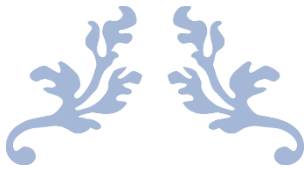


ACM 小學期實錄

2019 年 6 月 24 日~2019 年 7 月 7 日

張曉昊

2019-7-23

目录

Day 1

主题：绪论

时间：2019 年 6 月 24 日 星期一

主要内容：

排序算法：冒泡排序，快速排序

复杂度分析

STL：sort，堆，队列，vector，map

位运算

枚举：二进制枚举，折半枚举

题目：

+1	A. JM 的另一个 A + B Problem	基础运算
+	B. 寒域爷的奇怪代码	复杂度分析 答案提交
+	C. JM 的无限操作	排序
+	D. JM 的完美集合	折半枚举
+	E. JM 的俄罗斯套娃	模拟
+5	F. zzj & liaoy 想要去旅行	

关于ACM-ICPC

- ACM国际大学生程序设计竞赛（简称ACM-ICPC或ICPC）是由国际计算机协会（ACM）主办的，一项旨在展示大学生创新能力、团队精神和在压力下编写程序、分析和解决问题能力的年度竞赛。
- ACM：Association for Computing Machinery
- ICPC：International Collegiate Programming Contest

关于ACM-ICPC（续）

- 比赛期间，每队3人使用1台电脑在5个小时内使用C/C++/Java编写程序解决8到13个编程问题。有些比赛中可以使用Python。
- 程序完成之后提交裁判运行，运行的结果会判定为AC（正确）WA（错误）CE （编译错误） TLE（超时）MLE（超出内存限制）RE（运行错误）或其它结果并立刻被参赛队知晓。
- 每队在正确完成一题后，组织者将在其位置上升起一只代表该题颜色的气球。
- 可以多次提交，在AC之前每次错误的提交都将额外产生20min的罚时。
- 最终各队伍按照AC题数为第一关键字、罚时为第二关键字排行。

XJTUACM

基础知识

齐智

6月24日晚上 Update

- 1. 修正了一些错别字与拼写错误
- 2. 注意map的下标访问方式，mp[3]当mp中并没有一个为3的键时，会首先执行一次插入操作，默认插入值为0.
- 3. 修正了折半枚举例题的时间复杂度计算(45页)
  - 正确的算式是：2^(n/2) * (n/2 + log(2^(n/2))) = O(n * 2^(n/2))
  - 中间的“计算和”和“插入map“是并列的，应当加而不是乘。
- 4. 关于语言本身的特性推荐在文档中查询:
  - C++文档:http://www.cplusplus.com/reference/
  - 学有余力的同学可以了解一下STL里的string，set，unordered_map，lower_bound，upper_bound，deque，priority_queue，这些也比较常用
  - JAVA文档:https://docs.oracle.com/javase/8/docs/api/

课程内容

- 一些常识
- 排序算法
- 时间复杂度
- STL选讲
- 位运算与二进制枚举
- 折半枚举
- 初学者常见错误选讲
- 一些常识(二)

一些常识(一)

科学记数法，常用类型，引用

科学记数法

- xey 表示 x * 10^y 。
- 例如 1e5 表示 10^5， 2e9 表示 2 * 10^9 。

常用类型

- int 4字节 [-2^31，2^31 - 1] （很旧的标准可能不是）
- long long 8字节 [-2^63，2^63 - 1]
- unsigned long long 8字节 [0, 2^64 - 1]
- char 1字节 [-128，127]
- double 8字节, 约17位精度
- bool 占用空间1字节,实际只有1位

C++程序

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
int main()
{
        return 0;
}
```

引用

- C++的函数默认为值传递，可以在接受参数时加上&符号表示引用传递

```
void swap(int a,int b)
{
        int t=b;
        b=a;
        a=t;
}
```
没有任何效果

```
void swap(int &a,int &b)
{
        int t=b;
        b=a;
        a=t;
}
```
成功交换

排序算法

冒泡排序

- 思想：排序n趟，每趟找到一个最小的元素

- void bubble_sort(int arr[], int n)
- {

```
for(int i=0; i<len; ++i) //排序n趟
    for(int j=len-1; j>i; --j)
        if(arr[j]<arr[j-1])
            swap(arr[j],arr[j-1]);
```

- }

- 例子：6 5 4 3 2 1

快速排序

- 思想：对于乱序数组A，选取一个轴值x（通常为A[1]），把比x小的值放在左边，把比x大的值放在数组右边。然后再对左边的这些数和右边的这些数分别排序。

- 例子：
- 将[3 1 2 0 5 4]排序。
- [3 1 2 0 5 4]
- [1 2 0] 3 [5 4]
- [[0] 1 [2]] 3 [[4] 5]]
- 0 1 2 3 4 5

快速排序主算法

```
void quicksort(int l, int r){
    if (l >= r)return;
    int mid = partition(l, r);
    quicksort(l, mid - 1);
    quicksort(mid + 1, r);
}
```

快速排序划分算法：

通过交换元素使得任意时刻l左边的数一定比x小，r右边的数一定比x大

```
int partition(int l, int r){
    int x = a[l]; //轴值
    while (l < r){
        while (l < r && a[r] >= x)r--;
        if (l < r)a[l++] = a[r];
        while (l < r && a[l] <= x)l++;
        if (l < r)a[r--] = a[l];
    }
    a[l] = x; //此时l=r，处于中间位置
    return l;
}
```

时间复杂度

时间复杂度

- 如何评价一个算法的运行快慢？

- 大O记号：
- 1. 忽略较小项与常数
- 2. 可以变坏

- 分析如下程序的时间复杂度：

```
for (int i = 1; i <= n; i++){
    for (int j = 1; j <= n; j++)
        sum += i*j;
}
```

```
for (int i = 1; i <= n; i += i){
    for (int j = 1; j <= n; j++)
        sum += i*j;
}
```

```
for (int i = 1; i <= n; i ++){
    for (int j = 1; j <= n; j+=i)
        sum += i*j;
}
```

```
for (int i = 1; i <= n; i++){
    for (int j = i; j <= n; j++)
        sum += i*j;
}
```

```
for (int i = 1; i <= n; i += i){
    for (int j = i; j > 0; j--)
        sum += i*j;
}
```

调和级数和1+1/2+1/3+…+1/n = ln n+C

n*(lnn+C)=O(nlogn)

时间复杂度表

时间复杂度	10	10^2	10^3	10^4	10^5	10^6
n	10	100	1000	1万	10万	100万
nlogn	33	664	9965	13万	166万	1993万
n(logn)^2	255	8790	1.76万	289万	4269万	5.88亿
nsqrt(n)	31	1000	3.16万	100万	3162万	10亿
n^2	100	1万	100万	1亿	100亿	1万亿
n^3	1000	100万	10亿	1万亿	10^15	10^18
2^n	1000	10^30	inf	inf	inf	inf
n!	363万	inf	inf	inf	inf	inf

递归函数的时间复杂度

- 画出递归树

考虑求斐波那契第n项的程序

```
int fib(int n){
    if (n == 1)return 1;
    if (n == 2)return 1;
    return fib(n - 1) + fib(n - 2);
}
```

快速排序的时间复杂度:
1. 主要部分时间复杂度
2. 划分函数的时间复杂度

STL选讲

大幅简化编程难度

sort

- sort(数组首地址，数组尾地址)
- sort(数组首地址，数组尾地址，排序规则)

- 内部实现“主要”使用快速排序
- 默认排序规则是从小到大排序

排序规则

从大到小排序

```
bool cmp(int a,int b)
{
        return a>b;
}

int n=5, arr[M]={1,5,4,2,3};
sort(arr, arr+n, cmp);
```

可以认为cmp表示sort的询问，当cmp返回1时表示a和b无需交换

栈

- 后进先出，像一群人进入一个非常狭窄的山洞。
- 实现时，只需维护数据数组与栈顶指针.

```
stack<int> s;
s.push(x);
s.pop();
s.top();
s.size();

while(s.size())
{
    cout << s.top() <<endl;
    s.pop();
}
```

上述操作均为O(1)

队列

- 先进先出，像一群人排队打饭。
- 实现时，需要维护数据数组与队头指针，队尾指针。

```
queue<int> q;
q.push(x);
q.pop();
q.front();
q.size();

while(q.size())
{
    cout << q.front() <<endl;
    s.pop();
}
```

上述操作均为O(1)

vector

- 变长数组，适用于预先不知道开多大，或者想要轻快简便传参的情况。

```
vector<int> vc;

vc.push_back(x);
vc.size();
vc[i];
for(int i=0; i<vc.size(); ++i)
        vc[i];
sort(vc.begin(),vc.end())
```

map

- 映射,关联数组
- 以键值对的方式存储，可以使用类似数组的方式去操作。

```
map<int,int> mp;
mp[3] = 5;
mp[3]++;
printf(“%d”,mp);
for(auto p:mp)
        cout << p.first << p.second << endl;
```

位运算与二进制枚举

二进制枚举

位运算

- 11的二进制表示是1011, 5的二进制表示是0101
- 按位与 (与逻辑与&&的区别) 11&5=
- 按位或 (与逻辑或||的区别) 11|5=
- 按位异或 11^5=
- 左移 11<<1=
- 右移 11>>1=
- 按位取反 ~11=

- 如何将一个整数的某位 置成1?
- 如何将一个整数的某位 置成0?
- 如何将一个整数的某位 翻转?
- 如何判断一个整数的某位是否为1?

二进制枚举

- 问题：给定一个大小为 n 的集合，判断该集合中存在多少个子集，满足子集中所有元素之和恰好为 0 。(n ≤ 20)

二进制枚举

```
int n, a[n], ans = 0;
for (int mask=0, tot=1<<n; mask<tot; mask++) {
      int sum = 0;
      for (int i=0; i<n; i++) {
            if ((mask>>i) & 1) sum+=a[i];
      }
      if (sum == 0) ans ++;
}
```

折半枚举

折半枚举

- 问题：给定一个大小为 n 的集合，判断该集合中存在多少个子集，满足子集中所有元素之和恰好为 0 。
- 当 n ≤ 20 时，20 * 2^20 ≈ 2e7，可以暴力枚举。
- 当 n ≤ 35 时呢?

折半枚举

- 我们考虑将集合分成两半，大小分别为 n/2 和 n - n/2 ，记为 A 和 B 。
- 首先暴力枚举 A 的所有子集，得到 2^(n/2) 个和，将其记录下来。
- 然后暴力枚举 B 的所有子集，那么对于 B 的每个子集和 sumB ，我们只需要知道存在多少个 A 的子集和 sumA = - sumB 即可。

折半枚举

- 用二进制枚举求 A 的所有子集的时间复杂度为 n/2 * 2^(n/2) = O(n * 2^(n/2))。
- 如果对于每个 sumB ，都暴力查找所有的 sumA 看有哪些符合条件，那么时间复杂度为 (n - n/2) * 2^(n - n/2) * 2^(n/2) = O(n * 2^n)，事实上没有任何改善。
- 我们需要想办法加速查找满足条件的 sumA 的过程。

折半枚举

- 注意到我们只关心有多少个元素等于某个值，我们刚刚才学了map容器，它可以支持 O(logn) 的查找。我们只需要用map记录每个 sumA 值出现了多少次，就可以在枚举 B 的子集时进行快速查找。
- 设 mp 是一个 map<int, int> ，记 mp[x] 表示子集和为 x 的 A 的子集个数。那么我们在暴力枚举 A 的子集时就可以求出这个map。
- 时间复杂度为 2^(n/2) * (n/2 + log(2^(n/2))) = O(n * 2^(n/2))。虽然对于 n = 35 算出来仍然比较大，但是实际上是显然跑不满的。

折半枚举

- 通常，当遇到暴力枚举不可接受、而暴力枚举其中一部分就可以接受的数据范围时，该问题就有可能是需要运用折半枚举法的。
- 运用折半枚举的条件是很明确的。首先要保证枚举一半也是可以接受的，其次要保证我们可以通过保存半边解的方法加速查找，例如上述问题中就可以运用map解决，也就是说，充分利用了已知的条件：半个集合的所有可能的子集和。有些情况下就没有办法加速查找，那么折半枚举是无意义的。

POJ 2785

- 问题：给定四个大小为 n 的数组 A, B, C, D ，求存在多少个四元组 (a, b, c, d) ，满足 a ∈ A, b ∈ B, c ∈ C, d ∈ D 且 a + b + c + d = 0。(n ≤ 4000)。

POJ 2785

- 暴力枚举的时间复杂度为 O(n^4) ，显然是不可接受的。但该问题可以运用折半枚举解决。
- 首先暴力枚举 (a, b) ，那么我们会得到 n^2 个 a + b 的结果，我们将其存入map容器中。
- 然后暴力枚举 (c, d) ，那么我们对于每个 c + d 的结果，只需要检查存在多少个 (a, b) 满足 a + b = - (c + d) 即可。
- 时间复杂度为 O(n^2 * logn)。

一些常识(二)

字典序，模运算

字典序

- 有两个序列 a[1], a[2], …, a[n] 和 b[1], b[2], …, b[m]，称字典序 a < b，当且仅当存在一个下标 i ∈ [1, max(n, m)]，满足 a[i] < b[i]，且对于任意的下标 j ∈ [1, i)，满足 a[j] = b[j]。若 i 在序列 x 中越界，则认为 x[i] 无限小。
- 例如 a = “aabbc”, b = “aacbc”，则字典序 a < b 。
- 例如 a = [1, 2, 1], b = [1, 1, 6, 7]，则字典序 a > b 。
- 例如 a = [1, 1], b = [1, 1, 1]，则字典序 a < b 。
- 例如 a = [1, 2, 3], b = [1, 2, 3]，则字典序 a = b。

模运算

- 给定一个正整数p，任意一个整数a，一定存在等式 a = kp + r，其中k, r是整数，且 0 ≤ r < p。
- 称k为n除以p的商，r为n除以p的余数。
- 对于正整数p和整数a, b，定义取模运算：a % p（或 a mod p），表示a除以p的余数，即a % p = r。

模运算运算规则

- (a + b) % p = (a % p + b % p) % p
- (a − b) % p = (a % p − b % p) % p
- (a * b) % p = (a % p * b % p) % p
- a^b % p = (a % p)^b % p

- 其中a^b表示a的b次方。
- 注意，除法的模运算不满足该规则。
- 除法取模需要用到数论中的逆元，以后课程会讲。

C++11

- C++一个比较新的版本，有很多简化编程的特性，我们的OJ以及正式比赛中全部支持。（有些比较老旧的OJ不支持）
- 举例，map的遍历
- 旧版本的C++

```
for(map<int,int>::iterator it=mp.begin(); it!=mp.end(); ++it)
        cout << it->first  << " " << it->second << endl;
```

- C++11及以后

```
for(auto p:mp)
        cout << p.first " " << p.second << endl;
```

本机如果使用devC++或code::blocks的话，需要手动打开C++11开关

初学者常见错误

只列举了一部分

运算结果溢出

- 在C/C++中，int类型的有效表示范围为[-2^31, 2^31 - 1]。在旧标准中并非如此，但现在都是这样。
- 当运算结果超出int范围时，可以考虑使用long long int类型，取值范围为[-2^63, 2^63 - 1]。
- 可以用typedef缩减代码长度。
- 当运算结果过大且没有取模时，应当考虑使用大整数运算。

```
int a = 2e9;
int b = 2e9;
int c = a + b;
cout << c << endl;
```

```
long long a = 2e9 + 2e9;
typedef long long ll;
ll b = 1e18;
```

非法内存访问

- 非法内存访问会导致Runtime Error，即RE，包括但不限于：
  - 数组越界
  - 递归深度过大，即递归爆栈
  - 对野指针取值
- 但是数组越界不一定会导致RE，有时会访问到其他变量上，此时会很难debug，因此要格外小心。

```
#include <iostream>
using namespace std;
int a, b[2], c;
int main() {
    cin >> b[-1];
    cout << b[2];
    return 0;
}
```

大数组开成局部变量

- 在很多编译环境下，对局部栈空间的限制非常大，而竞赛题中经常用到1e5乃至1e6级别的数组，开成局部变量可能导致CE或RE，此时应当将其开成全局变量。

```
int main() {
    int a[1000000]; // 有6个0
    return 0;
} /*(X)*/
```

```
int a[1000000]; // 有6个0
int main() {
    return 0;
} /*(√)*/
```

读入效率问题

```
//Cin的优化
int main() {
    cin.sync_with_stdio(false); cin.tie(0);
    string str;
    cin >> str;
    return 0;
}
```

```
//更快的手写读入
inline int read(){
    int x=0,f=1;char ch=getchar();
    while(ch<'0'||ch>'9') {if(ch=='-')f=-1;ch=getchar();}
    while(ch>='0'&&ch<='9') {x=x*10+ch-'0';ch=getchar();}
    return x*f;
}
```

尽量不要使用float

- 因为竞赛中的浮点数运算经常有精度要求，而float的精度很低，在很多情况下都是不够用的，所以索性就不要去使用它，用double。

A. JM 的另一个 A + B Problem

内存限制：512 MiB

时间限制：1000 ms

题目描述

JM 觉得 XJTUOJ 上的题目 1000 : A + B Problem 太无趣了，就把它 duang~ duang~ 地变了一下：

给定两个正整数 A 和 B，求 $(A+B)\%P$ 的值，其中 $P=10^9+7$。

虽然这也是一个 A + B Problem，但是很容易出错哦？

也请不要使用高精度大整数运算，没有必要的。

输入格式

一行两个正整数，A 和 B。

输出格式

一行一个非负整数，表示答案。

数据范围与提示

$1 \le A,B \le 2^{64}-1$

解题思路

仅涉及基础的加减乘除及输入输出。注意数据范围，使用 unsigned long long 型。并利用公式 $(A+B)\%P=(A\%P+B\%P)\%P$ 进行计算。

代码

```
int main()
{
    ullong a, b, mn = 1000000007, ans;
    scanf("%llu%llu", &a, &b);
    a %= mn;
    b %= mn;
    ans = a + b;
    ans %= mn;
    printf("%llu", ans);
    return 0;
}
```

备注

代码省略了头文件和数据类型的定义。若无特别说明，均按照下文定义：

```
#include <bits/stdc++.h>
using namespace std;
typedef long long llong;
typedef unsigned long long ullong;
```

请注意，若使用的是 Visual Studio，则不能使用万能头文件 bits/stdc++.h。以下是在 Visual Studio 中常用的头文件的声明：

```cpp
#include <map>
#include <set>
#include <ctime>
#include <cmath>
#include <queue>
#include <stack>
#include <vector>
#include <string>
#include <cstring>
#include <cstdio>
#include <cstdlib>
#include <iostream>
#include <algorithm>
#include <functional>
```

B. 寒域爷的奇怪代码

题目类型：答案提交

题目描述

寒域爷最近突然精神抖擞，写下了大篇代码。刚刚学完时间复杂度的 JM 试图在一晚上弄懂寒域爷在代码中蕴含的深邃的思想，但是它却难以分析寒域爷的代码。于是愤怒的 JM 把你抓来了，要你为他讲解寒域爷代码的时间复杂度，如果他听不懂的话就把你喂给寒域爷制造更多的代码。

代码 1

该函数出自两年前的 ACM 小学期课程中，用于实现求从 1 加到 x 的和。

```cpp
int sum(int _Xx) {
    int ret = 0;
    for (int i = 1; i <= _Xx; i++) ret += i;
    return ret;
}
```

代码 2

代码 1 被寒域爷魔改后，大大降低了时间复杂度，仍然是用于实现求从 1 加到 x 的和。

```cpp
inline int sum(int _Xx) { return (1 + _Xx) * _Xx / 2; }
```

代码 3

这一坨代码出自刚刚接触 C 语言的小透明寒域，该函数用于将 arr 数组中的 num 个元素排序。

```cpp
void bubble_sort(int* arr, int num) {
for (int i = 0; i < num - 1; i++)
        for (int j = 0; j < num - i - 1; j++)
            if (arr[j] < arr[j + 1]) arr[j] ^= arr[j + 1] ^= arr[j] ^= arr[j + 1];
}
```

代码 4

在寒域爷偶然间翻看自己以前写的破烂代码的时候，他将代码 3 修改成了下面的样子。该函数用于将 arr 数组中的 num 个元素排序。

```cpp
#include <queue>
void heap_sort(int* arr, int num) {
    std::priority_queue<int> heap;
    for (int i = 0; i < num; i++) heap.push(arr[i]);
    for (int i = 0; i < num; i++) {
        arr[i] = heap.top();
        heap.pop();
    }
}
```

代码 5

这个代码出自寒域爷闲暇时候用来测试 Node.js 语言和 C 语言的速度差距。该函数用于求斐波那契数列的第 x 项。

```cpp
long long int fibonacci(int _Xx) {
    if (_Xx <= 2) return 1;
    return fibonacci(_Xx - 1) + fibonacci(_Xx - 2);
}
```

代码 6

JM 看到寒域的垃圾代码后劈头盖脸训斥了寒域一顿，并修改成了下面的函数。该函数用于求斐波那契数列的第 x 项($x < 1000$)。

```cpp
#include <cstring>
long long int fibonacci(int _Xx) {
    static long long int save[1000];
    static bool first = true;
    if (first) first = false, memset(save, -1, sizeof(save));
    if (save[_Xx] != -1) return save[_Xx];
    if (_Xx <= 2) return 1;
    return save[_Xx] = fibonacci(_Xx - 1) + fibonacci(_Xx - 2);
}
```

代码 7

该函数用于计算最长的每个字母最多出现一次的子字符串的长度，出处已经不可考。

```cpp
#include <string>
int lengthOfLongestSubstring(std::string s) {
    int i, j, len = 0, maxx = 0;
    bool sign[30];
    memset(sign, 0, sizeof(sign));
    for (i = 0, j = 0; j < s.length(); ++j) {
        while (sign[s[j] - 'a'] == 1) {
            sign[s[i] - 'a'] = 0;
            ++i;
        }
        sign[s[j] - 'a'] = 1;
        maxx = std::max(maxx, j - i + 1);
    }
    return maxx;
}
```

输出格式

这是一道提交答案题，你应该把你分析的结果按照下面的格式直接写在提交中。

如果满足多个时间复杂度，请选择最早出现的那一个。

时间复杂度	你要写的内容
$O(1)$	1
$O(\log n)$	logn
$O(n)$	n
$O(n\log n)$	nlogn
$O(n^2)$	n2
$O(n^3)$	n3
$O(n^4)$	n4
$O(2^n)$	2n
$O(n!)$	n!

数据范围与提示

注意本题的提交方式和通常的题目不同，仔细看，如你所见，在提交界面左侧有**代码 1.usr**，**代码 2.usr**，……，**代码 7.usr** 这 7 个小节，你需要**在每一小节都填写答案**，填写的内容与题目描述中的 7 份代码一一对应。

答案

```
n
1
n2
nlogn
2n
n
n
```

C. JM 的无限操作

内存限制：512 MiB

时间限制：2000 ms

题目描述

JM 有一个长为 n 的数组 a，数组的下标为 $1,2,\ldots,n$。他可以进行任意多次如下这种操作：

• 对于两个正整数 i 和 j $(1 \le i,j \le n)$，如果此时数组 a 满足 $a_i + a_j$ 是偶数，则交换 a_i 和 a_j 的值。

现在 JM 想知道，他在任意次操作之后，能得到的字典序最小的数组是什么？

输入格式

第一行一个正整数 n，表示数组的长度。

接下来一行 n 个正整数，表示数组中的元素。

输出格式

输出一行 n 个正整数，表示能得到的字典序最小的数组。

数据范围与提示

$1 \le n \le 2 \cdot 10^5$

$1 \le a_i \le 10^9$

解题思路

每次操作可以交换两个奇数或两个偶数，对应序号位置奇偶性始终不变。故对奇数和偶数分别存储和排序，在对应位置记录奇偶性，并在排序后放置最小的奇数或偶数即可。

代码

```
int n, tmp, tp;
vector<int> nums[2], jud;
int main()
{
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &tmp);
        nums[tmp % 2].push_back(tmp);
        jud.push_back(tmp % 2);
    }
    for (int i = 0; i < 2; i++)
        sort(nums[i].begin(), nums[i].end());
    for (int i = 0; i < n; i++)
    {
        tp = jud.at(i);
        printf("%d ", nums[tp].at(0));
        nums[tp].erase(nums[tp].begin());
    }
    return 0;
}
```

D. JM 的完美集合

内存限制：512 MiB

时间限制：2000 ms

题目描述

JM 是强迫症晚期患者，他执着于集合的完美性，他认为，如果一个集合中所有元素之和恰好为 0，那么这个集合是完美的。

给定一个大小为 n 的可重复集合 A，判断该集合存在多少个非空子集是完美的。

可重复集合的意思是集合中可能有重复的元素，此时也可能存在多个相同的满足条件的子集，要按照多个来算。

输入格式

第一行一个正整数 n，表示集合大小。

第二行 n 个整数，表示集合中的元素。

输出格式

一行一个非负整数，表示满足条件的非空子集的个数。

数据范围与提示

$1 \le n \le 35$

$-5 \cdot 10^7 \le A_i \le 5 \cdot 10^7$

解题思路

$1 \le n \le 35$，使用折半枚举，复杂度为 $O(2^{n/2})$，是可以接受的。

由于要求是非空子集，故答案要减去 1。

代码

```
int n, tmp, rge, ans = -1;
vector<int> nums;
map<int, int> ct;
void schl(int lc, int ansn)
{
    if (lc == rge)
    {
        ct[ansn]++;
        return;
    }
    schl(lc + 1, ansn);
    schl(lc + 1, ansn + nums.at(lc));
}
void schr(int lc, int ansn)
{
    if (lc == n)
    {
        if (ct.count(-ansn))
            ans += ct[-ansn];
        return;
    }
    schr(lc + 1, ansn);
    schr(lc + 1, ansn + nums.at(lc));
}
```

```cpp
int main()
{
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &tmp);
        nums.push_back(tmp);
    }
    rge = n / 2;
    schl(0, 0);
    schr(rge, 0);
    printf("%d", ans);
    return 0;
}
```

E. JM 的俄罗斯套娃

内存限制：512 MiB

时间限制：2000 ms

题目描述

神奇的 JM 最近沉迷玩俄罗斯套娃不能自拔，于是他灵机一动，想搞一个俄罗斯套娃数组，但他的电脑昨天被寒域爷浇了一罐肥宅快乐水，现在还在抢修中，所以他万不得已只能找你帮忙了。

俄罗斯套娃数组的定义如下（下面简称为套娃数组）：

1. 套娃数组是一个不定长数组，初始为空。数组的下标为 $1,2,\ldots,n$，其中 n 为数组长度。
2. 套娃数组中的元素可以是正整数，也可以是一个套娃数组。
3. 套娃数组允许在任意位置插入和删除一个元素；如果插入的元素是一个套娃数组，则插入的肯定是一个空的数组；如果删除的元素是一个套娃数组，则该数组内部的所有元素都将被一并删除，不留痕迹。
4. 此外，我们有一个唯一的光标，光标会指向一个套娃数组，初始时指向最外层的套娃数组。

作为一个 dalao，qz 是非常严谨的，他会用一些输入的操作来检测你帮 JM 实现的俄罗斯套娃数组是否正确。有四种操作：

1 p v：表示在当前所指的数组的下标 **p** 的**后面**插入元素。如果 $p=0$，则表示插入到数组的最前面。如果当前数组为空，则保证 $p=0$。如果 $v=0$，则表示插入一个空的套娃数组，否则表示插入一个正整数 **v**。

2 p：表示删除当前所指的数组在下标 **p** 处的元素。

3 p：表示将光标指向当前所指的数组在下标 **p** 处的数组，即改变光标所指的数组。如果下标 **p** 处是一个正整数，则不做任何事，光标不变。如果 $p=0$，则将光标退回到上级数组。如果当前已经指向最外层，则不做任何事，光标不变。

4：输出当前所指的数组中的所有元素，具体输出格式参考下面的样例。注意，如果元素是一个套娃数组，则只输出一个**[]**，不递归输出里面嵌套的内容。

保证输入的所有操作都是合法的，即保证：

1. 对于操作 1 有 $0 \le p \le n$ 且 $v \ge 0$ 。
2. 对于操作 2 有 $1 \le p \le n$。
3. 对于操作 3 有 $0 \le p \le n$。
4. 不会出现 $1,2,3,4$ 以外的操作。

输入格式

第一行一个正整数 q ，表示有 q 次操作。

接下来 q 行，每行表示一个操作，输入格式如前文中题目所述。

输出格式

对于每次操作 4，输出一行，表示当前所指的数组。

数据范围与提示

$1 \le q \le 50$

$1 \le v \le 100$ ，除了插入一个套娃数组时 $v = 0$ 。

解题思路

本题为一道模拟题，数据范围不大。对每个数组分配一个编号，即其在 vector 中的下标，并存储活动数组的编号。插入一个套娃数组时，在该位置存储套娃数组的编号。改变光标时，将目标数组改为活动数组即可。为实现返回上层数组，每一个数组需存储其父数组的编号。

代码

```
struct vdta
{
    int tpe, dta, vno;
};
struct vit
{
    vector<vdta> dta;
    int par;
};
int n, tp, p, v, now = 0, ed = -1;
vector<vit> vrr;
map<int, int> ct;
vector<vdta>::iterator isl;
void addVector(int now)
{
    vrr.push_back(vit({ vector<vdta>(), now }));
    ed++;
}
int main()
{
    scanf("%d", &n);
    addVector(0);
    for (int i = 0; i < n; i++)
```

```
    {
        scanf("%d", &tp);
        switch (tp)
        {
        case 1:
            scanf("%d%d", &p, &v);
            isl = vrr.at(now).dta.begin() + p;
            if (v)
                vrr.at(now).dta.insert(isl, vdta({0, v, now }));
            else
            {
                addVector(now);
                vrr.at(now).dta.insert(isl, vdta({ 1, 0, ed }));
            }
            break;
        case 2:
            scanf("%d", &p);
            vrr.at(now).dta.erase(vrr.at(now).dta.begin() + p - 1);
            break;
        case 3:
            scanf("%d", &p);
            if (p)
                now = vrr.at(now).dta.at(p - 1).vno;
            else
                now = vrr.at(now).par;
            break;
        case 4:
            printf("{");
            for (unsigned j = 0; j < vrr.at(now).dta.size(); j++)
            {
                if (j > 0)
                    printf(" ");
                vdta tpdta = vrr.at(now).dta.at(j);
                if (tpdta.tpe)
                    printf("[]");
                else
                    printf("%d", tpdta.dta);
            }
            printf("}\n");
        }
    }
    return 0;
}
```

F. zzj & liaoy 想要去旅行

内存限制：512 MiB

时间限制：2000 ms

题目描述

zzj 和 liaoy 都很喜欢旅行，他们现在打算计划新的一次旅行。在一番讨论之后他们将景点根据种类的不同，例如海滩、山脉、历史人文景观等等分成了 4 类，同时对于每一个想去的景点他们都有一个权值。我们将这 4 类景点分别记为 A,B,C,D 四个数组，数组中记录的是该类的各个景点的权值。

现在 zzj 和 liaoy 打算在 4 类景点中各选出一个，使得选出的 4 个景点中，权值最高的在 A,B 两种里面，权值最低的在 C,D 两种里面。也就是说，要选出一个四元组 (a,b,c,d)，满足 $a\in A,b\in B,c\in C,d\in D$ 且 $\max(a,b,c,d)=a \text{ or } b, \min(a,b,c,d)=c \text{ or } d$。

现在 zzj 和 liaoy 想知道他们一共有多少种符合要求的选法。

输入格式

输入一共四行，每行表示一个种类景点的期望值信息，依次是 A,B,C,D 四类。

每行开始是一个整数 n，表示该类景点一共有 n 个。接下来包含 n 个数字，表示这 n 个景点的期望值。

输出格式

输出一行一个非负整数，表示一共有多少种选法。

数据范围与提示

$1\le n\le 1000$

$1\le A_i,B_i,C_i,D_i\le 2000$

解题思路

对所有景点按照期望值进行排序，之后选择一个最大值，若该最大值为 A,B 种景点，则固定最大值后选择最小值，若该最小值为 C,D 种景点，则在区间内计算可能的组合数。复杂度为 $O(n^2)$，可以接受。在排序时，相同期望值的情况下，为不遗漏任何一种情况，选择最大值应优先选择 A,B 种景点，选择最小值应优先选择 C,D 种景点。

答案可能为 $n^4\le 1000^4$，故应使用 llong 型存储答案。

该种做法的复杂度为 $O(n^2)$。本题的最优解据说是 $O(n)$，但是本人并不知道如何实现。

代码

```
llong st, nd, stn[4], sz, tpn[2], tpst[4], ans = 0;
vector<prdd> dta;
bool cmp(prdd a, prdd b)
{
    if (a.second == b.second)
        return a.first < b.first;
    return a.second > b.second;
}
int main()
{
```

```
    for (int i = 0; i < 4; i++)
    {
        scanf("%lld", &st);
        stn[i] = st;
        for (int j = 0; j < st; j++)
        {
            scanf("%lld", &nd);
            dta.push_back(prdd(i, nd));
        }
    }
    sz = dta.size();
    sort(dta.begin(), dta.end(), cmp);
    for (int i = 0; i < sz; i++)
    {
        stn[dta.at(i).first]--;
        if (dta.at(i).first > 1)
            continue;
        for (int j = 0; j < 4; j++)
            tpst[j] = stn[j];
        for (int j = sz - 1; j > i; j--)
        {
            tpst[dta.at(j).first]--;
            if (dta.at(j).first < 2)
                continue;
            tpn[0] = tpst[(dta.at(i).first + 1) % 2];
            tpn[1] = tpst[(dta.at(j).first + 1) % 2 + 2];
            ans += tpn[0] * tpn[1];
        }
    }
    printf("%lld", ans);
    return 0;
}
```

备注

代码省略了数据类型的定义。若无特别说明，均按照下文定义：

```
typedef pair<int, int> prdd;
```

Day 2

主题：贪心

时间：2019 年 6 月 25 日 星期二

主要内容：

贪心思想

相邻交换法

题目：

+	A. jwp 的采购之旅	贪心
+2	B. jwp 的区间游戏	贪心
+2	C. jwp 的排队难题	相邻交换法
+2	D. JM 的祖传零钱箱	贪心，位运算
+5	E. JM 的西伯利亚特快专递	贪心
+7	F. JM 的睡前故事	

人类本质
因此大多数贪心始于直觉
一个贪心的正常过程:

- 1.出于直觉或经验想出的贪心方案
- 2.尝试证明
- 3.若证明失败,进入4，否则进入6
- 4.尝试构造反例，若构造成功，进入7，否则进入5
- 5.大胆/时间紧迫/不关心时间，进入6，否则开始待机
- 6.写出代码并AC
- 7.GG

- 你有T元钱去买书，一共n本书，第i本a_i元，最多买几本
- solution:按照价格从小到大买即可

- 消灭小怪兽：有两只怪兽，血量分别为$A, B \leq 10^9$，攻击力分别为$C, D \leq 10^9$,第i回合你的攻击力为i，但每次攻击只能打一只怪兽，而且每个回合,场上还活着的怪兽都会打你一下，问在你将两只怪兽全部打死前，你最少掉多少血
- 最少的总攻击次数是一定的。为什么呢??
- 也就是说后死的那只怪兽总共打你的次数是一定的，那么也就是说你要尽量早打死第一只怪兽，所以先使劲冲着那个血少的打即可，最后一下可能浪费，从前面的回合里调整一下

- 有$n \leq 10^5$种火柴棒，第i种火柴棒长度2^i,有a_i个，求这些火柴棒最多拼成多少个三角形
- solution:要么是等边三角形，要么是底边比腰短的等腰三角形，当考虑到第i种时，前面的剩余零散火柴棒只能做底边，因此可以不区分，下面说贪心策略：从短到长一种一种考虑。一直维护一个变量表示前面（比当前短的）还剩下多少零散火柴棒。枚举到i时，先尽量跟前面更短的零散的配对形成等腰（1短2长），然后再形成等边（三长）。还剩下的加入零散的火柴棒集合。时间复杂度$O(n)$。证明??

- 清一色：作为一个染手狂魔，石户霞平生最大的愿望就是清一色八连庄，因此她发明了一套只有一种花色的麻将，这样她就可以实现自己把把清一色的梦想了！由于这套麻将只有一种花色，因此我们可以直接用数字给每张牌标号，比如“五筒”可以记为“五”。这副牌里有$1, 2, 3....n$这$n \leq 10^5$种牌，每种牌有$a_i \leq 10^9$张，问这套牌最多能组成几套面子。面子：每套面子由三张牌组成，面子有两种1.顺子：形如$(i-1, i, i+1)$的连续三张牌2.刻子：形如(i, i, i)的三张同样的牌
- solution:这题根本就不是贪心嘻嘻嘻，但看起来真的很像贪心有没有!
- 发现一个重要的事情：同一种顺子不可能有超过三个!! 然后就可以dp了，复杂度$O(n)$，之后会讲

- 模板题：给出n个区间$[l_i, r_i]$，要求每个区间里选一个点，使这些点互不重复。
- solution:按照左端点为第一关键字，右端点为第二关键字把区间排序，依此考虑，每次尽量往左选点，时间复杂度$O(nlogn)$。证明??

- 三维题：给出n个空间中的平行六面体，要求每个平行六面体里选一个点，使这些点互不重复。
- 魂淡你们难道没有发现三个维度的限制和方案都是完全独立的??
- solution:对三个维度按照上一题依此做一遍即可

- ddd的期末：众所周知ddd在一学期中上课数不会超过5节，因此他必须在考试周里进行预习才能使自己不被退学，考试周有$n \leq 10^3$天，有$m \leq 10^3$门考试,第i门考试是在考试周的第t_i天，占p_i个学分，ddd预习能力全校顶尖，可以保证用某个没有安排考试的日期预习某门科目使该科目获得满分，如果一场考试来了但他没有为这个科目预习，那么他就会获得0分，如果一天安排了考试那么ddd就不能在这一天进行预习，求ddd最多获得多少学分

- solution:将所有科目按照p从大到小排序，按照顺序贪心地安排预习，对于每一科，都要让这一科的预习时间尽可能晚，如果发现某一科前面的所有日期都安排满了那这科就GG了，贪心正确性证明？？
- bonus:当数据范围是10^6时，可以使用并查集来维护每一天的前面的第一个没有安排考试和预习的日期，关于并查集会在数据结构课程中讲到
- bonus2:另一种做法，使用map等STL即可做到$O(nlogn)$

- 人渣的本愿：众所周知妹子杰是个人渣，经常冒充妹子然后与妹子们假装进行橘里橘气的约会，XJTU中有$n \leq 10^3$个妹子，第i个妹子可以在$[l_i, r_i] \leq 10^9$中的某一天和妹子杰约会,妹子杰与第i个妹子约会能获得p_i的喜悦值，由于妹子杰的肾不太好，因此他每天最多约会一次，由于妹子杰是喜新厌旧的人渣，每个妹子他最多只会约一次，那么妹子杰能够获得的总的最大喜悦值是多少呢？
- solution:这个题用到了两个贪心，首先，将妹子们按照p从大到小排序，然后我们可知，从大到小安排，如果当前的妹子不能和和前面已选的最优妹子集合同时安排，那么就不选她，这样一定是最优的，那么如何得到当前妹子集合是否可行呢？等价于n个区间，要求每个区间里选一个点，使得点互不重复是否可行，这个问题比较简单，复杂度$O(n^2)$
- bonus:如果加上$l_i \leq l_{i+1}, r_i \leq r_{i+1}$这个条件，那么本题可以用硬核的线段树技巧做到$O(nlogn)$，这里就不再讲了

- 相邻交换法：出现在各种关于"寻找最优排列"的问题中。
- 问题引入：排序
- 一个数组是排序好的，等价于这个数组中无逆序对
- 等价于这个数组中无相邻逆序对
- 等价于不存在一种临项交换的方法，使得逆序对数变小
- 等价于不存在相邻两项满足$a_i > a_{i+1}$
- 另一个例子：排序不等式

- general definition：
- 设$f(A)$是关于排列A的函数，$<$是找到的一个关于元素的全序关系，满足交换相邻项时，若$a_i < a_{i+1}$,则交换后f一定减小，否则交换后f一定增加,那么,$f(A)$最小的排列即为将所有元素按照这个$<$排序所得的排列
- 意义：将最优化$f(A)$的问题转化为找到满足条件的$<$关系的问题，之后就是普通的排序即可

- 给出$n \leq 10^5$个一次函数，$f_i(x) = a_i x + b_i$，给出x，求$f(f(f(....(x))))$的最大值，所有数字都是大于1的正整数
- solution:按照$\frac{a-1}{b}$的大小定义小于号，排序即可

- 有$n \leq 10^3$堆果子，每堆果子重a_i，每一次你可以用$a_i + a_j$的体力把i, j两堆果子合并，新的这堆果子重$a_i + a_j$,重复上述操作直到只有一堆果子，问最小用多少体力将所有果子合并
- solution:每次选择重量最小的果子进行合并，暴力复杂度$O(n^2)$
- extension:哈夫曼树
- bonus:使用数据结构，如堆/平衡树(map),可以将时间复杂度优化到$O(nlogn)$
- bonus2:如果只有相邻的果子堆才能合并呢？？

- 给定$n \leq 10^5$个区间，选最多的点使得每个区间最多一个点，假设这些区间的并为$[1, m], m \leq 10^9$
- solution:将所有区间$[l, r]$变成标记，然后把这些标记打在相应的l处，从左往右扫，并一直维护一个"界"，以及“新界”，初始的"界"和“新界”均为0，一旦你扫到比"界"大的地方，就立刻选点，并把界更新成新界，在小于界的范围往右走不能选点，但要一直更新新界

- 有$n \leq 10^5$个怪兽，打每个怪兽都必须先掉$a_i > 0$血再回$b_i > 0$血，你可以按照任意顺序打怪兽，请问你初始最少需要多少血
- solution:这是一道需要仔细思考的应用相邻交换法的经典题目。
首先，$b - a$大于零的一定比$b - a$小于零的要先打
在$b - a$大于零的怪物中，a越小越先打
在$b - a$小于零的怪物中，b越大越先打。证明？？
这样我们就能得到一个全序关系$<$,排序即可，$O(nlogn)$

A. jwp 的采购之旅

内存限制：512 MiB

时间限制：2000 ms

题目描述

负责任的 jwp 到商店采购校赛的奖品，总共有 n 件奖品要购买，第 i 件商品的价格为 A_i，但是 jwp 发现自己囊中羞涩，他只有 m 元钱，为了下一次来采购能够轻松一些，他决定这次买尽可能多的奖品。现在他希望你帮忙计算此次最多买多少奖品。

输入格式

第一行两个整数 n,m，分别代表奖品数量 n 和 jwp 带的钱数 m。

接下来一行 n 个正整数，分别代表第 i 件商品的价格 A_i。

输出格式

输出一行一个非负整数，表示 jwp 此次最多能买的奖品数量。

数据范围与提示

$1 \le n \le 10^5$

$1 \le m \le 10^6$

$1 \le A_i \le 10^3$

解题思路

贪心，为使购买的奖品最多，每次均选择价格最低的奖品。

代码

```
int n, m, tmp, ans = 0;
vector<int> mns;
int main()
{
    scanf("%d%d", &n, &m);
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &tmp);
        mns.push_back(tmp);
    }
    sort(mns.begin(), mns.end());
    while (mns.size() >= 0)
    {
        if (mns.at(0) > m)
            break;
        ans++;
        m -= mns.at(0);
        mns.erase(mns.begin());
    }
    printf("%d", ans);
```

```
    return 0;
}
```

B. jwp 的区间游戏

内存限制：512 MiB

时间限制：2000 ms

题目描述

某天，jwp 兴致勃勃的玩起了区间游戏，有 n 个区间 $[a_i, b_i]$，jwp 要在数轴上选择 k 个点，使它们覆盖所有的区间，即对任意一个区间 $[a_i, b_i]$，总是存在某个点 t 满足 $a_i \le t \le b_i$。

现在，jwp 想知道对眼前的 n 个区间，最小的 k 是多少。

输入格式

第一行一个正整数 n。

接下来 n 行，每行两个正整数 a_i, b_i。

输出格式

输出一行一个正整数 k 表示答案。

数据范围与提示

$1 \le n \le 10^6$

$1 \le a_i \le b_i \le 10^6$

解题思路

考虑贪心，将每个区间按照其右端点从小到大排序。将最小的未覆盖区间的右端点放置一个点，这是这个点能放的最右的位置，能最大化地利用该点。此时，所有左端点小于此点坐标的区间均已被覆盖，不需要继续考虑。

代码

```
int n, tpa, tpb, ans = 0, xnow = -1;
vector<prdd> itv;
bool cmp(prdd a, prdd b)
{
    return a.second < b.second;
}
int main()
{
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
    {
        scanf("%d%d", &tpa, &tpb);
        itv.push_back(prdd(tpa, tpb));
    }
    sort(itv.begin(), itv.end(), cmp);
    for (unsigned i = 0; i < itv.size(); i++)
    {
```

```
        if (itv.at(i).first <= xnow)
            continue;
        ans++;
        xnow = itv.at(i).second;
    }
    printf("%d", ans);
    return 0;
}
```

C. jwp 的排队难题

内存限制：512 MiB

时间限制：2000 ms

题目描述

jwp 竟然兼职做导游！某天，jwp 带着 n 个游客去动物园游玩，但是可恶的园长不予许 jwp 统一买票，所以所有的游客只能排队买票。

每一个游客的脾气和办事效率是不一样的，意味着每一个游客有一个暴躁值 a_i 和一个买票需要花费的时间 b_i（即他从开始买票到买票结束所花费的时间），这个游客最终的不愉快值可以表示为他的暴躁值 a_i 乘以排队及买票花费的总时间（即从第一个游客开始买票，到第 个游客买票结束）。假设每一个游客买完票会立刻离开，同时下一个游客开始买票。jwp 决定安排游客的买票顺序，从而使所有游客的不愉快值之和最小化。他想知道这个最小值是多少。

输入格式

第一行一个正整数 n 。

接下来 n 行，每行两个正整数 a_i, b_i 。

输出格式

一行一个整数表示答案。

数据范围与提示

$1 \le n \le 10^5$

$1 \le a_i, b_i \le 100$

解题思路

考虑相邻交换法。对于两个人 i 和 j ，其总不愉快值应比交换后小，设已用时间为 t，有 $a_i(t+b_i)+a_j(t+b_i+b_j) \le a_j(t+b_j)+a_i(t+b_i+b_j)$，展开，消去得 $a_j b_i \le a_i b_j$。据此进行排序即可。

最后一人的不愉快值可能为 $a_i b_i n \le 10^9$，总不愉快值可能为 $a_i b_i n^2 \le 10^{14}$，故应使用 llong 型存储答案。

代码

```
int n, tpa, tpb;
llong ans = 0, tot = 0;
vector<prdd> trs;
```

```cpp
bool cmp(prdd a, prdd b)
{
    return a.second * b.first < a.first * b.second;
}
int main()
{
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
    {
        scanf("%d%d", &tpa, &tpb);
        trs.push_back(prdd(tpa, tpb));
    }
    sort(trs.begin(), trs.end(), cmp);
    for (unsigned i = 0; i < trs.size(); i++)
    {
        tot += trs.at(i).second;
        ans += trs.at(i).first * tot;
    }
    printf("%lld", ans);
    return 0;
}
```

D. JM 的祖传零钱箱

内存限制：512 MiB

时间限制：2500 ms

题目描述

JM 的太爷爷的太爷爷的太爷爷……的太爷爷喜欢攒零钱，并把攒零钱作为祖训传承至今，现在已经攒了整整一箱子的硬币，JM 清点了一下发现箱子里一共有 n 枚硬币。众所周知，古代的钱币放到现在都很值钱，所以这些硬币的价值并不相同。不过，由于寒域爷施了石志魔法，这些硬币的价值都是 2 的幂次。也就是说，对于任意的第 i 枚硬币，其价值 a_i 一定满足 $a_i = 2^d$，其中 d 是某个非负整数。

现在，石乐志的 JM 想拿这些硬币去买菜，他有 q 种想买的菜。因为石乐志，JM 认为硬币的价值并不重要，而硬币的数量是最重要的。所以在买菜之前，他想事先知道，对于每种价格为 b_j 的菜，他最少需要用掉多少枚硬币才能将其买下。另外，由于 JM 有强迫症，他不希望他在买菜的时候还需要找零。

他希望你帮他分别计算买这 q 种菜的所需硬币数。注意，是分别计算，每种菜的求解之间是互相独立的。

输入格式

第一行两个正整数 n 和 q，表示 JM 的硬币数和 JM 想买的菜的种类数。

接下来一行 n 个正整数 a_i，表示每个硬币的价值。输入保证 $a_i = 2^d$，其中 d 是某个非负整数。

接下来 q 行，每行一个正整数 b_j，表示每种菜的价格。

输出格式

输出 q 行，每行一个正整数，表示买下每种菜所需的最少硬币个数。如果无论如何也无法凑齐价格（注意不能找零），则输出 -1 。

数据范围与提示

$1 \le n, q \le 2 \cdot 10^5$

$1 \le a_i, b_j \le 2 \cdot 10^9$

解题思路

按照数位存储、管理硬币数量。因为小面值货币可以转化为大面值货币，而大面值货币不能转化为小面值货币，故对每种菜从高位向低位进行查询：若该位有足够的对应面值货币，则继续进行查询，否则将不足的部分视作两倍的下一位货币需求数，继续进行查询。若查询结束后仍需要货币，则不能凑齐价格。

代码

```
llong n, q, a, b = 0, pw[32], cts[32], ans = 0;
int main()
{
    pw[0] = 1;
    for (int i = 1; i < 32; i++)
        pw[i] = 2 * pw[i - 1];
    scanf("%lld%lld", &n, &q);
    for (llong i = 0; i < n; i++)
    {
        scanf("%lld", &a);
        for (int j = 0; j < 32; j++)
            if (pw[j] == a)
            {
                cts[j]++;
                break;
            }
    }
    for (llong i = 0; i < q; i++)
    {
        scanf("%lld", &a);
        for (int j = 31; j >= 0; j--)
        {
            if (a >> j & 1)
                b++;
            if (cts[j] >= b)
            {
                ans += b;
                b = 0;
            }
            else
```

```
            {
                ans += cts[j];
                b -= cts[j];
                b *= 2;
            }
        }
        if (b == 0)
            printf("%lld\n", ans);
        else
            printf("-1\n");
        b = 0;
        ans = 0;
    }
    return 0;
}
```

E. JM 的西伯利亚特快专递

内存限制：512 MiB

时间限制：2000 ms

题目描述

今天 JM 收到了一份来自西伯利亚的特快专递，里面装了一个字符串 s，仅包含小写英文字母。于是 JM 决定跟 qz 和寒域爷一起玩一个游戏：

JM 手里拿着字符串 s，qz 手里拿着字符串 t，寒域爷手里拿着字符串 u，初始时 t 和 u 均为空。有两种操作：

将字符串 u 的首部字符取出，插入字符串 t 的尾部。

将字符串 t 的尾部字符取出，插入字符串 u 的尾部。

JM，qz 和寒域爷随意地进行上述两种操作，直到字符串 s 和 t 均为空时才停止。现在 JM 想知道，他们能得到的字典序最小的字符串 u 是什么。

注意，对于一个字符串 s，其长度为 n，其中字符的下标为 $1,2,\ldots,n$，则我们定义 s 的首部字符为 s_1，尾部字符为 s_n。

输入格式

一行一个字符串 s。

输出格式

一行一个字符串 u 表示答案。

数据范围与提示

$1 \le |s| \le 10^5$

解题思路

贪心地选择最小的字母放入 u 中，并将中途的字母放入 t 中，之后将 t 中所有小于等于剩余串中最小的字母 $min(s)$ 的字母从后往前地拿出，直至遇到比 $min(s)$ 大的字母位置，重复以上流程。最终获得的答案是字典序最小的。

代码

```
int st = 0, chrc[26];
string str, ans = "", strc, tmp;
char getMinC()
{
    for (int i = 0; i < 26; i++)
        if (chrc[i] > 0)
            return i + 'a';
    return 0;
}
int main()
{
    cin >> str;
    for (int i = 0; i < str.length(); i++)
        chrc[str.at(i) - 'a']++;
    for (int i = 0; i < str.length(); i++)
        if (str.at(i) == getMinC())
        {
            ans += str.at(i);
            chrc[str.at(i) - 'a']--;
            for (int j = tmp.length(); j > 0; j--)
                if (tmp.at(j - 1) <= getMinC())
                {
                    ans += tmp.at(j - 1);
                    tmp.erase(j - 1, 1);
                }
                else
                    break;
        }
        else
        {
            tmp += str.at(i);
            chrc[str.at(i) - 'a']--;
        }
    for (int i = tmp.length(); i > 0; i--)
        ans += tmp.at(i - 1);
    cout << ans;
    return 0;
}
```

F. JM 的睡前故事

内存限制：512 MiB

时间限制：2000 ms

题目描述

在某个不可描述的一维世界，寒域爷和 qz 爷瓜分了世界上的所有城市。现在这个世界也迎来了科技革命，为了不在科技上落后于对方，电缆的建设成为了寒域爷和 qz 爷统治下的两大国的第一要务。

我们可以把一维世界视为 x 轴，上面有 n 个城市，每个城市都有一个整数的 x 坐标，所有城市的坐标互不相同。每个城市的归属是下列三中情况之一：

1. 城市标记为 `S`，表示它属于寒域爷的 The Empire of Skyair 。
2. 城市标记为 `F`，表示它属于 qz 爷的 The Republic of LittleFall 。
3. 城市标记为 `N`，表示它是中立的 Neutral 。

电缆是沿 x 轴直线铺设的，每条电缆可以连接两个城市，但是并不会把它铺设路线上途经的所有城市都连接起来。例如有三座城市的坐标为 $x_1 = 1, x_2 = 2, x_3 = 3$，那么如果铺设一条从 1 号城市到 3 号城市的电缆，并不会把城市 2 也接入。但是电缆的通信是具有传递性的，也就是说，如果希望把这三座城市连接起来，显然我们可以先连接 1 和 2，再连接 2 和 3，这样是最节约的。此外，电缆的长度就是两端城市的坐标之差。

寒域爷和 qz 爷的两大国并不总是和睦相处的，例如有时候他们会进行商业竞争云云，所以在铺设电缆时，寒域爷和 qz 爷不得不考虑对方掐断己方的电缆的可能性。但是出于核平主义的考虑，中立城市不应当被斗争所波及，因此电缆的铺设就有了限制条件：

1. 站在寒域爷的角度，电缆的铺设方案必须保证：即使将 qz 爷治下的所有城市和与之相连的电缆都删除，只考虑寒域爷的城市和中立的城市，任意两座城市之间仍然可以通过电缆直接或间接通信。
2. 站在 qz 爷的角度，电缆的铺设方案必须保证：即使将寒域爷治下的所有城市和与之相连的电缆都删除，只考虑 qz 爷的城市和中立的城市，任意两座城市之间仍然可以通过电缆直接或间接通信。

铺设的电缆的数量是任意的，连接的城市也是任意的，不受任何限制。电缆本身没有归属，也就是说，两个人不需要各自独立地去铺，我们考虑的是电缆的总长度。

现在，寒域爷和 qz 爷正在进行商业谈判，商讨如何铺设电缆，既能满足两人的需求，又能保证铺设电缆的长度最短。因为他们很懒，所以他们把你绑了过去，如果你给不出答案，就会把你丢进海里喂鱼。请你给出满足条件所需的最短的电缆铺设总长。

输入格式

第一行一个正整数 n，表示城市个数。

接下来 n 行，每行一个整数 x_i 和一个大写字母 c_i，表示每座城市的坐标和归属。保证 $c_i \in \{'S', 'F', 'N'\}$。

保证城市是按 x 坐标递增的顺序输入的。

输出格式

输出一行一个正整数表示答案。

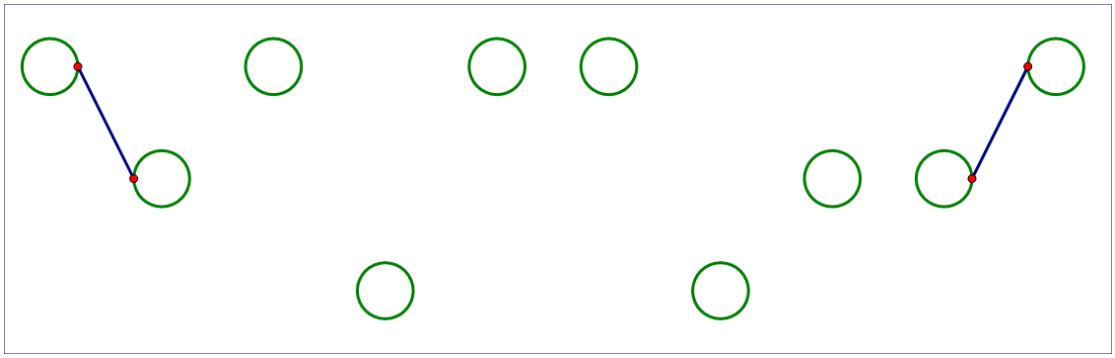
数据范围与提示

$2 \le n \le 2 \cdot 10^5$

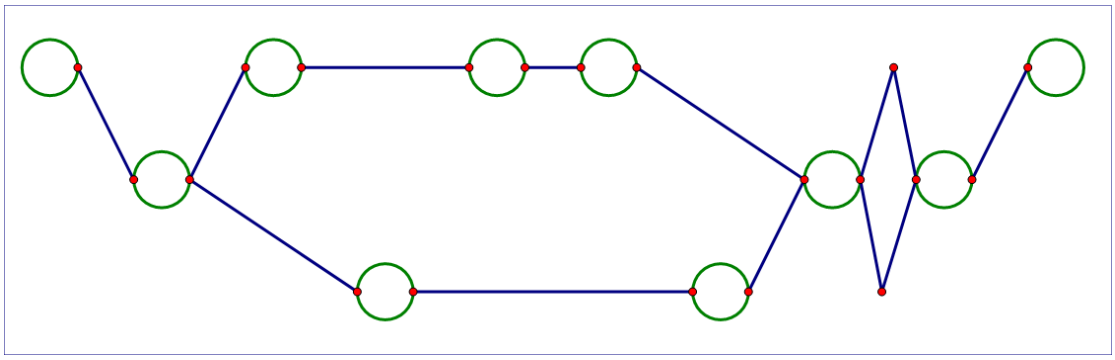
$|x_i| \le 10^9$

解题思路

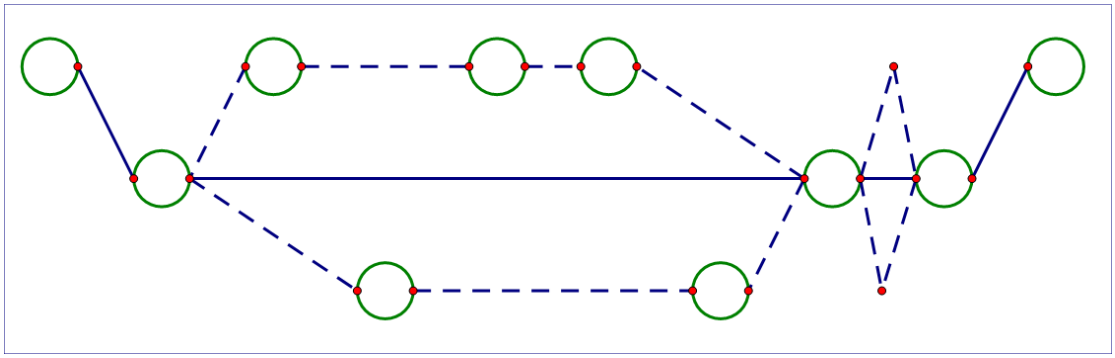
将题目中的城市与线路用图片表示：



图上每一个圆圈代表一座城市，上面一行的城市记为 S，下面一行的城市记为 F，中间一行的城市（中立城市）记做 N。考虑两个中立城市的连接：要保证敌对的双方都能通过各自的线路连接到两个中立城市，可以将中立城市通过两方的两条支路顺序连接：



容易发现，还有另一种连法，即直接将两个中立城市通过中间一行的道路进行连接，这一条连线是公共的：



同时，容易发现，为保证所有己方与中立城市连通，上下两支路最多各节省一条线路。考虑贪心，节省最长的两条线路。若节省的部分小于两中立城市的直接距离，则维持前种连法，否则改为后种连法，并节省这两条最长线路。对于每两个中立城市组成的区间都这样考虑，就能得到最优解。

起点至第一个中立城市和最后一个中立城市至终点直接各自顺序连接即为最优，无需进行上文的讨论。

代码

```
struct city
{
    llong x;
    char tpy;
    llong cmspa;
};
int n;
llong ans, mspa = llinf, nwn = 0;
vector<city> lne, tpc[3];
int typeInt(char c)
{
    switch (c)
    {
    case 'F': return 0;
    case 'S': return 1;
    case 'N': return 2;
    }
    return -1;
}
int main()
{
    cin.sync_with_stdio(false);
    cin.tie(0);
    cin >> n;
    for (int i = 0; i < n; i++)
    {
        llong tmpx;
        char tmpt;
        cin >> tmpx >> tmpt;
        city tmpc = { tmpx , tmpt, 0 };
        tpc[typeInt(tmpt)].push_back(tmpc);
        if (tmpt == 'N')
        {
            tpc[0].push_back(tmpc);
            tpc[1].push_back(tmpc);
        }
    }
    for (int y = 0; y < 2; y++)
    {
        for (int i = 0; i < tpc[y].size(); i++)
        {
            if (i > 0)
                if (tpc[y].at(i).x - tpc[y].at(i - 1).x > mspa)
                {
```

```
                    ans += mspa;
                    mspa = tpc[y].at(i).x - tpc[y].at(i - 1).x;
                }
                else
                    ans += tpc[y].at(i).x - tpc[y].at(i - 1).x;
            if (tpc[y].at(i).tpy == 'N')
            {
                if (nwn > 0)
                    tpc[2].at(nwn).cmspa += mspa;
                mspa = 0;
                nwn++;
            }
        }
        if (nwn > 0)
            ans += mspa;
        nwn = 0;
        mspa = llinf;
    }
    for (int i = 1; i < tpc[2].size(); i++)
        if (tpc[2].at(i).cmspa > tpc[2].at(i).x - tpc[2].at(i - 1).x)
            ans += tpc[2].at(i).x - tpc[2].at(i - 1).x;
        else
            ans += tpc[2].at(i).cmspa;
    printf("%lld", ans);
    return 0;
}
```

备注

代码省略了常数的定义。若无特别说明，均按照下文定义：

```
const int dinf = 0x7fffffff;
const llong llinf = 0x7fffffffffffffff;
```

由于涉及字符的读入，故可使用 cin 读入。在使用 cin 读入时，为保证读入速度，使用了以下两个语句。若如此，则不能使用 scanf 函数。

```
cin.sync_with_stdio(false);
cin.tie(0);
```

Day 3

主题：二分与分治

时间：2019 年 6 月 26 日 星期三

主要内容：

快速幂：快速幂，矩阵快速幂

二分查找

分治法：归并排序，逆序对

题目：

+	A. jwp 爱跑步	快速幂
+3	B. jwp 的数学是跟谁学的	归并排序，逆序对
+	C. 小秋的完美数字	二分查找，交互
+	D. 寒域爷的兔子	矩阵快速幂，斐波那契数列
+	E. JM 的星系战争	二分查找，Special Judge
+	F. JM 的特快列车	最大最小值
+15	G. JM 的毒瘤题	分治
−	H. nocriz 参加学习运动	

快速幂

快速幂解决的问题是：给定a和n，求a^n的值。算法的时间复杂度为$O(\log n)$。
快速幂的思想是基于分治的，它的正确性依据只是幂数的运算法则，非常简单。
如果n是偶数，可以把a^n转化为$(a^2)^{n/2}$。
如果n是奇数，可以把a^n转化为$a*a^{n-1}$。
显然，这可以将运算次数降低到$O(\log n)$的级别。

至于代码实现，我们初始化答案为$res=1$，它的意义是令$res=a^0$。
当n是偶数时，我们令$a=a^2$，$n=n/2$；当n是奇数时，我们令$res=res*a$，$n=n-1$。不断重复直到$n=0$为止。由于当n为奇数时，$n-1$必为偶数，因此我们可以将奇数和偶数规约成一种情况：每次如果n为奇数那么将$res=res*a$,然后$a=a^2$. $n=\lfloor \frac{n}{2} \rfloor$。
快速幂的原理也可以从二进制的角度去分析，但这样就比较复杂了。
首先我们要知道，任意一个正整数x都可以表示为$x=\sum_{i=0}^{\log n} c_i*2^i$的形式，其中$c_i=0,1$。例如13的二进制表示为1101，
即$13=2^3+2^2+0+2^0$。
根据上面的原理，我们可以对a^n的幂次n进行分解，即：
$a^n=a^{\sum_{i=0}^{\log n} c_i*2^i}=\prod_{i=0}^{\log n} a^{c_i*2^i}$。
这证明了运算次数可以降低到$O(\log n)$的级别。


```
int pow(int a,int b)
{
        int ans=1;
        while(b)
        {
                if(b&1)ans=1LL*ans*a%p;
                a=1LL*a*a%p;
                b>>=1;
        }
        return ans;
}
```


求$(a+b\sqrt{c})^n$，所有数字值域10^9

- $(a_1+b_1\sqrt{c})(a_2+b_2\sqrt{c})=a_1a_2+b_1b_2c+(a_1b_2+a_2b_1)\sqrt{c}$
- 永远是$x+y\sqrt{c}$的形式，我们开一个专门的结构体存储这种数字即可，然后进行普通快速幂

矩阵快速幂

求n阶矩阵$A=a_{ij}$的p次方，矩阵乘法一次$O(n^3)$
时间复杂度$O(n^3 logp)$

求斐波那契数列的第n项，$n\le 10^{18}$

- $\begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix}=\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} F_{n-1} \\ F_{n-2} \end{pmatrix}$
- $\begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix}=\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-1}\begin{pmatrix} 1 \\ 0 \end{pmatrix}$
- 矩阵快速幂即可，复杂度$O(logn)$

请求出下列数列的第$n\le 10^{18}$项

- $a_n=3a_{n-1}+4a_{n-2}+8a_{n-3}$
- $a_n=3a_{n-1}+4a_{n-2}+8n^2+6n+9$
- $a_n=3S_{n-1}+5a_{n-1}+6, S_n=\sum_{i=1}^{n} a_i$
- $a_n=a_{n-1}a_{n-2}$
- bonus:最后一个题目中,$a_n=a_1^{F(n-1)}a_2^{F(n-2)}$
- 哪些递推数列不能用这类方法呢？

- 有$n\le 10^{18}$个砖块你可以给每个砖块染红/黄/蓝/绿四种颜色之一，求有多少种染法使得最终红色砖块数量和绿色砖块数量都是偶数
- 从左往右染色，设a_i, b_i, c_i分别表示当染了i个方块后，红绿皆为偶，红绿皆为奇，红绿其中一个为奇的方案数，则:

$$\begin{cases} a_n=2a_{n-1}+c_{n-1} \\ b_n=2b_{n-1}+c_{n-1} \\ c_n=2a_{n-1}+2b_{n-1}+2c_{n-1} \end{cases}$$

用矩阵转移即可

- 有一个5阶矩阵A,求$S_n = \sum_{i=1}^{n} A^i$
- 矩阵是可以分块的，对于这个，我们可以有：

$$\begin{pmatrix} S_n \\ A^{n+1} \end{pmatrix} = \begin{pmatrix} I & I \\ 0 & A \end{pmatrix} \begin{pmatrix} S_{n-1} \\ A^n \end{pmatrix}$$

这样就又可以愉快地矩阵快速幂了

二分查找

概念

二分查找可以在$O(\log n)$的时间内在一个有序连续数组内查找某个元素，比朴素线性查找要快得多，是一种极为常用的基础算法。

原理

最基本的二分查找解决的问题是：考虑一个长为n的有序数组a，给定一个数x，要求查找是否存在一个元素a_i，满足$a_i = x$，如果存在则给出其下标i。算法的时间复杂度为$O(\log n)$。

二分查找的思想是基于分治的，但是因为它极为常用、变化多端、却又很简单，所以它通常被从分治中剥离开来。它的正确性要求保证数组a是有序的。

为便于描述，设数组下标为$1..n$，按升序排列。

算法维护一个窗口$[left, right]$，初始时窗口为$[1, n]$。

令$mid = (left + right)/2$，容易发现，因为数组是有序的，

当$a_{mid} < x$时，显然我们所求的下标满足$i \in [mid + 1, right]$。正确性很显然，我们假设$i \in [left, mid - 1]$，那么数组就不是有序的了，显然矛盾。同理，当$a_{mid} > x$时，满足$i \in [left, mid - 1]$。

注意到每次窗口大小都会减半，因此最多$O(\log n)$次就可以将窗口大小缩减到1，此时必定能$O(1)$地判断出该元素是否存在。

二分查找非常简单，并且极为常用，在有序条件下我们经常可以用二分来代替枚举，降低时间复杂度。


```
int binary_search(int left, int right, int val)
{
        while (left <= right)
        {
                int mid = (left + right) / 2;
                if (a[mid] == val)
                        return mid;
                if (a[mid] < val)
                        left = mid + 1;
                else
                        right = mid - 1;
        }
}
```


二分查找的扩展非常多，比如在一个升序数组中，求出比x大的最小的元素，或是求出比x小的最大的元素，等等。

事实上，这些扩展都是大同小异的，只要你真正地理解了二分查找的原理，这些变体你可以轻松地自己实现。如果你还不能轻松做到，说明你对二分查找的领悟还不够到位。

下面举几个常见的例子。为便于描述，设数组下标为$1..n$，按升序排序

1. *lower_bound*：求出第一个满足$a_i \geq x$的元素的下标i，若不存在则返回$n + 1$。
2. *upper_bound*：求出第一个满足$a_i > x$的元素的下标i，若不存在则返回$n + 1$。


```
int lower_bound(int left, int right, int val)
{
        int res = left;
        while (left <= right)
        {
                int mid = (left + right) / 2;
                if (a[mid] < val){//upper_bound <=
                        left = mid + 1;
                        res = left;
                }
                else  right = mid - 1;
        }
        return res;
}
```


有些题目并不是显式地存在一个可排序的数组，但是它的答案具有单调性，此时我们仍然可以运用二分策略。

最简单的例子，比如我们有一个比较复杂的单调函数$f(x) = t$，给定t的值，让你求它的解x。

设答案的取值范围为$x \in [left, right]$，那么每次对于当前值mid，计算出$f(mid)$的值，和t进行比较，据此修改窗口。其余的部分就是上述的二分查找了，如出一辙。

这个策略俗称为二分答案+check。只要问题的解具有单调性，我们就可以二分答案，然后我们通常会写一个check函数（因为这个过程比较复杂，直接整个塞进二分的代码里会非常恶心），据此进行二分。

- 1.初始上限下限需要足够
- 二分求n的平方根，$(n < m)$,$[0, \lceil \sqrt{m} \rceil]$
- 2.$l = mid + 1, r = mid$。防止死循环,永远当作在找后缀区间的首位
- 3.小数二分时，谨慎设置*eps*,必要时可用循环替代
- 4.擅用STL中的各种*lower_bound*和*upper_bound*

```cpp
//normal binary search
while(l<r)
{
mid=(l+r)>>1;
if(check(mid))l=mid+1;
else r=mid;
}
//decimal binary search
while(r-l>eps)
//binary search using 'for'
for(i=0;i<30;i++)
```



```cpp
//binary search in STL
pos=lower_bound(a,a+n)-a;
pos=lower_bound(vec.begin(),vec.end())-vec.begin();
iter=mp.lower_bound(x);
iter=set.lower_bound(x);
```


- 数轴上有$n \leq 10^5$个点，点的坐标x_i的绝对值小于10^9,你有$k \leq 10^5$块等长的木板，你想要将这些点全部覆盖，那么这些木板最短需要多长呢？
- solution:显然，长度为x的木板若能盖住，则长度大于x的木板也能盖住，所以可以对长度x二分，每次看当前长度是否可行，可行性检查是$O(n)$的贪心问题，复杂度$O(nlogn)$

经典应用1：最大最小值，最小最大值

- 农民约翰有用$C \leq 10^5$只牛，然后他有$n \leq 10^5$个隔间，每个隔间都有自己的坐标位置，如何安排把牛安排进隔间才能使牛两两之间距离的最小值最大，求最大的最小值
- 二分最小值，等价于两两之间的距离都大于这个最小值，贪心从前往后放置牛即可。一个小case，当$C > n$时答案为0

- 有$n \leq 10^5$个人从左到右站成一排，每个人都属于一个组织，一共只有$k \leq 10^5$个组织，用数字$x \in [1,k]$表示，第i个人属于组织s_i,一开始每个人手里都有一颗宝石，现在依次发出了$q \leq 10^5$条指令，每条指令形如$a_i \in [1,k], b_i \in [0,1]$,表示每个$a_i$组织的人都会把他手中的所有宝石交给他左边/右边的人，如果是第一个人往左传或者第n个人往右传，那么很遗憾的是他的所有宝石就丢掉了,当所有指令结束后，问还有多少宝石没有丢掉
- 核心观察：如果把所有宝石编号，那么它们从左到右的相对顺序不会变化！因此我们只需二分从左往右第一个没有丢掉的宝石和从右往左第一个没有丢掉的宝石即可，时间复杂度$O(qlogn)$

- 给定一个第一象限的$n \leq 10^5$个点的凸多边形，第i个点为$A_i = (x_i, y_i)$。共$q \leq 10^5$次询问，每次给定第二象限的一个点P，求这个点与这个凸多边形的两条切线
- 将这个凸多边形的x坐标最小/最大的点拿出来，将多边形分割成上凸壳和下凸壳，可知两条切线对应的切点一定分别在上凸壳和下凸壳上。
- 设切点为Ai,$(A_i - A_{i-1}) \times (A_i - P)$与$(A_{i+1} - A_i) \times (A_i - P)$符号相反，这个点可以二分得到。时间复杂度$O(qlogn)$

分治法

- 求一个数列中的最大子段和？
- 暴力枚举$O(n^2)$个区间，时间复杂度$O(n^2)$
- 分治:$solve(l,r)$
  在$[l,r]$中，设$mid = (l+r)/2$.有三类子段：
- 1.在$[l, mid]$中的子段，这一部分可以在$solve(l, mid)$中分治处理
- 2.在$[mid+1, r]$中的子段，这一部分可以在$solve(mid+1, r)$中分治处理
- 3.包含mid的子段，我们只需要从mid开始，往左和往右分别求出最大的前缀/后缀，加起来即可。
- 基本流程：$solve(l,r)$
- $solve(l, mid)$
- $solve(mid+1, r)$

- 从mid找出最大的前缀/后缀并合并。
- 时间复杂度：$T(n) = 2T(\frac{n}{2}) + O(n) = O(nlogn)$
- 用前缀和算法可以优化到$O(n)$

- 初始有一张$2^n \times 2^n$的棋盘，$n \leq 10$,一开始这个棋盘上有一个位置(x, y)被占据了，其他地方都是空的，请设计一种方案，用L型多米诺骨牌铺满所有空着的地方
- $solve(n, P)$
  将棋盘分成四块A, B, C, D，把含有P的那块称为A
  放置一块骨牌使得B, C, D都被占据了一个格子
  $solve(n-1, P1)$
  $solve(n-1, P2)$
  $solve(n-1, P3)$
  $solve(n-1, P4)$
- 注意边界条件，当$n = 0$时直接返回

归并排序与逆序对

归并排序：利用分治法排序
算法流程：调用$solve(l, r)$

- $A = solve(l, mid)$,将数组前一半排序
- $B = solve(mid + 1, r)$,将数组后一半排序
- $return \quad merge(A, B)$,将A, B两个有序数组按照顺序归并成新的有序数组，这一步是$O(r - l)$的
- $T(n) = 2T(\frac{n}{2}) + O(n) = O(nlogn)$


```
int n,i,a[105],tmp[105];
void solve(int l,int r)
{
        int hd1,hd2,hd,i;
        if(l>=r)return;
        int mid;
        mid=(l+r)>>1;
        solve(l,mid);
        solve(mid+1,r);
        hd1=l;hd2=mid+1;hd=l;
        while(hd1<=mid||hd2<=r)
        {
                if(hd1>mid)tmp[hd++]=a[hd2++];
```



```
                else
                {
                        if(hd2>r)tmp[hd++]=a[hd1++];
                        else
                        {
                        if(a[hd1]<a[hd2])
                                tmp[hd++]=a[hd1++];
                        else
                                tmp[hd++]=a[hd2++];
                        }
                }
        }
        for(i=l;i<=r;i++)a[i]=tmp[i];
}
```


逆序对，数组中的(i, j)对数满足$i < j$且$a_i > a_j$
在归并排序的$merge$步骤，每加入一个A中元素，都将比这个元素小的B中元素的数量加入答案。(为什么呢？)

- 给定一个长度为n的数组，判断这个数组是否满足：对于任意一个子段，这个子段中至少有一个只出现了一次的元素。
- solution：首先有一种思路是：找到一个在数组中只出现了一次的元素，设为k，然后判断$[l, k]$和$[k + 1, r]$即可。但如果只是朴素的从头到尾找这个k的话，最坏时间复杂度是$O(n^2)$
- 进行如下改进，从两头向中间同时找这个k，时间复杂度的递推式变为$T(n) = maxT(k) + T(n - k) + minO(k), O(n - k)$,则$T(n) = O(nlogn)$这个时间复杂度可以用决策树证明

经典应用2：0/1分数规划

- 有$n \leq 10^5$块金属，第i块金属有体积和质量分别是v_i, m_i,求一个金属块的子集,要求子集中至少有$k \leq 10^5$块金属，使得它们的质量和除以体积和最大，只需要输出最大的质量和与体积和的比即可
- 分数规划：$\lambda = \frac{\sum m}{\sum v}$,即$\sum(m - \lambda v) = 0$
- 也就是说，对于一个λ,如果存在一个子集使得$\sum(m - \lambda v) > 0$则最终答案一定大于λ，否则最终答案小于等于λ,因此我们二分λ即可，每次将一个物品的价值设置为$m - \lambda v$，然后取前k大价值的物品加起来，看是否大于零即可。

- 有$n \leq 10^5$个电池，初始电量为$a_i \leq 10^9$，每分钟消耗$b_i \leq 10^9$，要求在接下来的$m \leq 10^5$分钟内每个电池都必须保证电量非负。你可以买一个充电宝，花x元钱可以买到一个每分钟充x电量的充电宝，每分钟你都可以用这个充电宝给某个电池充电(必须充电完整的一分钟)。每个电池容量无上限，问最少花多少钱完成任务？若无论如何都无法保持住n个电池电量，输出-1
- 二分x是显然的，然后我们需要验证每分钟充x电量的充电宝是否能满足需求，贪心即可，在每一分钟，我们用充电宝优先给**最快即将没电**的电池续命，这个可以用数据结构(map)维护最快即将没电的电池，时间复杂度$O(mlognlogA)$
- bonus:请试着提出一个时间复杂度$O(mlogA)$的算法

A. jwp 爱跑步

内存限制：512 MiB

时间限制：1000 ms

题目描述

（梦里）jwp 成为了一名出色的田径运动员，他每次跑步的时候喜欢喝肥宅快乐水，假设他最开始打算跑的距离是 1 米，每喝一瓶肥宅想跑的距离就乘以 A，他一共带了 B 瓶肥宅快乐水，环形操场的长度为 C。

假设 jwp 从 0 米处出发顺时针开始跑，他想知道，跑完步之后距离起点的距离（从起点顺时针到终点）的距离。

输入格式

一行三个整数 A,B,C。

输出格式

输出一行一个整数表示答案。

数据范围与提示

$1 \le A,B,C \le 10^9$

解题思路

求$A^B\%C$，使用快速幂即可。

代码

```
int A, B, C, ans = 1;
int main()
{
    scanf("%d%d%d", &A, &B, &C);
    while (B > 0)
    {
        if (B & 1)
            ans = (1LL * ans * A) % C;
        A = (1LL * A * A) % C;
        B = B >> 1;
    }
    printf("%d", ans);
    return 0;
}
```

B. jwp 的数学是跟谁学的

内存限制：512 MiB

时间限制：1000 ms

题目描述

jwp 的数学是跟谁学的？jwp 的数学当然是跟体育老师学的！一年一度的运动会表演又要到来了，n 个同学站成了一排，如果有两个人 A 和 B，A 站在 B 前面，但是比 B 个子要高，

那这就是不和谐的，则不和谐度 +1 。体育老师要 jwp 帮忙计算一下当前队列的总不和谐度，你能帮帮他吗？

输入格式

第一行一个整数 n 。

接下来一行 n 个整数，分别表示从前往后每个人的身高 H_i 。

输出格式

输出一行一个整数表示答案。

数据范围与提示

$1 \leq n \leq 10^6$

$1 \leq H_i \leq 10^6$

解题思路

本题中的不和谐度就是逆序对数。使用归并排序。在合并两个序列时，每从前一个序列取一个数时，答案就会增加等同于后一个序列取出的数的数目。最终的答案即为排序前的序列的逆序对数。

代码

```
int n, tmp;
llong ans = 0;
vector<int> hi, tphi;
void sol(int l, int r)
{
    if (r - l <= 1) return;
    int hdl, hdr, hd, mid = (l + r) >> 1;
    sol(l, mid);
    sol(mid, r);
    hd = hdl = l;
    hdr = mid;
    while (hd < r)
        if (hdl >= mid)
            tphi.at(hd++) = hi.at(hdr++);
        else if (hdr >= r)
        {
            tphi.at(hd++) = hi.at(hdl++);
            ans += hdr - mid;
        }
        else if(hi.at(hdl) > hi.at(hdr))
            tphi.at(hd++) = hi.at(hdr++);
        else
        {
            tphi.at(hd++) = hi.at(hdl++);
            ans += hdr - mid;
        }
```

```
    for (int i = l; i < r; i++)
        hi.at(i) = tphi.at(i);
}
int main()
{
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &tmp);
        hi.push_back(tmp);
        tphi.push_back(tmp);
    }
    sol(0, n);
    printf("%lld", ans);
    return 0;
}
```

备注

代码中的 sol 函数处理的是区间 $[l,r)$。

C. 小秋的完美数字

内存限制：512 MiB

时间限制：1000 ms

题目类型：交互

题目描述

这是一道**交互题**，交互题是指题目的输入会根据你的输出所产生变化。

小秋想和你玩个游戏，你需要猜出小秋最喜欢的那个数字是多少。每次小秋会告诉你，你猜测的数字是大了、小了、还是正好。

你最多可以给出 100 次猜测，猜测方式如下：

输出一行，一个整数，表示你所猜测的数字。

之后系统会给出一个整数 x ($x \in \{1,-1,0\}$)，表示猜测结果，其中 1 表示你猜的数大了，-1 表示你猜的数小了，0 表示你猜的数是对的。

输入格式

初始不会有任何输入。

交互输入的方式：

每当你进行一次猜测，就可以从标准输入流中读取一个整数，表示猜测结果，其中 1 表示你猜的数大了，-1 表示你猜的数小了，0 表示你猜的数是对的。

当读入一个 0 后，应当**立即终止程序**，此后**不要再继续读入**，因为你已经猜对了。

输出格式

交互输出：

最多可以猜测 100 次，对于每次猜测，输出一行，包含一个 $[1,10^9]$ 以内的整数并换行。

注意：

每次输出之后，必须刷新输出缓冲区，否则将会得到 TLE 的结果。

记得输出时要输出换行符。

刷新输出缓冲区的方式如下：

C/C++:　　`fflush(stdout);`

Java:　　`System.out.flush();`

Python:　　`stdout.flush();`

Pascal:　　`flush(output);`

其它语言请参见各自文档。

数据范围与提示

答案在 $[1,10^9]$ 以内。

猜测次数必须在 100 次以内。

每次输出之后必须刷新缓冲区。

（如果你没有思路，可以和身边的男/女朋友玩一下这个游戏）

解题思路

二分查找。

代码

```
int l = 1, r = 1000000001, ans = -2, mid;
int main()
{
    while (true)
    {
        mid = (l + r) >> 1;
        printf("%d\n", mid);
        fflush(stdout);
        scanf("%d", &ans);
        if (ans == 0)
            break;
        if (ans > 0)
            r = mid;
        else
            l = mid;
    }
    return 0;
}
```

备注

代码处理的是区间 $[l,r)$。

D. 寒域爷的兔子

内存限制：512 MiB

时间限制：1000 ms

题目描述

寒域爷养了一窝兔子。

众所周知，兔子是繁殖能力相当强的动物，古有谚语归纳生物的繁殖速度：

兔一鼠二猫三狗四猪五羊六牛七马八人九骆驼十。

因此我们可以得知兔子每月就能繁殖一窝。

假设小兔子需要一个月长大，在第二个月就能繁殖。每对成年兔子每月能产下一对小兔子。现在（第一个月）寒域爷有一对小兔子。

但是 cyy 开了辣兔肉连锁机构，由于黄老仙的毒奶，他自己养殖的兔子远远达不到订单要求。为了万恶的资本，cyy 会把寒域爷的兔子偷走以凑成他的辣兔肉订单，且对小兔子和成年兔子一视同仁。cyy 在第 M 个月要完成他的订单，每份订单需要用掉 K 只兔子（不论公母），这时 cyy 会从寒域爷那里偷走尽可能多的整订单。

问寒域爷在被 cyy 偷之后还能剩下多少只兔子。

输入格式

一行两个正整数 M 和 K 。

输出格式

一行一个整数表示剩下多少只兔子。

数据范围与提示

$1 \le M \le 2^{64}-1$

$1 \le K \le 2^{15}$

一对兔子指一公一母两只。

解题思路

第 M 个月拥有的兔子对数即为斐波那契数列的第 M 项 $F(M)$，只数即为 $2F(M)$。由斐波那契数列的性质，可得：$\begin{pmatrix}F(n+1)\\F(n)\end{pmatrix}=\begin{pmatrix}1&1\\1&0\end{pmatrix}\begin{pmatrix}F(n)\\F(n-1)\end{pmatrix}=\begin{pmatrix}1&1\\1&0\end{pmatrix}^{n}\begin{pmatrix}F(1)\\F(0)\end{pmatrix}$。定义矩阵乘法，并应用矩阵快速幂即可。

代码

```
struct mt22
{
    int a, b;
    int c, d;
};
struct mt12
{
    int a, c;
};
```

```
int k, ans;
mt22 upd = { 1, 1, 1, 0 };
mt12 fans = { 1, 0 };
ullong m;
mt12 mip(mt22 x, mt12 y)
{
    mt12 ans;
    ans.a = (1LLU * x.a * y.a + 1LLU * x.b * y.c) % k;
    ans.c = (1LLU * x.c * y.a + 1LLU * x.d * y.c) % k;
    return ans;
}
mt22 mip(mt22 x, mt22 y)
{
    mt22 ans;
    ans.a = (1LLU * x.a * y.a + 1LLU * x.b * y.c) % k;
    ans.b = (1LLU * x.a * y.b + 1LLU * x.b * y.d) % k;
    ans.c = (1LLU * x.c * y.a + 1LLU * x.d * y.c) % k;
    ans.d = (1LLU * x.c * y.b + 1LLU * x.d * y.d) % k;
    return ans;
}
mt22 fpow(mt22 x, ullong b)
{
    mt22 ans = { 1, 0, 0, 1 };
    while (b > 0)
    {
        if (b & 1)
            ans = mip(x, ans);
        x = mip(x, x);
        b >>= 1;
    }
    return ans;
}
int main()
{
    scanf("%llu%d", &m, &k);
    upd = fpow(upd, m);
    fans = mip(upd, fans);
    ans = (2 * fans.c) % k;
    printf("%d", ans);
    return 0;
}
```

备注

两个结构体的含义：mt22 即为矩阵 $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$，mt12 即为矩阵 $\begin{pmatrix} a \\ c \end{pmatrix}$。

E. JM 的星系战争

内存限制：512 MiB

时间限制：1000 ms

评测方式：Special Judge

题目描述

今天，带恶人 JM 掀起了一场超大规模的星系战争，为了阻止疯狂的 JM，你需要前往 n 个星球去招募勇士（为了部落！）。这些星球的编号为 $1, 2, \ldots, n$，地球（也就是你的初始所在）为 1 号星球。为了掩人耳目（？）你必须按照编号顺序依次访问这些星球，然后再回到地球上，也就是按照 $1 \to 2 \to \cdots \to n \to 1$ 的顺序访问。

当你驾驶飞船访问每个星球时，你的飞船都需要在星球上着陆然后再起飞，这两个过程都是需要耗费大量的燃料的。因为这是一艘神(zhi)奇(zhang)的飞船，它只在着陆和起飞时有大量消耗，至于在星球之间飞行所耗费的燃料，少到可以忽略不计。

对于编号为 i 的星球，着陆所需的燃料为 M_1/a_i，起飞所需的燃料为 M_2/b_i，其中 M_1 和 M_2 分别表示着陆或起飞**之前**飞船的总重。飞船的总重可以表示为 $M = m + f$，其中 m 表示飞船的净重，f 表示飞船上**当前**搭载的燃料重量。

例如在编号为 的星球上着陆之前有 $m = 9, f = 3, a_i = 8, b_i = 2$，那么着陆需要 $(9 + 3) / 8 = 1.5$ 吨燃料，而起飞需要 $(9 + 1.5) / 2 = 5.25$ 吨燃料，因为着陆以后燃料被消费掉了，起飞之前 $f = 1.5$ 而不是 3。

注意，整个过程是最开始从地球起飞，然后依次在 $2, \ldots, n$ 号星球着陆然后起飞，最后在地球着陆。

因为 JM 的军队已经将这些星球上的燃料掠夺一空，所以你没法在路上补充燃料，你的燃料只能在出发前全部装好。

同样是因为 JM 的军队的疯狂掠夺，现在燃料已经十分紧缺，你希望花费最少的燃料来完成招募任务。请计算访问这些星球并返回地球所需的最少燃料。上述所有重量的计数单位均相同。

输入格式

第一行两个正整数 n 和 m，表示星球个数和飞船净重。

接下来 n 行，每行两个正整数 a_i, b_i，分别表示每个星球的着陆参数和起飞参数，其作用如题目所述。

输出格式

输出一行一个浮点数表示答案。但是如果无论装载多少燃料都无法完成招募任务，则输出一个整数 -1 不带浮点。

当答案的绝对值误差不超过 10^{-4} 时即会被评测通过。

保证如果有解，则解小于10^9。

数据范围与提示

$2 \le n \le 2 \cdot 10^5$

$1 \le m, a_i, b_i \le 10^6$

解题思路

对答案进行二分查找。查找次数为 $\log(mrg/eps) = \log(10^{14})$，每次走遍全程查询燃料数目是否允许，复杂度为 $O(n)$，总复杂度可以接受。

本题现在的 $mrg = 10^{18}$，则不能使用 double 或 long double 直接进行二分，而是应先用二分得到整数范围，之后在两个整数之间进行小数的二分。鉴于这相当毒瘤，故没有代码。

代码

```
const double mrg = 1e9, efs = 1e-5;
int n, m;
vector<prdd> balls;
bool check(double f)
{
    double tf = f;
    for (int i = 0; i < n; i++)
    {
        tf -= (m + tf) / balls.at(i).second;
        tf -= (m + tf) / balls.at((i + 1) % n).first;
        if (tf < 0)
            return false;
    }
    return true;
}
void search(double str, double end)
{
    if (end - str <= efs)
    {
        if (check(end))
            printf("%lf", end);
        else
            printf("-1");
        return;
    }
    double mid = (str + end) / 2;
    if (check(mid))
        search(str, mid);
    else
        search(mid, end);
}
int main()
{
    scanf("%d%d", &n, &m);
    for (int i = 0; i < n; i++)
    {
        int t1, t2;
        scanf("%d%d", &t1, &t2);
```

```
            balls.push_back(prdd(t1, t2));
        }
        search(0, mrg);
        return 0;
}
```

F. JM 的特快列车

内存限制：512 MiB

时间限制：1000 ms

题目描述

由于 JM 太菜了，他总是要去向强无敌的神仙 cyy 请教问题，但是神仙 cyy 住在深山老林，路途遥远，太不方便，因此 JM 洗劫了史前巨富 jwp 的宝库，并用这笔钱建造一段铁路。

铁路是蜿蜒崎岖的，不过这并不重要，因为我们知道准确的铁轨长度。

除了起点站（JM 家）和终点站（cyy 家）以外，铁路上一共有 n 个中转站，编号为 $1,2,\ldots,n$，第 i 个中转站和起点站之间的距离为 a_i，火车可以在中转站把油加满，但是加油总是很费时间的，所以 JM 并不打算让火车频繁停站加油。此外，终点站和起点站之间的距离为 k。

另一方面，JM 建好铁路以后突然发现钱不够了，而史前巨富 jwp 已经布下十八重防御以免再次被洗劫，所以 JM 只好设法减少开销了。他打算拆除一些中转站以获取一些流动资金。

火车的耗油量和铁轨长度是 1 比 1 的关系，这也就意味着，两个中转站之间的距离决定了火车至少要具备的储油量，如果太远的话，火车没来得及开到下一个中转站就瘫痪了，那就 GG 了。

其实上面那些都是废话，甚至题目的要求都是反的（错乱），请以下面的要求为准。

现在 JM 打算拆除**恰好** m 个中转站，并在此基础上，使得两个站（包括起点站、终点站、所有中转站）之间的**最短距离最大**。也就是说，要令 $\min\{d_{i+1}-d_i \mid i\in[0,n]\}$ 的值最大，其中 $d_0=0, d_{n+1}=k$。

输入格式

第一行三个整数 n,m,k，其中 n 表示除了起点和终点以外的中转站的个数，m 表示要拆除的中转站的个数，k 表示终点站和起点站之间的距离。

接下来一行 n 个正整数，表示 n 个中转站和起点站之间的距离。保证不会有两个中转站具有相同的距离值。

输出格式

一行一个正整数，表示两个站之间的最短距离的最大值。

数据范围与提示

$1\le n\le 10^5$

$0\le m\le n$

$1\le k\le 10^9$

$1\le d_i<k, (i\in[0,n])$

$d_i\ne d_j, (i\ne j)$

解题思路

二分查找最小值。将所有中转站按坐标排序，若两站距离比查找值小，则拆除一个中转站。若拆除的中转站个数大于 m，则是不允许的；若拆除的中转站的个数小于等于 m，任意拆除更多的中转站，都使得最小值大于等于查询值，是允许的。查找次数为 $\log k$ 次，每次查询的复杂度为$O(n)$，总复杂度为 $O(n \log k)$，是允许的。

代码

```
int n, m, dis, tmp;
vector<int> sta;
bool check(int mxdis)
{
    int rmm = m, nwdis = 0;
    for (int i = 1; i <= n + 1; i++)
    {
        nwdis += sta.at(i) - sta.at(i - 1);
        if (nwdis < mxdis)
            if (rmm > 0)
                rmm--;
            else
                return false;
        else
            nwdis = 0;
    }
    return rmm >= 0;
}
void search(int l, int r)
{
    if (r - l == 1)
        printf("%d", l);
    else
    {
        int mid = (l + r) >> 1;
        if (check(mid))
            search(mid, r);
        else
            search(l, mid);
    }
}
int main()
{
    scanf("%d%d%d", &n, &m, &dis);
    sta.push_back(0);
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &tmp);
```

```
        sta.push_back(tmp);
    }
    sta.push_back(dis);
    sort(sta.begin(), sta.end());
    search(0, dis + 1);
    return 0;
}
```

备注

代码中 search 函数处理的是区间 $[l,r)$。

G. JM 的毒瘤题

内存限制：512 MiB

时间限制：1000 ms

题目描述

回文串是一种“对称”的字符串，若字符串 s 是一个回文串，则 s 满足 $s=\text{reverse}(s)$，其中 $\text{reverse}(s)$ 表示字符串 s 的翻转。

cyy 非常喜欢回文串，今天他掏出了一个构造回文串的算法：

在第 1 步，字符串为 $p_1=\text{c}_1$。

在第 2 步，字符串为 $p_k=p_{k-1}+c_k+p_{k-1}$，其中 $+$ 表示字符串之间的连接。

上述算法中的 c_i 表示一个未知字符，这样的字符一共有 k 种，分别为 $c_1,c_2,\dots,c_k$。

就在 cyy 开心地构造着回文串时，毒瘤的 JM 想起了最长公共子串问题，和这个回文串结合了一下，弄出了一道毒瘤（其实并不）题：

为便于描述，规定 $s[l,r]$ 表示 s 的子串 $s_ls_{l+1}\dots s_r$，其中 s_i 表示 s 的第 i 个字符，且 $i\in[1,|s|]$，其中 表示 $|s|$ 的长度。例如 $s=$ **abcde**，则 $s[2,3]=$ **bc**。

给定 q 次询问，每次给定五个正整数 k,l_1,r_1,l_2,r_2，求 p_k 的两个子串 $p_k[l_1,r_1]$ 和 $p_k[l_2,r_2]$ 的最长公共子串的长度是多少。

例如 $k=3$，则 $p_3=$ **abacaba**，那么：

$p_3[1,2]$ 和 $p_3[4,5]$ 的最长公共子串为 **a**，其长度为 1；

$p_3[1,4]$ 和 $p_3[3,6]$ 的最长公共子串为 **ab** 或 **ac**，其长度为 2。

输入格式

第一行一个正整数 q，表示询问次数。

接下来 q 行，每行五个正整数 k,l_1,r_1,l_2,r_2，表示一次询问。

输出格式

对于每次询问，输出一行一个非负整数表示答案。

数据范围与提示

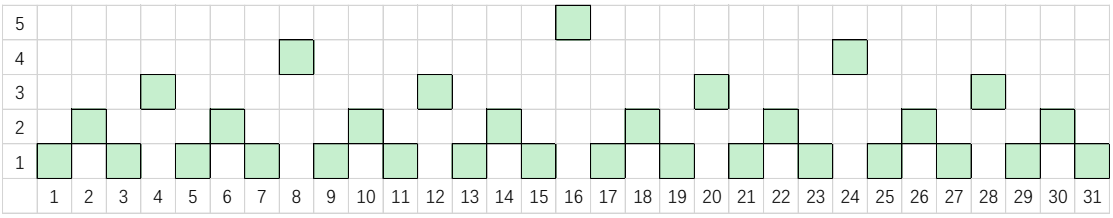
$1\le q\le 10^5$

$1\le k\le 30$

$1\le l_i\le r_i\le 2^k-1, i\in[1,2]$

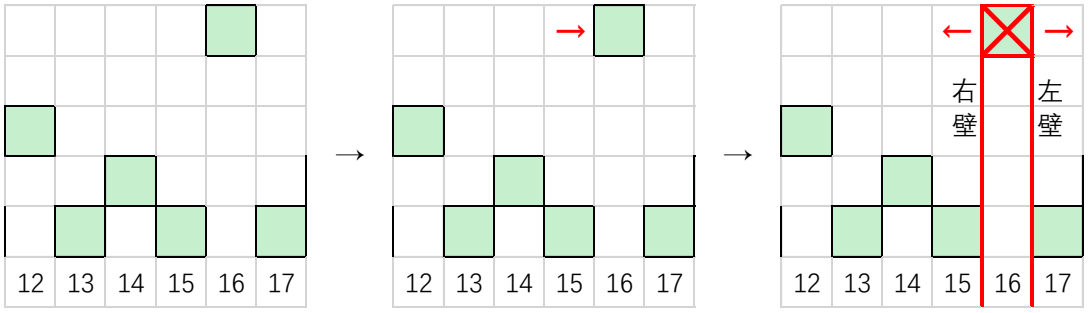
解题思路

将字符串转化为下面的图：



其中横坐标 x 表示这一列为 s 的第 x 个字母 s_x，纵坐标 y 表示 s_x 为 c_y，不妨称之为层数。由于字符串长度为 2^k，由上图中所表现的性质考虑分治：

从高层数向低层数依次向两个子串查询：设查询的层数为 v，若两子串中均含有层数为 v 的字母，容易发现，两子串一定存在以 c_v 为中心的子串，之后与答案比较取较大值。每次查询结束后，将查询的子串以 c_v 为切点分割为两个子串，他们的最高层数均小于 v，再对两子串的所有分割后的子串两两上述查询。如图：



待操作子串　　查询　　分隔

但是，由于对所有分割后的子串都要各取一个进行查询，容易发现，这种做法复杂度依然为 $O(2^k)$。但是，由上图的分割可以发现：分割后的子串要么是紧贴 p_{v-1} 的左壁，要么是紧贴 p_{v-1} 的右壁，故只需对两种情况存储最长的子串即可。

从图中可以发现，这本质是一个完全二叉树，每个字符的层数与其位置 x 的二进制数有关。利用这个性质，使用位运算的方式，每次查询和分割的复杂度为 $O(1)$，每次询问共计查询 $\log(2^k)=k$ 次，总复杂度为 $O(qk)$，是可以接受的。

代码

```
struct lar
{
    prdd l[2][32];
};
int q, k, l1, r1, l2, r2, ans = -1;
prdd is[2];
lar eg[2], ep[2];
bool cml(prdd itv, int lv)
{
    if (itv.first == 0)
        return false;
    return (itv.second >> (--lv) & 1) - ((itv.first - 1) >> (lv) & 1);
}
```

```
int latml(prdd itv, int lv)
{
    if (itv.first == 0)
        return 0;
    return (itv.second >> (lv - 1)) << (lv - 1);
}
int getlen(prdd itv)
{
    return itv.second - itv.first;
}
prdd mxlen(prdd x, prdd y)
{
    if (getlen(x) > getlen(y))
        return x;
    else if (getlen(x) < getlen(y))
        return y;
    if (x.first == 0)
        return y;
    else
        return x;
}
int main()
{
    scanf("%d", &q);
    for (int fq = 0; fq < q; fq++)
    {
        scanf("%d", &k);
        for (int i = 0; i < 2; i++)
            scanf("%d%d", &is[i].first, &is[i].second);
        eg[0].l[0][k] = eg[0].l[1][k] = is[0];
        eg[1].l[0][k] = eg[1].l[1][k] = is[1];
        int a[2];
        for (int i = k; i > 0; i--)
            for (a[0] = 0; a[0] < 2; a[0]++)
                for (a[1] = 0; a[1] < 2; a[1]++)
                {
                    int x = a[0], y = a[1], ni = i - 1;
                    if (cml(eg[0].l[x][i], i) && cml(eg[1].l[y][i], i))
                    {
                        int    rx = eg[0].l[x][i].second % (1 << i),
                            ry = eg[1].l[y][i].second % (1 << i),
                            minr = min(rx, ry),
                            lx = eg[0].l[x][i].first % (1 << i),
                            ly = eg[1].l[y][i].first % (1 << i),
                            maxl = max(lx, ly);
```

```
                        ans = max(ans, minr - maxl);
                    }
                    for (int tp = 0; tp < 2; tp++)
                        if (cml(eg[tp].l[a[tp]][i], i))
                        {
                            int    leg = eg[tp].l[a[tp]][i].first,
                                reg = eg[tp].l[a[tp]][i].second,
                                tplv = latml(eg[tp].l[a[tp]][i], i);
                            prdd lp = prdd(leg, tplv - 1),
                                rp = prdd(tplv + 1, reg);
                            eg[tp].l[0][ni] = mxlen(eg[tp].l[0][ni], lp);
                            eg[tp].l[1][ni] = mxlen(eg[tp].l[1][ni], rp);
                        }
                        else
                        {
                            int T = a[tp];
                            prdd ts = eg[tp].l[T][i];
                            eg[tp].l[T][ni] = mxlen(eg[tp].l[T][ni], ts);
                        }
                }
            printf("%d\n", ans + 1);
            ans = -1; eg[0] = ep[0]; eg[1] = ep[1];
        }
        return 0;
}
```

H. nocriz 参加学习运动

内存限制：512 MiB

时间限制：1000 ms

题目描述

马是中国古代的主要交通工具，在华夏大地上散布着 n 个城市，有 m 条**有向**的骑马线路，也就是构成了一张 n 个点 m 条边的有向图。图上的节点编号为 $1,2,\ldots,n$ 。注意，图上可能有重边或自环。

这 m 条线路被分成了 c 种等级，对于其中的第 i 种等级，只有知识点达到了 w_i 才能骑马过这条线路。

nocriz 现在在长安城（ 1 号点），他要赶到京城（ n 号点）参加青年大学习。作为一个爱国青年，初始时他知识点为 0 。但他实在是太爱学习了，以至于他每骑一次马（即经过一条边）就能使他的知识点 $+1$ 。

由于科技和时代所限，**nocriz** 不会通过其他方法来获取知识点。**nocriz** 不喜欢步行，也乘不起轿子，所以他不会通过除骑马以外的其他方法来到达另一个城市。

现在他想知道他是否能赶到京城，如果能，他还想知道他最少要经过多少条边才能到达。于是他找你帮他计算，在当代完成计算的你可以获得 1 ~~德育~~分。

输入格式

第一行三个整数 n,m,c 。

接下来 m 行，每行三个整数 u,v,lv ，表示有一条从 u 到 v 的线路，等级为 lv 。接下来一行 c 个整数，第 i 个表示 w_i 。

输出格式

若不能到达，则输出一行一个字符串**"Impossible"**（不包括引号），否则输出一行一个整数，表示他最少要经过的边数。

样例

样例输入 1

```
3 2 2
1 2 1
2 3 2
0 2
```

样例输出 1

```
Impossible
```

样例输入 2

```
3 3 2
1 2 1
2 1 1
2 3 2
0 2
```

样例输出 2

```
4
```

数据范围与提示

$2 \le n \le 180$

$1 \le c \le 50$

$0 \le m \le 4 \cdot 10^4$

$0 \le w_i \le w_{i+1} \le 10^9$

本题有一定难度，需要具备图论的知识和一定的比赛经验。

图上可能有重边或自环。

Day 4

主题：搜索

时间：2019 年 6 月 27 日 星期四

主要内容：

DFS，BFS

剪枝，回溯，记忆化搜索，状态压缩

题目：

+	A. tyx 的蜜汁爱好	DFS，线性筛
+	B. 愚蠢的 tyx	DFS，剪枝，八皇后
+	C. 马话家 tyx	DFS，记忆化搜索
+	D. 膨胀的 tyx	BFS
+2	E. zzj & liaoy 想要去摄影	BFS，状态压缩
+1	F. JM 的觉醒之路	DFS，折半枚举
+	G. nocriz 的爆搜题	

什么是搜索？

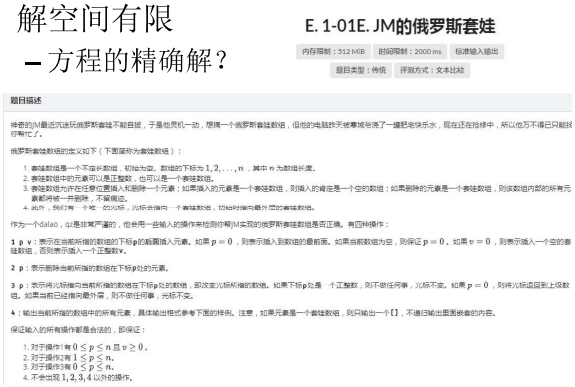
- “解空间”：
  - 所有可能的情况所构成的空间
  - 答案一定处于空间中
- 搜索整个空间，得到答案
  - 暴力，检查所有情况
  - 优化，检查部分情况

为什么要搜索？

- 计算机够快
  - 一秒计算量10^8级别
  - 解空间不是非常大，完全可以暴力
- 除了搜索没啥好方法
  - 旅行商问题
  - 猜数字
  - etc..

是不是所有问题都能搜索？

- 解空间有限
  - 方程的精确解？



E. 1-01E. JM的俄罗斯套娃

内存限制：512 MiB　时间限制：2000 ms　标准输入输出

题目类型：传统　评测方式：文本比较

题目描述

搜索算法

- 搜索算法是有目的的穷举一个问题的部分或所有的可能情况，从而求出问题的解的一种方法
- 搜索算法实际上是根据初始条件和扩展规则构造一棵“搜索树”并寻找符合目标状态的节点的过程。搜索算法的使用第一步在于搜索树的建立，完成搜索的过程就是找到一条从根结点到目标结点的路径，类似于图或树的遍历

搜索算法

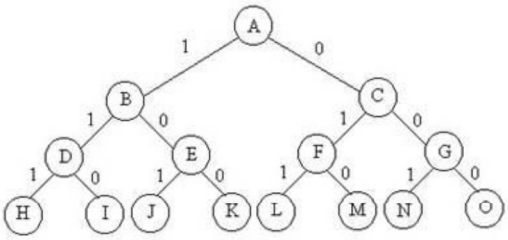
求解过程

- 状态（state）是对一个问题在某一时刻进展情况的数学描述。
- 状态空间（state space）是一个表示某问题全部可能状态及其关系的图，包含三个集合，即问题初始状态集合S，操作符集合F以及目标状态集合G。实际上状态空间图由节点及连接节点的边构成，结点是所有的状态，边对应状态转移。

- Step1：
  将问题表示为状态空间图，解表示为目标节点
  Step2：
  选择合适的搜索算法求解从初始节点到目标节点的路径

搜索树

用最多3个0或1能组成那些字符串？



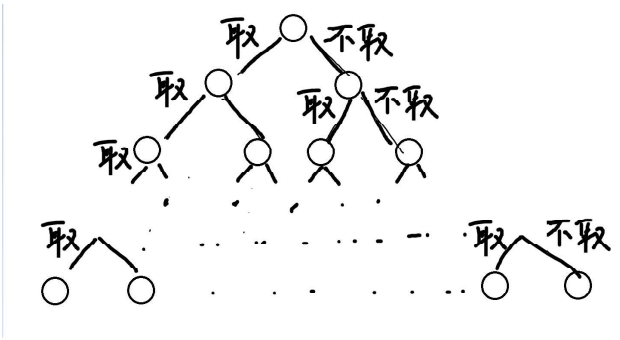
一个简单的例子

- 有20个数
- 你可以从中取任意个，但不能不取
- 将取出的求和
- 最多可以得到多少种不同的和？

暴力搜索每种情况

- 第1个数：取 or 不取
- 第2个数：取 or 不取
- ....
- 第20个数：取 or 不取

- 得到一个和，加到set里



代码实现

```
void dfs( int t, int sum ) {
    if( t == 20 ) {
        q.insert( sum );
        return;
    }
    dfs( t + 1, sum + a[ t ] );
    dfs( t + 1, sum );
}
```

常见的搜索

- 深度优先搜索
  - DFS
- 广度优先搜索
  - BFS

深度优先搜索

- 首先扩展最新产生的节点。
- 从初始节点开始。在其子节点中选择一个节点考察，若不是目标节点，则再在该子节点中选择一个子节点进行考察，一直向下搜索，直到某个节点既不是目标节点又不能继续扩展，之后选择兄弟节点进行考察。

DFS的实现

算法描述

- 递归

```
(1) 访问顶点v; visited[v]=1;
(2) w=顶点v的第一个邻接点;
(3) while (w存在)
        if (w未被访问)
          从顶点w出发递归执行该算法;
        w=顶点v的下一个邻接点;
```

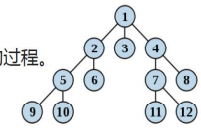
- 非递归

```
(1) 栈S初始化; visited[n]=0;
(2) 访问顶点v; visited[v]=1; 顶点v入栈S
(3) while(栈S非空)
        x=栈S的顶元素(不出栈);
        if(存在并找到未被访问的x的邻接点w)
                访问w; visited[w]=1;
                w进栈;
        else
                x出栈;
```

广度优先搜索

思想

- 搜索以接近起始节点的程度依次扩展，是一个分层搜索的过程。



1.访问顶点v0
2.依次访问v0的各个未被访问的邻接点
3.从这些邻接点出发，再访问它们的邻接点，直到所有的节点均被访问。

BFS的实现

算法描述

- 使用**队列**以保持遍历过的顶点顺序，以便按出队的顺序再去访问这些顶点的邻接顶点。

```
(1) 初始化队列Q; visited[n]=0;
(2) 访问顶点v; visited[v]=1; 顶点v入队列Q;
(3)  while (队列Q非空)
            v=队列Q的队首元素出队;
            w=顶点v的第一个邻接点;
            while (w存在)     //将所有邻接点入队
                    如果w未访问，则访问顶点w;
                    visited[w]=1;
                    顶点w入队列Q;
                    w=顶点v的下一个邻接点。
```

有什么区别？

- DFS：
  - 内存消耗小，每层只维护一个节点
  - 求得的解未必最优
  - 主要用于判断可行性
- BFS：
  - 内存消耗大，每层维护若干个节点
  - 求得的解一定最优
  - 终于用于求最优解

迷宫问题

模拟算法流程

结论：所以根据广度优先搜索的话，搜索到终点时，该路径一定是最短的。

BFS搜索顺序是第一层->第二层->第三层->第N层。
假设终点在第N层，则搜索到的路径长度肯定是N，且这个N一定是最短的。
我们用简单的反证法来证明：假设终点在第N层上边出现过，例如第M层，M<N，那么我们在搜索的过程中，肯定是先搜索到第M层的，此时搜索到第M层的时候发现终点出现过了，那么最短路径应该是M，而不是N了。

带权边不可以

迷宫问题

实现

N-Find a way

题目描述

- Yifenfeis's home is at the countryside, but Merceki's home is in the center of city. So yifenfei made arrangements with Merceki to meet at a KFC. There are many KFC in Ningbo, they want to choose one that let the total time to it be most smallest.
  Now give you a Ningbo map, Both yifenfei and Merceki can move up, down ,left, right to the adjacent road by cost 11 minutes.

•宜分菲的家在农村，但梅尔奇基的家是城市的中心。 所以宜分菲与梅尔奇基安排在肯德基见面。 宁波有很多肯德基，他们想选择一个时间最少的一家。 现在给你一张宁波地图，伊芬菲和梅切斯基都可以上下左右移动到邻近的道路上。求两人到达所需时间最短肯德基的总时间之和。

N-Find a way

Sample Input

```
5 5
Y, 0, 0, @, 0,
0, #, 0, 0, 0,
0, 0, 0, 0, 0,
@, #, 0, M, 0,
#, 0, 0, 0, #
```

n行，每行m个元素。

'Y'代表宜芬菲的初始位置。
'M' 代表梅尔奇基的初始位置。
'#' 代表被禁止的路段;
'0' 代表可通行的路段.
'@' 代表KCF

Sample Output

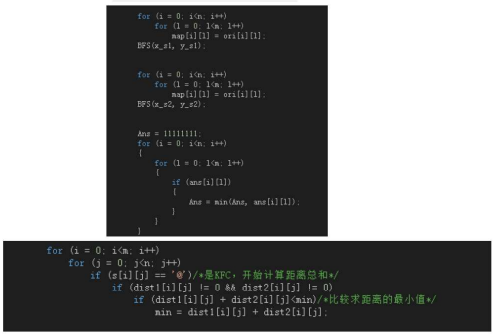
```
6
```

解题思路

本题为求迷宫最短路径问题，使用BFS下搜索到的第一条路径就是最短路径。
分别以两个人为起点，对图进行两次的BFS，求出各自到KCF得最短距离，之后比较多个KCF的距离，输出最短的距离。

N-Find a way

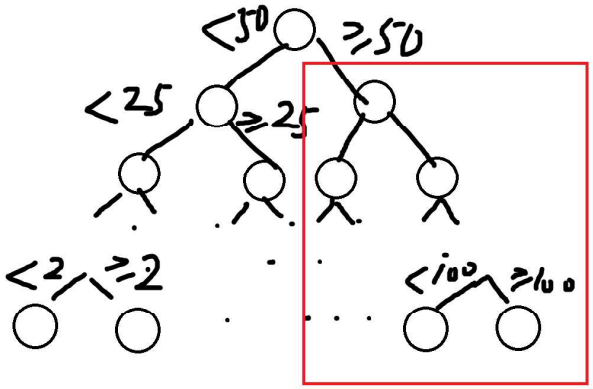
实现



优化——剪枝

- 为什么可以剪枝？
  - 在搜索树中，可能有一些子树满足相同的性质A
  - 如果答案不满足性质A，则所有这些点都不满足
  - 判断一个点，砍掉一堆点

猜数字



之前的搜索思路

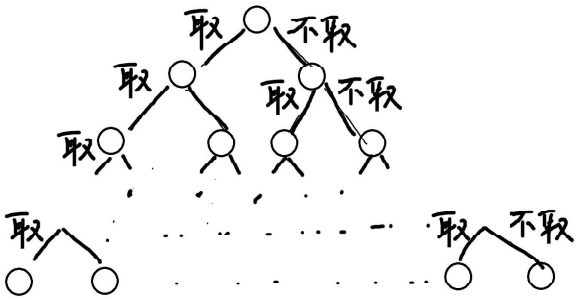
- 枚举所有方案
- 判断该方案是否可行

- 是否可以优化？？

一个不怎么恰当的例子

- 背包问题
- N个物品，容量为M的背包
- 每个物品有价值Ai和体积Bi
- 怎么装可以获得尽可能多的价值？

搜索的做法



- 如果在中间的某一步，体积已经超过了M
- 还有必要继续搜索吗？

- 剪枝1（可行性剪枝）

- 对于每一个节点，我们记录他已经取得物品的总价值为w，总体积为v，剩下的物品的总价值为q
- 假如我们之前已经获得了某一个方案的价值总和为x
- 假如w+q<x，是否还有必要继续搜索？

- 剪枝2（最优性剪枝）

新的搜索思路

- 边搜索，边判断
- 如果当前状态不可能导出最优解，则回溯
- 不需要等到最后才判定是否能得到最优解，跳过了中间一系列不可能导出最优解的过程
- ——因为走的少，所以走得快

回溯？？

- 之前我们说过，搜索就是在搜索树中遍历
- 如果我们搜索到了叶节点，或者该节点的所有子节点我们都已经搜索过了，现在应该干啥？
- 回溯——回到父节点，继续搜索

简而言之

- 走迷宫
- 走到死胡同了，倒回去，换一个路口继续走
- 如果上一个岔道没有新的岔路未探索了，再倒回去

- 回溯

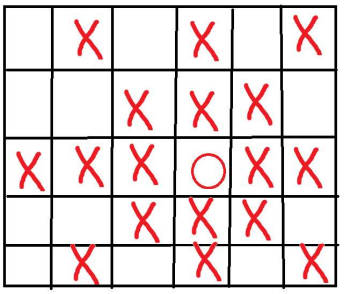
剪枝+回溯？

- 通俗地说，本来走迷宫，我们走到死胡同才会倒回去
- 现在我们nb了，眼睛带透视的，可以看到当前位置的前方若干步
- 不需要走到死胡同，我们就可以判定这条路一定是死路，从而直接回退，而不是非要撞到墙才回头

经典应用题

- N皇后问题
- 给定一个N*N的棋盘，如何放置尽可能多的皇后，使得皇后之间两辆不攻击？
- 皇后的攻击范围是：该行，该列，以及该点所在的两条对角线

皇后的攻击范围



思路分析

- 我们按行来，每一行遍历该行的每一列，尝试将皇后放在这里
- 如果能放下，则进入下一行
- 如果不能放下，则尝试下一列
- 如果最后一列尝试完了，则回溯
- 如果进入了N+1行，表示前N行都已经放完了，则为一个合法方案

如何判断是否可行？

- 暴力检查？
  - 每到（i,j），我们依次向左，左上，上，右上搜索，看看有没有已经放置的棋子
  - 可以，但是太慢
- 通过记录一些额外的信息，完成O（1）的检查
  - 咋整？

怎么快速判断可行

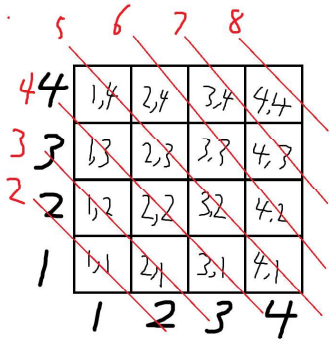
- 行判断
  - 由于我们是按行搜索的，因此行一定满足
- 列判断
  - 用row[i]记录第i列是否已经放置了棋子
- 对角线判断
  - 怎么表示对角线？

怎么表示对角线？

- 主对角线

容易看出，同一条主对角线上的点，x+y相同

不同主对角线上的点，则不同



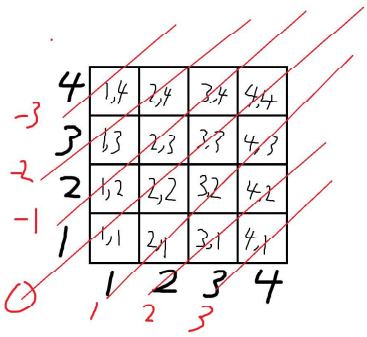
怎么表示对角线？

- 副对角线

容易看出，同一条副对角线上的点，x-y相同

不同主对角线上的点，则不同

注意负值的处理
--加上偏移量
--分类



判断的总结

- 用row[i]表示第i列是否有棋子
- 用md[i]表示第i条主对角线是否有棋子
- 用cd[i]表示第i条副对角线是否有棋子

新的算法流程

- 按行遍历，对于每一行，依次搜索每一列
- 检查改点对应的row, md, cd，若均为false，则将其标记为true，进入下一行
- 若有一个为true，则搜索下一列
- 若没有下一列，则回溯
- 对已经放置棋子的点回溯时，记得清空row, md, cd

```
void search(int x) {
    if (x > n) {
        ++ans;
    } else {
        for (int i = 1; i <= x; ++i)
            if (!c[i] && !x1[i + x] && !x2[x - i]) {
                res[x] = i;
                c[i] = x1[i + x] = x2[x - i] = 1;
                search(x + 1);
                c[i] = x1[i + x] = x2[x - i] = 0;
            }
        for (int i = x + 1; i <= n; ++i)
            if (!c[i] && !x1[i + x] && !x3[i - x]) {
                res[x] = i;
                c[i] = x1[i + x] = x3[i - x] = 1;
                search(x + 1);
                c[i] = x1[i + x] = x3[i - x] = 0;
            }
    }
}
```

复习

有一个5*5的棋盘，有一个马在（1,1）的位置，按照马的走法，最少走一步，最多走三步，他可以到达哪些地方？

dfs写法：

```
int a[ 5 ][ 5 ];
int mi[ 5 ][ 5 ];
int movx[] = { -1, -2, -2, -1, 1, 2, 2, 1 };
int movy[] = { -2, -1, 1, 2, 2, 1, -1, -2 };
int ans;

void dfs( int, int, int );

int main() {
        memset( mi, MAX_INF, sizeof( mi ) );
        dfs( 0, 0, 0 );
        for( int i = 0; i < 5; ++i ) {
                for( int j = 0; j < 5; ++j )
                        printf( "%d\t", a[ i ][ j ] );
                printf( "\n" );
        }
        return 0;
}

void dfs( int x, int y, int t ) {
        int nx, ny;
        if( t == 3 )
                return;
        for( int i = 0; i < 8; ++i ) {
                nx = x + movx[ i ];
                ny = y + movy[ i ];
                if( nx < 0 || nx >= 5 || ny < 0 || ny >= 5 )
                        continue;
                if( t + 1 < mi[ nx ][ ny ] ) {  //有更近的到达方法
                        if( mi[ nx ][ ny ] == MAX_INF_VAL )     //第一次到达该点
                                a[ nx ][ ny ] = ++ans;
                        mi[ nx ][ ny ] = t + 1;
                        dfs( nx, ny, mi[ nx ][ ny ] );
                }
        }
}
```

2	16	17	7	4
14	0	1	18	11
15	3	0	5	6
21	13	9	10	19
22	12	20	8	0

bfs写法

```
struct p {
        int x, y, t;
};
int a[ 5 ][ 5 ];
int movx[] = { -1, -2, -2, -1, 1, 2, 2, 1 };
int movy[] = { -2, -1, 1, 2, 2, 1, -1, -2 };
int ans;
queue< p > q;

void bfs( int, int, int );

int main() {
        bfs( 0, 0, 0 );
        for( int i = 0; i < 5; ++i ) {
                for( int j = 0; j < 5; ++j )
                        printf( "%d\t", a[ i ][ j ] );
                printf( "\n" );
        }
        return 0;
}

void bfs( int xx, int yy, int tt ) {
        int x, y, t;
        int nx, ny;
        p pp;
        q.push( p{ xx, yy, tt } );
        while( !q.empty() ) {
                pp = q.front();
                q.pop();
                x = pp.x, y = pp.y, t = pp.t;
                if( t == 3 )    //为什么不用管后面的?
                        return;
                for( int i = 0; i < 8; ++i ) {
                        nx = x + movx[ i ];
                        ny = y + movy[ i ];
                        if( nx < 0 || nx >= 5 || ny < 0 || ny >= 5 )
                                continue;
                        if( a[ nx ][ ny ] )     //已经访问过
                                continue;
                        a[ nx ][ ny ] = ++ans;
                        q.push( p{ nx, ny, t + 1 } );
                }
        }
}
```

3	20	9	14	4
19	0	1	10	17
8	2	0	13	5
22	7	16	6	21
12	18	11	15	0

思考

- 为什么DFS写法中需要记录mi[x][y]，用来表示到达(x,y)的最小步数，而BFS不需要？

```
int a[ 5 ][ 5 ];
int movx[] = { -1, -2, -2, -1, 1, 2, 2, 1 };
int movy[] = { -2, -1, 1, 2, 2, 1, -1, -2 };
int ans;

void dfs( int, int, int );

int main() {
        dfs( 0, 0, 0 );
        for( int i = 0; i < 5; ++i ) {
                for( int j = 0; j < 5; ++j )
                        printf( "%d\t", a[ i ][ j ] );
                printf( "\n" );
        }
        return 0;
}

void dfs( int x, int y, int t ) {
        int nx, ny;
        if( t== 3 )
                return;
        for( int i = 0; i < 8; ++i ) {
                nx = x + movx[ i ];
                ny = y + movy[ i ];
                if( nx < 0 || nx >= 5 || ny < 0 || ny >= 5 )
                        continue;
                if( a[ nx ][ ny ] )     //已经访问过
                        continue;
                a[ nx ][ ny ] = ++ans;
                dfs( nx, ny, t + 1 );
        }
}
```

2	16	17	7	4
14	0	1	18	11
15	3	0	5	6
21	13	9	10	19
22	12	20	8	0

2	16	0	7	4
14	0	1	0	11
15	3	0	5	6
0	13	9	10	0
0	12	0	8	0

总结BFS和DFS

- 都能遍历整个解集树
- 在找“最短”、“最小”、“最近”的时候用BFS，因为它是按层的
- 在找唯一解的时候用DFS，因为能少跑很大一部分解集树
- 都能用的时候用DFS，因为好写

- 注意有时候DFS会爆栈
- 需要自己模拟递归

记忆化搜索

- 洛谷 P1464 Function

题目描述

对于一个递归函数$w(a,b,c)$

- 如果$a \le 0$ or $b \le 0$ or $c \le 0$就返回值1.
- 如果$a > 20$ or $b > 20$ or $c > 20$就返回$w(20,20,20)$
- 如果$a < b$并且$b < c$ 就返回$w(a,b,c-1)+w(a,b-1,c-1)-w(a,b-1,c)$
- 其它的情况就返回$w(a-1,b,c)+w(a-1,b-1,c)+w(a-1,b,c-1)-w(a-1,b-1,c-1)$

这是个简单的递归函数，但实现起来可能会有些问题。当a,b,c均为15时，调用的次数将非常的多。你要想个办法才行.

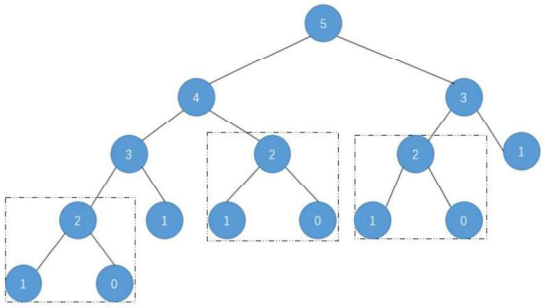
```cpp
#include <cstdio>
long long int w(long long int a, long long int b, long long int c) {
    if ((a <= 0) || (b <= 0) || (c <= 0)) return 1;
    if ((a > 20) || (b > 20) || (c > 20)) return w(20, 20, 20);
    if ((a < b) && (b < c)) w(a, b, c - 1) + w(a, b - 1, c - 1) - w(a, b - 1, c);
    return w(a - 1, b, c) + w(a - 1, b - 1, c) + w(a - 1, b, c - 1) - w(a - 1, b - 1, c - 1);
}
int main() {
    long long int a, b, c;
    while (1) {
        scanf("%lld%lld%lld", &a, &b, &c);
        if ((a == -1) && (b == -1) && (c == -1)) break;
        printf("w(%lld, %lld, %lld) = %lld\n", a, b, c, w(a, b, c));
    }
    return 0;
}
```

测试点信息

#1	#2	#3	#4	#5
TLE	TLE	TLE	TLE	TLE

类比一下斐波那契数列

- F[n]=F[n-1]+F[n-2]




```cpp
int f[ 21 ][ 21 ][ 21 ];
int deal( int, int, int );

int main() {
    int x, y, z;
    memset( f, 0, sizeof( f ) );
    while( true ) {
        scanf( "%d%d%d", &x, &y, &z );
        if( x == -1 && y == -1 && z == -1 )
            break;
        printf( "w(%d, %d, %d) = %d\n", x, y, z, deal( x, y , z ) );
    }
    return 0;
}

int deal( int x, int y, int z ) {
    if( x <= 0 || y <= 0 || z <= 0 )
        return 1;
    if( x > 20 || y > 20 || z > 20 )
        return deal( 20, 20, 20 );
    if( f[ x ][ y ][ z ] )
        return f[ x ][ y ][ z ];
    if( x < y && y < z )
        return f[ x ][ y ][ z ] = deal( x, y, z - 1 ) + deal( x, y - 1, z - 1 ) - deal( x, y - 1, z
);
    return f[ x ][ y ][ z ] = deal( x - 1, y , z ) + deal( x - 1, y - 1, z ) + deal( x - 1, y, z - 1
) - deal( x - 1, y - 1, z - 1 );
}
```

算过的数一直算？

- 算过的数还用算吗？
  - 当然不用了
- 怎么解决
  - 开个数组记录一下

现在你会的优化

- 剪枝
- 回溯
- 记忆化

还不够

- 很多时候状态并不好表示
  - 状压搜索

来试试

AveryBoy又又又被关在一个n*m的迷宫里，这次还有了检查人员防止他逃跑。，并在迷宫的某些地方安装了带锁的门，钥匙藏在迷宫另外的某些地方。刚开始AveryBoy被关在(sx,sy)的位置，离开迷宫的门在(ex,ey)的位置。AveryBoy每分钟只能从一个坐标走到相邻四个坐标中的其中一个。检查人员每t分钟回地牢视察一次，若发现AveryBoy不在原位置便把他拎回去。经过若干次的尝试，AveryBoy已画出整个迷宫的地图。现在请你帮他计算能否再次成功逃出迷宫。只要在检查人员下次视察之前走到出口就算离开迷宫，如果检查人员回来的时候刚好走到出口或还未到出口都算逃亡失败。

每组测试数据的第一行有三个整数n,m,t(2<=n,m<=20,t>0)。接下来的n行m列为迷宫的地图，其中包括：

. 代表路
* 代表墙
@ 代表AveryBoy的起始位置
^ 代表地牢的出口
A-J 代表带锁的门,对应的钥匙分别为a-j
a-j 代表钥匙，对应的门分别为A-J

Sample Input

```
@A.B.
a*.*.
*..*^
c..b*

4 5 16
@A.B.
a*.*.
*..*^
c..b*
```

Sample Output

```
16
-1
```

这怎么写

- 最初的bfs，状态仅有三个参数
- x, y, t
- 现在多了一个参数，拥有的钥匙情况

- 怎么表示？

状态压缩

- 最多也就10把钥匙
- 用一个10位的二进制数来记录钥匙的拥有情况
- 如果有第i把钥匙，则第i位置1，否则为0
- 也就2^10=1024种状态

- 20*20*1024*4

Find The Multiple

题目描述

- Given a positive integer n, write a program to find out a nonzero multiple m of n whose decimal representation contains only the digits 0 and 1. You may assume that n is not greater than 200 and there is a corresponding m containing no more than 100 decimal digits.
- 给出一个整数n，(1 <= n <= 200)。求出任意一个它的倍数m，要求m必须只由十进制的'0'或'1'组成。

Sample Input

2 6 19 0

输入0结束

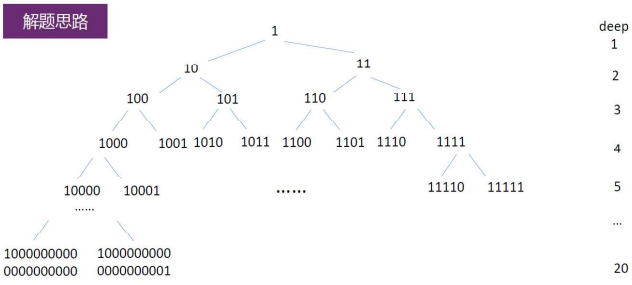
分别求2、6、19的倍数m，且m只由0和1组成

Sample Output

10
100100100100100100
111111111111111111

Find The Multiple

解题思路

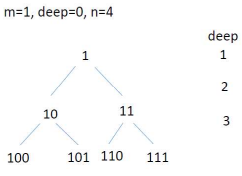


使用深度优先搜索，从1开始，如果搜索到m则输出，否则搜索m×10和m×10+1，直到得出答案。

Find The Multiple

实现

```
//t:表示0和1组成的数， deep:位数，n:给定的数
void dfs(unsigned long long m, int deep, int n)
{
   剪枝
   if(ok || deep == 20)  //退出的标志
      return;
   else if(m%n == 0)   //可以整除则输出
   {
      cout<<m<<endl;
      ok = 1;
      return;
   }
   else
   {dfs(m*10, deep+1, n);   //个位是0
    dfs(m*10+1, deep+1, n);  //个位是1
   }
}
```



如果要求最小值呢？

DFS还行吗？

例如：13

有一些解：
1001
1000000001
1000000000000001

DFS：1-10-100-1000-1000.....---1000000000000001

显然不是最优解

BFS可以吗？

为什么？

A计划

题目描述

可怜的公主在一次次被魔王掳走一次次被骑士们救回来之后，而今，不幸的她再一次面临生命的考验。魔王已经发出消息说将在**T时刻**吃掉公主，因为他听信谣言说吃公主的肉也能长生不老。年迈的国王正是心急如焚，告招天下勇士来拯救公主。不过公主早已习以为常，她深信智勇的骑士Acmer肯定能将她救出。 现据密探所报，公主被关在一个两层的迷宫里，迷宫的入口是S（0，0，0），**公主的位置用P表示**，**时空传输机用#表示**，**墙用*表示**，**平地用.表示。**

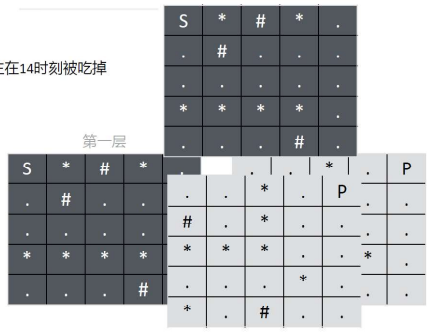
A计划

题目要求

- 骑士一进入时空传输机就会被转到另一层的相对位置，但如果被转到的位置是墙的话，那骑士们就会被撞死。
- 骑士在一层中只能前后左右移动，**每移动一格花1时刻。**
- 层间的移动只能通过时空传输机，**且不需要任何时间。**

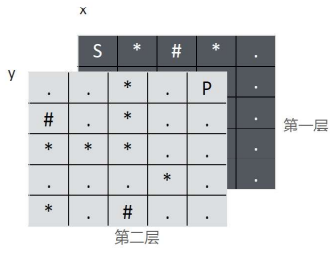
A计划

Sample Input
1 ——输入一个测试用例
5 5 14 ——迷宫大小为5*5，公主在14时刻被吃掉
S*#*.
.#...
..... ——第一层
****.
...#.
..*.P
#.*..
***.. ——第二层
...*.
*.#..
Sample Input
NO



A计划

样例输入



A计划

思路

- 因为现在每个点多了一个层数信息（Z坐标）。所以不仅仅可以前后左右移动，并且可以上下穿越。 第一反应是使用广度优先搜索（使用队列），和以前唯一不同的就是要判断Z坐标的变化。
- 遇到上下两层都是#的，无法移动。
- 双层BFS

```
struct Node
{
    int z;
    int x;
    int y;  //记录层数
    int step;  //记录步数
};
```

小技巧

为了避免每次穿越时都要注意穿过去的是不是墙，可以在输入数据时做这样的操作：如果当位置可以穿越，则判断穿越过去的是不是墙，**如果是则将当前位置设为墙。**

A. tyx 的蜜汁爱好

内存限制：512 MiB

时间限制：1500 ms

题目描述

tyx 最近沉迷于数学，并且他突然对素数产生了奇妙的兴趣

凑巧的是，tyx 手里有 n 个数字 $x_1, x_2, \ldots, x_n$ 。

tyx 看着手里的一堆不知道从哪里的数字，突然很好奇，他如果从中选出若干个数相加，能有多少种取数的方案，使得得到的和为一个素数？

当然，由于 tyx 有强迫症，他不仅希望知道满足条件的方案数，也想知道他总共可以得到多少个不同的素数。

遗憾的是，tyx 的脑子因为被寒域爷灌了一罐肥宅快乐水，现在似乎不怎么好使，因此他抱住了你的大腿希望你可以帮助他解决。

输入格式

第一行一个整数 n ，表示 tyx 手里的数的个数。

第二行 n 个整数，表示 $x_1, x_2, \ldots, x_n$ 。

输出格式

第一行一个整数 x ，表示满足和为素数的取数方案数。

第二行一个整数 y ，表示 tyx 总共可以得到多少个不同的素数。

数据范围与提示

$1 \le n \le 20$

$1 \le x_i \le 2 \cdot 10^6$

解题思路

暴力枚举搜索。复杂度为 $O(2^n)$。因需要判断一个数是否为质数，故使用线性筛预处理，复杂度为 $O(nx_i)$，后续检查的复杂度为 $O(1)$。

代码

```
int n, xi[20], alk = 0;
bool bolist[50000000];
map<int, int> st;
void fillInAllZhishu()
{
    vector<int> ps;
    int range = 2000000 * n;
    for (int i = 2; i <= range; i++)
        bolist[i] = true;
    bolist[0] = bolist[1] = false;
    for (int i = 2; i <= range; i++)
    {
        if (bolist[i])
```

```
            ps.push_back(i);
        for (int j = 0; j < ps.size() && ps.at(j) <= range / i; j++)
        {
            bolist[i * ps.at(j)] = false;
            if (i % ps.at(j) == 0)
                break;
        }
    }
}
void check(int ths, int ans)
{
    if (ths >= n)
    {
        if (bolist[ans])
        {
            alk++;
            st[ans]++;
        }
        return;
    }
    check(ths + 1, ans);
    check(ths + 1, ans + xi[ths]);
}
int main()
{
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
        scanf("%d", &xi[i]);
    fillInAllZhishu();
    check(0, 0);
    printf("%d\n%u", alk, st.size());
    return 0;
}
```

B. 愚蠢的 tyx

内存限制：512 MiB
时间限制：1500 ms

题目描述

在学习了今天的课程以后，tyx 也对 N 皇后问题产生了兴趣，可惜他想了很久还是不知道怎么写，于是又一次抱住你的大腿请求你的帮助。

他想知道对于 $N \times N$ 的棋盘，最多有多少种满足条件的摆法？

所谓满足条件的摆法，即每一行、每一列、每一条对角线最多仅能有一个皇后。

当然，tyx 为了确保你不是随便报了一个数字来忽悠他，他需要你给出按照字典序从小到大排序的前 3 组解（如果不足 3 种，则全部输出）。

一个解的序列是这样的定义的：设 a_i 表示第 i 行的皇后放在了第 a_i 列的位置，其中 $1 \le i \le N, 1 \le a_i \le N$，那么这个解所对应的的序列就是 $a_1, a_2, \ldots, a_n$ 。

注意，tyx 非常讨厌数学好的人，因此你如果直接用公式计算出了答案，tyx 可能会打死你

输入格式

一行一个整数 N 。

输出格式

首先是最多 3 行，表示字典序最小的前 3 组解。若解的总数不足 3，则输出所有的解即可。

接下来一行一个整数 x，表示满足条件的摆法总数。**请不要用公式计算答案哦~**

数据范围与提示

$1 \le N \le 13$

解题思路

DFS 搜索，复杂度为 $O(N^N)$。需进行剪枝：由皇后的性质可知，每次放新皇后时，只需放在所在列、所在主对角线、所在次对角线均没有皇后的格子上，用数组存储。容易发现，主对角线的横纵坐标差相同，次对角线横纵坐标和相同，可以作为对角线的特征值。

由于 N 并不大，且结果固定，迫不得已时可以打表。迫不得已时，也可以使用公式计算。

代码

```
int n, ans = 0, optans[3][13], nans[13];
bool row[13], diam[26], dias[26];
map<int, int> st;
void put(int l)
{
    if (l >= n)
    {
        if (ans < 3)
            for (int i = 0; i < n; i++)
                optans[ans][i] = nans[i];
        ans++;
        return;
    }
    for (int i = 0; i < n; i++)
        if (!row[i] && !diam[l - i + 12] && !dias[l + i])
        {
            row[i] = diam[l - i + 12] = dias[l + i] = true;
            nans[l] = i + 1;
            put(l + 1);
            row[i] = diam[l - i + 12] = dias[l + i] = false;
            nans[l] = 0;
        }
}
```

```cpp
int main()
{
    scanf("%d", &n);
    put(0);
    for (int i = 0; i < min(ans, 3); i++)
    {
        for (int j = 0; j < n; j++)
            printf("%d ", optans[i][j]);
        printf("\n");
    }
    printf("%d", ans);
    return 0;
}
```

C. 马话家 tyx

内存限制：512 MiB

时间限制：1500 ms

题目描述

众所周知，tyx 是一个大马话家，由于马话说的太多，因此 tyx 本人也逐渐马化。具体表现为，tyx 现在走路也只会走马步，就像中国象棋里的马一样，并且他现在热爱吃草。

现在 tyx 正处在一块巨大的 n 行 m 列的马场上，马场上的每个点 (i,j) 都有一定量的草 $a_{i,j}$ 。

作为马话家，tyx 自然与一般的马不一样，他每到达一个点，就会将这个点的所有草全吃光，并且他不会重复到达一个点两次。

tyx 为了不让自己吃的太多，因此他行进的路线是有讲究的，他的下一个点的草的数量必须要少于当前所在的点上原有的草的数量。当然，tyx 还是尽可能的想要多吃一些草，因此他可以任意选择一个点作为他的起点，并规划一条合理的路线。

现在 tyx 想知道，他最多可以吃到多少草？

输入格式

第一行两个整数 n,m ，表示马场的大小。

接下来 n 行，每行 m 个正整数，表示马场上每个点的草的数量 $a_{i,j}$ 。

输出格式

输出一行一个整数 x ，表示 tyx 最多能吃到的草的数量。

数据范围与提示

$1 \le n,m \le 2000$

$1 \le a_{i,j} \le 2 \cdot 10^6$

解题思路

中国象棋中，马走日字形，有 8 个点可以到达。

DFS 搜索，某点的最大食草量为其可到达的点的最大食草量中的最大值，加上该点的草量，复杂度为 $O(8^{mn})$ 。但是，容易发现，单纯使用 DFS，有很多点是重复计算多次的。故可以在每点的最大食草量计算出来后存储起来。

代码

```
struct cao
{
    int nofc, nonc = -1;
};
int n, m, ans = 0;
cao ecao[2000][2000];
const int way[8][2] = { {-2, -1}, {-1, -2}, {1, -2}, {2, -1},
                        {2, 1}, {1, 2}, {-1, 2}, {-2, 1} };
int ccl(int l, int r)
{
    if (ecao[l][r].nonc == -1)
    {
        int mx = 0;
        for (int i = 0; i < 8; i++)
        {
            int nl = l + way[i][0], nr = r + way[i][1];
            if (nl >= 0 && nl < n && nr >= 0 && nr < m)
                if (ecao[nl][nr].nofc < ecao[l][r].nofc)
                    mx = max(mx, ccl(nl, nr));
        }
        ecao[l][r].nonc = mx;
    }
    return ecao[l][r].nofc + ecao[l][r].nonc;
}
int main()
{
    scanf("%d%d", &n, &m);
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            scanf("%d", &ecao[i][j].nofc);
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            ans = max(ans, ccl(i, j));
    printf("%d", ans);
    return 0;
}
```

D. 膨胀的 tyx

内存限制：512 MiB

时间限制：1500 ms

题目描述

由于 tyx 抱过了这么多大腿，他变得越来越棒棒了，一般的迷宫根本拦不住他。为了治一治他，你决定为他单独定制一款无限迷宫。

所谓无限迷宫是指，由一个 $N \times M$ 的迷宫单元经过无限平铺得到的迷宫，即将无数份迷宫单元平铺在一个二维平面上。定制好以后，你将 tyx 扔到了迷宫里，不妨假定 tyx 落在了起点处。

出于对无限迷宫的恐惧，tyx 想要尽可能逃离这里。那么问题来了，tyx 能不能逃到距离起点无限远的地方去呢？

输入格式

第一行两个整数 N, M，用来描述迷宫单元的尺寸。

接下来是一个 $N \times M$ 的字符矩阵，用来描述这个迷宫，每个字符一定属于以下三种：

1. 字符`'.'`代表这个点是空地。
2. 字符`'S'`代表这个点是起点。
3. 字符`'#'`代表这个点是墙，不可以走。

输出格式

输出一行一个字符串“Yes”或“No”（不包括引号），“Yes”表示 tyx 可以逃到无限远的地方，“No”表示不可以。

数据范围与提示

$1 \leq N, M \leq 2000$

解题思路

从起点开始，进行 BFS 的探索。

由于是无限个相同的迷宫平铺，将起点迷宫坐标表示为 $(0,0)$，从边界走出是允许的，并会进入相邻的迷宫的对应位置，迷宫坐标也会相应地改变。若点 (n,m) 能同时在 BFS 过程中在两个不同的迷宫 (a_i, b_i) 和 (a_j, b_j) 中访问，则存在一个可以到达的点 (n,m)，使得存在一条通路，能从当前迷宫 (a_0, b_0) 到达另一个迷宫 $\left(a_0 + (a_i - a_j), b_0 + (b_i - b_j)\right)$。由数学归纳法可知，重复以上操作可以到达距离起点无穷远的地方。访问一个点后，记录该点的所在的迷宫坐标，若再次访问，比较本次的迷宫坐标和存储的迷宫坐标即可。

代码

```
const int way[4][2] = { {1, 0}, {-1, 0}, {0, 1}, {0, -1} };
struct cao
{
    bool vd;
    int dx = dinf, dy = dinf;
};
int n, m, ans = 0;
vector<prdd> bfsn, tmp;
cao ecao[2000][2000];
int pmd(int a, int p)
{
    int ans = a % p;
    return ans < 0 ? ans + p : ans;
}
```

```
void bfsnp(prdd p)
{
    for (int i = 0; i < 4; i++)
    {
        int nl = p.first + way[i][0], nr = p.second + way[i][1];
        if (!ecao[pmd(nl, n)][pmd(nr, m)].vd) continue;
        int nx = ecao[p.first][p.second].dx,
            ny = ecao[p.first][p.second].dy;
        if (nl < 0 || nr < 0 || nl >= n || nr >= m)
        {
            nx += way[i][0];
            ny += way[i][1];
        }
        if (ecao[pmd(nl, n)][pmd(nr, m)].dx == dinf)
        {
            tmp.push_back(prdd(pmd(nl, n), pmd(nr, m)));
            ecao[pmd(nl, n)][pmd(nr, m)].dx = nx;
            ecao[pmd(nl, n)][pmd(nr, m)].dy = ny;
        }
        else if (nx != ecao[pmd(nl, n)][pmd(nr, m)].dx ||
                 ny != ecao[pmd(nl, n)][pmd(nr, m)].dy)
        {
            bfsn.clear();
            tmp.clear();
            ans++;
        }
    }
}
int main()
{
    cin.sync_with_stdio(false);
    cin.tie(0);
    cin >> n >> m;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
        {
            char tmp;
            cin >> tmp;
            if (tmp == '#')
                ecao[i][j].vd = false;
            else
                ecao[i][j].vd = true;
            if (tmp == 'S')
            {
                ecao[i][j].dx = ecao[i][j].dy = 0;
```

```
                bfsn.push_back(prdd(i, j));
            }
        }
    while (!bfsn.empty())
    {
        for (int i = 0; i < bfsn.size(); i++)
            bfsnp(bfsn.at(i));
        bfsn = tmp;
        tmp.clear();
    }
    if (ans)
        printf("Yes");
    else
        printf("No");
    return 0;
}
```

E. zzj & liaoy 想要去摄影

内存限制：512 MiB

时间限制：1500 ms

题目描述

摄影是 liaoy 的一大爱好，碰巧 zzj 也很感兴趣。于是他们打算这个周末从交大出发，到西安市内中意的一些地点去摄影。zzj 和 liaoy 想要知道他们这次活动最少需要走多少路程（不需要返程）。

西安市的地图可以视作为一个 $n \times m$ 的矩形网格图，图上标注了交大的位置（起点）、空地、墙壁、计划地点。墙壁是不可通行的，其他地形都是可通行的。zzj 和 liaoy 从交大出发，每一步可以选择上下左右四个方向移动，目标是到达每个计划地点至少一次，求达成该目标所需的最少步数。

输入格式

第一行两个正整数 n 和 m，表示地图的大小。

接下来一共是 $n \times m$ 的字符串，表示地图的详细地形，其中`'p'`表示交大的位置，`'.'`表示空地，`'#'`表示不可通过的墙，`'@'`表示一个计划地点。

输出格式

输出一行一个非负整数，表示达成目标所需的最少步数。

如果他们不可能完成摄影计划，则输出 -1。

数据范围与提示

$2 \leq n, m \leq 30$

地图上至少存在一个计划地点，且计划地点的数量不超过 10 个。

数据范围不算太大，时间复杂度不卡常数。

解题思路

BFS 搜索。当到达一个计划地点后，搜索状态会发生改变，即从未访问过该计划地点变为访问过该计划地点。由于计划地点不超过 10 个，共有 $2^{10} = 1024$ 种状态，为每个计划地点分配一个编号，使用状态压缩。

代码

```
const int way[4][2] = { {1, 0}, {-1, 0}, {0, 1}, {0, -1} };
struct cao
{
    bool vd, fnd[1024];
    int pic;
};
struct bfnt
{
    int f, s, ti;
};
int n, m, res = -1, ans = 0, picno = 0;
vector<bfnt> bfsn, tmp;
cao ecao[32][32];
void bfsnp(bfnt p)
{
    for (int i = 0; i < 4; i++)
    {
        int nl = p.f + way[i][0], nr = p.s + way[i][1];
        if (nl < 0 || nl >= n || nr < 0 || nr >= m || !ecao[nl][nr].vd)
            continue;
        int nti = p.ti;
        if (ecao[nl][nr].pic >= 0)
            nti |= 1 << ecao[nl][nr].pic;
        if (!ecao[nl][nr].fnd[nti])
        {
            ecao[nl][nr].fnd[nti] = true;
            tmp.push_back(bfnt{ nl, nr, nti });
            if (nti == (1 << picno) - 1)
            {
                res = ans;
                bfsn.clear();
                tmp.clear();
            }
        }
    }
}
int main()
{
    cin.sync_with_stdio(false);
```

```
    cin.tie(0);
    cin >> n >> m;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
        {
            char tmp;
            cin >> tmp;
            if (tmp == '#')
                ecao[i][j].vd = false;
            else
                ecao[i][j].vd = true;
            if (tmp == 'p')
            {
                bfsn.push_back(bfnt{ i, j, 0 });
                ecao[i][j].fnd[0] = true;
            }
            if (tmp == '@')
                ecao[i][j].pic = picno++;
            else
                ecao[i][j].pic = -1;
        }
    while (!bfsn.empty())
    {
        ans++;
        for (int i = 0; i < bfsn.size(); i++)
            bfsnp(bfsn.at(i));
        swap(bfsn, tmp);
        tmp.clear();
    }
    printf("%d", res);
    return 0;
}
```

F. JM 的觉醒之路

内存限制：512 MiB

时间限制：2000 ms

题目描述

为了使你成仙，JM 特地为你设计了觉醒之路，只要通过这条路你就能飞天成仙。

觉醒之路可以看作是一个 $n \times m$ 的网格图，左上角为 $(1,1)$，右下角为 (n,m)。网格图上的每个点 (i,j) 都有一个权值 $a_{i,j}$。在网格图上的移动是只能向下或向右的，也就是说，从点 (i,j) 出发走一步只能走到 $(i+1,j)$ 或 $(i,j+1)$，当然，目标点的坐标不能越界。

初始时你手中的数值为 0，每到达一个点 (i,j)，都会对你手中的数值异或该点的权值，包括起点和终点。

觉醒之路的觉醒值为 k，现在 JM 希望你求出存在多少条从 $(1,1)$ 到 (n,m) 的路径，使得走过终点后手中的数值恰好等于觉醒值。

输入格式

第一行三个整数 n,m,k，表示网格图的长和宽，以及觉醒值。

接下来 n 行，每行 m 个整数 $a_{i,j}$，表示网格图上每个点的权值。

输出格式

输出一行一个整数表示答案。

数据范围与提示

$1 \le n,m \le 20$

$0 \le k \le 10^{18}$

$0 \le a_{i,j} \le 10^{18}$

解题思路

DFS 搜索，复杂度为 $O(C_{n+m}^{n})$，复杂度较大。由于异或运算满足结合律，且异或存在逆运算，考虑折半枚举，复杂度为 $O(2^{(n+m)/2+1} \cdot (n+m)/2)$，可以接受。

代码

```
const int way[2][2] = { {1, 0}, {0, 1} };
int n, m, sp;
llong k, ans = 0, dbg = 0;
llong ecao[32][32];
map<int, llong> sves[40];
void dfs(int x, int y, llong nk)
{
    if (x + y == sp)
    {
        sves[x][ecao[x][y] ^ nk]++;
        return;
    }
    if (x + 1 < n)
        dfs(x + 1, y, nk ^ ecao[x][y]);
    if (y + 1 < m)
        dfs(x, y + 1, nk ^ ecao[x][y]);
}
void rdfs(int x, int y, llong nk)
{
    if (x + y == sp)
    {
        ans += sves[x][k ^ nk];
        return;
    }
```

```
    if (x - 1 >= 0)
        rdfs(x - 1, y, nk ^ ecao[x][y]);
    if (y - 1 >= 0)
        rdfs(x, y - 1, nk ^ ecao[x][y]);
}
int main()
{
    cin.sync_with_stdio(false);
    cin.tie(0);
    cin >> n >> m >> k;
    sp = (n + m - 2) >> 1;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            cin >> ecao[i][j];
    dfs(0, 0, 0);
    rdfs(n - 1, m - 1, 0);
    printf("%lld", ans);
    return 0;
}
```

G. nocriz 的爆搜题

内存限制：512 MiB

时间限制：1000 ms

题目描述

现有一 $n \times m$ 的方阵，其中有一些块已经被放入了数字，其他的为 0。你要在其中空的位置上放上从 1 到 c 的数字，使得其中任何一条从左上到右下的仅包含向右或向下移动的路径都不包含同一个数字两次或两次以上。求方案数对 $998,244,353$ 取模后的结果。

输入格式

第一行三个整数 n, m, c 。

接下来 n 行每行 m 个整数，代表方阵。

输出格式

一行一个整数，代表答案。

数据范围与提示

$1 \le n, m, c \le 10$

解题思路

若 $n + m \ge c$，则无论如何都不能满足条件，直接特判掉即可。

不考虑数字，只考虑 $n \times m$ 的空图，可以从左上至右下进行 BFS，搜索可行的方案。在选择数字时，不选择具体的数字，而是选择使用变量来表示。在一个点选择数字时，一共有以下两类选择：选择全图已经选择过，但是本路上没有选择过的变量；或是选择一个没有出现过的数字，并作为一个新的变量。若选择前者，则对出现过的变量一一枚举即可，后者的前提是变量的总数小于 c 。若一个点没有变量可供选择，且不能产生新的变量，则这种选择是

不合法的。选择完成后，一张图由各不相同的若干变量 $a_1, a_2, \ldots, a_x$ 组成，之后根据给出的数字给这些变量赋值，且不能给同一个变量先后赋多个值，也不能给多个变量赋同一个值，否则这种选择是不合法的。设共有 x 个变量，其中为 y 个变量赋了值，没有确定值的变量将会有 $(c-y)(c-y-1)\ldots(c-x+1)=(c-y)!/(c-x)!$ 种选择，这个选择数就是答案的一部分。之后进行回溯，给出新的选择。

该种做法在数据较大时可能会超时，有两种优化：一是始终保留一个变量给右下角的点，即在生成新变量时要求变量的总数小于 $c-1$，能够剪枝去掉部分情况；二是对 $n+m=c+1$ 的情况进行特判：这种情况的合法选择只有一种，即每条次对角线必须选取相同的变量，答案为 $(c-y)!$。

代码

```
const int p = 998244353;
struct save
{
    int n, ua;
};
int n, m, c, ans = 0;
int pc[12][12], mp, pw[12];
save sve[12][12];
void argp(int x, int y, int rg);
void bfs(int ts, int rg);
int main()
{
    pw[0] = 1;
    for (int i = 1; i < 12; i++)
        pw[i] = i * pw[i - 1];
    scanf("%d%d%d", &n, &m, &c);
    if (n + m > c + 1)
        printf("0");
    else
    {
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= m; j++)
                scanf("%d", &pc[i][j]);
        if (n + m == c + 1)
        {
            vector<int> usd;
            for (int i = 2; i <= c + 1; i++)
            {
                int tn = 0;
                for (int j = 1; j <= n; j++)
                    if (i - j > 0 && i - j <= m)
                        if (pc[j][i - j] != 0 && pc[j][i - j] != tn)
                            if (tn == 0)
                            {
                                tn = pc[j][i - j];
```

```
                                for (int a = 0; a < usd.size(); a++)
                                    if (tn == usd.at(a))
                                        goto baded;
                                usd.push_back(tn);
                            }
                            else
                                goto baded;
                        }
                        printf("%d", pw[c - usd.size()]);
                    }
                    else
                    {
                        bfs(2, 0);
                        printf("%d", ans);
                    }
                }
                return 0;
            baded:
                printf("0");
                return 0;
            }
            void argp(int x, int y, int rg)
            {
                if (y > m)
                    argp(x + 1, y - 1, rg);
                else if (x > n || y <= 0)
                    bfs(x + y + 1, rg);
                else
                    for (int i = 1; i <= rg + 1 && i < c; i++)
                    {
                        if (sve[x][y].ua >> i & 1)
                            continue;
                        sve[x][y].n = i;
                        sve[x][y].ua ^= (1 << i);
                        argp(x + 1, y - 1, max(i, rg));
                        sve[x][y].ua ^= (1 << i);
                    }
            }
            void bfs(int ts, int rg)
            {
                if (ts >= n + m)
                {
                    sve[n][m].n = rg + 1;
                    rg++;
                    int ct[12] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
```

```
        vector<int> us;
        for (int x = 1; x <= n; x++)
            for (int y = 1; y <= m; y++)
            {
                int xyn = sve[x][y].n;
                if (pc[x][y] != 0 && pc[x][y] != ct[xyn])
                {
                    if (ct[xyn] == 0)
                    {
                        ct[xyn] = pc[x][y];
                        for (int i = 0; i < us.size(); i++)
                            if (ct[xyn] == us.at(i))
                                return;
                        us.push_back(ct[xyn]);
                    }
                    else
                        return;
                }
            }
        int nans = pw[c - us.size()] / pw[c - rg];
        ans = (ans + nans) % p;
        return;
    }
    for (int x = 1; x <= n; x++)
        if (ts - x > 0 && ts - x <= m)
            sve[x][ts - x].ua |= sve[x][ts - x - 1].ua;
argp(1, ts - 1, rg);
}
```

Day 5

主题：动态规划

时间：2019 年 6 月 28 日 星期五

主要内容：

动态规划

背包：0/1 背包，完全背包

子序列：最长公共子序列，最长上升子序列

题目：

+3	A. JM 的 GG 之路	动态规划
+	B. 鸽子王 qz	最长上升子序列
+	C. 西餐厅的 mm	0/1 背包，完全背包
+3	D. mm 和 tt 吃糖果	动态规划
+4	E. mm 逗黑白猫	BFS
+1	F. mob 的科学麻将	动态规划，数论
–	G. mob 的《麻将与概率系统导论》	

引入

走台阶问题

走台阶问题

- 要走n级台阶，每次可以走一级，也可以走两级，求方案数
- 1<=n<=1e6
- N=4时
- 1111，112，121，211，22，共5种

- F(n)表示走n级台阶的方案数
- F(n)=F(n-1)+F(n-2)
- F(1)=1, F(2)=2
- 代码实现及复杂度

- 1.递归
(记忆化)

```
int save[M] = {1, 1};
int solve(int n)
{
    if(save[n]) return save[n];
    return save[n] = solve(n - 1) + solve(n - 2);
}
```

2.递推

```
int save[M] = {1, 1};
int solve(int n)
{
    for(int i = 2; i <= n; i++)
        save[i] = save[i - 1] + save[i - 2];
    return save[n];
}
```

3. 斐波那契第n项_矩阵快速幂/公式法

动态规划理论

概念，应用条件，解法，复杂度分析，实现方式

- 动态规划（DP）不是一种特定的算法，而是一种解决问题的思想。
- Dynamic Programming，这里的programming不是指编程，而是指一种表格推演的形式，动态规划和【表格】关联十分紧密。
- 动态规划一般指的是，由局部的最优解得到全局的最优解。

动态规划解题方法

- 想出来以下三个点，一道题目就完成了80%
- 状态表示：如何将实际问题用表格（即数组）的形式来表示
- 状态转移：如何由已知状态得到未知状态
- 状态边界：哪些状态不用转移就可以得到

动态规划复杂度分析

- 状态数 × 转移复杂度
- 即：表项个数 * 填每个表项所需时间

动态规划的问题要求

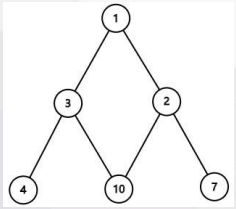
- 有以下三个特性的问题适用动态规划求解
- 最优子结构：全局最优解包含局部最优解
- 重叠子问题：子问题出现大量重叠
- 无后效性：每次决策不会影响之后的决策

动态规划的写法

- 记忆化搜索：
  - 递归+记忆化
  - 不用考虑遍历顺序
- 递推
  - 从状态边界向前递推
  - 无函数调用消耗，较为高效

数字三角形问题

- 有一个由非负整数组成的三角形，第一行只有一个数，除了最下行之外每个数的左下方和右下方各有一个数.
- 从第一行的数开始，每次可以往左下或右下走一格，直到走到最下行，把沿途经过的数全部加起来，如何才能让这个和尽量大.
- 数据范围：
  - 共有N行，$2 \leq N \leq 1000$
  - 每个数小于1e9



- 状态表示：dp[i][j] 表示走到第 i 层第 j 个点能得到得最大值
- 状态转移：dp[i][j] = max(dp[i-1][j], dp[i-1][j-1]) + save[i][j]
- 状态边界：dp[1][1] = save[1][1].

- $O(N^2)$状态
- $O(1)$转移
- 总时间复杂度$O(N^2)$

石子合并问题

- 给出一排n个石子堆，每堆石子都有石子数v_i，可以选取相邻的两个两堆石子$i, i+1$合并得到一堆新的石子放到原位置，但每次合并都需要付出代价$v_i + v_{i+1}$。现在需要把这一排石子合成一堆，求最小代价
- 数据范围$2 \leq n \leq 100, 1 \leq v_i \leq 10^9$
- 4
- 4 2 3 4

- 状态表示：dp[i][j] 表示 i 到 j 的石子合并成一堆的最小代价
- 状态转移：dp[i][j] = min(dp[i][k]+dp[k+1][j])+sum[i][j]，i<=k<j
- 状态边界：dp[i][i] = 0
- 转移顺序?

- 状态数$O(N^2)$
- 转移$O(N)$
- 复杂度$O(N^3)$

石子合并2

- 在一个**圆形**操场的四周摆放N堆石子,现要将石子有次序地合并成一堆.规定每次只能选相邻的2堆合并成新的一堆，并将新的一堆的石子数，记为该次合并的代价。
- 试计算出将N堆石子合并成1堆的最小代价和最大代价.

01背包

- 给定一个容量为v的背包和n个物品,
- 每个物品体积Ci，价值Wi.
- 问最多能向背包中装多少价值的物品?

样例输入：
3 5
1 5
2 12
4 20

样例输出：25

- 1<= n<=1000
- 1<=v, c_i<=1000
- 0<=w_i<=1e9

- 贪心做法
- 搜索做法(手写)
- 复杂度O(2^n)
- 状态表示了解一下?
- dp[i][j]表示考虑前 i 个物品，有容量为 j 时可得到的最大收益

- 状态表示：dp[i][j]表示前 i 个物品，有容量为 j 时的最大价值
- 状态转移：dp[i][j] = max(dp[i-1][j], dp[i-1][j-c]+w)
- 状态边界：dp[0][0] = 0

- 时间复杂度：O(nv)
- 空间复杂度：O(nv)

```
26          for(int i = 1; i <= n; i++)
27              for(int j = 0; j <= v; j++)
28              {
29                  dp[i][j] = dp[i - 1][j];
30                  if(j >= volume[i])
31                      dp[i][j] = max(dp[i][j], dp[i - 1][j - volume[i]] + value[i]);
32              }
```

DP的空间复杂度优化

- 注意到，每一行的状态仅有上一行推出。
- 有这种性质的问题都可以使用一个叫作滚动数组的空间优化。
- 把n行的状态优化成两行。

- 而背包问题，每个状态只会由正上方和左上方的状态转移得到。
- 此时可以把空间优化成一行，但是求值顺序要反过来。
- 如果求值顺序不是反过来的，会发生什么?

```
34          for(int i = 1; i <= n; i++)
35              for(int j = v; j >= volume[i]; j--)
36                  dp[j] = max(dp[j], dp[j - volume[i]] + value[i]);
```

满包问题

- 有n(<=100)个物品，每个物品的价格为ai(<=1000)，现在身上有m(<=10000)元，问

- 最多能花多少钱?
- 有几种把钱花光的方式?

- 状态表示：dp[i][j]表示前 i 个物品装满 j 体积背包的方法数
- 状态转移：dp[i][j] = dp[i-1][j] + dp[i-1][j-ai]
- 状态边界：dp[0][0] = 1

完全背包

- 给定一个容量为v的背包和n种物品，
- 每种物品有无限个且有体积c和价值w
- 问最多能向背包中装多少价值的物品?

- 1≤n ≤1000
- 1≤v, c_i≤1000
- 0≤ w_i ≤ 1e9

- naive做法：

- 状态表示：dp[i][j]表示前 i 个物品，有容量为 j 时的最大价值
- 状态转移：dp[i][j] = max{dp[i-1][j-kc]+kw}，0<=k<=j/w
- 状态边界：dp[0][0] = 0

- 时间复杂度：O(nvv)

- 画表大法好：01背包的空间优化为什么要倒着做?

最长公共子序列 LCS

- 给定两个串A,B,求他们的最长公共子序列的长度
- 子序列/子串
- 数据范围$1 \leq |A|, |B| \leq 1000$

String A	a	c	b	a	e	d
String B	a	b	c	a	d	f

- 状态规划：dp[i][j]表示A的前i个字符与B的前j个字符的LCS长度
- 状态边界：dp[i][0]=dp[0][i]=1
- 状态转移：dp[i][j]=max(dp[i-1][j], dp[i][j-1], dp[i-1][j-1]+(A[i]==B[j]))

- 状态数nm
- 转移$O(1)$
- 复杂度$O(nm)$

最长上升子序列 LIS

- 给定一个数字序列,求它的最长上升子序列的长度，要求子序列中的元素严格上升。

- 6
- 1 5 4 2 3 6
- 输出:4

- 16
- 0 8 4 12 2 10 6 14 1 9 5 13 3 11 7 15
- 输出:6

- 解法1：给自己排序后与自己求一个LCS
- 解法2：
  状态表示：设dp[i]是以S[i]为结尾的LIS长度
  状态边界：dp[i] = 1
  状态转移：dp[i] = max{dp[j]+1}，(j<i && S[j]<S[i])

时间复杂度?

- 如何优化时间?

有些转移，是永远不会用到的，比如 0 5 4 xxxx， 之后的转移，要么从0转移出去，要么从4转移出去，绝对不会从5转移出去。

维护一个数组len[i]，表示长度为 i 的子序列最小的末尾。（推演）

可以看到，len数组具有单调性，于是可以通过二分查找来实现dp数组的转移。

二分查找的实现方法?
std::lower_bound()
std::upper_bound()

- 转化例题：

- 给定两个1到n排列,求最长公共子序列，数据范围3e5

总结

动态规划

动态规划

- 状态表示、状态转移、状态边界
- 最优子结构、重叠子问题、决策无后效性
- 时间复杂度计算
- 时空复杂度优化

- 后续内容：
- 背包进阶：多重背包、二维费用背包、依赖背包、...
- 时间优化：单调dp、数据结构优化dp、斜率dp、四边形dp、...
- dp类型：状压dp、概率dp、博弈论、...
- dp与其它内容相结合

A. JM 的 GG 之路

内存限制：512 MiB

时间限制：1000 ms

题目描述

昨天的觉醒之路似乎很多人都走不过，那就不能觉醒了，只能 GG 了。JM 为你量身打造了 GG 之路。

这个路就很好走了，他是一个 n 阶的台阶，每阶的编号为 $1,2,\dots,n$，初始你就站在 1 号台阶上。每一步你有 k 种不同的步长可以走。现在你需要求出，你有多少种不同的走法可以走到 n 号台阶上。每种步长都没有使用次数的限制。

因为方案数可能很多，你需要输出答案对 10^9+7 取模。

输入格式

第一行两个正整数 n 和 k，表示台阶数和可选的步长种类数。

接下来一行 k 个正整数，表示你可选的步长有哪些。保证所有的步长互不相同。

输出格式

输出一行一个非负整数表示答案。

数据范围与提示

$2 \le n \le 10^5$

$1 \le k \le \min(n-1, 50)$

解题思路

设步长为 $c_1, c_2, \dots, c_k$。到达第 x 阶一共有 k 种走法：从第 $x-c_1$ 阶走 c_1 步长，从第 $x-c_2$ 阶走 c_2 步长……。考虑动态规划，从第 0 阶到达第 x 阶的走法数 $dp(x) = \sum_{i=1}^{k} dp(x-c_i)$。$dp(n-1)$ 即为答案。状态边界为：$dp(ng) = 0 (ng < 0)$，$dp(0) = 1$。状态数为 $O(n)$，状态转移为 $O(k)$，总复杂度为 $O(nk)$，可以接受。

代码

```
const int p = 1000000007;
int n, k;
int mtd[210000], stps[50];
int main()
{
    scanf("%d%d", &n, &k);
    n--;
    for (int i = 0; i < k; i++)
        scanf("%d", &stps[i]);
    for (int i = 1; i <= n; i++)
        for (int j = 0; j < k; j++)
            if (i - stps[j] > 0)
                mtd[i] = (mtd[i] + mtd[i - stps[j]]) % p;
            else if (i == stps[j])
                mtd[i] = (mtd[i] + 1) % p;
```

```
    printf("%d", mtd[n]);
    return 0;
}
```

B. 鸽子王 qz

内存限制：512 MiB

时间限制：1000 ms

题目描述

众所周知，qz 是一个万年鸽子王，并且他可以从鸽人中获得一些莫名其妙的快感？

现在有 N 个人按顺序对 qz 发出了邀请，如果 qz 咕咕掉第 i 个人，就可以获得 a_i 点快感。

当然，由于 qz 的贪得无厌，他通过咕咕获得快感必须越来越高，如果一个人能带给他的快感小于或等于前一个人，他就不屑于咕掉他。

现在 qz 想知道，对于这样的邀请，他最多可以咕掉多少个？

输入格式

第一行一个整数 N，表示发出邀请的总人数。

第二行 N 个整数，第 i 个表示咕掉第 i 个人可以获得的快感 a_i。

输出格式

输出一行一个整数，表示 qz 最多可以咕掉多少人。

数据范围与提示

$1 \le N \le 10^5$

$1 \le a_i \le 10^9$

解题思路

求最长上升子序列的长度即可。

代码

```
int n, mx;
int pgt[110000], gglt[110000];
int main()
{
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
        scanf("%d", &pgt[i]);
    for (int i = 0; i < n; i++)
    {
        gglt[lower_bound(gglt, gglt + mx, pgt[i]) - gglt] = pgt[i];
        if (lower_bound(gglt, gglt + mx, pgt[i]) == gglt + mx) mx++;
    }
    printf("%d", mx);
    return 0;
}
```

C. 西餐厅的 mm

内存限制：512 MiB

时间限制：1000 ms

题目描述

mm 要搬寝室了，作为一个吃货，她认为和室友 tt、cc 饯别的最好方式是大吃一顿，于是她们来到点评还不错的悦阅西餐厅。

看到菜单后，mm 十分忧愁，她胃口是有限的，可是什么都想吃！餐单上有 n 份菜，她给每份菜定义了饱腹度 x_i 和美味度 y_i，她的胃容量只有有限的 m。

另外，有的菜 mm 认为比较健康比如蔬菜沙拉，无论吃多少份都没有问题；然而对于油荤的披萨和牛排，mm 认为一餐吃超过一份会发胖，这是她不能接受的。

mm 想知道自己在既不被撑死也不发胖的情况下能吃到的菜的美味度之和为多少？

输入格式

第一行两个正整数 n, m。

接下来 n 行，每行三个数 x_i, y_i, k_i，表示每种菜的饱腹度、美味度、最高份数量。其中 $k_i \in \{1, -1\}$，当 $k_i = -1$ 时表示份数无限制，当 $k_i = 1$ 时表示只能吃一份。

输出格式

输出一行一个整数，表示 mm 吃到的菜的美味度之和。

数据范围与提示

$1 \le n \le 200$

$1 \le m \le 2 \cdot 10^5$

$1 \le x_i \le 2 \cdot 10^5$

$1 \le y_i \le 10^5$

$k_i \in \{1, -1\}$

解题思路

0/1 背包与完全背包的结合。考虑胃容量容量为 j，新增第 i 个菜品时的最大美味度$dp(i,j)$，若 $k_i = 1$，则 $dp(i,j) = \max(dp(i-1,j), dp(i-1,j-x_i) + y_i)$；若 $k = -1$，则 $dp(i,j) = \max(dp(i-1,j-nx_i) + ny_i)$。$dp(n,m)$ 即为答案。状态边界为 $dp(0,-|n|) = 0$。

代码

```
struct dsh
{
    llong spa, bty;
    bool lim;
};
int n, m, knd = 0;
llong dp[210000];
vector<dsh> tds;
void upt(int i, int j)
{
```

```
    if (j - tds.at(i).spa >= 0)
        dp[j] = max(dp[j], dp[j - tds.at(i).spa] + tds.at(i).bty);
}
int main()
{
    scanf("%d%d", &n, &m);
    for (int i = 0; i < n; i++)
    {
        llong a, b, c;
        scanf("%lld%lld%lld", &a, &b, &c);
        if (c == -1)
            tds.push_back(dsh{ a, b, false });
        else
            tds.push_back(dsh{ a, b, true });
    }
    for (int i = 0; i < n; i++)
        if (tds.at(i).lim)
            for (int j = m; j > 0; j--)
                upt(i, j);
        else
            for (int j = 1; j <= m; j++)
                upt(i, j);
    printf("%lld", dp[m]);
    return 0;
}
```

D. mm 和 tt 吃糖果

内存限制：512 MiB

时间限制：1000 ms

题目描述

mm 和 tt 来到渡渡鸟王国旅行，渡渡鸟王国是一片 $n \times n$ 块格子的花园，每块格子有一个坐标，左上角为入口 $(1,1)$，右下角为出口 (n,n)。

花园的 k 个格子上有一定数量的糖果，作为两枚吃货，mm 和 tt 致力于收集尽可能多的糖果。

两人初始是在入口为 $(1,1)$，由于旅行时间紧张，两人只能向目的地方向行走（即向右走或向下走）；另外，为了防止花园地面塌陷，tt 决定等 mm 先行到达出口再从入口出发。

mm 和 tt 每到达一个有糖果的格子，便会拿走格子上的糖果，当然，对于同一个格子，mm 拿走了格子上的糖果，tt 再到达时便没有糖果可拿了。作为好朋友，mm 和 tt 会将拿到的糖果一起分享，她们想知道如何走才能使拿到的糖果数之和最大。

输入格式

第一行两个整数 n,k。

接下来 k 行，每行三个正整数 x_i,y_i,s_i，表示在 (x_i,y_i) 处有 s_i 颗糖果。

保证对于任意的 $i \neq j$ 不存在 $x_i = x_j$ 且 $y_i = y_j$。

输出格式

输出一行一个整数，表示 mm 和 tt 能获得的最大糖果总数。

数据范围与提示

$1 \le n \le 20$

$0 \le k \le n^2$

$1 \le x_i, y_i \le n$

$1 \le s_i \le 10^5$

解题思路

两人先后出发与两人同时出发并无本质区别。考虑当 mm 到达点 (i_m, j_m) 且 tt 到达点 (i_t, j_t) 时能获得的最大糖果总数 $dp(i_m, j_m, i_t, j_t)$，设 (x, y) 处有 $s(x, y)$ 个糖果，设

$$mfdp(i_m, j_m, i_t, j_t) = \max \begin{pmatrix} dp(i_m - 1, j_m, i_t - 1, j_t), dp(i_m, j_m - 1, i_t - 1, j_t), \\ dp(i_m - 1, j_m, i_t, j_t - 1), dp(i_m, j_m - 1, i_t, j_t - 1) \end{pmatrix}$$

采用 BFS 搜索，一个点不会先后被两人到达，只可能同时到达。若 $i_m = i_t, j_m = j_t$，则

$$dp(i_m, j_m, i_t, j_t) = mfdp(i_m, j_m, i_t, j_t) + s(i_m, j_m)$$

否则

$$dp(i_m, j_m, i_t, j_t) = mfdp(i_m, j_m, i_t, j_t) + s(i_m, j_m) + s(i_t, j_t)$$

$dp(n, n, n, n)$ 即为答案。状态边界为 $dp(a, b, c, d) = 0$，其中 $\min(a, b, c, d) \le 0$。

代码

```
int n, k, mp[24][24];
int sum[24][24][24][24];
int max(int a, int b, int c, int d)
{
    return max(max(a, b), max(c, d));
}
void upt(int x, int y, int i, int j)
{
    int fdp[2][2];
    fdp[0][0] = sum[x - 1][y][i - 1][j];
    fdp[1][0] = sum[x][y - 1][i - 1][j];
    fdp[0][1] = sum[x - 1][y][i][j - 1];
    fdp[1][1] = sum[x][y - 1][i][j - 1];
    sum[x][y][i][j] = max(fdp[0][0], fdp[0][1], fdp[1][0], fdp[1][1]);
    sum[x][y][i][j] += mp[x][y];
    if (x != i || y != j)
        sum[x][y][i][j] += mp[i][j];
}
int main()
{
    scanf("%d%d", &n, &k);
    for (int i = 0; i < k; i++)
    {
```

```
        int x, y, c;
        scanf("%d%d%d", &x, &y, &c);
        mp[x][y] = c;
    }
    for (int i = 2; i <= 2 * n; i++)
        for (int x = 1; x <= n; x++)
            if (i - x > 0 && i - x <= n)
                for (int y = 1; y <= n; y++)
                    if (i - y > 0 && i - y <= n)
                        upt(x, i - x, y, i - y);
    printf("%d", sum[n][n][n][n]);
    return 0;
}
```

E. mm 逗黑白猫

内存限制：512 MiB

时间限制：1000 ms

题目描述

mm 在渡渡鸟幼儿园发现一堆黑白猫玩具，mm 可是很仰慕围棋大佬，于是决定把它们摆在棋盘上当围棋（并不）玩耍。

黑白猫盘由一个 4×4 的方格构成，棋盘的每一方格中放有 1 只猫，共有 8 只白猫和 8 只黑猫。在棋盘上拥有 1 条公共边的 2 个方格称为相邻方格。在玩黑白猫游戏时，每一步可将任何 2 个相邻方格中猫互换位置。mm 要做的就是通过多次互换位置把给定的初始猫排列状态变成目标猫排列状态。

输入格式

输入文件共有 8 行。前 4 行是初始猫排列状态，后 4 行是目标猫排列状态。

每行有 4 个数字 0 或 1（无空格），分别表示该行放置的猫颜色，其中 0 表示白猫，1 表示黑猫。

输出格式

第一行输出互换步数 n 。

接下来 n 行，每行 4 个正整数（无空格），分别表示该步交换的两个相邻方格的位置。例如， $abcd$ 表示将棋盘上 (a,b) 处的猫与 (c,d) 处的猫换位。

注意，如果有多种合法方案，你的方案需要保证：

1. 所需步数最小。
2. 输出时保证 $a<c$ 或 $a=c,b<d$ 。
3. 在满足以上两个条件的前提下，输出字典序最小的方案。

解题思路

一共有 16 个位置，每个位置有两种状态，考虑状态压缩。一共有 24 种交换方式，BFS 搜索初状态至末状态即可。在每一个状态记录互换步数及方式，相同步数取字典序最小的。

或者进行两次搜索，第一次搜索由末状态至初状态，进行 BFS 搜索，得出互换步数，并标记所有能在互换步数之内到达末状态的状态。第二次搜索由初状态至末状态，进行 DFS 搜索，每次搜索字典序最小的方案，且只访问第一次搜索标记过、使得距离末状态步数更小的方案。

代码

使用第一种方案的代码：

```
const int way[24] = { 49152,34816,24576,17408,12288,8704,4352,3072,2176,153
6,1088,768,544, 272,192,136,96,68,48,34,17,12,6,3 };
const int wayop[24] = { 1112,1121,1213,1222,1314,1323,1424,2122,2131,2223,2
232,2324,2333, 2434,3132,3141,3233,3242,3334,3343,3444,4142,4243,4344 };
const int ways[24][2] = { { 15,14 },{ 15,11 },{ 14,13 },{ 14,10 },{ 13,12 }
,{ 13,9 },{ 12,8 },  { 11,10 },{ 11,7 },{ 10,9 },{ 10,6 },{ 9,8 },{ 9,5 },{
 8,4 },{ 7,6 }, { 7,3 },{ 6,5 },{ 6,2 },{ 5,4 },{ 5,1 },{ 4,0 },{ 3,2 },{ 2
,1 },{ 1,0 } };
struct wy
{
    int st = -1;
    string w;
};
int from, to, tsp;
wy sve[65540];
vector<int> bfs, tmp;
int bin2dec(int bn)
{
    const int pws[4] = { 1, 10, 100, 1000 };
    int ans = 0;
    for (int i = 3; i >= 0; i--)
        if (bn >= pws[i])
        {
            bn -= pws[i];
            ans += 1 << i;
        }
    return ans;
}
int main()
{
    for (int i = 0; i < 4; i++)
    {
        int orgs;
        scanf("%d", &orgs);
        from += bin2dec(orgs) << (12 - 4 * i);
    }
    for (int i = 0; i < 4; i++)
    {
        int orgs;
```

```
        scanf("%d", &orgs);
        to += bin2dec(orgs) << (12 - 4 * i);
    }
    bfs.push_back(from);
    sve[from].st = 0;
    while (!bfs.empty())
    {
        for (int i = 0; i < bfs.size(); i++)
        {
            int ts = bfs.at(i);
            for (int j = 0; j < 24; j++)
                if ((ts >> ways[j][0] & 1) != (ts >> ways[j][1] & 1))
                {
                    int np = bfs.at(i) ^ way[j];
                    if (sve[np].st == -1)
                    {
                        sve[np].st = sve[ts].st + 1;
                        sve[np].w = sve[ts].w + char(j);
                        tmp.push_back(np);
                    }
                    else if (sve[np].st == sve[ts].st + 1)
                        sve[np].w = min(sve[np].w, sve[ts].w + char(j));
                }
        }
        if (sve[to].st != -1)
        {
            printf("%d\n", sve[to].st);
            for (int i = 0; i < sve[to].w.length(); i++)
                printf("%d\n", wayop[sve[to].w.at(i)]);
            break;
        }
        swap(bfs, tmp);
        tmp.clear();
    }
    return 0;
}
```

使用第二种方案的代码：（更高效）

```
const int way[24] = { 49152,34816,24576,17408,12288,8704,4352,3072,2176,153
6,1088,768,544, 272,192,136,96,68,48,34,17,12,6,3 };
const int wayop[24] = { 1112,1121,1213,1222,1314,1323,1424,2122,2131,2223,2
232,2324,2333, 2434,3132,3141,3233,3242,3334,3343,3444,4142,4243,4344 };
const int ways[24][2] = { { 15,14 },{ 15,11 },{ 14,13 },{ 14,10 },{ 13,12 }
,{ 13,9 },{ 12,8 }, { 11,10 },{ 11,7 },{ 10,9 },{ 10,6 },{ 9,8 },{ 9,5 },{
8,4 },{ 7,6 }, { 7,3 },{ 6,5 },{ 6,2 },{ 5,4 },{ 5,1 },{ 4,0 },{ 3,2 },{ 2,
1 },{ 1,0 } };
struct wy
```

```
{
    int st = -1;
};
int from, to, tsp;
wy sve[65540];
vector<int> bfs, tmp;
int bin2dec(int bn)
{
    const int pws[4] = { 1, 10, 100, 1000 };
    int ans = 0;
    for (int i = 3; i >= 0; i--)
        if (bn >= pws[i])
        {
            bn -= pws[i];
            ans += 1 << i;
        }
    return ans;
}
int main()
{
    for (int i = 0; i < 4; i++)
    {
        int orgs;
        scanf("%d", &orgs);
        from += bin2dec(orgs) << (12 - 4 * i);
    }
    for (int i = 0; i < 4; i++)
    {
        int orgs;
        scanf("%d", &orgs);
        to += bin2dec(orgs) << (12 - 4 * i);
    }
    bfs.push_back(to);
    sve[to].st = 0;
    while (!bfs.empty())
    {
        for (int i = 0; i < bfs.size(); i++)
        {
            int ts = bfs.at(i);
            for (int j = 0; j < 24; j++)
                if ((ts >> ways[j][0] & 1) != (ts >> ways[j][1] & 1))
                {
                    int np = bfs.at(i) ^ way[j];
                    if (sve[np].st == -1)
                    {
```

```
                    sve[np].st = sve[ts].st + 1;
                    tmp.push_back(np);
                }
            }
        }
        if (sve[from].st != -1)
        {
            printf("%d\n", sve[from].st);
            break;
        }
        swap(bfs, tmp);
        tmp.clear();
    }
    tsp = from;
    while (tsp != to)
        for (int j = 0; j < 24; j++)
            if ((tsp >> ways[j][0] & 1) != (tsp >> ways[j][1] & 1))
            {
                int np = tsp ^ way[j];
                if (sve[tsp].st - sve[np].st == 1)
                {
                    tsp = np;
                    printf("%d\n", wayop[j]);
                    break;
                }
            }
    return 0;
}
```

备注

状态压缩的方案，左图为状态压缩前的位置编号，右图为对应位置在二进制数中的位数。

11	12	13	14
21	22	23	24
31	32	33	34
41	42	43	44

→

15	14	13	12
11	10	9	8
7	6	5	4
3	2	1	0

用 ways 数组进行检测：若对应位不同，则可以交换，0 变为 1，1 变为 0，相当于进行了一次异或运算，将状态与预先处理好的 way 数组中的对应值取异或即可。最后得到答案后输出，wayop 数组中的对应值即为题目所求的原始位置编号，输出即可。

F. mob 的科学麻将

内存限制：512 MiB

时间限制：1000 ms

题目描述

不知不觉间，mob 已经在盐中的肉体改造部待了一年了，体魄强健的他很快又发现了新的爱好——“麻将”！虽然 mob 是地球上超能力最为强大的少年，但当他与 Saki，Teru 等天才麻将少女对战时，他却由于执念从不使用超能力！但由于他无法像对手一样看破牌山，因此评估自己的配牌，决定本局究竟是进攻还是防守成了一件非常重要的事情。mob 很快找到了一种方法去量化自己的起手配牌价值，以下是他自己总结出的经验公式：

一套麻将有 n 张牌，第 i 张牌的点数为 a_i，这张牌的任何一个子集都是一种可能的“配牌”，mob 会用以下方式决定拿到某种配牌时应该进攻还是防守：设一种配牌所含有的牌的点数分别为 $b_1, b_2, \dots, b_k$ (这是一种含 k 张牌的配牌)，设 $b\{k\}$ 数列中某个子序列的价值为组成这个子序列的点数和，设一种配牌的价值为相应 $b\{k\}$ 数列的**所有非空子序列的价值的和**。

如：配牌 $[1,2,3]$，它的所有非空子序列有 $[1][2][3][1,2][1,3][2,3][1,2,3]$，其价值分别为 $1,2,3,3,4,5,6$，因此这个配牌的价值为 $1+2+3+3+4+5+6=24$。

若一种配牌的价值是 m 的倍数，那么称这个配牌为好的配牌。

问：一套麻将中有多少个子集是“好的配牌”。

由于答案可能非常大，你只需要输出答案对 10^9+7 取模的结果。

输入格式

第一行两个正整数 n, m。

下面一行 n 个正整数用空格隔开，第 i 个正整数表示 a_i。

输出格式

输出一行一个正整数表示答案。

数据范围与提示

$n \le 2000$

$m \le 5000$

$0 \le a_i \le m$

解题思路

考虑一个只选取了前 i 个数，在 $\bmod m$ 意义下价值为 j，长度为 k 的子集的个数 $dp(i,j,k)$。该子集增加第 $i+1$ 个数，其价值增加 $j+2^k a_i$，长度 $+1$；不增加，其价值不变，长度不变。$dp(i,2j+2^k a_i,k) = dp(i-1,j,k-1) + dp(i-1,2j+2^k a_i,k)$。答案为 $\sum_{k'=1}^{n} dp(n,0,k')$。复杂度为 $O(n^2 m)$，不能接受。考虑优化。

一个子集增加一个数后，其价值变化满足 $j_{k+1} = 2j_k + 2^k a_i$，记 $2^{k-1}h_k = j_k$，变形得到 $h_{k+1} = h_k + a_i$。另一方面，根据算术基本定理，存在两个自然数 a, b，满足 $m = 2^a \cdot b$，其中 $a \le \log m \le 13$，b 为奇数。由于我们的目的是使得价值 $2^{k-1}h_k$ 在 $\bmod m$ 意义下为 0，若 $k-1 \ge a$，则 $2^{k-1}h_k \equiv 0 \pmod m$ 当且仅当 $b | h_k$，与 k 的具体的大小无关。故不需要记录 j_k，仅需记录 h_k，且 $k \ge 14$ 时均按照 $k = 14$ 处理即可。复杂度为 $O(nm\log m)$。

代码

```
const int p = 1000000007;
int n, m, gans = 0;
int as[2020], pw[20];
const int rg = 15;
int san[rg + 1][5020];
int nmod(int a, int mp)
{
    int ans = a % mp;
    if (ans < 0)
        ans += mp;
    return ans;
}
int main()
{
    pw[0] = 1;
    for (int i = 1; i < 20; i++)
        pw[i] = (pw[i - 1] * 2) % p;
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++)
        scanf("%d", &as[i]);
    san[0][0] = 1;
    for (int i = 1; i <= n; i++)
    {
        for (int j = min(i, rg); j > 0; j--)
            for (int x = 0; x < m; x++)
            {
                int nf = (x + as[i]) % m;
                san[j][nf] = (san[j][nf] + san[j  - 1][x]) % p;
            }
        for (int x = 0; x < m; x++)
        {
            san[rg - 1][x] = (san[rg - 1][x] + san[rg][x]) % p;
            san[rg][x] = 0;
        }
    }
    for (int i = 1; i <= min(n, rg - 1); i++)
        for (int x = 0; x < m; x++)
            if ((x * pw[i - 1]) % m == 0)
                gans = (gans + san[i][x]) % p;
    printf("%d", gans);
    return 0;
}
```

G. mob 的《麻将与概率系统导论》

内存限制：512 MiB

时间限制：10000 ms

题目描述

在打麻将的同时，mob 也在十分刻苦地学习《概率论与数理统计》，他逐渐从书本和麻将实践中得到了一些启发，发现麻将的概率问题复杂性来源于其模型的高度复杂性，于是他试图发明一套新的概率方法来求解这些过于复杂的概率和统计问题。功夫不负有心人，他最终找到了规律并发明了一种他称之为“概率系统”的理念，其中有一种统计问题是整个理论的核心：

有 n 个概率模式，每个概率模式有两个参数 p_i, b_i，这个概率模式组成全集 U，对任何一个 U 的子集 S，我们称它的 mob 值为：S 中所有概率模式对应的 b_i 的异或和。此外，我们称 S 是合法的当且仅当：S 中所有概率模式所对应的 b_i 两两按位与的结果均为 0，我们称一个 S 出现的概率为 S 中所有概率模式对应的 p_i 的积。

为了验证你是否掌握了这套概率系统理论，mob 给了你 m 个询问，每个询问都给出一个 x_i，询问：

所有 mob 值为 x 的合法子集的出现概率之和为多少？

由于这个数可能很大，你只需要输出答案对 998 244 353 取模的结果即可

输入格式

第一行一个正整数 n，表示概率模式数。

第 $2-(n+1)$ 行每行两个正整数 p_i, b_i，分别表示第 i 种概率模式的两个参数。

接下来一行一个正整数 m 表示询问个数。

接下来 m 行每行一个正整数 x_i，表示询问。

输出格式

输出 m 行，第 i 行一个正整数表示第 i 次询问的答案。

样例

样例输入	样例输出
3 1 1 1 2 1 3 3 1 2 3	1 1 2

数据范围与提示

$n, m \leq 10^5$

$0 \leq p_i < 998\,244\,353$

$0 < b_i < 2^{17}$

$0 < x_i < 2^{17}$

Day 6

主题：基础数据结构

时间：2019 年 7 月 1 日 星期一

主要内容：

并查集

优先队列：优先队列，priority_queue

二叉搜索树：set，map

线段树：前缀和，线段树，lazy 标记，树状数组

题目：

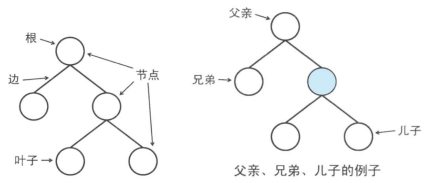
+5	A. nocriz 送温暖	前缀和，求逆元，快速幂
+	B. nocriz 修路(改)	并查集
+5	C. nocriz 坐火车	树状数组
+	D. JM 的小学数学题	优先队列
+1	E. nocriz 维护括号序列	线段树
+	F. nocriz 与队列计算机	线段树
–	G. nocriz 和巨佬谈人生	优先队列，树上倍增
–	H. nocriz 与'Swappy Tree'	

简介

此次课内容较多，因此可能讲完难度也有一点大。我做了充分的准备，希望能够成功。
例题往往出自真实比赛，有一定思维难度，但无算法难度，旨在给同学们对算法竞赛更深入的体验，请放心食用。
如果你会做题，可以考虑简单讲一讲。
如果你不会做某道题，这很正常，如果能够在讲课时间内切完所有的题，就不必来了。课件上很多题目都是希望大家课下看会。
当然，会有部分简单题目。

Overview

树



父亲、兄弟、儿子的例子

图中的圆圈代表“节点”，线代表“边”。我们把树的根所表示的节点提到整棵树的最上面。每个节点在保存了各自的信息之外，都还拥有儿子节点。把自己作为儿子的节点叫做父亲节点。拥有同一父亲的节点互为兄弟节点。没有儿子的节点叫做叶子节点。“二叉树”是树中所有节点的儿子个数都不超过 2 的树。

图与树

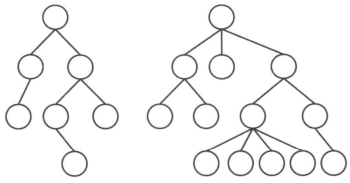

二叉树的例子和非二叉树的树的例子

图 G 是一对 (V, E)，其中 V 是有穷集，E 是边集。
自由树 (无根树) 是连通的，无回路的无向图。
有根树是一颗自由树，他有一颗与其他点不同的结点，称为树的根。

并查集

并查集 (Disjoint-Set) 是一种可以动态维护若干个不重叠的集合，并支持合并与查询的数据结构。详细地说，并查集包括如下两个基本操作：

- Get，查询一个元素属于哪一个集合。
- Merge，把两个集合合并成一个大集合。

并查集-2

为了具体实现并查集这种数据结构，我们首先需要定义集合的表示方法。在并查集中，我们采用“代表元”法，即为每个集合选择一个固定的元素，作为整个集合的“代表”
使用一棵树形结构存储每个集合，树上的每个节点都是一个元素，树根是集合的代表元素。整个并查集实际上是一个森林 (若干棵树)。我们仍然可以维护一个数组 fa 来记录这个森林，用 fa[x] 保存 x 的父节点。特别地，令树根的 fa 值为它自己。

并查集-3

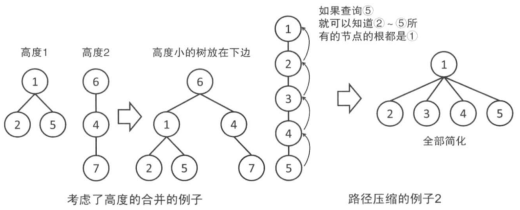
这样一来，在合并两个集合时，只需连接两个树根 (令其中一个树根为另一个树根的子节点，即 fa[root1]=root2)。不过在查询元素的归属时，需要从该元素开始通过 fa 存储的值不断递归访问父节点，直至到达树根。

复杂度分析与优化

如果我们有 $2n$ 次操作，那么我们可以首先拿出前 n 次操作连接出一棵深度为 $n+1$ 的树，而后再用 n 次询问，每次询问最深的那个结点，这样我们就达到了 $O(n^2)$ 的复杂度，这显然是不能够实用的。
优化 1: 按秩合并
在每次合并都过程中，将大小比较小的集合接到大小比较大的集合上面，这样就可以保证树的深度不超过 $O(\log n)$ 了。
简单证明：每一次走向父亲时，子树的大小都最小翻倍。

复杂度分析与优化

优化 2: 路径压缩
同时采用了按秩合并和路径压缩两种优化后，复杂度可以达到 $O(n\,\alpha(n))$，基本接近线性 α 为反阿克曼函数，增长速度远比对数函数缓慢。($\forall N \leq 2^{2^{10^{19729}}}, \alpha(N) < 5$)。著名科学家 R.E. Tarjan 在 1975 年发表的论文中给出了证明。
在日常应用中，往往只进行路径压缩优化，代码简短，复杂度为 $O(n \log n)$ 可以满足大部分做题需要。



考虑了高度的合并的例子　　路径压缩的例子2

并查集的实现

下面是著名选手 Ildar Gainullin 的实现

```
int dsu[N];
int fnd(int x){
    return (x == dsu[x])?x:dsu[x] = fnd(dsu[x]);
}
int merge(int u,int v){
    dsu[fnd(u)] = fnd(v);
}
```


简单练习题

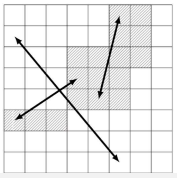
你现在有 n 个点，操作 m 次。($2 \leq n, m \leq 10^5$)
每次将两个点连接在一起并问有多少个联通分量。

- 一开始有 n 个联通分量
- 每次询问两个点是否在一个集合，如果在一个集合什么也不做，不然合并集合，联通分量个数减一

CERC2018 - Shooter Island

有一个 $N * M$ 的冰湖，$N \leq 100,000, M \leq 50$。共有 $Q(Q \leq 200,000)$ 次操作，每次操作可能有两种:

1. 给出 $(x_1, y_1), (x_2, y_2)$，以此为顶点内的所有冰都被轰炸清除，该区域将被液态水填满。
2. 给出 $(x_1, y_1), (x_2, y_2)$，问一个周长为 0.31416 的圆形小船能不能从一点走向另一点。



CERC2018 - Shooter Island - Solutions

我们维护一个 $N * M$ 的网格，在上面做一个并查集。
这样，询问就可以简单的解决了。但是我们要如何维护并查集呢？如果我们遍历所有被清空的区间内的块，显然会超时。但是我们可以注意到，操作之前操作过的块是没有意义的。于是我们可以另外维护 50 个并查集，同时在这些并查集中维护集合中最大的数。复杂度 $O(N * M * \alpha(N * M) + Q * M * \alpha(N))$

优先队列

优先队列为支持以下操作的数据结构:

- 插入一个数
- 获取最小值并且将其从堆中删除

堆的 STL 使用

堆的具体实现是使用完全二叉树。在算法竞赛中，我们往往使用 C++ STL 中的 priority_queue 来代替手写堆。

```
#include <queue>
...
priority_queue< int,vector<int>,greater<int> > pq;
pq.push(2);
cout<<pq.top()<<endl;
pq.pop();
```

POJ 2431 - Expedition

你需要驾驶一辆卡车行驶 L 单位距离。最开始时，卡车上有 P 单位的汽油。卡车每开 1 单位距离需要消耗 1 单位的汽油。在途中一共有 N 个加油站。第 i 个加油站在距离起点 A_i 单位距离的地方，可以给卡车加 B_i 单位汽油。假设卡车的燃料箱的容量是无限大的。
那么请问卡车是否能到达终点？如果可以，最少需要加多少次油？

POJ 2431 - Expedition - solution

我们稍微变换一下思考方式。在卡车开往终点的途中，只有在加油站才可以加油。但是，如果认为“在到达加油站时，就获得了一次在之后的任何时候都可以加 B_i 单位汽油的权利”。而在之后需要加油时，就认为是在之前经过的加油站加的油就可以了。那么，因为希望到达终点时加油次数尽可能少，所以当燃料为 0 了之后再进行加油，这样就能够获得最优的决策。贪心选择能加油量 B_i 最大的加油站，便是正确决策。

POJ 2431 - Expedition - solution

为了高效地进行上述操作，我们可以使用从大到小的顺序依次取出数值的优先队列。在经过加油站时，往优先队列里加入 B_i。
当燃料箱空了时，

- 如果优先队列也是空的，则无法到达终点。
- 否则取出优先队列中的最大元素，并用来给卡车加油。

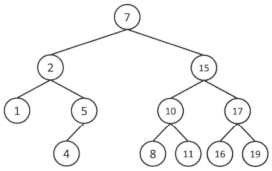
二叉搜索树

二叉搜索树是能够高效进行以下操作的数据结构：

- 插入一个数
- 删除一个数
- 求比一个值 x 小的数中最大的一个（求前驱）
- 求比一个值 x 大的数中最小的一个（求后继）

所谓高效是指随机情况下一次操作期望时间复杂度为 $O(\log n)$。

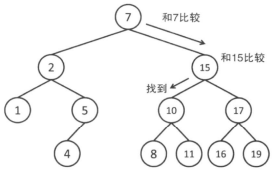
二叉搜索树-结构



二叉搜索树是满足以下条件的二叉树。

- 一个结点的值不小于它的左子树中任意节点的值
- 一个结点的值不大于它的右子树中任意节点的值

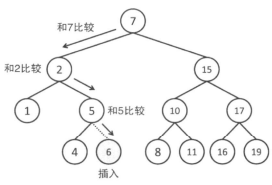
操作：搜索



在上图的二叉搜索树中查询是否存在 10：

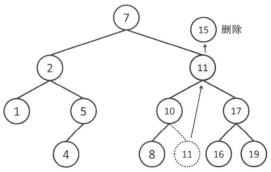
- 根节点的数值是 7，比 10 小，所以往右儿子节点走。
- 走到的节点的数值是 15，比 10 大，所以往左儿子节点走。
- 走到的节点是 10

操作：插入



- 试图查找这个数值的节点，就可以知道其对应的节点的所在位置
- 在那个位置插入新的节点即可

操作：删除



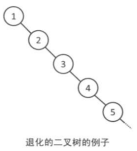
如果删除了 15 所在的节点，那么它的两个儿子 10 和 17 就悬空了。
于是，把 11 提到 15 所在的位置就可以解决问题

操作：删除-2

一般来说，需要根据下面几种情况分别进行处理。

- 需要删除的节点没有左儿子，那么就把右儿子提上去。
- 需要删除的节点的左儿子没有右儿子，那么就把左儿子提上去。
- 以上两种情况都不满足的话，就把左儿子的子孙中最大的节点提到需要删除的节点上。

二叉搜索树-复杂度退化

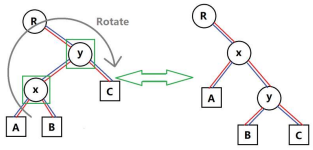


退化的二叉树的例子

考虑一下根据之前的说明，在向二叉树插入节点时，如果以 $1, 2, 3, 4, 5, \ldots$ 的顺序插入的话会产生什么问题。如果这样插入节点的话，二叉搜索树就会变得如同一条链表一般。如果按照这个顺序插入 n 个元素，树的高度就会变成 n，那么所有的操作就都需要 $O(n)$ 时间才能完成了。所有操作本该都是 $O(\log n)$ 时间的二叉搜索树，每次操作变成了 $O(n)$。

平衡树简介

关键便在于树高过高，
而平衡二叉树恰好能避免这样的问题。平衡二叉树为了避免这种退化的情况发生，巧妙地使用旋转操作来保持树的平衡。
信息学竞赛中常用的有 splay、treap、替罪羊树，这里不再涉及。



STL 简介

set 和 map 的使用

C++ STL 中已经有现成的平衡树可以使用，就是 set 和 map。
set 和 map 的本质都是红黑树，还挺快的。

STL 简介

C++ STL 中 set 的使用

```
#include <set>
set<int> S;
set<int>::iterator it;
S.insert(233);
S.erase(233);
S.count(233);
it = S.lower_bound(234);
it--;
```


STL 简介

C++ STL 中 map 的使用

```
#include <map>
map<int,int> Mp;
Mp[0] = 1;
cout<<Mp[1]<<endl;
```


STL 简介

set，map 的部分函数们

- begin()–返回指向第一个元素的迭代器
- clear()–清除所有元素
- count()–返回某个值元素的个数
- empty()–如果集合为空，返回 true
- end()—返回指向最后一个元素的迭代器
- erase()–删除集合中的元素
- find()–返回一个指向被查找到元素的迭代器
- insert()–在集合中插入元素
- lower_bound()–返回指向大于（或等于）某值的第一个元素的迭代器
- size()–集合中元素的数目
- upper_bound()–返回大于某个值元素的迭代器

前缀和与差分

简单问题 1.1-前缀和

现有一长度为 n 的数列 a，有 q 次询问，每次给出 l, r，求 $a_l + a_{l+1} + \ldots + a_r$
$n, q \leq 10^6$

$O(n)$ 时间建立前缀和数列 s，即 $\forall 1 \le i \le n, s_i = \sum_{j=1}^{i} a_j$，则 $a_l + a_{l+1} + \ldots + a_r = s_r - s_{l-1}$，询问可以 $O(1)$ 回答，问题解决。

```
s[1] = a[i];
for(int i=2;i<=n;i++)s[i] = s[i-1]+a[i];
for(int i=0;i<q;i++){
    cin>>l>>r;
    cout<<s[r]-s[l-1]<<endl;
}
```


简单问题 1.1a-差分

现有一长度为 n 的数列 a，有 q 次询问，每次给出 l, r, x，求 $a_l, a_{l+1}, \ldots, a_r$ 都加上 x.
在所有询问结束后一次输出整个数列。
$n, q \le 10^6$
构造差分数组，每次在差分数组上进行操作，而后通过差分数组得到最终答案。

类似前缀和的其他应用

使用 $O(n)$ 时间复杂度预处理，$O(1)$ 复杂度求 $C_a^b, (0 \le b \le a \le n) \% p, p$ 是质数

$C_n^m = \frac{n!}{m!(n-m)!}$，则需要预处理阶乘 $\bmod p$ 的值。
如何在模意义下进行除法？在模意义下，除以 x 就等于乘以 $x^{-1} \equiv x^{p-2}$，x 的逆元。

```
int fac[N],invfac[N];
fac[0] = 1;
for(int i=1;i<=n;i++)fac[i] = 1ll*fac[i-1]*i%p;
invfac[n] = mpow(fac[n],p-2);// 快速幂
for(int i=n-1;i>=0;i--)
    invfac[i] = 1ll*(i+1)*invfac[i+1]%p;
```


Atcoder M-SOLUTIONS2019 - E - Product of Arithmetic Progression

考虑下述的长度为 n 的等差数列:
$x, x+d, x+2d, \ldots, x+(n-1)d$
求所有数的乘积，对 $1,000,003$（一个质数）取模。
$1 \le Q \le 10^5, 0 \le x_i, d_i \le 1,000,002, 1 \le n_i \le 10^9$，保证所有输入的数字均为整数。

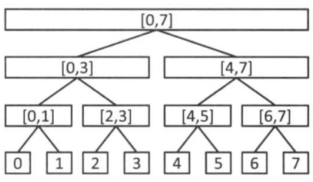
Atcoder M-SOLUTIONS2019 - E - Product of Arithmetic Progression - solution

首先分析一下问题，如果 $d = 1$，那么问题就变得相对简单了许多。而后我们又可以发现下面这个等式:

$$\prod_{i=0}^{n-1} x + id = d^n \prod_{i=0}^{n-1} \frac{x+id}{d} = d^n \prod_{i=0}^{n-1} (xd^{-1} + i)$$

如果在 $xd^{-1}, xd^{-1}+1, \ldots, xd^{-1}+n-1$ 之间有一个 1000003 的倍数，那么答案就是 0. 不然答案就是 $(xd^{-1}+n-1)!/(xd^{-1}-1)!$
另外，如果 $d = 0$，那么直接调用一次快速幂就可以求出答案。

线段树

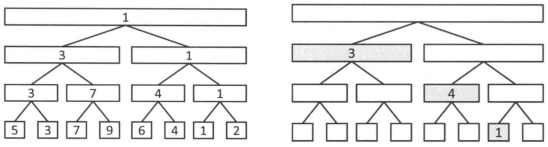


线段树是一棵完全二叉树 (所有的叶子的深度都相同，并且每个节点要么是叶子要么有 2 个儿子的树)，树上的每个节点都维护一个区间。根维护的是整个区间，每个节点维护的是父亲的区间二等分后的其中一个子区间。当有 n 个元素时，对区间的操作可以在 $O(\log\ n)$ 的时间内完成。

RMQ 问题

现有一长度为 n 的数列 a，有 q 次询问或操作
询问：每次给出 l, r，求 $\min a_l, a_{l+1}, \ldots, a_r$
操作：给出 p, x，将 a_p 赋值为 x
$n, q \le 10^5$

线段树具体操作



在使用线段树解决这个问题时，具体的操作是使用最少的线段树上区间（树的结点）不重叠不遗漏地将所需要修改的区间覆盖，并且只需要对这些线段树上的区间取最小值即可。
可以证明，需要进行操作对区间个数是 $O(\log\ n)$ 个。

线段树的实现

```
#define mid ((l+r)/2)
void build_tree(int id,int l,int r){
    if(l == r){value[id] = a[l]; return;}
    build_tree(id*2,l,mid);
    build_tree(id*2+1,mid+1,r);
    value[id] = min(value[id*2],value[id*2+1]);
}
```


线段树的实现-2

```
#define mid ((l+r)/2)
int query(int id,int l,int r,int ql,int qr){
    if(ql<=l && r<=qr){return value[id]}
    int ans = inf;
    if(ql<=mid)
        ans = min(ans,query(id*2,l,mid,ql,qr));
    if(qr>mid)
        ans = min(ans,query(id*2+1,mid+1,r,ql,qr));
    return ans;
}
```

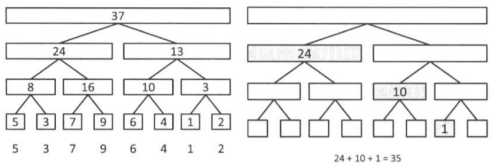

线段树的实现-3

```
#define mid ((l+r)/2)
void operate(int id,int l,int r,int p,int x){
    if(l == r){value[id]=x;return;}
    if(p<=mid)
        operate(id*2,l,mid,p,x);
    else
        operate(id*2+1,mid+1,r,p,x);
    value[id] = min(value[id*2],value[id*2+1]);
}
```


简单问题 1.2

现有一长度为 n 的数列 a，有 q 次询问或操作
询问：每次给出 l, r，求 $a_l + a_{l+1} + \ldots + a_r$
操作：给出 l, r, x，将 $a_l, a_{l+1}, \ldots, a_r$ 加 x
$n, q \leq 10^5$

线段树具体操作



在使用线段树解决这个问题时，具体的操作是使用最少的线段树上区间（树的结点）不重叠不遗漏地将所需要修改的区间覆盖，并且只需要对这些线段树上的区间进行操作即可。
可以证明，需要进行操作对区间个数是 $O(\log\ n)$ 个。

Lazy 标记

线段树在进行加操作时，不会将每一个结点都进行更新，如果那样做的话时间复杂度便不会得到改善。线段树在加操作时，会在对应都结点上打上一个 lazy 标记，代表这个结点所表示的区间中所有的点值都被加上了一个值。
在每一次访问一个结点时，都需要将其 Lazy 标记“下放”。具体的讲，就是首先用 Lazy 标记更新其值，然后将两个子节点的标记修改，将这个结点的标记清空。

线段树的实现

```
#define mid ((l+r)/2)
void push_down(int id,int l,int r);
void build_tree(int id,int l,int r){
    if(l == r){value[id] = 233; return;}
    build_tree(id*2,l,mid);
    build_tree(id*2+1,mid+1,r);
    value[id] = value[id*2]+lazy[id*2]*(mid-l+1)
          +value[id*2+1]+lazy[id*2+1]*(r-mid);
}
```

线段树的实现-2

```
#define mid ((l+r)/2)
int query(int id,int l,int r,int ql,int qr){
    if(ql<=l && r<=qr)
        return value[id]+lazy[id]*(r-l+1);
    push_down(id);
    int ans = 0;
    if(ql<=mid)
        ans = ans+query(id*2,l,mid,ql,qr);
    if(qr>mid)
        ans = ans+query(id*2+1,mid+1,r,ql,qr);
    return ans;
}
```


线段树的实现-3

```
#define mid ((l+r)/2)
void operate(int id,int l,int r,int ql,int qr,int x
    if(ql<=l && r<=qr){lazy[id]+=x;return;}
    push_down(id);
    if(ql<=mid)
        operate(id*2,l,mid,ql,qr,x);
    if(qr>mid)
        operate(id*2+1,mid+1,r,ql,qr,x);
    value[id] = value[id*2]+lazy[id*2]*(mid-l+1)
    +value[id*2+1]+lazy[id*2+1]*(r-mid);
}
```


线段树小例题 1

现有一长度为 n 的数列 a，有 q 次询问或操作
询问：每次给出 l, r，求 $a_l + a_{l+1} + \ldots + a_r$
操作 1：给出 l, r, x，将 $a_l, a_{l+1}, \ldots, a_r$ 加 x
操作 2：给出 l, r, x，将 $a_l, a_{l+1}, \ldots, a_r$ 乘 x
$n, q \leq 10^5$

线段树小例题 1 - Solution

设置两种 lazy 标记：lazy1,lazy2.
值 = value*lazy1 + lazy2*(r-l+1);

```
void push_down(int id,int l,int r){
    value[id] = value[id]*lazy1[id]
    +lazy2[id]*(r-l+1);
    lazy1[id*2]*=lazy1[id];
    lazy2[id*2]=lazy2[id*2]*lazy1[id]+lazy2[id];
    lazy1[id*2+1]*=lazy1[id];
    lazy2[id*2+1]=lazy2[id*2]*lazy1[id]+lazy2[id];
    lazy1[id] = 1;lazy2[id] = 0;
}
```


线段树小例题 2

现有一长度为 n 的数列 a，有 q 次询问或操作
询问：每次给出 l, r，求 $a_l, a_{l+1}, \ldots, a_r$ 中次小的数字
操作：给出 p, x，将 a_p 赋值为 x
$n, q \leq 10^5$

线段树小例题 2 - Solution

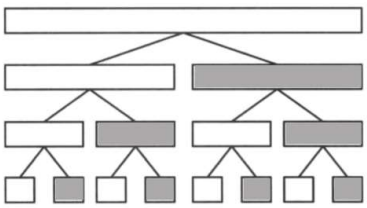
维护两个 value, value1, value2, 代表当前区间中的最小值和次小值。这个题目不需要 lazy 标记。
这是去年的 F 题。今年题目还要难一些。

简单问题 1.2 - 弱化

现有一长度为 n 的数列 a，有 q 次询问或操作
询问：每次给出 l, r，求 $a_l + a_{l+1} + \ldots + a_r$
操作：给出 p, x，将 a_p 加 x
$n, q \leq 10^6$

树状数组的原理

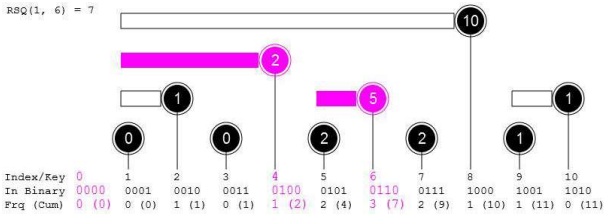
显然这个比上一道题还容易，可以使用线段树。
但是如果我们能够计算 (从 1 到 r 的和)-(从 1 到 s-1 的和)，同样可以得到 s 到 r 的和。也就是说，只要对于任意 i，我们都能计算出 1 到的部分和就足够了。



灰色：不需要用到的节点

我们可以发现，线段树上每个节点的右儿子的值都不需要了（在计算时如果要使用这个点的值，那么它的左边的兄弟的值也一定会用到，这个时候只需要使用它们的父亲的值就可以了）。

二进制运算？



树状数组的代码实现

```
void add(int p,int x){
    while(p<N){
        A[p]+=x; p+=(p&(-p));
    }
}
int query(int p){
    int ret = 0;
    while(p){
        ret+=A[p]; p-=(p&(-p));
    }
    return ret;
}
```


树状数组的优缺点

优点：

- 常数小（由于随机数有一半的位为 1，常数为 0.5），比线段树快很多很多。
- 代码简短易于书写

缺点：

- 不好理解
- 不易于拓展

树状数组简单例题 1

现有一长度为 n 的数列 a，有 q 次询问或操作
询问：给出 p，询问 a_p 的值
操作：每次给出 l, r, x，将 $a_l, a_{l+1}, \ldots, a_r$ 都加 x
$n, q \leq 10^6$
维护差分数列即可。

树状数组简单例题 2

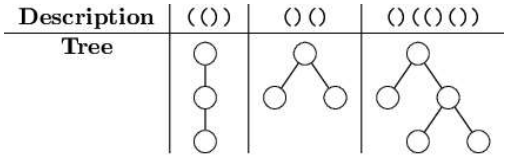
现有一长度为 n 的数列 a，有 q 次询问或操作
询问：每次给出 l, r，求 $a_l \oplus a_{l+1} \oplus \ldots \oplus a_r$
操作：给出 p, x，将 a_p 异或上 x
$n, q \leq 10^6$
直接将运算符变成异或即可。

杂题选讲???

题目难度有所增加，旨在提供一些**提高**练习题。

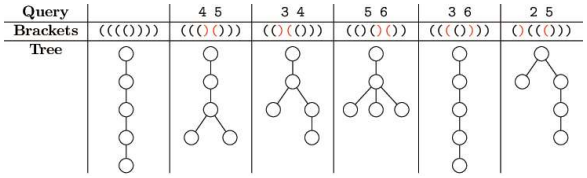
如果时间不够，将会跳过一部分题目。如果时间足够
题目出处会写清楚，如果上课没搞懂，可以自行学习。

CF Round 556 Div.1 C - Tree Generator™



用括号序列来表示一颗树。每次交换两个括号，输出树的直径长度。
树的直径：树中最长的一条路径。

CF Round 556 Div.1 C - Tree Generator™



样例数据。

CF Round 556 Div.1 C - Tree Generator™ - Solution

定义一个结点的" 深度" $h(x)$ 为一个结点和根结点之间边数 $+1$
树上任意一条路径 u, v 都有一个最近公共祖先 $w = \mathsf{lca}(u, v)$，则其路径长度为 $h(u) + h(v) - 2 * h(w)$。

CF Round 556 Div.1 C - Tree Generator™ - Solution

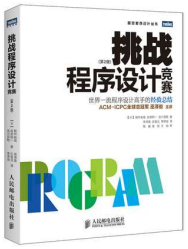
我们不妨对括号序列做如下理解：左括号代表走一条向更深的边到自己的一个儿子，右括号代表走到自己的父结点。就发现括号序列就能够代表 dfs 时候访问的顺序。设 $\mu(t)$ 为在括号序列 t 处时候的深度。
设 t_u 为我们处在 u 结点时候的时间（如果有很多，所有的都可以作为 t_u）
那么 $h(\mathrm{lca}(u, v)) = \min_{t_l \in [t_u, t_v]} \mu(t_l)$，显然便有
$d(u, v) = \max_{t_l \in [t_u, t_v]} (\mu(t_u) + \mu(t_v) - 2\mu(t_l))$
于是直径便为
$\max_{u,v} d(u, v) = \max_{t_u \le t_l \le t_v} (\mu(t_u) - 2\mu(t_l) + \mu(t_v))$

CF Round 556 Div.1 C - Tree Generator™ - Solution

于是我们可以用线段树维护下面的这些值，假设 $a \le b \le c$

- δ (子串的 +1,-1 之和)
- $\max \mu(a)$
- $\max -2\mu(b)$
- $\max(\mu(a) - 2\mu(b))$
- $\max(-2\mu(b) + \mu(c))$
- $\max(\mu(a) - 2\mu(b) + \mu(c))$

参考书籍



A. nocriz 送温暖

内存限制：512 MiB

时间限制：1500 ms

题目描述

nocriz 是个友善的出题人。

有 n 个数 $a_1, a_2, \ldots, a_n$，有 q 次询问，每次问 $a_l \times a_{l+1} \times \ldots \times a_r \bmod 998\,244\,353$ 的值。

输入格式

第一行一行两个整数 n, q。

第二行一行 n 个整数 $a_1, a_2, \ldots, a_n$。

接下来 q 行，每行两个整数 l_i, r_i。

输出格式

q 行，每行一个整数，代表答案。

数据范围与提示

$1 \le n, q \le 10^6$

$1 \le a_i \le 998{,}244{,}352$

由于输入量较大，请避免使用 cin/cout，使用较为快速的 scanf/printf。

解题思路

预处理前缀积及其逆元即可。获取前缀积的逆元时，只需求一次逆元，其余的可以使用递推的方法得到：

$$\left(\prod_{i=1}^{l} a_i\right)^{-1} \equiv a_{l+1}\left(\prod_{i=1}^{l+1} a_i\right)^{-1} \pmod{998\,244\,353}$$

代码

```
const int p = 998244353;
int n, qe;
int ns[1100000], pt[1100000], rs[1100000];
int nmod(int a, int mp)
{
    int ans = a % mp;
    if (ans < 0)
        ans += mp;
    return ans;
}
llong quickmod(llong a, llong b)
{
    llong sum = 1;
    while (b)
    {
        if (b & 1)
```

```
            sum = (sum * a) % p;
        b >>= 1;
        a = (a * a) % p;
    }
    return sum;
}
int main()
{
    scanf("%d%d", &n, &qe);
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &ns[i]);
        if (i > 0)
            pt[i] = (1LL * ns[i] * pt[i - 1]) % p;
        else
            pt[i] = ns[i];
    }
    rs[n - 1] = quickmod(pt[n - 1], p - 2) % p;
    for (int i = n - 2; i >= 0; i--)
        rs[i] = (1LL * rs[i + 1] * ns[i + 1]) % p;
    for (int i = 0; i < qe; i++)
    {
        int l, r, ans = 1;
        scanf("%d%d", &l, &r);
        if (l > 1)
            ans = (1LL * ans * rs[l - 2]) % p;
        ans = (1LL * ans * pt[r - 1]) % p;
        printf("%d\n", ans);
    }
    return 0;
}
```

B. nocriz 修路(改)

内存限制：512 MiB

时间限制：1000 ms

题目描述

nocriz 同学有个好朋友，叫渡渡鸟公爵。

在渡渡鸟，鸭子，鹦鹉和鸽子的联合王国中，没有一条路，因为大部分的鸟都会飞行。遗憾的是，渡渡鸟不会飞行！

所以为了旅行的更快，渡渡鸟们开始在王国中修建道路。在王国中有 N 座城市，城市的编号为 $1,2,\ldots,N$，然后渡渡鸟公爵——渡渡鸟的伟大领袖计划修建 M 条道路，道路的编号为 $1,2,\ldots,M$。但是渡渡鸟公爵非常吝啬，他不希望任何两座城市之间可以通过不同的简单路径互相到达，他觉得这是对资源的浪费。所谓的从 到 的一条简单路径，就是一条依次经过节点 $a,v_1,v_2,\ldots,v_k,b$ 的路径（b 可以为 0），满足序列中不会有某个节点出现两次。

所以当计划开始时，渡渡鸟的修路工人按照伟大的计划一条一条地修建道路，但是如果建造了这条道路会导致，存在两座不同的城市 a,b 可以通过不同的简单路径互相到达，那么这条路就会被跳过而不会被修建，工人们转而考虑下一条道路。

在计划开始之前，渡渡鸟公爵希望你告诉他有多少条路最终会被修建，这些路又是哪些。

输入格式

第一行包含两个整数 N,M 。

接下来 M 行，每行包含两个整数 u_i,v_i ，代表计划中的第 i 条路。

输出格式

第一行包含一个整数，将要修建的道路数。

第二行包含升序排列的要修建的道路的编号，用一个空格互相隔开。

数据范围与提示

$1 \leq N,M \leq 2 \cdot 10^5$

$1 \leq u_i,v_i \leq N$

解题思路

并查集。一条路的两端若不在同一个集合则合并这两个集合，否则不能修建。

代码

```
int n, m, k, ans = 0;
int fts[500010];
vector<int> anss;
int fnd(int x)
{
    return (fts[x] == 0) ? x : fts[x] = fnd(fts[x]);
}
void mrg(int f, int t)
{
    fts[fnd(f)] = fnd(t);
}
int main()
{
    scanf("%d%d", &n, &k);
    for (int i = 0; i < k; i++)
    {
        int a, b;
        scanf("%d%d", &a, &b);
        if (fnd(a) == fnd(b))
            continue;
        ans++;
        anss.push_back(i + 1);
        mrg(a, b);
    }
```

```
    printf("%d\n", ans);
    for (int i = 0; i < ans; i++)
        printf("%d ", anss.at(i));
    return 0;
}
```

C. nocriz 坐火车

内存限制：512 MiB

时间限制：1000 ms

题目描述

nocriz 是一名喜欢旅游的同学，在来到渡渡鸟王国时，坐上了颜色多样的火车。

nocriz 同学在车上，车上有 n 个车厢，每一个车厢有一种颜色。

他想知道对于每一个 x，$\{(i,x,j)|i<x<j,l_x\le col_i=col_j\le r_x\}$ 这个集合包含多少元素。

换句话说，就是要求每一个车厢两边有多少对颜色相同的车厢，并且这一对车厢的颜色要在 l_x 到 r_x 之间。

col_i 代表 i 号车厢的颜色。

l_x,r_x 代表颜色限制。

输入格式

一行一个整数，代表 n 。一行 $3n$ 个整数，第 $3i-2$ 个代表 col_i，第 $3i-1$ 个代表 l_i，第 $3i$ 个代表 r_i。

输出格式

一行 n 个整数，代表答案。

数据范围与提示

$1\le n\le 5\cdot 10^5$

$1\le col_i,l_i,r_i\le 5\cdot 10^5$

解题思路

查询 $[l_i,r_i]$ 区间内的结果总数，可以使用前缀和，但修改效率过低，故可以使用树状数组。一个车厢两边颜色为 c 的对数为该车厢左侧颜色为 c 的车厢数乘该车厢右侧颜色为 c 的车厢数，记做 $p(c)=a(c)b(c)$。当即将查询第 i 个车厢时，$b(col_i)$ 减少 1，$p(col_i)$ 减少 $a(col_i)$。对第 i 个车厢查询结束后，$a(col_i)$ 增加 1，$p(col_i)$ 增加 $b(col_i)$。之后进行下一个车厢的查询。

$b(c)$ 的初值即颜色为 c 的车厢总数，可以在读入的过程时候计算得到。

代码

```
int n;
int l[525000], r[525000], c[525000], ct[525000], cn[525000];
llong tar[525000];
int main()
{
    scanf("%d", &n);
```

```
    for (int i = 0; i < n; i++)
    {
        scanf("%d%d%d", &c[i], &l[i], &r[i]);
        ct[c[i]]++;
    }
    for (int i = 0; i < n; i++)
    {
        llong ans = 0;
        const int ntc = c[i];
        int tl = l[i] - 1, tr = r[i], tc = c[i];
        ct[ntc]--;
        for (int bt = 0; bt < 20; bt++)
            if ((tc >> bt) & 1)
            {
                tar[tc] -= cn[ntc];
                tc += (1 << bt);
            }
        for (int bt = 0; bt < 20; bt++)
        {
            if ((tr >> bt) & 1)
            {
                ans += tar[tr];
                tr -= (1 << bt);
            }
            if ((tl >> bt) & 1)
            {
                ans -= tar[tl];
                tl -= (1 << bt);
            }
        }
        printf("%lld ", ans);
        tc = ntc;
        for (int bt = 0; bt < 20; bt++)
            if ((tc >> bt) & 1)
            {
                tar[tc] += ct[ntc];
                tc += (1 << bt);
            }
        cn[ntc]++;
    }
    return 0;
}
```

D. JM 的小学数学题

内存限制：512 MiB

时间限制：1000 ms

题目描述

不知道已经成为大学生的你是否还记得中位数这个东西，我们似乎很少再用到它，今天 JM 就想让你做一下这个小学数学题。

初始你手里什么也没有，接下来 JM 会按顺序给你 n 个数。当你手中的数的个数为奇数时，你需要告诉 JM 你手里这堆数的中位数是多少。

输入格式

第一行一个正整数 n，表示给你的数的个数。

接下来一行 n 个整数，表示依次给你的这些数 a_i。

输出格式

输出 x 行，每行一个整数，表示答案。

当 n 为奇数时 $x = n/2 + 1$，当 n 为偶数时 $x = n/2$，其中除法为下取整。

数据范围与提示

$1 \le n \le 2 \cdot 10^5$

$|a_i| \le 10^9$

解题思路

维护两个优先队列，分别维护比中位数小的最大值和比中位数大的最小值。插入时，将值与中位数比较，放入对应的优先队列中。询问时，若两个优先队列大小相同，则中位数不变；否则将中位数放入元素较少的优先队列中，并从另一个优先队列中取出一个值并成为新的中位数。

代码

```
int n, md;
multiset<int> s, b;
int main()
{
    scanf("%d%d", &n, &md);
    printf("%d\n", md);
    for (int i = 1; i < n; i++)
    {
        int nw;
        scanf("%d", &nw);
        if (nw > md)
            b.insert(nw);
        else
            s.insert(nw);
        if ((i & 1) == 0)
        {
```

```
            while (s.size() != b.size())
                if (s.size() > b.size())
                {
                    b.insert(md);
                    multiset<int>::iterator ls = s.end();
                    ls--;
                    md = *ls;
                    s.erase(ls);
                }
                else
                {
                    s.insert(md);
                    md = *b.begin();
                    b.erase(b.begin());
                }
            printf("%d\n", md);
        }
    }
    return 0;
}
```

E. nocriz 维护括号序列

内存限制：512 MiB

时间限制：1000 ms

题目描述

nocriz 是一个懒得编题面的同学。

括号序列是一个只由左括号(和右括号)构成的序列；进一步的，一个合法的括号序列是指左括号和右括号能够一一匹配的序列。如果用规范的语言说明，一个合法的括号序列可以有以下三种形式：

1. **S=""**，即 S 是空串，则 S 是一个合法的括号序列。
2. **S=XY**，其中 X,Y 均为合法的括号序列，则 S 也是一个合法的括号序列。
3. **S=(X)**，其中 X 为合法的括号序列，则 S 也是一个合法的括号序列。

给出一个长度为 n 的括号序列，有 q 次操作，每次翻转一个括号，问括号序列是否合法。

输入格式

第一行两个整数 n,q，代表括号序列长度和操作次数。

接下来一行一个字符串，代表括号序列。

接下来 q 行，每行一个整数 p，代表翻转括号的编号。

输出格式

一行一个字符串，仅包含 **0** 或 **1**，代表每次的答案。

数据范围与提示

$1 \le n, q \le 2 \cdot 10^5$

解题思路

考虑一个字符串，未配对的括号只有前括号和后括号两种。当两个字符串拼接在一起时，前一个字符串的未配对前括号可以与后一个字符串的未配对后括号配对。使用线段树，每个节点维护该区间里的未配对前后括号数即可。显然，一个字符串合法当且仅当未配对前括号数和未配对后括号数均为 0。

代码

```
struct psum
{
    int l, r;
};
int n, q;
string bkt;
psum va[525000];
void uplr(int id)
{
    va[id].l = va[id * 2].l;
    va[id].r = va[id * 2 + 1].r;
    if (va[id * 2].r > va[id * 2 + 1].l)
        va[id].r += va[id * 2].r - va[id * 2 + 1].l;
    else
        va[id].l += va[id * 2 + 1].l - va[id * 2].r;
}
void bdt(int id, int l, int r)
{
    if (l == r)
    {
        if (bkt[l] == '(')
            va[id].r = 1;
        else
            va[id].l = 1;
        return;
    }
    int mid = (l + r) >> 1;
    bdt(id * 2, l, mid);
    bdt(id * 2 + 1, mid + 1, r);
    uplr(id);
}
void spbkt(int id, int l, int r, int p)
{
    if (l == r)
    {
```

```
        swap(va[id].l, va[id].r);
        return;
    }
    int mid = (l + r) >> 1;
    if (p <= mid)
        spbkt(id * 2, l, mid, p);
    else
        spbkt(id * 2 + 1, mid + 1, r, p);
    uplr(id);
}
int main()
{
    cin.sync_with_stdio(false);
    cin.tie(0);
    cin >> n >> q >> bkt;
    bdt(1, 0, n - 1);
    for (int i = 0; i < q; i++)
    {
        int lo;
        cin >> lo;
        spbkt(1, 0, n - 1, lo - 1);
        if (va[1].l == 0 && va[1].r == 0)
            printf("1");
        else
            printf("0");
    }
    return 0;
}
```

备注

代码处理的是区间 $[l,r]$。

F. nocriz 与队列计算机

内存限制：512 MiB

时间限制：2000 ms

题目描述

nocriz 是一位来到了西安的同学。

他在参观兵马俑时，了解到在遥远的秦朝，为了进行复杂的计算，秦始皇发明了队列计算机。兵马俑就是秦始皇为了在地下进一步发展科技而制作的队列计算机模型。

在此题中，队列计算机被简化成长度为 n 的一个士兵的队列，其中计算元件（士兵）有两种，其中 a_i 为这个士兵的特征值：

- 加法士兵负责从左边读入一个数字 x，并将 $(x + a_i) \bmod 998\,244\,353$ 告诉下一个士兵。

• 乘法士兵负责从左边读入一个数字 x，并将 $(x \times a_i) \bmod 998\,244\,353$ 告诉下一个士兵。

每一次，秦始皇有两个操作：

• 更换一个士兵，给出新士兵的类型和特征值。

• 告诉第 l 个士兵一个数，求第 r 个士兵说出的数字。

秦始皇了解到 **nocriz** 学习了数据结构，要考考 **nocriz** 同学，但是他把锅丢给了你。

输入格式

一行两个整数 n, q；接下来 n 行，每行两个整数 typ_i, a_i：

• $typ_i = 0$：士兵为乘法士兵。

• $typ_i = 1$：士兵为加法士兵。

接下来 q 行，每行开始一个整数 op：

• $op = 0$：接下来输入 p, typ_p, a_p，代表修改操作。

• $op = 1$：接下来输入 l, r, x，代表询问操作。

输出格式

对于每次询问操作，输出一行一个数代表答案。

数据范围与提示

$n, q \le 5 \cdot 10^5$

所有出现过的数字在 $[1, 998\,244\,353)$ 区间中

解题思路

一个数 x_0 经过一个区间后，其值变为 $x_1 = ax_0 + b$。考虑线段树，每个节点只需维护该区间的 a, b 即可。若 x_0 先后经过了相邻的两个区间，其值变为：

$$x_2 = a_2x_1 + b_2 = a_2(a_1x_0 + b_1) + b_2 = a_1a_2x_0 + (b_1a_2 + b_2)$$

故将两个区间合并后，$a = a_1a_2, b = b_1a_2 + b_2$。若一个区间内没有乘法士兵，则 $a = 1$；若一个区间内没有加法士兵，则 $b = 0$。

代码

```
struct psum
{
    int mp = 1, pl;
};
int n, q, tps[525000], ais[525000];
psum va[1050000];
void upa(int id)
{
    va[id].mp = (1LL * va[id * 2].mp * va[id * 2 + 1].mp) % p;
    va[id].pl = (1LL * va[id * 2].pl * va[id * 2 + 1].mp) % p;
    va[id].pl = (1LL * va[id].pl + va[id * 2 + 1].pl) % p;
}
void bdt(int id, int l, int r)
{
```

```
    if (l == r)
    {
        if (tps[l] == 0)
            va[id].mp = ais[l];
        else
            va[id].pl = ais[l];
        return;
    }
    int mid = (l + r) >> 1;
    bdt(id * 2, l, mid);
    bdt(id * 2 + 1, mid + 1, r);
    upa(id);
}
void cgsd(int id, int l, int r, int lo, int tp, int ai)
{
    if (l == r)
    {
        if (tp == 0)
            va[id] = { ai, 0 };
        else
            va[id] = { 1, ai };
        return;
    }
    int mid = (l + r) >> 1;
    if (lo <= mid)
        cgsd(id * 2, l, mid, lo, tp, ai);
    else
        cgsd(id * 2 + 1, mid + 1, r, lo, tp, ai);
    upa(id);
}
int sch(int id, int l, int r, int sl, int sr, int sx)
{
    if (l == sl && r == sr)
        return (1LL * sx * va[id].mp + va[id].pl) % p;
    int mid = (l + r) >> 1;
    if (sr <= mid)
        return sch(id * 2, l, mid, sl, sr, sx);
    if (sl > mid)
        return sch(id * 2 + 1, mid + 1, r, sl, sr, sx);
    int x1 = sch(id * 2, l, mid, sl, mid, sx);
    return sch(id * 2 + 1, mid + 1, r, mid + 1, sr, x1);
}
int main()
{
    scanf("%d%d", &n, &q);
```

```
    for (int i = 0; i < n; i++)
        scanf("%d%d", &tps[i], &ais[i]);
    bdt(1, 0, n - 1);
    for (int i = 0; i < q; i++)
    {
        int op, a, b, c;
        scanf("%d%d%d%d", &op, &a, &b, &c);
        if (op == 0)
            cgsd(1, 0, n - 1, a - 1, b, c);
        else
            printf("%d\n", sch(1, 0, n - 1, a - 1, b - 1, c));
    }
    return 0;
}
```

备注

代码处理的是区间 $[l,r]$。

G. nocriz 和巨佬谈人生

内存限制：512 MiB

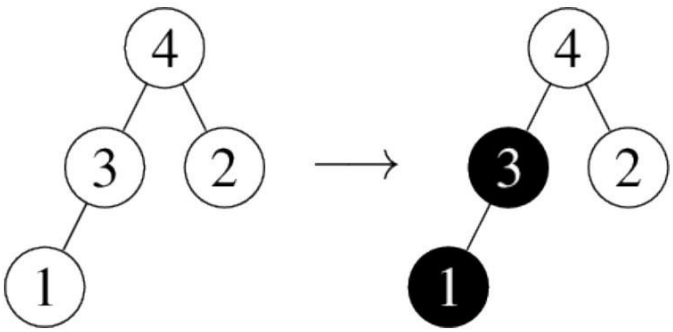
时间限制：1000 ms

题目描述

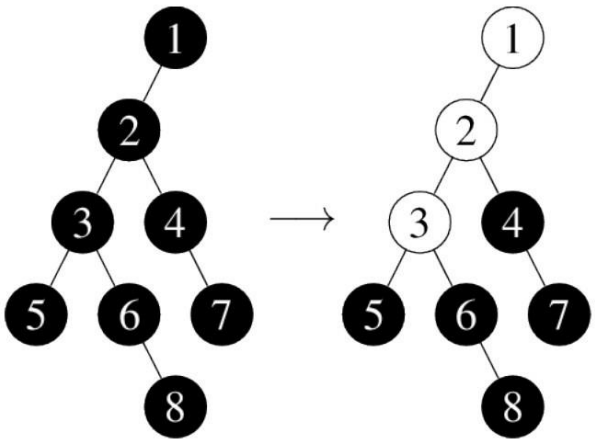
nocriz 是一名对人生充满向往的同学。

他和巨佬 **AprilGrimoire** 谈起了人生。**nocriz** 提出，人生就如一个不回溯的搜索，你并不知道前面有什么。**AprilGrimoire** 说，其实作为巨佬的人，是可以知道前面有什么的。**nocriz** 在 Orz 之余，说即使对于巨佬有一些路能够走的人是有限的，比如 **AprilGrimoire** 巨佬就可以走一条只有 ~~50~~ ~~15~~ 4 他自己 1 个人可以走的路。

AprilGrimoire 巨佬将人生建模成了一颗树。树上不同的结点有着不同的优劣程度。一个人从根节点出发，每一次朝着可达结点编号最小的空子结点出发。具体的说，假如当前节点有且仅有一个空儿子节点，那么人就会移动到那个节点。假如有多个空儿子节点，那么人会遵循如下过程：假设人所在当前节点为 P，人会先寻找以 P 节点为根的子树中编号最小并且和当前结点路径上有空结点的的节点，然后判断这个节点属于哪个以 P 的空儿子为根的子树，最后移动到那个空儿子节点中。人会一直移动直到到达没有空儿子的节点。如图所示：两个人从节点 4 处开始了他们的人生。因为编号最小的节点 1 在以 3 为根的子树中，所以第一个人走到了节点 3，然后……最后移动到了节点 1。同理，第二个人从节点 4 走到了节点 3 并且停了下来。



还有可能有时候人消失了，那么原先的节点就变为空节点，那么假如它的父节点有人，那么人就会走过来。比如，如图所示:假如我们在 5, 7 和 8 号节点的人消失了，那么在一系列人移动之后，1, 2, 3 号节点就变空了。



nocriz 觉得这实在太真实，于是他记录下了这个模型，并想用计算机模拟模拟。

有一颗个节点的树，有两种操作：

- 从树根逐次生成 k 个人。
- 将某个节点清空，节点上的人会消失，其他人会填过来。

输入格式

第一行两个正整数 n 和 q。n 代表有 n 个节点，q 代表有 q 次操作。 下面 n 行，每行一个整数。第 i 行的整数表示编号为 i 的节点的父亲节点的编号。如果整数为 0 则说明该节点是根节点。下面 q 行，每行表示一个操作。

- 操作一：**1 k** 表示从树根逐次生成 k 个人。
- 操作二：**2 x** 表示将编号为 x 的节点清空。

保证所有操作合法，不会有人无处可去也不会清空空节点。

输出格式

对于操作一，输出人最后走到的节点编号。

对于操作二，输出清空节点之后，有移动的人的数量。

样例

输入样例

```
8 4
0
1
2
2
3
3
4
6
1 8
```

```
2 5
2 7
2 8
```

输出样例

```
1
3
2
2
```

数据范围与提示

$1 \le n, q \le 10^5$

H. nocriz 与'Swappy Tree'

内存限制：512 MiB

时间限制：1000 ms

题目描述

nocriz 是一个好学的同学。

nocriz 同学了解到有一种叫 **Swappy Tree** 的神奇的树。

Swappy Tree 是一棵高度为 $n+1$，有着 2^n 个叶子的完全二叉树。叶子从左向右分别写着为 $1, 2, \dots, 2^n$。 对于非叶子节点，在所有时刻，我们将从上往下第 i 层，从左往右第 j 个的点称为 $2^{i-1}+j-1$。 这里，我们定义 **swap(x)** 为交换 x 号节点的左右子树（交换后第 i 个非叶子节点依然按上文中的方法定义）。 **Swappy Tree** 上要进行以下两种操作：

- 给定 k $(k \le 2^n)$，求出从左往右第 k 个叶子上的值。
- 给定 a, b $(1 \le a \le b \le 2^n - 1)$，依次进行 **swap(a)**, **swap(a+1)**, … , **swap(b)**。

你需要代替 **nocriz** 同学实现一棵 **Swappy Tree**。 具体来说，给定 n 和 q 个询问，请对其处理并做出一棵 **Swappy Tree**。

输入格式

第一行包含两个整数 n, q。

接下来 q 行，每行包括三个整数 t_i, a_i, b_i，表示一组操作。

若 $t_i = 1$，a_i 即为询问中的 k，保证 $b_i = 0$。

若 $t_i = 2$，a_i 与 b_i 即为修改中的 a 与 b。

输出格式

对于每一个询问，在新的一行中输出答案。

样例

输入样例

```
3 10
2 5 5
1 1 0
1 2 0
```

```
1 3 0
1 4 0
2 1 3
1 2 0
1 3 0
1 5 0
1 6 0
```

输出样例

```
1
2
4
3
8
5
4
3
```

数据范围与提示

$1 \le n \le 17$

$1 \le q \le 2 \cdot 10^5$

Day 7

主题：字符串理论

时间：2019 年 7 月 2 日 星期二

主要内容：

单模式匹配：哈希，KMP

Trie 树

多模式匹配：AC 自动机

题目：

+10	A. 一姬的后缀自动机	单模式匹配，哈希
+	B. 一姬的 BM 算法	单模式匹配，KMP
+	C. JM 的荧光棒工厂	fail 数组
+3	D. 一姬的三倍满自动机	trie 树
+2	E. JM 的模板题	AC 自动机
–	F. 一姬的役满自动机	

第一节 单模式匹配

第二节 Trie树

第三节 多模式匹配

第四节 应用

第一节 单模式匹配

1.0 字符串

定义1：基本术语
（1）字母表：用∑表示，字母表大小记为|∑|；
（2）字母表上的字符：用小写字母a,b,c等表示；
（3）字符串：用大写字母S,T,W,X,Y,Z等表示；
（4）字符串的长度：|S|。
（5）空串：用ε表示，空串长度为0；
（6）字符串的连接：记为ST；

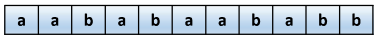
定义2：
（1）前缀：设S=XY，X,Y可空，称X是S的前缀；
（2）后缀：设S=XY，X,Y可空，称Y是S的后缀；
（3）子串：设S=XWY，X,Y,W可空，称W是S的子串；
将S中从位置i到位置j这一子串记为S[i,j]。

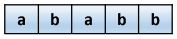
定义3：匹配
若W是S的子串，称W匹配了S，称S为主串或文本（串），W为子串或模式（串）。

例1：字符串abaababa被模式a匹配了5次，被模式aba匹配了3次。字符串aaaaaaa被aaaa匹配了4次。

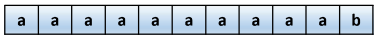
1.1 哈希算法引入

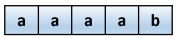
朴素匹配算法





时间复杂度$O(n^2)$





1.1 哈希算法引入

将哈希算法用于字符算匹配的原理非常简单。对于每个起始位置，我们不是 $O(m)$ 地直接比较字符串是否匹配，而是$O(1)$地比较长度为m的字符串子串的哈希值与T的哈希值是否相等。虽然即使哈希值相等字符串也未必相等，但如果哈希值是随机分布的话，不同的字符串哈希值相等的概率是很低的，如果哈希函数的值域足够大，可以当作这种情况不会发生。

选取两个合适的互素常数b和$h(l < b < h)$,假设字符串$C = c_1c_2 \dots c_m$,定义哈希函数$H(C) = (c_1b^{m-1} + c_2b^{m-2} + c_3b^{m-3} + \cdots + c_mb^0) \ mod \ h$其中b是基数，相当于把字符串看作b进制数。

这样，字符串$S = s_1s_2 \dots s_n$从位置$k+1$开始长度为m的字符串子串S[k+1..k+m]的哈希值，就可以利用从位置人开始的字符串子串S[k..k+m-1]的哈希值，直接进行如下计算。

$$H(S[k+1..k+m])$$
$$= (H(S[k..k+m-1]) \times b - s_kb^m + s_{k+m}) mod \ h$$


```
int h = 998244353,b = 19260817;
char s[MAX_N];
int hash[MAX_N],pow_b[MAX_N] = {1};
for(int i=1;i<=n;i++){
    pow_b[i] = 1ll*pow_b[i-1]*b%h;
    hash[i] = (1ll*hash[i-1]*b+s[i]-'a'+1)%h;
}
int get_hash(int r,int l){
    return
    (hash[r]+mod-1ll*pow_b[r-l+1]*hash[l-1]%mod)%mod;
}
```

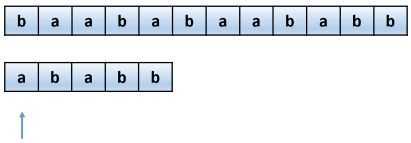
$$H(C,m) = (c_1b^{m-1} + c_2b^{m-2} + c_3b^{m-3} + \cdots + c_mb^0) \ mod \ h$$
$$H(C,l,r) = \left(c_lb^{r-l} + c_{l+1}b^{r-l-1} + \cdots + c_rb^0\right) \ mod \ h$$
$$= H(C,r) - H(C,l)b^{r-l+1}$$

更通用的任意长度哈希方法.

模数的选择(尽量还是要选择质数): 绝大多数情况下，不要选择一个 10^9 级别的数，因为这样随机数据都会有 Hash 冲突，根据生日悖论，随便找上至少一对 Hash 值相等的串(参见 BZOJ 3098 Hash Killer II)。最稳妥的办法是选择两个 10^9级别的质数，只有模这两个数都相等才 判断相等，但常数略大，代码相对难写，目前暂时没有办法卡掉这种 写法(除了卡时间让它超时)(参见 BZOJ 3099 Hash Killer III)。
如果能背过或在考场上找出一个 10^{18} 级别的质数，也相对靠谱，主要 用于前一种担心会超时，后一种担心被卡。
偷懒的写法就是直接使用 unsigned long long，不手动进行取模，它溢 出时会自动对 2^{64} 进行取模，如果出题人比较良心，这种做法也不会 被卡，但这个是完全可以卡的，卡的方法参见 BZOJ 3097 Hash Killer I。

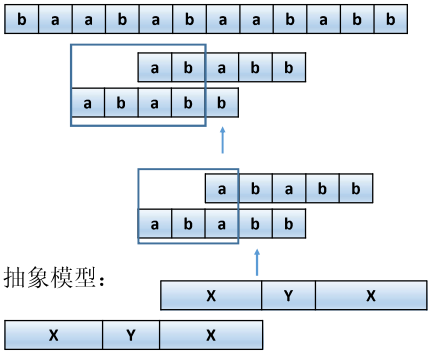
如果想要比较两个子串，而不是判断两个子串是否相等，这也可 以用 Hash 完成。
因为比较两个字符串的实质是比较两个字符串的第一个不相等的 字符，所以直接通过二分后 Hash 判是否相等求出它们的最长公 共前缀，比较下一个字符即可。
时间复杂度 O(log n)。

1.2 KMP算法引入



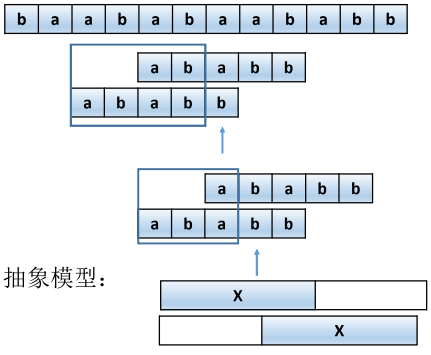
可以看出，充分利用已匹配的信息，模式串可以跳跃地向右移动。每次模式串移动后，指针之前内容均被匹配。每次操作，要么指针向右移动，要么模式串向右移动。且指针总是向右移动$|S|$次，模式串向右移动不超过$|S|$次，这意味着算法时间复杂度可以达到线性。

问题的关键在于模式串向右滑动多少！

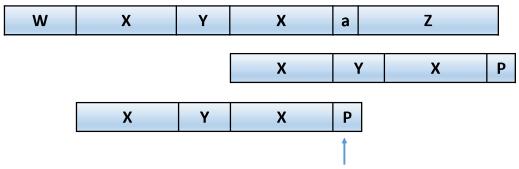


抽象模型：

问题的关键在于模式串向右滑动多少！

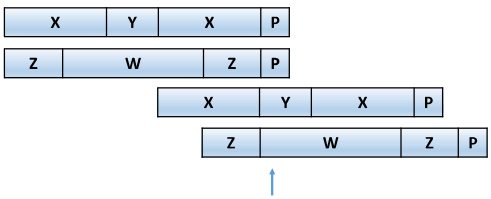


抽象模型：



也就是在模式串中找到子串X，分别在开头和结尾出现！满足该条件当且仅当匹配指针之前的串！

若有多个这样的X，显然应选择长度最大的一个。
这是因为，根据朴素算法的思想，模式串每次只向右滑动1。而KMP算法实质上只是过滤了不满足要求的情况，故仍应滑动的尽可能少，自然得证。



选择长度最大的一个X将模式串滑动后，若不匹配指针位置，则需要进一步滑动到长度较小的位置。于是我们就这样不断滑动，直到指针位置匹配为止。这就是KMP的基本思想。

可以发现，最大X的选取只和模式串有关，和主串无关，它是模式串的固有性质。我们有必要加以研究。

1.3 border、fail及其性质

定义4：border和Border
设X≠S满足X是S的前缀，且是S的后缀，则称X是S的一个border。S的所有border之集记作border(S)。border(S)中长度最大的X记作Border(S)。显然ε是任何非空串的border。

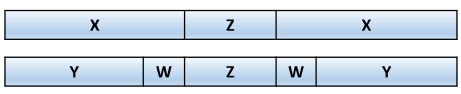
定义5：fail数组
设|S|=n，则fail数组是一个有n个元素的数组，其中fail[i]=|Border(S[0,i])|，规定fail[0]=0。显然fail[i]≤i。

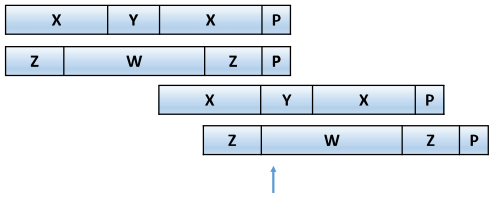
故之前所说的“最大X”实际上就是Border。

定理1：border(S)中所有字符串的长度可表示为：fail[|S|-1],fail[fail[|S|-1]-1],……,0。

证明：首先，|Border(S)|=fail[|S|-1]。
将border(S)中所有字符串X从大到小排序，设X和Y为排序后相邻两字符串，下证|Y|=fail[|X|-1]。

一方面，Y是border(X)；
另一方面，Y一定是border(X)中最长的字符串。

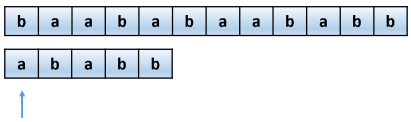




当指针位置不匹配时，需要滑动模式串。由定理1知，滑动过程中，匹配长度依次为（设初始匹配串为S）：fail[|S|-1], fail[fail[|S|-1]-1],……,0。

假设我们能求出fail数组（对i从0到|S|-1）,那么就得到如下的KMP算法。

算法1：KMP算法



i	fail[i]
0	0
1	0
2	1
3	2
4	0

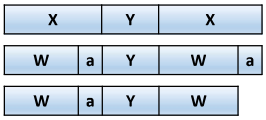
初始化：匹配长度j=0，指针i=0；
重复以下步骤，直到指针i达到主串末尾：
（1）若当前指针位置匹配，则i++；
此时若j达到模式串末尾，则匹配成功！
（2）否则，重复j=fail[j-1]，直到当前指针位置匹配；或者j=0。

算法1：KMP算法

```
#define MAXN 2000001
char s[MAXN];
int fail[MAXN];
bool search(char *str)
{
  for (int i = 0, j = 0; str[i]; i++){
    while (j && str[i] != s[j])j = fail[j - 1];
    if (str[i] == s[j] && !s[++j])return true;
  }
  return false;
}
```

定理2：KMP算法的时间复杂度为Θ(|T|)，T为文本。
证明已由前面指出。

下面讨论 fail数组的建立算法。
通常方法是递推的建立，即由已知的fail[0]到fail[i-1]推出fail[i]。思想如下：



设Border(S[0,i])=X，且X非空，则X可以写成Wa；
故W是S[0,i-1]的border。

故问题转化为求最长的border(S[0,i-1])，设为W，满足S[|W|]=S[i]。

由定理1，我们已经能够由已知的fail[0]到fail[i-1]找到所有的border(S[0,i-1])，然后逐一判断S[|W|]=S[i]的条件是否满足即可。

算法2：fail数组的建立

```
void make_fail()
{
  for (int i = 1, j = 0; s[i]; i++){
    while (j && s[i] != s[j])j = fail[j - 1];
    if (s[i] == s[j])fail[i] = ++j;
    else fail[i] = 0;
  }
}
```

定理3：fail数组的建立时间复杂度为Θ(|S|)。

证明：只考虑内层循环，每执行一次j至少减1。
而除内层循环外，j最多被增加|S|，这就证明了内层循环执行次数为O(|S|)。
故建立时间复杂度为Θ(|S|)。

事实上，search函数与make_fail函数是极为相似的，上述证明同样适用于KMP算法。

定理4：设模式串为S，文本为T，则KMP算法实现单模匹配的时间复杂度为Θ(|S|+|T|)。

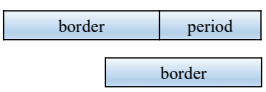
KMP算法是一个十分高效、简洁、优美的算法！

1.4 border、fail的进一步性质

定义6：周期与循环节
设字符串S满足存在整数k≤|S|，使得对任意i∈[0,|S|-k)，有S[i]=S[i+k]，则称k为S的一个周期。
若k为S的一个周期且k | |S|，则称S严格以k为周期。S的最小严格周期中将S[0,k-1]称为S的循环节。

例2：字符串abaabaaba以9、6、3为周期。
字符串aaaa以1、2、3、4为周期，以1、2、4为严格周期。

定理5：S以k（k<|S|）为周期当且仅当S[0,|S|-k-1]是S的border。



证明： 设S以k为周期，则S[i]=S[i+k]，故S[0,|S|-k-1]是S的border；
设S[0,|S|-k-1]是S的border，则对i∈ [0,|S|-k-1]，有S[i]=S[i+k]。

定理6：S以k （k<|S|）为严格周期当且仅当S[0,|S|-k-1]是S的border且k | |S|。

定理7：若|S|-fail[|S|-1] | S，则S的最小严格周期为|S|-fail[|S|-1]。

定理8：若p,q均是S的严格周期，则gcd(p,q)是S的严格周期。

定理9：若p,q均是S的周期，且p+q≤|S|，则gcd(p,q)是S的周期。

证明：不妨设p<q，先证明S以q-p为周期。
（1）i≥p，则S[i]=S[i-p]=S[i+q-p]
（2）i<|S|-q，则S[i]=S[i+q]=S[i+q-p]
按照欧几里得算法思想，即证明S以gcd(p,q)为周期。

本节课暂不继续深入下去，有兴趣者可进一步研究字符串周期理论。

1.5 KMP算法的应用

编程题1：给定模式S和文本T，判断S在T中出现了多少次？出现位置可以相交。|S|,|T|≤10^6。

```
#define MAXN 2000001
char s[MAXN];
int fail[MAXN];
int search(char *str)
{
  int ans = 0;
  for (int i = 0, j = 0; str[i]; i++){
    while (j && str[i] != s[j])j = fail[j - 1];
    if (str[i] == s[j] && !s[++j])ans++;
  }
  return ans;
}
```


编程题2：给定模式S和文本T，T中最多找到多少个S？出现位置不能相交。|S|,|T|≤10^6。

思路：采取如下贪心：T中一旦被S匹配，即选取该匹配。

```
#define MAXN 2000001
char s[MAXN];
int fail[MAXN];
int search(char *str)
{
  int ans = 0;
  for (int i = 0, j = 0; str[i]; i++){
    while (j && str[i] != s[j])j = fail[j - 1];
    if (str[i] == s[j] && !s[++j]){ans++;j=0;}
  }
  return ans;
}
```


编程题3：实验中想要测试转子转动情况下受力F的正弦函数。现测得了t=1,2,…,n时刻的F值，试分析该数据得出转子转动的周期。n≤10^6。

样例：

10

1 3 4 3 1 -1 -2 -1 1 3

输出：

8

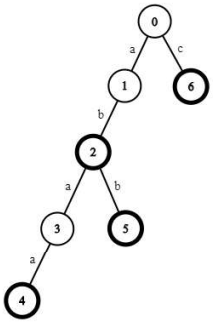
字符串理论不仅仅用于解决字符串！

第二节 Trie树

2.1 Trie树定义与操作

定义7：Trie树是一棵树，每条边上的值为∑上的符号，且满足同一节点和孩子相连的所有边上符号均不相同。

每个由字符串组成的集合都对应一棵Trie树。



Trie树的存储：

```
#define LETTER 26
struct Trie{
  int num;
  int next[LETTER];
}trie[500001];
int cnt;
```

num表示结点标记。

Trie树的初始化：

```
void init(){
  cnt = 0;
  memset(trie, 0, sizeof(Trie));
}
```

表示一棵空树。

Trie树的插入：

```
void insert(char *s)
{
  int cur = 0;
  for (int i = 0; s[i]; i++){
    int pos = trie[cur].next[s[i]-'a'];
    if (!pos){
      pos = ++cnt;
      memset(trie[cnt], 0, sizeof(Trie));
    }
    cur = pos;
  }
  trie[cur].num++;
}
```

时间复杂度： $\Theta(|S||\sum|)$

Trie树的查找：

```
int search(char *s)
{
  int cur = 0;
  for (int i = 0; s[i]; i++){
    cur = trie[cur].next[convert(s[i])];
    if (!cur)return -1;
  }
  return trie[cur].num;
}
```

时间复杂度： $\Theta(|S|)$

Trie树另一种存储方式：

`int next[LETTER];`改为`map<int, int> next;`

此时插入和查询操作均为$\Theta(|S|\log|\sum|)$

Trie树的删除：
只需要查找到删除的字符串，将num属性减1即可。

Trie树还可以用于统计信息。例如，对每个结点记录一个total属性，可用于记录子树中有多少个字符串。

Trie树的遍历：

```
char temp[1001];
void dfs(int i, int h)
{
  if (trie[i].num){
    temp[h] = 0;
    printf("%s %d\n", temp, trie[i].num);
  }
  for (int j = 0; j < LETTER; j++){
    if (trie[i].next[j]){
      temp[h] = j+'a';
      dfs(trie[i].next[j], h + 1);
    }
  }
}
```

调用dfs(0,0)实现遍历。

2.2 Trie树应用

应用1：实现字符串排序
设排序量为n，单位串长O(S)，总串长L，则
传统排序：时间复杂度O(nSlogn)
Trie树：时间复杂度O (|L||∑|)

应用2：取代字符串集合
Trie树插入复杂度O(|S||∑|)，集合插入复杂度O(|S|logn)
Trie树查询复杂度O(|S|)，集合查询复杂度O(|S|logn)

总之，当字母表不大时，Trie树十分高效。

应用3：二进制上的应用

编程题4：给定n个数，试选出两个，使其异或值最大。$2 \le n \le 10^6$，每个数不超过10^9。

思路：每个数看做一个01串，将这些串建立Trie树，很容易想出如何在Trie树中寻找异或值最大、最小的两个值。

时间复杂度$O(n \log 10^9)$。

第三节 多模式匹配

多模式匹配是给出多个模式串，查询文本中是否存在每个模式串。

考虑KMP算法，设模式串有n个，总长度为L，文本为T，则KMP算法时间复杂度为Θ(L+n|T|)。

显然通常|T|远大于L，算法十分低效。

是否存在一个算法能把n去掉呢？

答案：把KMP和Trie树相结合的AC自动机！

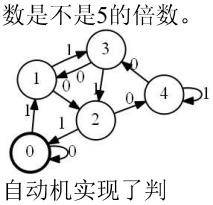
3.1 自动机简介

定义8：确定性有限状态自动机，简称DFA
是一个5元组M = $(\sum, Q, q_s, F, tr)$，其中
- ∑为一个有限字符集，其中每个字符c称为一个输入符号
- Q为一个有限状态集合，
- $q_s \in Q$称为开始状态
- $F \subseteq Q$称为结束状态集合
- $tr \in Q \times \Sigma \to Q$ 为状态转换函数

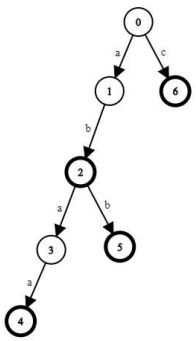
tr(q,c) = q' (q,q' ∈ Q, c ∈ Σ)
表示在状态q输入符号C之后，自动机M将转化到下一个状态q',此时q'称为q的一个后继状态。

对于给定的串S，从开始状态出发，沿边上对应的字符前进，若最终能到达结束状态，称该自动机接受串S；否则（例如无路可走），拒绝S。

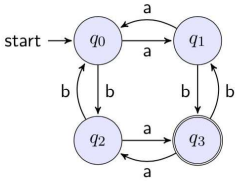
例3：以下自动机实现了判断一个2进制整数是不是5的倍数。

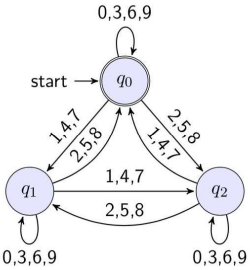


例4：以下自动机实现了判断字符串是不是abaa、abb、ab、c中的一个。



练习：描述下面两个自动机识别的所有串



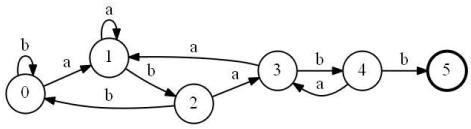


3.2 AC自动机引入

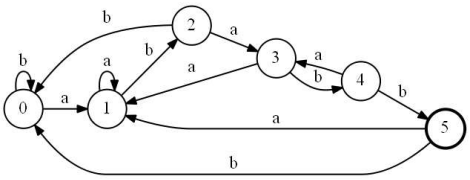
先考虑单模式匹配：利用KMP的思想，自动机状态i表示已经匹配了前i个字符，最后一个状态是结束状态。下面只要想办法构造转移函数即可。以下假设字母表∑={a,b}。

i	fail[i]
0	0
1	0
2	1
3	2
4	0

a	b	a	b	b

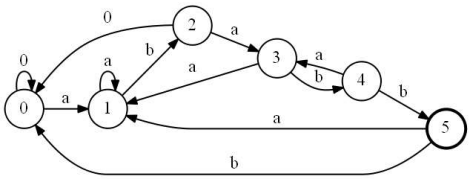


考虑到实际应用中，往往要求找到文本中的所有模式串，故结束状态处同样也加上转移。



这就是一个AC自动机！
显然，给定文本S，AC自动机的匹配时间复杂度为O(|S|)。

可以看出，对于单模式而言，给定fail数组，构造自动机的转移函数方法如下：
对于状态i，符号a，若s[i+1]=a，显然δ(i,a)=i+1；
否则，j从i开始，不断令j=fail[j-1]，直到s[j]=a，
此时δ(i,a)=j+1；
若最后j=0且s[j]≠a，则δ(i,a)=0。



3.3 AC自动机的构造

下面考虑多模式串下AC自动机的构造。以模式串abaa,aa,ba为例。

为了使用之前单模式串的思想，需要将单模式串中的fail数组拓展到多模式串。

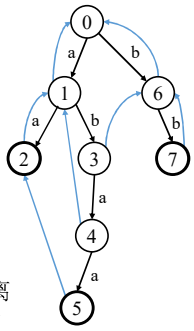
为了将多个模式串统一成一个整体，并方便后续操作，先构造一棵对应的Trie树。之后的AC自动机均是以Trie树为基础构造的。

Trie树的结点是AC自动机的状态，Trie树的边对应了一部分转移函数。开始状态、结束状态也都确定了。

引入fail指针：和KMP中的fail数组完全类似，指向匹配尽可能多字符的状态，如右图所示。

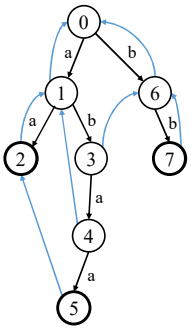
两个关键问题：
（1）如何构造fail指针？
（2）如何根据fail指针构造自动机？

fail数组是一个递推构造的过程，fail指针同样如此。注意到fail指针始终指向离Trie树树根更近的位置，因而fail指针则需要按照状态离根的距离来递推构造。



初始化： Trie树根结点的fail指针为-1。
Trie树第一层的结点，fail指针指向根结点。
其他结点：设结点i的父亲为j，j到i边的符号为a。
重复j=trie[j].fail，直到结点j有符号a出发的边，则
trie[i].fail=trie[j].next[a];
若j已经为-1，则结点i的fail指针指向根结点。

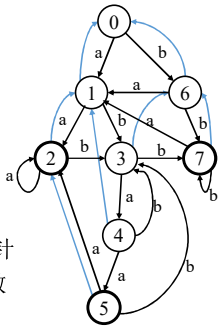
该算法是沿BFS顺序进行的。



根据fail指针构造自动机：

为了求δ(Q,a)，从Q开始不断沿fail指针开始走，直到走到的状态有符号a出发的边，设该边指向的结点为P，则δ(Q,a)=P。

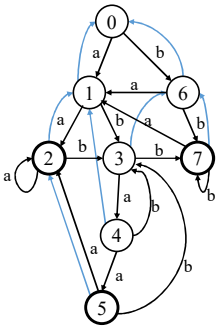
定理10：采用上述思想，构造fail指针的时间复杂度为Θ(L)，构造转移函数的时间复杂度为Θ(L|∑|)，其中L为Trie树中结点个数。



3.4 AC自动机的匹配

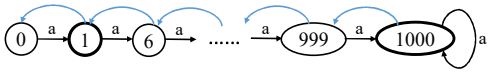
例5：在自动机上运行文本abaababbb。

可以发现，到达状态5后，不仅匹配了模式abaa，也匹配了模式aa。一般地，每到达一个状态，即使该状态不是结束状态，也要沿fail指针不断前进，检查是否和更短的模式串匹配。



事实上，按刚才的做法，每次到达一个状态，都沿fail指针向上找匹配，这样的复杂度可能非常大。这和之前求fail指针是不同的。下面是一个例子：

例6：模式串为a和aaaaaaa……（设有1000个a）共2个，文本为aaaaaaa……（设有10^6个a），自动机如下所示：



可以发现，到达状态1000后，每次都要沿fail指针向前999次，才能找到匹配的模式串a，时间复杂度相当高。

解决办法：每个结点增加一个属性，表示该结点沿fail指针到达的最近一个结束状态，记为match。

递推求match非常容易：

```
trie[i].match = trie[i].num ? i :
trie[trie[i].fail].match;
```

求match属性的时间复杂度为Θ(L)，其中L为Trie树中结点个数。

增加属性match后，每到达一个状态，沿match属性前进，每次必然匹配。

3.5 AC自动机的实现

结构体定义与初始化：

```
#define LETTER 26
struct Trie{
    int num, fail;
    int next[LETTER];
}pool[500001];
Trie* const trie = pool + 1;
int cnt;
void init(){
    cnt = 0;
    memset(pool, 0, 2 * sizeof(Trie));
    trie[0].fail = -1;
}
```

这里使用了一点技巧。

实际情况中，fail指针和状态转移函数构造是同时进行的，均按照BFS顺序。

这样做的好处：

（1）求fail指针简化为一句话：设结点i有一条沿符号a到结点j的边，则j的fail指针为

```
trie[trie[i].fail].next[a];
```

（2）求转移函数简化为一句话：设Trie树中j的父亲是i，则状态j沿字符b的转移为

```
trie[trie[i].fail].next[b];
```

值得注意的是，这是一个常数优化，没有改变时间复杂度。

算法3：AC自动机构造

```
void build()
{
  queue<int> q; q.push(0);
  while (!q.empty()){
    int t = q.front(); q.pop();
    for (int i = 0; i < LETTER; i++){
      int cur = trie[t].next[i];
      if (cur){
        q.push(cur);
        trie[cur].fail = trie[trie[t].fail].next[i];
        trie[cur].match = trie[cur].num ? cur :
          trie[trie[cur].fail].match;
      }
      else cur = trie[trie[t].fail].next[i];
    }
  }
}
```


定理11：AC自动机的构造复杂度为Θ(L |∑|)。

算法4：AC自动机匹配

```
int search(char *s)
{
    int ret = 0, cur = 0;
    for (int i = 0; s[i]; i++){
        cur = trie[cur].next[convert(s[i])];
        for (int temp = trie[cur].match; temp;
            temp = trie[trie[temp].fail].match)
            ret += trie[temp].num;
    }
    return ret;
}
```

定理12：AC自动机匹配算法时间复杂度为Θ(|S|+m)，其中m为匹配成功次数。

3.6 AC自动机的基本应用

编程题5：给定一些模式串，问文本T被这些模式串匹配共多少次。$|T| \le 10^6$，模式串各不相同，且长度和L不超过10^6。

思路：考虑极端情况，模式串为a,aa,aaa,……，文本为aaa……，此时匹配次数可能达到$10^6 \times 1414$，若一个一个数，肯定会超时。此时需要预处理出每个状态能匹配多少次。时间复杂度为Θ(L |∑|+|T|)。

更多应用在后面介绍。

第四节 应用

4.1 KMP与串的去重连接

编程题6：字符串连接有时是需要首尾去重的。例如，将串“(1+2)*3=3*3”和串“3*3=9”相连接，我们希望得到“(1+2)*3=3*3=9”。现在给定串S和T，请进行尽可能长的首尾去重，并输出去重连接结果。$|S|, |T| \le 10^6$。

思路：用串T匹配S，找到首次匹配到S末尾时的位置，这段区间即为最大去重长度。采用KMP算法，时间复杂度Θ($|S| + |T|$)。

4.2 AC自动机与DP

编程题7：DNA改造
已知，DNA上的某些片段将导致遗传性疾病。现在给出一些有疾病的DNA片段（总长度L不超过1000），并给出一条DNA链S（长度不超过10000）。S上可能含有疾病的片段。问最少改变S上的几个碱基对，可以使其不包含有疾病的DNA。注意，DNA链上的碱基对只有A,G,C,T四种。例如有疾病的DNA片段为“A”和“ TG ”，S=TGAATG，则答案为4。

对所有疾病DNA建立AC自动机。
以状态dp[i][j]表示前i个字符到达状态j的情况下最少修改几个字符，则转移方程为：

```
dp[i + 1][cur] = min(dp[i + 1][cur], dp[i][j] + (convert(s[i]) != k));
```

其中k为读入的下一个字符，

```
cur = trie[j].next[k];
```

注意：`cur`状态不能匹配任意疾病字符串，这可由`match`属性判断。
初始条件：`dp[0][0]=0`;其他为无穷
时间复杂度：O(L|S||∑|)。

4.3 AC自动机与矩阵快速幂

编程题8：一个程序由01代码组成，若这个01串中含有某种序列将会被360识别为病毒。请问长度为n的程序中不会被360识别为病毒的程序有几个？360的病毒库大小只有不到100个01字符，$n \le 10^9$。答案对 10^9+7 取模。

定理14：设图G的邻接矩阵为A，则从s经过k条边到t的路径数为矩阵A^k第s行第t列的元素值。

思路：将AC自动机看做有向图，去掉所有是病毒的结束状态后，求0状态经过k条边到所有状态的路径数之和。时间复杂度$O(L^3 \times \log n)$。

4.4 AC自动机的其他应用

与图论中强连通分量结合，解决是否存在无限长的文本，不包含任意指定模式串。

与线性方程组求解相结合，解决概率问题。
有n个人，每人说一段1到6构成的序列，然后开始掷骰子，最先出现自己说的序列的人获胜。求每人获胜的概率。

时间关系，不展开介绍了。

进阶内容：

1、后缀数组

2、后缀自动机

难点：

自动机与图论、数据结构的结合应用

A. 一姬的后缀自动机

内存限制：512 MiB

时间限制：1000 ms

题目描述

一姬已经学习 ACM 七天啦！今天她熟练掌握地了后缀自动机的各种应用。不过她正在打麻将，没有时间去写代码，帮她写一个后缀自动机吧！

以下是后缀自动机的经典应用题：

给定一个长度为 S 的只含小写英文字母的字符串 ，求这个字符串中有多少个本质不同的非空子串？

输入格式

一行一个只由小写英文字母组成的字符串 S 。

输出格式

输出一行一个整数表示答案。

数据范围与提示

$n \leq 1000$

解题思路

算出所有子串的哈希值并放入 set 中即可。计算哈希可以使用递推的方法：

$$Hash(l,r) \equiv Hash(l-1,r-1)b - S_{l-1}b^{r-l+1} + S_r \pmod h$$
$$Hash(0,r) \equiv Hash(0,r-1)b + S_r \pmod h$$

使用双模哈希，防止撞车。（不然怎么会 WA 了 10 发）

代码

```
const int h = 1000000007, p = 998244353, b = 19260817;
llong n, ans = 0, hss = 0, hssp = 0, hssl = 0, hsslp = 0;
llong pw[2000], pwp[2000];
string ck;
set<prdd> tt;
inline int trans(char x)
{
    return x - 'a' + 1;
}
int main()
{
    cin.sync_with_stdio(false);
    cin.tie(0);
    cin >> ck;
    pw[0] = pwp[0] = 1;
    for (int i = 1; i < 2000; i++)
        pw[i] = (1LL * pw[i - 1] * b) % h;
        pwp[i] = (1LL * pwp[i - 1] * b) % p;
```

```
    n = ck.length();
    for (int le = 1; le <= n; le++)
    {
        tt.clear();
        hssl = (1LL * hssl * b + trans(ck.at(le - 1))) % h;
        hsslp = (1LL * hsslp * b + trans(ck.at(le - 1))) % p;
        hss = hssl; hssp = hsslp;
        tt.insert(prdd(hss, hssp));
        for (int i = le; i < n; i++)
        {
            llong bhss = 1LL * hss * b + trans(ck.at(i)),
                snhss = 1LL * trans(ck.at(i - le)) * pw[le];
            hss = (bhss - snhss) % h;
            llong bhssp = 1LL * hssp * b + trans(ck.at(i)),
                snhssp = 1LL * trans(ck.at(i - le)) * pwp[le];
            hssp = (bhssp - snhssp) % p;
            tt.insert(prdd(hss, hssp));
        }
        ans += tt.size();
    }
    printf("%lld", ans);
    return 0;
}
```

B. 一姬的 BM 算法

内存限制：512 MiB

时间限制：1000 ms

题目描述

一姬又学习了用于 $O(n^2)$ 求解递推数列最短递推式的算法：BerlekampMassy 算法（简称 BM 算法），具体请见：毛啸《关于数列递归式的一些研究》。

好吧，以上是一姬骗你的，事实上 BM 算法是一种朴素的字符串匹配算法，用于求解以下问题：

给定一个字符串 A，和另一个字符串 B，问 B 是否是 A 的某个子串？

cyy 觉得这题太简单了，于是把它改了一下：

给定一个环形字符串 A，和一个字符串 B，问 B 是否是 A 的某个子串？

环形字符串表示将一个字符串首尾连接形成的环状结构，例如 **cde** 并没有在链式字符串 **deaaaaaac** 中出现，但是它可以在环形字符串 **deaaaaaac** 中出现，也可以在环形字符串 **aaaaaaced** 中出现

请你帮一姬解决这个问题。

输入格式

第一行一个只有小写英文字母的字符串 A。

第二行一个只有小写英文字母的字符串 B。

输出格式

如果 B 是 A 的某个子串，输出“Yes”，否则输出“No”（不包括引号，**注意字母大小写**）。

数据范围与提示

$1 \le |B| \le |A| \le 10^6$

解题思路

典型的单模式匹配，唯一的不同是环形字符串，转化为将模式 B 与文本 $A[A_1, A_{|B|})$ 和文本 $reverse\left(A[A_1, A_{|B|})\right)$ 匹配即可，任何一个文本匹配成功即成功。

代码

```
int c, s;
int fail[1250000];
string cirs, rcirs, subs;
void mfail()
{
    for (int i = 1, j = 0; i < s; i++)
    {
        while (j && subs[i] != subs[j])
            j = fail[j - 1];
        if (subs[i] == subs[j])
            fail[i] = ++j;
        else
            fail[i] = 0;
    }
}
bool search(string cirs)
{
    for (int i = 0, j = 0; i < c; i++)
    {
        while (j && cirs[i] != subs[j])
            j = fail[j - 1];
        if (cirs[i] == subs[j] && subs[++j] == 0)
            return true;
    }
    return false;
}
int main()
{
    cin.sync_with_stdio(false);
    cin.tie(0);
    cin >> cirs >> subs;
    c = cirs.length();
    s = subs.length();
    mfail();
```

```
    for (int i = 0; i < s - 1; i++)
        cirs += cirs.at(i);
    rcirs = cirs;
    reverse(rcirs.begin(), rcirs.end());
    bool ans = search(cirs) || search(rcirs);
    if (ans)
        printf("Yes");
    else
        printf("No");
    return 0;
}
```

C. JM 的荧光棒工厂

内存限制：512 MiB

时间限制：1000 ms

题目描述

黑心商人 JM 雇佣了 10000 个 wzk 来生产魔法荧光棒，这种魔法荧光棒拿在 dalao 的手上就会闪闪发光。精通字符串理论的 wzk 在工作的时候突发奇想：如果这些荧光棒可以一根接一根地绑在一起，那该有多好！

被批量生产的荧光棒可以视为一个仅包含小写字母的字符串 ，没错这些荧光棒都是一模一样的。两个荧光棒可以被绑在一起，那么他们必须在被绑处有公共的部分，这个部分的长度叫作损失长度。

例如对于 $s=$ **abcccab** ，他可以以 $L=2$ 的损失长度绑在一起，就像这样：

```
abcccab
    abcccab
```

例如对于 $s=$ **aacaa** ，他可以以 $L=1$ 或 $L=2$ 的损失长度绑在一起，就像这样：

```
aacaa
  aacaa
```

or:

```
aacaa
   aacaa
```

当然你不能这样绑，因为这毫无意义，绑了跟没绑一样：

```
aacaa
aacaa
```

现在 wzk 随意地制造了一种荧光棒，他想知道有多少种不同的非零的损失长度，使得两根荧光棒可以被绑在一起。损失长度必须小于荧光棒的总长。

输入格式

输入一行一个字符串 s 。

输出格式

第一行一个非负整数 x ，表示有多少种不同的损失长度。

接下来一行 x 个正整数，表示每种损失长度，用空格隔开，要求按从小到大的顺序输出。

数据范围与提示

$1 \le |s| \le 2 \cdot 10^5$

本题 I/O 量可能较大，请避免使用 cin/cout。

解题思路

由 Border 函数定义可知，最长的损失方案即 $Border(s)$。可以证明，第 i 长的损失方案 $s_i = Border(s_{i+1})$。不断地访问 fail 数组即可。

代码

```
int fail[270000], sl, ans;
string str;
vector<int> brds;
void mfail()
{
    for (int i = 1, j = 0; i < sl; i++)
    {
        while (j && str[i] != str[j])
            j = fail[j - 1];
        if (str[i] == str[j])
            fail[i] = ++j;
        else
            fail[i] = 0;
    }
}
int main()
{
    cin.sync_with_stdio(false);
    cin.tie(0);
    cin >> str;
    ans = sl = str.length();
    mfail();
    while (ans > 1)
    {
        ans = fail[ans - 1];
        if (ans > 0)
            brds.push_back(ans);
    }
    printf("%u\n", brds.size());
    for (int i = brds.size() - 1; i >= 0; i--)
        printf("%d ", brds.at(i));
    return 0;
}
```

D. 一姬的三倍满自动机

内存限制：512 MiB

时间限制：3000 ms

题目描述

一姬想要设计一种机器，使自己不断地和牌上分，但是她觉得麻将是可以让四个人都得到快乐的，至少她不希望雀魂玩家减一这种事情的发生。因此她不希望有人被飞，所以她想要你设计一款三倍满自动机，使得自己在闲家和牌收益最大又不会让人被飞。

由于她只和三倍满，所以做出决策以及提高打点是十分重要的，在自动机中，这需要解决以下问题实现：

给定 n 个数 $a_1, a_2, \ldots, a_n$，给定一个 x，求 $a_i \oplus a_j \oplus x$ 的最大值 $(i \neq j)$，其中 $\oplus$ 表示异或运算。

输入格式

第一行两个用空格分开的正整数 n, x。

第二行 n 个整数，用空格隔开，第 i 个表示 a_i。

输出格式

输出一行一个正整数表示答案

数据范围与提示

$2 \leq n \leq 10^6$

$0 \leq a_i, x \leq 10^9$

解题思路

考虑 Trie 树，按位将所有数字存入 Trie 树中，求一个 a_i 使得 $a_i \oplus x$ 最大的方法很简单：考虑贪心，从高位到低位尽量选取与 x 对应位数字不同的 a_i，复杂度为 $O(\log x)$。本题可以由上述方法扩展得到：取出一个 a_j，求一个 a_i 使得 $a_i \oplus (a_j \oplus x)$ 最大，记此最大值为 $f(j)$，答案即为 $\max(f(j))$。复杂度为 $O(n \log x)$。

代码

```
struct sgtrie
{
    int num, npid[2];
};
int n, x, nid = -1, ans = 0;
vector<sgtrie> trie;
vector<int> pts;
int crtep()
{
    trie.push_back(sgtrie({ 0, -1, -1 }));
    return ++nid;
}
void crtp(int p)
{
```

```
    int nid = 0;
    for (int i = 30; i >= 0; i--)
    {
        int bt = p >> i & 1, tp;
        if (trie.at(nid).npid[bt] < 0)
            tp = crtep(), trie.at(nid).npid[bt] = tp;
        nid = trie.at(nid).npid[bt];
        trie.at(nid).num++;
    }
}
int main()
{
    scanf("%d%d", &n, &x);
    crtep();
    for (int i = 0; i < n; i++)
    {
        int tmp;
        scanf("%d", &tmp);
        crtp(tmp);
        pts.push_back(tmp);
    }
    for (int ci = 0; ci < n; ci++)
    {
        int nwp = pts.at(ci), ts = x ^ nwp, nid = 0;
        bool ckn = true;
        for (int i = 30; i >= 0; i--)
        {
            int bt = (ts >> i & 1) ^ 1, npid = trie.at(nid).npid[bt];
            if (npid > 0 && trie.at(npid).num > (ckn && (nwp >> i == bt)))
                nid = npid;
            else
            {
                bt ^= 1;
                nid = trie.at(nid).npid[bt];
            }
            ts ^= bt << i;
            if (ckn && (nwp >> i) != bt)
                ckn = false;
        }
        ans = max(ans, ts);
    }
    printf("%d", ans);
    return 0;
}
```

E. JM 的模板题

内存限制：512 MiB

时间限制：1000 ms

题目描述

你今天好好听课了吗？

给定一个主串 S，给定 n 个模式串 T_i，你需要求出这些模式串**总共**在 S 中被匹配了多少次。匹配是可以重叠的。总之，你不需要考虑那么多，这就是一道 AC 自动机的模板题。

你感受到被 AC 自动机支配的恐惧了吗？

输入格式

第一行一个字符串 S。

第二行一个正整数 n 表示模式串的个数。接下来 n 行，每行一个模式串 T_i。

输出格式

输出一行一个非负整数表示答案。

数据范围与提示

$1 \le |S| \le 10^6$

$1 \le n \le 10^6$

$1 \le |T_i| \le 10^6$

$1 \le \sum_{i=1}^{n} |T_i| \le 10^6$

解题思路

AC 自动机。（注意优化，否则会变成 TLE 自动机！）

代码

```
struct sgtrie
{
    int num, fail, match, sve;
    int nexts[26];
};
vector<sgtrie> trie;
string txt, mode;
int nom, tid = 0;
void bldac()
{
    queue<int> q;
    q.push(0);
    while (!q.empty())
    {
        int t = q.front();
```

```
        q.pop();
        for (int i = 0; i < 26; i++)
        {
            int cur = trie.at(t).nexts[i];
            if (cur)
            {
                q.push(cur);
                if (t)
                    trie.at(cur).fail = trie.at(trie.at(t).fail).nexts[i];
                else
                    trie.at(cur).fail = 0;
                if (trie.at(cur).num)
                    trie.at(cur).match = cur;
                else
                    trie.at(cur).match = trie.at(trie.at(cur).fail).match;
                int ts = trie.at(cur).num,
                    fsv = trie.at(trie.at(cur).fail).sve;
                trie.at(cur).sve = ts + fsv;
            }
            else
                trie.at(t).nexts[i] = trie.at(trie.at(t).fail).nexts[i];
        }
    }
}
int search(string s)
{
    int ret = 0, cur = 0;
    for (int i = 0; i < s.length(); i++)
    {
        cur = trie.at(cur).nexts[s.at(i) - 'a'];
        ret += trie.at(cur).sve;
    }
    return ret;
}
int main()
{
    cin.sync_with_stdio(false);
    cin.tie(0);
    cin >> txt >> nom;
    trie.push_back(sgtrie());
    for (int i = 0; i < nom; i++)
    {
        cin >> mode;
        int nwid = 0;
        for (int j = 0; j < mode.length(); j++)
```

```
            if (trie.at(nwid).nexts[mode.at(j) - 'a'])
                nwid = trie.at(nwid).nexts[mode.at(j) - 'a'];
            else
            {
                trie.push_back(sgtrie());
                trie.at(nwid).nexts[mode.at(j) - 'a'] = ++tid;
                nwid = tid;
            }
        trie.at(nwid).num++;
    }
    bldac();
    int ans = search(txt);
    printf("%d", ans);
    return 0;
}
```

F. 一姬的役满自动机

内存限制：512 MiB

时间限制：1000 ms

题目描述

你最终还是辜负了初心，与八木唯缔结了契约，一姬内心的怒火可想而知，她决定收回先前的善良，制造出传说中的役满自动机打飞你们！

当然，役满自动机对打点有着更高的要求，自然也就需要更高明的程序解决以下这个更高深的问题：

有一个大数 M，众所周知在 $\bmod m$ 的意义下总共有 $0,1,2\ldots,M-1$ 共 M 个数字，现在把 M 这个数字中的其中 n 个给你，你的第 i 个数字是 a_i，你可以任意取一个你自己拥有的数字 x，再取一个你没有获得的数字 y，那么所有可以这样所产生的 $(x+y)\%M$ 便可以被称为“能生成的数字”，其他的数字称为“不能生成的数字”（数字指的是 $[0,M-1]$ 中的整数）。

请你求出有多少个不能生成的数字，它们分别是什么？

输入格式

第一行两个用空格隔开的正整数 n,M。

第二行 n 个整数，用空格隔开，第 i 个数字表示 a_i，a_i 应不重复，且**从小到大输出**

输出格式

第一行一个整数 k 表示不能生成的数字个数。

第二行 k 个数字用空格隔开，表示 k 个不能生成的数字。

数据范围与提示

$1 \le n \le 2 \cdot 10^5$

$n < M \le 10^9$

$0 \le a_i < M$

Day 8

主题：图论

时间：2019 年 7 月 3 日 星期三

主要内容：

图的表示：一维数组表示，邻接表，邻接矩阵

拓扑排序

最短路：Dijkstra，Bellman，Floyd

最小生成树：Kruskal，Prim

题目：

+	A. JM 的魔法果园	BFS
+	B. cty 的大型魔法	邻接表
+	C. qz 吃 cty	最小生成树，Kruskal
+4	D. jwp 的慈善晚会	最短路，Dijkstra
+2	E. cty 的等式与不等式	最短路，Floyd
−1	F. JM 的俸俸倪购美病	最短路，Dijkstra，BFS
−5	G. JM 的最短路径问题	BFS，字典序

什么是图?

- 答：点的集合和边的集合构成的系统
- 问：什么是点?
- 问：什么是边?

什么是图?

```
#.#.#
..#..
#####
..#..
#.#.#
```

点：矩阵的每个坐标(i,j)就是一个点
边：每个点跟与其相邻的点连一条边，如(1,1)和(1,2)连一条边

点：一个地铁站就是一个点

边：如果从a地铁站可以一站坐到b地铁站，就在这两个点之间连一条边

什么是图?

- 问：什么是点?
- 答：任何元素都可以是点
- 问：什么是边?
- 答：由两个点构成的点对

什么是图?

- 问：边的含义是什么?
- 答：可以是任何含义
- 问：把点和边分离开看，它们的含义还一样吗?
- 答：点和边必须成对出现，分离之后失去本来的含义

什么是图?

- 问：图的作用是什么?
- 答：两个作用：存储数据和处理数据
- 思考：图和数据结构的关系是什么？它和day6学的数据结构有何不同?

图的表示1 画图法

- 优点：直观形象
- 缺点：无法转化为代码

图的表示2 一维数组表示

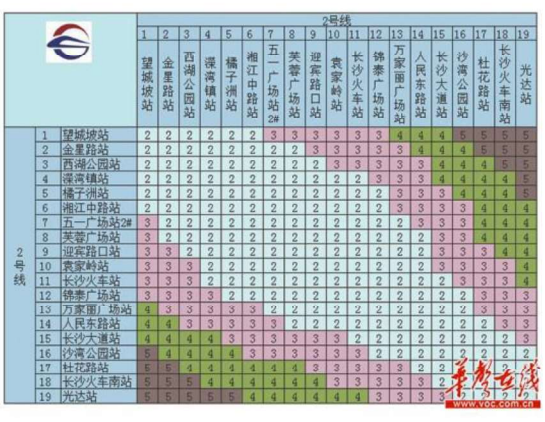
- 点的集合开一个数组表示，边的集合开一个数组表示

```
string v[] = {"小寨", "钟楼", "北大街", "五路口", "大差市"};
pair<int, int> e[] = {{0, 1}, {0, 2}, {1, 2}, {2, 3}, {1, 4}};
```

图的表示2 一维数组表示

- 优点：可以转化为代码，简单，空间复杂度低
- 缺点：遍历一个点所连的边，时间复杂度O(m)，m为图的总边数

图的表示



图的表示3 邻接矩阵

- 问：边权是什么？
- 答：边的一个值，用来量化表示边的含义
- 使用一个矩阵M，M(i, j)表示边(i, j)的边权，如果边(i, j)不存在，M(i, j) = INF

图的表示3 邻接矩阵

- 优点：可以转化为代码，简单
- 缺点：空间复杂度O(n^2)，遍历一个点所连的边，时间复杂度O(n)，n为图的总点数

图的表示4 邻接表

- 小寨：钟楼，北大街
- 钟楼：小寨，北大街，大差市
- 北大街：小寨，钟楼，五路口
- 五路口：北大街
- 大差市：钟楼

图的表示4 邻接表

- 使用一个链表数组或vector数组，第 i 个链表/vector中存放所有和 i 号点有边相连的点的编号

```
vector<int> adj_list;
adj_list[0].push_back(1);
adj_list[1].push_back(2);
```

图的表示4 邻接表

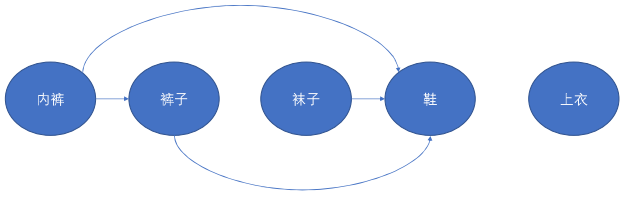
- 优点：可以转化为代码，空间复杂度为O(m)，遍历一个点所连的边，时间复杂度O(e)，m为总边数，e为连接该点的边的数量
- 缺点：无法O(1)得知两点之间是否连边

拓扑排序

- 问：边 (u, v) 和 边 (v, u) 一样吗？
- 答：无向图中一样，有向图中不一样
- 问：什么是无向图？什么是有向图？
- 答：有向图中边是有方向的，边连接的两个点，其中一个是起点，另一个是终点。无向图则不分起点和终点。

拓扑排序

- 将一个有向图中的所有点，按一个线性顺序排列，要求原图中不存在这样一条边，它起点的线性顺序在终点的线性顺序之后。



拓扑排序

```
void toposort()
{
    stack<int> s;
    for (int i = 1; i <= n; ++i)
        if (indeg[i] == 0)
            s.push(i);
    while (!s.empty())
    {
        int u = s.top();
        s.pop();
        for (int v : adj_list[u])
        {
            if (--indeg[v] == 0)
                s.push(v);
        }
    }
}
```

拓扑排序

- 每条边最多被处理一次，再加上预处理的遍历
- 时间复杂度 O(n + m)，n为点数，m为边数

拓扑排序

- 思考：任何图都有拓扑序吗？
- 思考：任何有向无环图（DAG）都有拓扑序吗？
- 思考：拓扑排序的结果是有向无环图（DAG）吗？
- 思考：修改拓扑排序的算法，在不增加渐进复杂度的情况下，使算法能够识别出环

最短路

- 问：什么是一条路？
- 问：路的权值怎么算？
- 答：路里面每条边的权值和
- 问：什么是两点间的最短路？
- 答：两点间权值最小的路

最短路 dijkstra

- 输入：带权图，起点s
- 输出：s到所有点的最短路

最短路 dijkstra

- 设 i 到 j 的最短路为 d(i, j)
- 前提：图中没有负权图
- 思想：i 到 j 的最短路，如果经过点 k，那么 d(i, j) = d(i, k) + d(k, j)
- 思想：当前距离起点最近的一个点，其最短路已不可能更短

最短路 dijkstra

```
void dijkstra(int start)
{
    dist[start] = 0;
    for (int i = 1; i <= n; ++i)
    {
        int min_val = INF;
        int min_id;
        for (int j = 1; j <= n; ++j)
            if (!vis[j] && dist[j] < min_val)
            {
                min_val = dist[j];
                min_id = j;
            }
        vis[min_id] = 1;
        for (int j = 1; j <= n; ++j)
            dist[j] = min(dist[j], min_val + w[min_id][j]);
    }
}
```

最短路 dijkstra

- 时间复杂度O(n^2)，n为点数

最短路 dijkstra

- 思考：能否优化？
- 思考：优先队列取出的内容，一定需要进一步松弛吗？
- 思考：能否进一步优化？
- 思考：一定是优化吗？

最短路 Bellman

- 输入：带权图，起点s
- 输出：s到所有点的最短路

最短路 Bellman

- 设 i 到 j 的最短路为 d(i, j)
- 前提：图中没有负环
- 思想：i 到 j 的最短路，如果经过点 k，那么 d(i, j) = d(i, k) + d(k, j)
- 思想：每轮对每条边执行松弛操作，经过 n 轮，可以得出最短路

最短路 Bellman

```
void bellman(int start)
{
    dist[start] = 0;
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= n; ++j)
            for (int v : adj_list[j])
                dist[v] = min(dist[v], dist[j] + w[j][v]);
}
```

最短路 Bellman

- 时间复杂度 O(nm)，n为点数，m为边数

最短路 Bellman

- 思考：每轮选边有没有什么技巧？它能优化最坏情况下时间复杂度吗？
- 思考：修改算法，使得在图有负环的情况下，算法可以识别

最短路 Floyd

- 输入：带权图
- 输出：所有点对之间的最短路

最短路 Floyd

- 思想：i 到 j 的最短路，如果经过点 k，那么 d(i, j) = d(i, k) + d(k, j)

最短路 Floyd

```
void floyd()
{
    for (int k = 1; k <= n; ++k)
        for (int i = 1; i <= n; ++i)
            for (int j = 1; j <= n; ++j)
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
}
```

最短路 Floyd

- 时间复杂度：O(n^3)，n为点数

最短路 Floyd

- 思考：dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
- 此时的 dist[i][k] 和 dist[k][j] 一定是最短路吗？
- 思考：既然不一定，如何保证算法的正确性？
- 思考：如果有负环，会发生什么？

最短路

- 思考：无向图和有向图有区别吗？
- 思考：什么时候用dijkstra，什么时候用Bellman，什么时候用Floyd？
- 思考：最长路能求吗？

POJ 1860

- n个货币兑换点，m种货币，每个点把一种货币兑换成另一种货币，有一定的汇率，初始你有一定的1号货币，你能不能通过一系列交换，使得你的1号货币增多？

洛谷 P1119

- n个点，m条边的带权无向图，一开始所有点都是坏的，接下来执行q次操作，操作分两种：
- 1. 修好一个点
- 2. 询问两个点之间的最短路径长度
- 一条路径不能经过任何一个坏点

最小生成树

- 问：什么是生成树？
- 答：在图上选择n-1条边，使得所有点连通，这n-1条边构成的树即为生成树
- 问：什么是生成树的权值？
- 答：树上所有边的权值和
- 问：什么是最小生成树？
- 答：权值最小的一棵生成树

最小生成树 Kruskal

- 输入：带权图
- 输出：最小生成树

最小生成树 Kruskal

- 思想：贪心，优先选择权值较小的边

最小生成树 Kruskal

```
int kruskal()
{
    int ret = 0;
    for (int i = 0; i < m; ++i)
        if (s.find(e[i].u) != s.find(e[i].v))
        {
            ret += e[i].w;
            s.merge(e[i].u, e[i].v);
        }
    return ret;
}
```

最小生成树 Kruskal

- 时间复杂度 O(mlogm)，m为边数

最小生成树 Kruskal

- 思考：修改算法，使得可以识别图不连通的情况

最小生成树 Prim

- 输入：带权图
- 输出：最小生成树

最小生成树 Prim

- 思想：贪心，将图上的点分成两类，已经在最小生成树里的点和未在最小生成树里的点，每次在第二类中选一个距离第一类点最近的点，加入最小生成树

最小生成树 Prim

```
int prim()
{
    int ret = 0;
    int cur = 1;
    dist[1] = 0;
    vis[1] = 1;
    for (int i = 2; i <= n; ++i)
    {
        for (int j = 1; j <= n; ++j)
            dist[j] = min(dist[j], w[cur][j]);
        int min_val = INF;
        int mid_id;
        for (int j = 1; j <= n; ++j)
            if (!vis[j] && dist[j] < min_val)
            {
                min_val = dist[j];
                min_id = j;
            }
        ret += min_val;
        cur = min_id;
    }
    return ret;
}
```

最小生成树 Prim

- 时间复杂度：O(n^2)，n为点数

最小生成树 Prim

- 思考：Prim 可以像 Dijkstra 那样优化吗？

最小生成树

- 思考：无向图的最小生成树和有向图的最小生成树有区别吗？我们刚才讲的算法能求有向图的最小生成树吗？
- 思考：证明最小生成树的最大边边权最小？
- 思考：不同的最小生成树，它们边权的构成一样吗？
- 思考：最大生成树能求吗？

校赛 H 题

甲方给了你一块$N \times M$的网格区域，外边界上的墙已经修筑好了，除此以外，每两个水平或垂直相邻的格子之间可以建造一堵墙，建造每堵墙的花费都是已知的。

现在他要求你在这片网格区域上建造若干堵墙，形成一个网格迷宫。甲方将会把这个迷宫作为一个游乐设施，他们会把一些玩家随机扔进迷宫中，落在迷宫内的随机一点上，然后玩家需要走一条不重复经过任意点的路径，走到一个指定目标点。

甲方希望你的迷宫能够最大地限制玩家，也就是说，对于一个被随机扔进迷宫任意一点(x_1, y_1)的人，无论他的指定目标点(x_2, y_2)是迷宫的哪一点，他都有且只有唯一一条不重复路径能够抵达目标点。同时，在满足条件的前提下，甲方还希望你以最小的花费建墙。

POJ 3159

- n个未知数，m条不等式
- 每条不等式都是 $x_i - x_j$ <= c，c是常数
- 求$x_1 - x_n$的最大值

POJ 3159

- 思考：不等式全部变成 $x_i - x_j$ >= c，改求$x_1 - x_n$的最小值
- 思考： $x_i - x_j$ >= c 和 $x_i - x_j$ <= c 以及 $x_i - x_j$ = c都有
- 思考：哪些情况无解

POJ 1679

- 求次小生成树

A. JM 的魔法果园

内存限制：512 MiB

时间限制：1000 ms

题目描述

JM 有一个魔法果园，里面一共有 n 棵苹果树，编号为 $1,2,\ldots,n$ 。这些苹果树之间通过 $n-1$ 条魔法线相连，每一条魔法线可以连接两棵不同的苹果树，不存在两条魔法线连接的两对苹果树是完全相同的。

由于昨天 cty 在山上胡乱释放大型魔法，导致果园受到了扰乱，魔法线的连接结构全都变了，这可把 JM 弄懵逼了。因为以前他是知道这些魔法线恰好能将这些苹果树全部连通的，意思是，对于任意两棵苹果树，它们总能经过若干条魔法线直接或间接地连接上。而现在它们是不是全部连通的就是个未知数了。

JM 还要忙着去把 cty 抓去喂 qz，所以他没空，希望你能帮他判断一下，现在这些苹果树是否还是全部连通的。如果不连通，那么 JM 就要花费大量的精力去修正了，如果连通的话，你还需要帮他重新构建游客指引。

因为魔法果园是一个 9A 级景点，游客参观量是非常大的。果园的 1 号苹果树最为健壮，它种在大门口，游客们首先会被引导到这里。接下来，游客们总是随着魔法线的指引进行后续的参观。因为果园只有一个出入口且游客们的体力有限，所以他们总是会希望知道，自己当前距离 1 号苹果树的最短距离是多少。

输入格式

第一行一个正整数 n 表示果园里的苹果树个数。

接下来 $n-1$ 行，每行两个正整数 u,v 表示编号为 u 的苹果树和编号为 v 的苹果树之间有一条魔法线。每条魔法线的距离长度都是 1 。

输出格式

第一行一个字符串“Yes”或“No”（不包括引号）表示果园里的苹果树是否全部连通。

如果答案为“Yes”，则接下来还有一行 个非负整数，其间用空格隔开，其中第 个数表示编号为 的苹果树距离编号为 的苹果树的最短距离是多少。

数据范围与提示

$1 \leq n \leq 2 \cdot 10^5$

解题思路

从 1 号点开始进行 BFS 即可。若经过的点不足 n 个，则不是全部连通的。

代码

```
int n, dis[1270000], dsn, opn = 0;
bool gkd[1270000];
vector<int> ways[1270000], bfs, tmp;
int main()
{
    scanf("%d", &n);
    for (int i = 0; i < n - 1; i++)
    {
```

```
        int u, v;
        scanf("%d%d", &u, &v);
        ways[u].push_back(v);
        ways[v].push_back(u);
    }
    bfs.push_back(1);
    while (!bfs.empty())
    {
        for (int i = 0; i < bfs.size(); i++)
        {
            int nwp = bfs.at(i);
            gkd[nwp] = true;
            for (int j = 0; j < ways[nwp].size(); j++)
            {
                int np = ways[nwp].at(j);
                if (!gkd[np])
                {
                    if (dis[np] != 0)
                        goto badend;
                    tmp.push_back(np);
                    dis[np] = dis[nwp] + 1;
                    opn++;
                }
            }
        }
        swap(bfs, tmp);
        tmp.clear();
    }
    if (opn != n - 1)
        goto badend;
    printf("Yes\n");
    for (int i = 1; i <= n; i++)
        printf("%d ", dis[i]);
    return 0;
badend:
    printf("No");
    return 0;
}
```

B. cty 的大型魔法

内存限制：512 MiB

时间限制：1000 ms

题目描述

JM 有一个魔法果园，里面一共有 n 棵苹果树，编号为 $1,2,\ldots,n$ 。这些苹果树之间通过 m 条魔法线相连，每一条魔法线以一棵苹果树 u 作为起点，以另一颗苹果树 v 作为终点，只要这条魔法线存在，我们就可以认为苹果树 受到了苹果树 的魔力保护。

假如你是 cty，你站在果园后面的山上，准备对果园里的苹果树施放大型魔法。大型魔法的施法目标是果园里任意一棵苹果树，如果目标苹果树没有被任何苹果树的魔力保护（也就是说，不存在任何一条魔法线的终点是目标苹果树），那么大型魔法会摧毁目标苹果树以及所有以目标苹果树为起点或终点的魔法线；如果目标苹果树受到了至少一棵苹果树的魔力保护，那么大型魔法无效。

你胡乱施放了一连串大型魔法，现在你想知道果园里面还剩多少棵苹果树。你需要很快地计算出来，以便于逃跑。如果程序 TLE 的话，你就要被 JM 抓去喂 qz 哦！ ^_^

输入格式

第一行三个正整数 n,m,k，分别表示果园里的苹果树个数、魔法线的个数、cty 施放的大型魔法个数。

接下来 m 行，每行两个正整数 u,v 表示存在一条以编号为 u 的苹果树作为起点，编号为 v 的苹果树作为终点的魔法线。

接下来 k 行，每行一个正整数 x，按顺序给出从第 1 次到第 k 次大型魔法的目标苹果树编号。输入数据保证每行给出的目标苹果树没有被之前的大型魔法摧毁。

输出格式

输出一个非负整数，表示果园里面还剩多少棵苹果树。

数据范围与提示

$1 \le n,m,k \le 2 \cdot 10^5$

$1 \le u,v,x \le n$

解题思路

对于每个点，存储该点的上游节点和下游节点。若没有上游节点，则该点被摧毁，所有下游节点的上游节点集合移除被摧毁的点；否则无事发生。

代码

```
int n, m, k;
set<int> from[270000], to[270000];
int main()
{
    scanf("%d%d%d", &n, &m, &k);
    for (int i = 0; i < m; i++)
    {
        int u, v;
        scanf("%d%d", &u, &v);
```

```
        to[u].insert(v);
        from[v].insert(u);
    }
    for (int i = 0; i < k; i++)
    {
        int dsty;
        scanf("%d", &dsty);
        if (from[dsty].empty())
        {
            n--;
            for (auto p : to[dsty])
                from[p].erase(dsty);
        }
    }
    printf("%d", n);
    return 0;
}
```

C. qz 吃 cty

内存限制：512 MiB

时间限制：1000 ms

题目描述

假如你是 cty，你在知道了魔法果园还剩多少棵苹果树之后就从山上逃走了，但不幸的是你还是被 JM 抓住了。

经过一番激♂烈的搏斗，你被 JM 带到了 qz 的洞穴里，qz 的洞穴由 n 个不连通的房间组成，编号为 $1, 2, \ldots, n$ 。qz 决定给你一次机会，如果你帮他在洞穴里修建通道，他就不吃你。

qz 已经告诉了你哪些房间之间可以修建通道，以及修建每条通道的代价，你需要选择其中的一些通道修建，使得所有房间都直接或间接通过通道连通，并且修建通道的总代价最小。

此外，在满足修建通道总代价最小的前提下，qz 还有一个要求：直接连接 1 号房间的通道要尽可能多，因为 1 号房间是餐厅，这样修建可以使 qz 吃饭方便一些。

修通道之前，你需要计算出在满足修建通道总代价最小的前提下，直接连接 1 号房间的通道最多能有多少条。qz 的耐心是有限的，所以如果程序 TLE 的话，qz 就会在 1 号房间吃掉你！ ^_^

输入格式

第一行两个正整数 n, m ，分别表示房间的个数和候选通道的个数。

接下来 m 行，每行三个整数 u, v, w ，表示可以修建一条直接连接 u 号房间和 v 号房间的通道，修建这条通道的代价为 w 。

数据保证一定有方案使得所有房间都直接或间接通过通道连通。

输出格式

输出一个非负整数，表示在满足修建通道总代价最小的前提下，直接连接 1 号房间的通道最多能有多少条。

数据范围与提示

$1 \le n \le 2 \cdot 10^5$

$n < m \le 2 \cdot 10^5$

$1 \le w \le 10^4$

$1 \le u, v \le n$

解题思路

最小生成树。在选择等长的路径时，优先选择连通 1 号房间的路径。

代码

```
struct side
{
    int from, to, val;
};
int n, m, dd = 0, ans = 0;
int fts[270000];
vector<side> sds;
int cmp(side a, side b)
{
    if (a.val == b.val)
        return min(a.from, a.to) < min(b.from, b.to);
    return a.val < b.val;
}
int fnd(int x)
{
    return (fts[x] == 0 || fts[x] == x) ? x : fts[x] = fnd(fts[x]);
}
void mrg(int f, int t)
{
    fts[fnd(f)] = fnd(t);
}
int main()
{
    scanf("%d%d", &n, &m);
    for (int i = 0; i < m; i++)
    {
        int u, v, w;
        scanf("%d%d%d", &u, &v, &w);
        sds.push_back(side({ u, v, w }));
    }
    sort(sds.begin(), sds.end(), cmp);
    for (int i = 0; i < m; i++)
    {
        side a = sds.at(i);
```

```
        if (fnd(a.from) == fnd(a.to))
            continue;
        else
        {
            dd++;
            if (a.from == 1 || a.to == 1)
                ans++;
            if (dd == n - 1)
                break;
            mrg(a.from, a.to);
        }
    }
    printf("%d", ans);
    return 0;
}
```

D. jwp 的慈善晚会

内存限制：512 MiB

时间限制：2000 ms

题目描述

热心公益的 jwp 又来举办慈善晚会了，这次他邀请到了巴菲特、马云等巨富，还邀请到了 cyy、wzk 等算法界泰斗。晚会一共邀请了 n 位尊贵的客人，每位客人都位于不同的城市，也就是说每座城市都有且仅有一位客人。这些城市的编号为 $1,2,\ldots,n$，jwp 决定将晚会放在 p 城市举办。

城市之间有 m 条**单向的**交通路径（两座城市间可能同时存在多条直接相连的路径），通过每一条路的花费时间为 t_i。这些客人都日理万机，工作繁忙，因此他们会选择时间最短的路径往返 p 城市，jwp 想知道客人中花费时间最长的人需要在路上花费多久。

输入格式

第一行三个正整数 n,m,p。

接下来 m 行，每行三个整数 u_i,v_i,t_i，分别表示一条交通路径的起点，终点和用时。

输入保证图是强连通的，即所有客人都能往返 p 城市。

输出格式

输出一行一个整数，表示花费时间最久的客人所需的时间。

数据范围与提示

$2 \le n \le 10^3$

$n \le m \le 10^5$

$1 \le t_i \le 100$

解题思路

由于路径是单向的，居住在 x 城的客人往返花费的时间为 $t_{\min}(x,p)+t_{\min}(p,x)$，使用 n 次 Dijkstra 算法即可。

代码

```
int n, m, pl, ans = 0;
bool vis[1050];
int dis[1050], acmp[1050];
mpdd ways[1050];
int dijkstra(int srt, int end)
{
    for (int i = 0; i <= n; i++)
        dis[i] = dinf;
    dis[srt] = 0;
    for (int i = 0; i <= n; i++)
        vis[i] = false;
    priority_queue<prdd, vector<prdd>, greater<prdd>> mins;
    mins.push(prdd(0, srt));
    while (!mins.empty())
    {
        prdd nmin = mins.top();
        mins.pop();
        if (nmin.second == end)
            return nmin.first;
        if (nmin.first > dis[nmin.second] || vis[nmin.second])
            continue;
        vis[nmin.second] = true;
        for (auto p : ways[nmin.second])
        {
            if (dis[p.first] > nmin.first + p.second)
            {
                dis[p.first] = nmin.first + p.second;
                mins.push(prdd(dis[p.first], p.first));
            }
        }
    }
    return dinf;
}
int main()
{
    scanf("%d%d%d", &n, &m, &pl);
    for (int i = 0; i < m; i++)
    {
        int u, v, t;
        scanf("%d%d%d", &u, &v, &t);
        if (ways[u][v] > 0)
            ways[u][v] = min(t, ways[u][v]);
        else
            ways[u][v] = t;
```

```
    }
    dijkstra(pl, -1);
    for (int i = 1; i <= n; i++)
        acmp[i] = dis[i];
    for (int i = 1; i <= n; i++)
    {
        acmp[i] += dijkstra(i, pl);
        ans = max(ans, acmp[i]);
    }
    printf("%d", ans);
    return 0;
}
```

E. cty 的等式与不等式

内存限制：512 MiB

时间限制：2000 ms

题目描述

出题人懒得想题面了，直接把题丢给了你。

你手里有 n 个未知数，分别为 $x_1, x_2, \ldots, x_n$，这些未知数之间需要满足 m 个条件，每个条件是以下三种形式之一：

1. $x_a \le x_b + c$
2. $x_a = x_b + c$
3. $x_a \ge x_b + c$

其中 x_a, x_b 代表两个不同的未知数，c 代表一个常数。

你需要回答 q 次询问，每次询问给出两个正整数 a, b，你需要计算出 $\max(x_a - x_b)$。

输入格式

第一行两个正整数 n, m, q，分别表示未知数的个数和条件的个数。

接下来 m 行，每行四个正整数 k, a, b, c，表示一个条件。$k = 1$ 时，$x_a \le x_b + c$；$k = 2$ 时，$x_a = x_b + c$；$k = 3$ 时，$x_a \ge x_b + c$。

接下来 q 行，每行两个正整数 a, b，表示一次询问。

输出格式

对于每次询问输出一行，总共输出 q 行。

每行一个整数，表示 $\max(x_a - x_b)$。

数据保证每次询问一定有整数解。

数据范围与提示

$1 \le n \le 5 \cdot 10^2$

$1 \le m, q \le 10^4$

$1 \le a, b \le n$

$-10^5 \le c \le 10^5$

解题思路

根据输入的不等式，可以得出以下结论：

1. 所有的不等式均可以转化为 $x_a - x_b \le c$ 的形式，其中 $x_a = x_b + c$ 等价于 $\begin{cases} x_a - x_b \le c \\ x_b - x_a \le c \end{cases}$。

2. 若同时满足 $\begin{cases} x_a - x_b \le c_1 \\ x_b - x_d \le c_2 \end{cases}$，则同时亦满足 $x_a - x_d \le c_1 + c_2$。

3. 若同时满足 $\begin{cases} x_a - x_b \le c_1 \\ x_a - x_b \le c_2 \\ \quad\vdots \\ x_a - x_b \le c_n \end{cases}$，则有 $\max(x_a - x_b) = \min_{i=1}^{n} c_i$。

将 x_a 作为节点，将 $x_a - x_b \le c$ 作为连接 x_a, x_b、边长为 c 的有向边建立图，则根据上述结论可以得到：任意一条从 x_a 到 x_b 的通路均为一个不等式，$\max(x_a - x_b)$ 即 x_a 到 x_b 的最短路。

由于 c 可能为负数，故使用 Floyd 算法，复杂度为 $O(n^3)$。

代码

```
int n, m, q;
int dis[512][512];
int infpls(int a, int b)
{
    if (a == dinf || b == dinf)
        return dinf;
    return a + b;
}
void Floyd()
{
    for (int k = 1; k <= n; k++)
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++)
                dis[i][j] = min(dis[i][j], infpls(dis[i][k], dis[k][j]));
}
int main()
{
    scanf("%d%d%d", &n, &m, &q);
for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n; j++)
            dis[i][j] = dinf;
        dis[i][i] = 0;
    }
    for (int i = 0; i < m; i++)
    {
        int k, a, b, c;
        scanf("%d%d%d%d", &k, &a, &b, &c);
```

```
        if (k == 1 || k == 2)
            dis[a][b] = min(dis[a][b], c);
        if (k == 2 || k == 3)
            dis[b][a] = min(dis[b][a], -c);
    }
    Floyd();
    for (int i = 0; i < q; i++)
    {
        int a, b;
        scanf("%d%d", &a, &b);
        printf("%d\n", dis[a][b]);
    }
    return 0;
}
```

F. JM 的俸俸倪购美病

内存限制：512 MiB

时间限制：2000 ms

题目描述

JM 今天懒得编题面了，于是他直接把题意赤裸裸地甩了出来：

给你一个 n 个点， m 条边的无向图，对于任意两点，它们之间的任意一条最短路上的边都是俸俸倪购美病的。

你要回答 q 次询问，每次询问给你 u,v 两点，你要回答对于 u,v 两点，这张图上有多少条边是俸俸倪购美病的。

输入格式

第一行三个正整数 n,m,q ，分别表示点的个数，边的个数，询问的个数。

接下来 m 行，每行三个正整数 u,v,w ，表示有一条连接 u 和 v 的无向边，边权为 w 。

接下来 q 行，每行两个正整数 u,v ，表示询问你对于 u,v 两点，这张图上有多少条边是俸俸倪购美病的。

数据保证图上无重边和自环。

输出格式

每个询问输出一行，一共输出 q 行。

对于每个询问，输出一个非负整数，表示对于询问的两点，这张图上有多少条边是俸俸倪购美病的。

数据范围与提示

$1 \le n \le 5 \cdot 10^2$

$1 \le m,q \le n \cdot (n-1)/2$

$1 \le u,v \le n$

$1 \le w \le 10^3$

解题思路

首先求得最短路，在求最短路的同时用 $frm(i,j)$ 存储从 i 到 j 的最短路上，j 点的上游节点。在求 i 到 j 的最短路总边数时根据 $frm(i,j)$ 数据经由所有最短路上的边进行反向 BFS，统计走过的边数即可。每次查询的复杂度为 $O(n)$，总复杂度为 $O(n^3)$，

代码

```
int n, m, q;
int dis[512][512];
bool dijkstraed[512];
vector<int> frm[512][512];
vector<prdd> pic[512];
struct tprdd
{
    int first, second, third;
};
struct gter_tprdd
{
    bool operator() (const tprdd &x, const tprdd &y)
    {
        return x.first > y.first;
    };
};
void dijkstra(int fr)
{
    dijkstraed[fr] = true;
    bool vis[512];
    for (int i = 0; i <= n; i++)
        dis[fr][i] = dinf;
    for (int i = 0; i <= n; i++)
        vis[i] = false;
    priority_queue<tprdd, vector<tprdd>, gter_tprdd> mins;
    mins.push(tprdd{ 0, fr, -1 });
    while (!mins.empty())
    {
        tprdd nm = mins.top();
        mins.pop();
        if (nm.first > dis[fr][nm.second])
            continue;
        frm[fr][nm.second].push_back(nm.third);
        if (vis[nm.second])
            continue;
        vis[nm.second] = true;
        for (auto p : pic[nm.second])
            if (dis[fr][p.first] >= nm.first + p.second)
            {
```

```
                dis[fr][p.first] = nm.first + p.second;
                mins.push(tprdd{ dis[fr][p.first], p.first, nm.second });
            }
    }
}
int bfs(int fr, int to)
{
    int ans = 0;
    bool bfsd[512];
    for (int i = 0; i <= n; i++)
        bfsd[i] = false;
    vector<int> bfsv, tmp;
    bfsv.push_back(fr);
    while (!bfsv.empty())
    {
        for (unsigned i = 0; i < bfsv.size(); i++)
        {
            int nw = bfsv[i];
            if (nw == to)
                continue;
            for (unsigned j = 0; j < frm[to][nw].size(); j++)
            {
                if (!bfsd[frm[to][nw][j]])
                {
                    tmp.push_back(frm[to][nw][j]);
                    bfsd[frm[to][nw][j]] = true;
                }
                ans++;
            }
        }
        bfsv = tmp;
        tmp.clear();
    }
    return ans;
}
int main()
{
    scanf("%d%d%d", &n, &m, &q);
    for (int i = 0; i < m; i++)
    {
        int u, v, w;
        scanf("%d%d%d", &u, &v, &w);
        pic[u].push_back(prdd(v, w));
        pic[v].push_back(prdd(u, w));
    }
```

```
    for (int i = 0; i < q; i++)
    {
        int u, v, ans;
        scanf("%d%d", &u, &v);
        if (!dijkstraed[u])
            dijkstra(u);
        ans = bfs(v, u);
        printf("%d\n", ans);
    }
    return 0;
}
```

G. JM 的最短路径问题

内存限制：512 MiB

时间限制：1000 ms

题目描述

JM 今天懒得编题面了，于是他直接把题意赤裸裸地甩了出来：

给定一个 n 个点、 m 条边的无向连通图，节点编号为 $1,2,\ldots,n$ ，边的编号为 $1,2,\ldots,m$ ，每条边的长度均为 1 。

设 $dist(u,v)$ 表示图上从节点 u 到节点 v 的最短路径长度。要求在这 m 条边中选择恰好 $n-1$ 条边，将其余边全部删除，使得 $sum=\sum_{i=1}^{n} dist(1,i)$ 的值最小。

现在，你需要求出使得 sum 最小的选边方案有多少种，并输出所有的合法方案。输出格式为 $b_1b_2\ldots b_m$ ，其中 $b_i=1$ or 0 表示编号为 i 的边选或不选。输出时按照 01 串 b 的字典序从大到小输出。具体可参考输出样例。

当然，在某些情况下，合法的方案数可能会非常大。因此另外给定一个正整数 k ，若合法的方案数超过 k 种，则你需要输出的合法方案数为 k ，并且接下来只输出字典序前 k 大的 k 种合法方案。

输入格式

第一行三个正整数 n,m,k ，其中 n,m 分别表示图的点数和边数， 表示你需要输出的最大方案数。

接下来 m 行，第 i 行表示编号为 i 的边，每行两个正整数 u 和 v ，表示这条双向边所连接的两个节点的编号。

输入保证图是连通的。

输出格式

第一行一个正整数 x ，表示使得 sum 最小的选边方案有多少种。如果方案数超过 k ，则输出 k 而不是实际的方案数，也就是说你需要输出 $\min(x,k)$ 。

接下来 $\min(x,k)$ 行，每行一个 01 串 $b_1b_2\ldots b_m$ ，表示一种合法的方案。输出时按照 01 串 b 的字典序从大到小输出。若合法的方案数超过 k 种，则只输出字典序前 k 大的 k 种合法方案。

数据范围与提示

$2\le n\le 2\cdot 10^5$

$n < m \le 2 \cdot 10^5$

$1 \le k \le 2 \cdot 10^5$

$m \cdot k \le 10^6$

本题输出量较大，请使用 scanf/printf 以免 I/O 时间过长导致 TLE。

解题思路

由于边权相同，故使用 BFS 计算各点到 1 号点的最短距离。任何一个 $dist(1,i) = d_i$ 的点 i，均有至少一个 $dist(1,j) = d_i - 1$ 的 j 与其有边直接连通。在多个满足上述条件的边中，选择任意多个均能保证 $dist(1,i)$ 不变。由于只能选择 $n-1$ 条边，故对于任意一个点 i，其仅能选至多 1 条边与距离更小的点连接，记可选的边集合为 S_i，边数为 σ_i。则总选择方案数 $x = \prod_{i=2}^{n} \sigma_i$。

接下来考虑字典序的问题：01 串字典序最大，意味着编号更小的边应更优先出现。在选择这 $n-1$ 个边时，从每个 S_i 中均选择编号最小的边，这样的方案是字典序最大的。之后选择编号最大的边，设其在集合 S_i 中，将其替换为该集合中比其编号大的下一个边。若不能选出这样的边，该边应该被“复位”，然后在该方案中其余的边中选择编号最大的边重复以上操作，直到替换成功为止。替换成功是，所有需要复位的边在此时进行复位：将所有需要复位的边替换为所在集合中比被替换的边编号大的下一个边。这样的复位方法能保证：能保证字典序变小（不会开倒车）；且不会存在一种方案满足字典序比原方案的小且比新方案的大（不会遗漏方案）。重复 $\min(x,k)-1$ 次产生新方案的过程即可。

代码

```
int n, m, k, ans = 1;
int dit[270000], usd[270000];
vector<int> frm[270000];
vector<prdd> sgans, pic[270000];
void bfs(int fr)
{
    for (int i = 0; i <= n; i++)
        dit[i] = dinf;
    int stp = 0;
    vector<int> bfsv, tmp;
    bfsv.push_back(fr);
    dit[fr] = 0;
    while (!bfsv.empty())
    {
        stp++;
        for (unsigned i = 0; i < bfsv.size(); i++)
        {
            int nw = bfsv[i];
            for (unsigned j = 0; j < pic[nw].size(); j++)
            {
                int np = pic[nw][j].first, wy = pic[nw][j].second;
                if (dit[np] >= stp)
                {
                    if (dit[np] > stp)
                        tmp.push_back(np);
```

```
                    dit[np] = stp;
                    frm[np].push_back(wy);
                }
            }
        }
        bfsv = tmp;
        tmp.clear();
    }
}
void optans()
{
    for (int j = 0; j < sgans[0].first; j++)
        printf("0");
    printf("1");
    for (int i = 1; i < sgans.size(); i++)
    {
        int lsd = sgans[i - 1].first,
            nsd = sgans[i].first;
        for (int j = lsd + 1; j < nsd; j++)
            printf("0");
        printf("1");
    }
    for (int j = sgans[n - 2].first + 1; j < m; j++)
        printf("0");
    printf("\n");
}
inline int ub(int x, int y)
{
    vector<int>::iterator b = frm[x].begin();
    return upper_bound(b, frm[x].end(), y) - b;
}
int main()
{
    scanf("%d%d%d", &n, &m, &k);
    for (int i = 0; i < m; i++)
    {
        int u, v;
        scanf("%d%d", &u, &v);
        pic[u].push_back(prdd(v, i));
        pic[v].push_back(prdd(u, i));
    }
    bfs(1);
    for (int i = 2; i <= n; i++)
    {
        sort(frm[i].begin(), frm[i].end());
```

```
        sgans.push_back(prdd(frm[i][0], i));
    }
    for (int i = 2; i <= n; i++)
    {
        ans *= frm[i].size();
        if (ans > k) break;
    }
    if (ans < k) k = ans;
    printf("%d\n", k);
    do
    {
        sort(sgans.begin(), sgans.end());
        optans();
        k--;
        if (k > 0)
            for (int loc = n - 2; loc >= 0; loc--)
            {
                int pt = sgans[loc].second;
                if (usd[pt] < frm[pt].size() - 1)
                {
                    for (int i = loc + 1; i < n - 1; i++)
                    {
                        int ipt = sgans[i].second;
                        usd[ipt] = ub(ipt, sgans[loc].first);
                        sgans[i].first = frm[ipt][usd[ipt]];
                    }
                    sgans[loc].first = frm[pt][++usd[pt]];
                    break;
                }
            }
        else
            break;
    } while (true);
    return 0;
}
```

Day 9

主题：数论

时间：2019 年 7 月 4 日 星期四

主要内容：

整除：最大公约数，最小公倍数，欧几里得算法，算术基本定理

质数：Eratosthenes 筛，线性筛

同余：二元线性模方程，扩展欧几里得算法

同余方程组：中国剩余定理，扩展中国剩余定理

欧拉函数：欧拉筛

同余系与逆元

组合数取模：杨辉三角，Lucas 算法，扩展 Lucas 算法

题目：

+2	A. jwp 来发糖果了	最小公倍数，欧几里得算法
+6	B. 脸盲的 zzy 和 jwp	扩展欧几里得
+4	C. zzy 数糖果	扩展中国剩余定理 exCRT
+	D. zzy 与飞行棋	欧拉函数，欧拉筛
+	E. 谁来救救 jwp	扩展 Lucas
–	F. jwp 的游戏策划	最短路

整除的性质

- 性质 1: $a|b, b|c \Rightarrow a|c$
- 性质 2: $a|b \Rightarrow a|bc$
- 性质 3: $a|b, a|c \Rightarrow \forall x, y, a|xb + yc$
- 性质 4: $a|b, b|a \Leftrightarrow a = \pm b$
- 性质 5: $a = kb \pm c \Leftrightarrow a, b$ 的公因数与 b, c 的公因数完全相同（利用性质 3 证明）

最大公约数、最小公倍数

- 定义: gcd(a,b)，(a,b) 表示 a,b 的最大公约数；lcm(a,b)，[a,b] 表示 a,b 的最小公倍数
- 基本结论
  (1) (a,b)=(-a,b) ； [a,b]=[-a,b]；
  (2) 对任意整数 k，(a,b)=(a,b+ka) ；
  (3) 设 m>0，则 (am,bm)=(a,b)m，[am,bm]=[a,b]m；
  (4) (a,(b,c))=(a,b,c)；
  (5) 若 c|a，c|b，则 c | (a,b) ；
  (6) 若 a|c，b|c，则 [a,b]|c；
  (7) |ab|=(a,b)[a,b]。
  (8) 设 a,b,m 是整数，(a,m)=1，则 (a,mb)=(a,b)。进一步，若 a|mb，则 a|b
- 辗转相除法原理:
  若 $a \equiv r \pmod b$, 则 $gcd(a, b) = gcd(b, r)$（利用性质 5 证明）

欧几里得算法

- 原理:
  若 $a \equiv r \pmod b$, 则 $gcd(a, b) = gcd(b, r)$
-
```
int gcd(int a, int b)
{
    return b ? gcd(b, a % b) : a;
}
```
- 算法的时间复杂度: $O(log(min(a, b)))$
- 举个栗子 gcd(14,36)
- 有可能爆 int

最小公倍数 LCM

求出 gcd 就可以了
lcm(a,b) = a*b / gcd(a,b)
计算的时候最好写成 lcm(a,b) = a / gcd(a,b) * b
为什么?

算术基本定理

素数和合数的概念

算术基本定理

任何一个大于 *1* 的自然数 n，都可以唯一分解成有限个质数的乘积。

$$n = p_1^{r_1} p_2^{r_2} ... p_k^{r_k}$$

其中 $p_1 < p_2 < ... < p_k$ 为质数，$r_1, r_2, ..., r_k$ 为正整数。

一些性质

定理

设 a 为合数，则 a 必有不超过根号 a 的素因子。

定理

若 $n = p_1^{r_1} p_2^{r_2} ... p_k^{r_k}$，则 n 的约数个数为

$$(1 + r_1)(1 + r_2)...(1 + r_k)$$

n 的所有约数和为

$$(1+p_1+p_1^2+...+p_1^{r_1})(1+p_2+p_2^2+...+p_2^{r_2})...(1+p_k+p_k^2+...+p_k^{r_k})$$

其中 $p_1 < p_2 < ... < p_k$ 为质数，$r_1, r_2, ..., r_k$ 为正整数。

POJ 1845 Sumdiv

计算 A^B 的所有约数的和模 9901
$0 \le A, B \le 50000000$
思路：求等比数列的和

- 方法一：二分求等比数列的和
- 方法二：等比数列求和公式 + 逆元

Eratosthenes 筛法

求 1 到 n 中的所有素数。
每次取出第一个没被筛掉的数 p，则为素数。然后将 p 的倍数筛掉。

```
bool vis[MAXN];
int prime[MAXN];
for (int i=2;i<=n;i++)
{
    if (!vis[i])
        prime[tot++] = i;
    for (int j=i*2;j<=n;j+=i)
        vis[j] = 1;
}
```

时间复杂度: $n/1 + n/2 + ... + n/p = O(nlogn)$ 实际上是 $O(nloglogn)$

线性筛

一个元素会被多次筛掉。让每个元素只被它最小的质因子筛掉，可以优化成线性。

```
for (int i=2;i<=n;i++)
{
    if (!vis[i])
        prime[tot++] = i;
    for (int j=0;j<tot && i * prime[j] <= n;j++)
    {
        vis[i * prime[j]] = 1;
        if (i % prime[j] == 0) break;
        //保证只被最小的质因子筛掉
    }
}
```


同余的定义和性质

定义

设 m 是正整数，对整数 a,b，若 $m|a-b$，则称 a 与 b 模 m 同余，记做 $a \equiv b \pmod m$ 定义 $a \bmod m$ 为 0 到 $m-1$ 中和 a 同余的整数。

可以写成 $a = b + km$

性质：

若 $a \equiv b \pmod m$, $c \equiv d \pmod m$，则

- 性质 1: $a + c \equiv b + d \pmod m$
- 性质 2: $a - c \equiv b - d \pmod m$
- 性质 3: $a * c \equiv b * d \pmod m$

BZOJ 1257

- 给定正整数 n 和 k，计算 $(k \bmod 1) + (k \bmod 2) + ... + (k \bmod n)$ 的值，$1 \le n, k \le 10^9$
- 模数不同，考虑把求余运算变成乘法和减法。
  $k \bmod i = k - \lfloor k/i \rfloor * i$，原式为 $n * k - \sum_{i=1}^{n} \lfloor k/i \rfloor * i$
- $\lfloor k/i \rfloor$ 在 $i \in [x, \lfloor k/\lfloor k/x \rfloor \rfloor]$ 内相同，在这一段中，计算一个等差数列的值
- 在 $i \in [1, k]$ 中，$\lfloor k/i \rfloor$ 最多只有 $2\sqrt{k}$ 个不同的值，时间复杂度 $O(\sqrt{k})$

解二元线性模方程

定义

二元线性模方程（二元一次不定方程）：

求解形如 $ax \equiv c \pmod b$ 或 $ax + by = c$ 的整数解

- 扩展欧几里得算法与裴蜀定理

裴蜀定理

设整数 a,b 不为 0，则方程组 $ax + by = c$ 有解当且仅当 $(a, b)|c$。

- 裴蜀定理特例
  若 a,b 互质，gcd(a,b)=1, 则存在 x,y 使得 $ax + by = 1$

因此求解 $ax + by = c$ 可以化为求解 $ax' + by' = gcd(a, b)$

$ax'/gcd(a, b) * c + by'/gcd(a, b) * c = c$

原方程的解为 $x = x'/gcd(a, b) * c, y = y'/gcd(a, b) * c$

推导

令 d=gcd(a,b)

$b*x+(a\%b)*y = d \Rightarrow b*x+(a-\lfloor a/b \rfloor *b)*y = a*y+b*(x-\lfloor a/b \rfloor *y)$

因此若 $b * x + (a\%b) * y = d$ 有解 x_0, y_0，那么 $a * x + b * y = d$ 有解 $x_1 = y_0, y_1 = x_0 - \lfloor a/b \rfloor * y_0$

可以迭代求出解

扩展欧几里得算法

求 $as + bt = gcd(a, b)$ 的解，返回 gcd(a,b)

```
int exgcd(int a, int b, int & x, int & y)
{
    if (!b) {x = 1;y = 0;return a;}
    int d = exgcd(b, a % b, y, x);
    y = y - a / b * x;//注意这里容易溢出
    return d;
}
```

时间复杂度与欧几里得算法的时间复杂度相同。

通解

定理

设整数 a,b 不为 0，方程组 $ax + by = c$ 有解 x,y，则 $x + \frac{b}{(a,b)}$ 和 $y - \frac{a}{(a,b)}$ 也是一组解，且所有解都可以写成 $x + \frac{kb}{(a,b)}$ 和 $y - \frac{ka}{(a,b)}$，其中 k 是任意整数。

若要求正整数解，用该定理变换。

定理

设 a,b 为正整数，则扩展欧几里得算法求出的解满足

$$|x| <= b, |y| <= a$$

证明.

由 $x_{k+1} = y_k + (a_k/b_k) * x_k, y_{k+1} = x_k$

得 $y_k = x_{k+1} - (a_k/b_k) * y_{k+1}$

下证任意时刻 $|x_k| \le b_k, |y_k| \le a_k$

基础：$(a_0, 0)$ 时解为 $x_0 = 1, y_0 = 0$

归纳：(a_k, b_k) 时解为 (x_k, y_k)

$(ta_k + b_k, a_k)$ 时解为 $(y_k, x_k - ty_k)$ □

例

判断 $ax+by+cz=n$ 是否存在非负整数解。
数据范围：$0 \le a,b,c < 2*10^5, n \le 10^{18}$
Input:
1 2 3 6
3 5 6 4
Output:
YES
NO
思路：若有解，则必有 $x < b$ 的解。枚举 x，用扩展欧几里得求 y,z，时间复杂度 $O(blogn)$

解线性同余方程组

定义

线性同余方程组

$$x \equiv a_1 \pmod{m_1}$$
$$\dots$$
$$x \equiv a_n \pmod{m_n}$$

中国剩余定理 CRT

定理

当 $m_1,...,m_n$ 两两互质时，上述线性同余方程在 $[0, m_1m_2...m_n]$ 上有唯一整数解。
设同余方程组有特解 x，则所有解可表示为 $x+km_1m_2...m_n$，其中 k 是任意整数。

求解过程

令 $M=m_1m_2...m_n, M_j=\frac{M}{m_j}, M_jy_j \equiv 1 \pmod{m_j}, j=1,2,...,n$
因为 $gcd(M_j,m_j)=1$，所以存在 M_j^{-1}，使 $M_j^{-1}M_j \equiv 1 \pmod{m_j}$，且存在 h 和 k 两个整数，使得 $hM_j+km_j=1$，$hM_j \equiv 1 \pmod{m_j}$，所以

$$y_j \equiv M_j^{-1} \pmod{m_j}$$

令 $x=M_1y_1a_1+M_2y_2a_2+...+M_ny_na_n$，易验证满足上述同余方程组。则 x 是一个解。
最小非负整数解为 $(x \bmod M+M) \bmod M$.
求解 y_j 可以用扩展欧几里得实现。
时间复杂度 $O(logm_1+...+logm_n)$

扩展中国剩余定理

当 $m_1,...,m_n$ 不两两互质时，方程组不一定有解，考虑代入法解同余方程组。
基础：a_1 是 $x \equiv a_1 \pmod{m_1}$ 的一个解
归纳：

- 设前 k-1 个方程组的一个解为 x，记 $m=lcm(m_1,m_2,...,m_{k-1})$，则 $x+i*m$ 是前 k-1 个方程组的通解。
- 对第 k 个方程，求出一个 t 使得

$$x+t*m \equiv a_k \pmod{m_k}$$

- 用扩展欧几里得解 t，并可以判断有没有解。

为什么模数变成了 lcm?

代入过程

假设要合并 $x \equiv a_1 \pmod{m_1}$
$x \equiv a_2 \pmod{m_2}$
就需要求出一个 x 满足 $x=a_1+m_1*k_1=a_2+m_2*k_2$

$$m_1*k_1-m_2*k_2=a_2-a_1$$

可由扩展欧几里得算法求出一组解 x_1,y_1，满足 $m_1*x_1+m_2*y_1=gcd(m_1,m_2)$ 在 $gcd(m_1,m_2)|(a_2-a_1)$ 时，有 $m_1*x_1*\frac{a_2-a_1}{gcd(m_1,m_2)}+m_2*x_2*\frac{a_2-a_1}{gcd(m_1,m_2)}=a_2-a_1$ （否则无解）
则 $k_1^*=x_1*\frac{a_2-a_1}{gcd(m_1,m_2)}$
通解 $k_1=k_1^*+m_2/gcd(m_1,m_2)*T$ 可以找到最小的正整数解 k_1，并代入 $x=a_1+k_1*m_1$ 得到

$$x=a_1+(k_1^*+T*m_2/gcd(m_1,m_2))*m_1$$

所以合并后的方程变成 $x \equiv A \pmod{lcm(m_1,m_2)}$

快速乘

为了解决上面过程中乘法爆 long long 的问题

```
typedef long long LL;
LL qmul(LL n, LL b, LL P) // a * b % P
{
    LL ans = 0;
    while (b)
    {
        if (b & 1) ans = (ans + a) % P;
        b >>= 1;
        a = (a + a) % P;
    }
    return ans;
}
```


hdu 1573 X 问题

题目描述：
求在小于等于 N 的正整数中有多少个 X 满足：X mod a[0] = b[0], X mod a[1] = b[1], X mod a[2] = b[2], …, X mod a[i] = b[i], …(0 < a[i] <= 10)。
输入：N,M，数组 a,b
例子：
Input
10 3
1 2 3
0 1 2
Output
1

hdu1573 X 问题 Solution

题目没有保证 a[i] 互质，用扩展中国剩余定理。

定理

若 $ac \equiv bc \pmod m$，则 $a \equiv b \pmod{\frac{m}{gcd(m,c)}}$。进一步，若 $gcd(c, m) = 1$，则 $a \equiv b \pmod m$

定理

若 $a \equiv b \pmod{m_i}$，$i = 1, 2, ..., n$，则 $a \equiv b \pmod{lcm(m_1, m_2, ..., m_n)}$

欧拉函数

定义

欧拉函数 $\varphi(n)$ 为 1 到 n 中与 n 互质的数的个数。

性质：

- 积性：(m,n)=1 时，$\varphi(mn) = \varphi(m)\varphi(n)$
- 设 p 是素数，$\varphi(p) = p - 1$
- 设 p 是素数，$k > 1$，则 $\varphi(p^k) = p\varphi(p^{k-1})$
- 设 p 是素数，且 p | n, 则 $\varphi(pn) = p\varphi(n)$

欧拉函数计算公式：$\varphi(n) = n(1 - \frac{1}{p_1})(1 - \frac{1}{p_2})...(1 - \frac{1}{p_k})$（容斥原理）

欧拉公式：设 a 与 m 互质，则

$$a^{\varphi(m)} \equiv 1 \pmod m$$

欧拉函数的性质

- 设 $(a, m) = 1$，则 $a^n \equiv a^{n \bmod \varphi(m)} \pmod m$
- 设 n 是满足 $a^n \equiv 1 \pmod m$ 的最小正整数，则有 $n|\varphi(m)$。n 记做 a 关于模 m 的阶。进一步，若 $a^x \equiv 1 \pmod m$，$a^y \equiv 1 \pmod m$，则 $a^{(x,y)} \equiv 1 \pmod m$。
- 设 $n > 1$，则所有小于 n 且与 n 互质的数的和为 $n\varphi(n)/2$。

例

求第 k 大的 $\varphi(n)$ 为合数的数 n.

$\varphi(1) = 1$
$\varphi(2) = 1$
$\varphi(3) = 2$
$\varphi(4) = 2$
$\varphi(5) = 4$ ✓
$\varphi(6) = 2$
$\varphi(7) = 6$ ✓
...

$$\varphi(n) = n(1 - \frac{1}{p_1})(1 - \frac{1}{p_2})...(1 - \frac{1}{p_k})$$

k=1 时，答案是 5; k>1 时，答案是 k+5

事实上，

定理

若 $n > 2$，$\varphi(n)$ 必定是偶数。

求欧拉函数

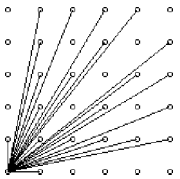
- 求 1 到 n 的所有 $\varphi(i)$：采用欧拉筛的思想，时间复杂度 $O(n)$。
- 求 $\varphi(n)$：用分解质因数的方法，时间复杂度 $O(\sqrt{n})$

筛法求欧拉函数

```
void Euler()
{
    phi[1] = 1;
    for (int i=2;i<=n;i++)
    {
        if (!vis[i])
        {
            prime[tot++] = i;
            phi[i] = i-1;
        }
        for (int j=0;j<tot && i * primc[j] <= n;j++)
        {
            vis[i*prime[j]] = 1;
            if (i % prime[j] == 0)
            {
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            }
            else phi[i * prime[j]] = phi[i] * (prime[j]-1);
        }
    }
}
```


洛谷 2158 仪仗队

作为体育委员，C 君负责这次运动会仪仗队的训练。仪仗队是由学生组成的 N * N 的方阵，为了保证队伍在行进中整齐划一，C 君会跟在仪仗队的左后方，根据其视线所及的学生人数来判断队伍是否整齐 (如下图)。现在，C 君希望你告诉他队伍整齐时能看到的学生人数。



洛谷 2158 仪仗队题解

- 能看见的点关于 $y = x$ 对称
- $ANS(1) = 0$
- $ANS(n) = \sum_{x=0}^{n-1}\sum_{y=0}^{n-1}[gcd(x,y) == 1], n \geq 2$
- $= 2\sum_{x=1}^{n-1}\varphi(x) + 1$

欧拉定理推广

欧拉定理
若 a 和 n 互质，则 $a^{\varphi(m)} \equiv 1 \pmod m$

费马小定理
设 p 是素数，$(a,p) = 1$，则 $a^{p-1} \equiv 1 \pmod p$

推广 (降幂公式)
当 $x \geq \varphi(m)$ 时，$a^x \equiv a^{x \bmod \varphi(m)+\varphi(m)} \pmod m$

例

计算

$$\underbrace{a^{a^{a^{\cdots}}}}_{k个a} \bmod 100000000$$

$1 \leq k \leq 200, 1 \leq a \leq 10^{18}$
思路：按照题意递归计算
$a^x \equiv a^{x \bmod \varphi(n)+\varphi(n)} \pmod n$
当指数比欧拉函数大的时候，根据降幂公式可以取模
C = 1e8，预处理出需要用到的欧拉函数：$C, \varphi(C), \varphi(\varphi(C)), \ldots$ (200 个)

同余类与剩余系

定义
对于 $\forall a \in [0, m-1]$，集合 $\{a + km\}(k \in \mathbb{Z})$ 的所有数模 m 同余，余数都是 a。该集合成为一个模 m 的同余类，记为 $\bar{a}$. 模 m 的同余类有 m 个，分别为 $\bar{0}, \bar{1}, \ldots, \overline{m-1}$，它们构成 m 的完全剩余系。$1 \sim m$ 中与 m 互质的数代表的同余类有 $\varphi(m)$ 个，它们构成 m 的简化剩余系。

群

群 $(S, +)$ 是一个集合 S 和定义在 S 上的二元运算 $+$，它满足如下性质：

- 封闭性：如果 $a, b \in S$，那么 $a + b \in S$
- 单位元：存在一个元素 e，使得对于所有 $a \in S$ 都满足 $e + a = a + e = a$
- 结合律：对于任意 a,b,c 都满足 (a+b)+c=a+(b+c)
- 逆元：对每个 $a \in S$ 都存在唯一的元素 $b \in S$ 使得 a+b=b+a=e。把 b 称作 a 的逆元。

根据模加法和模乘法定义的群：

- 定义在集合 $Z_n = \{0, 1, \ldots, n-1\}$ 上
- 集合上的加法和乘法运算定义为：$[a]_n +_n [b]_n = [a + b]_n$ $[a]_n *_n [b]_n = [a * b]_n$

逆元

定义
设 m,a 为正整数，若存在正整数 a'，满足 $aa' \equiv 1 \pmod m$，则称 a' 是 a 关于模 m 的逆元，记为 a^{-1}.

一些定理：

1. a 的逆元存在当且仅当 $(a, m) = 1$；且逆元唯一。
2. 同余方程 $ax \equiv b \pmod m$ 当 $(a, m) = 1$ 时在区间 $0 \leq x < m$ 上有唯一解 $a^{-1}b$
3. 同余方程 $ax \equiv b \pmod m$ 有解当且仅当 $(a, m)|b$，且在区间 $0 \leq x < m/(a, m)$ 上有唯一解，在区间 $0 \leq x < m$ 上有 (a, m) 个解。

定理
同余方程 $ax \equiv b \pmod m$ 当 $(a, m) = 1$ 时在区间 $0 \leq x < m$ 上有唯一解 $a^{-1}b$

证明.
存在性：由裴蜀定理可知，存在 x,y 满足 $ax + my = (a, m) = 1$，则存在整数 b 使得 $x' = bx, y' = by$ 满足 $ax' + my' = b$，即 $ax \equiv b \pmod m$ 有解 bx。通解为 $bx + km$，则必在 $[0, m-1]$ 上有一个解。
唯一性：假设在 $[0, m-1]$ 上有两个解 x_1, x_2 满足 $x_1 \geq x_2$，则 $ax_1 \equiv b \pmod m$，$ax_2 \equiv b \pmod m$。则 $a(x_1 - x_2) \equiv 0 \pmod m$，且 $x_1 - x_2 \in [0, m-1]$，所以 $x_1 = x_2$ □

定理
同余方程 $ax \equiv b \pmod m$ 有解当且仅当 $(a, m)|b$，且在区间 $0 \leq x < m/(a, m)$ 上有唯一解，在区间 $0 \leq x < m$ 上有 (a, m) 个解。

证明.
令 $d = gcd(a, m)$，$ax \equiv b \pmod m$，则 $ax + my = b$

$$\frac{a}{d}x + \frac{m}{d}y = \frac{b}{d}$$

$gcd(\frac{a}{d}, \frac{m}{d}) = 1$，由定理 2，该方程在 $[0, \frac{m}{d} - 1]$ 上有唯一解。
若有特解 x，则所有解为 $\{x + k\frac{m}{d}, k \in \mathbb{Z}\}$，在模 m 的完全剩余系 $\{\bar{0}, \bar{1}, \ldots, \overline{m-1}\}$ 中，恰有 (a,m) 个解。 □
由此可以解释关于拓展欧几里得算法通解的定理。

求逆元

- 方法一：费马小定理
  设 p 是素数，且 gcd(a,p)=1，则 $a^{p-1} \equiv 1 \pmod p$
  当 m 为素数时，a^{m-2} 是 a 关于模 m 的逆元。（快速幂）
  当 m 不是素数时，$a^{\varphi(m)-1}$ 是 a 关于模 m 的逆元。
- 方法二：扩展欧几里得
  对 m 是否为素数没有限制。
  使用扩展欧几里得求 $ax \equiv 1 \pmod m$
  （化为 $ax + my = 1$）
- 两种方法的时间复杂度均为 $O(logm)$

线性时间求 1 到 n 中所有数模素数 m 的逆元

```
typedef long long LL;
int inv[N], m;
void get_inv(int n)
{
    inv[1] = 1;
    for (int i=2;i<=n;i++)
        inv[i] = (LL)(m-m/i) * inv[m % i] % m;
}
```

证明：设 $m = ki + b$，则 $ki \equiv -b \pmod m$，
$i^{-1} \equiv -b^{-1}k \equiv -b^{-1}(m-k) \pmod m$
即 $inv[i] = (LL)(m - m/i) * inv[m\%i]\%m$;

阶乘的逆元

求 1 到 n 的阶乘的逆元

```
fac[0] = 1;
for (int i=1;i<=n;i++)
    fac[i] = fac[i-1] * i % P;
invfac[n] = qpow(fac[n], P-2); //快速幂
for (int i=n;i>=1;i--)
    invfac[i-1] = invfac[i] * i % P;
```

可以用来求单个排列组合数取模

组合数取模

- 求 $C_n^m \bmod p$
- 根据 n,m,p 的范围和约束条件不同，求解方法不同。

n,m 较小，p 较大

$1 \le m \le n \le 1000, 1 \le p \le 10^9$

$$C_n^m = C_{n-1}^{m-1} + C_{n-1}^m$$

直接利用杨辉三角，递推计算组合数，时间复杂度 $O(n^2)$

n,m 较大，p 较小且 p 为素数

$1 \le m \le n \le 10^{18}, 2 \le p \le 10^5$ 且 p 是素数
直接计算显然不可能

Lucas 定理

如果 p 是素数，对整数 $1 \le m \le n$ 有，

$$C_n^m \equiv C_{n \bmod p}^{m \bmod p} * C_{n/p}^{m/p} \pmod p$$

也就是将 n 和 m 分别表示成 p 进制数，对 p 进制下的每一位计算组合数。
对 $C_{n \bmod p}^{m \bmod p}$ 的计算，用组合数阶乘的公式和逆元。
递归计算，出口为 m=0
时间复杂度 $O(plog_p^n)$

n,m 较大，p 较小但 p 可能为合数

扩展 Lucas

思路：考虑将 p 分解，得到同余方程组，再用中国剩余定理合并。

- 令 $p = p_1^{r_1} p_2^{r_2} ... p_k^{r_k}$，需要求出 $C_n^m \equiv c_i \pmod{p_i^{r_i}}, i=1,...k$
- $$C_n^m \equiv \frac{n!}{m!(n-m)!} \equiv \frac{\frac{n!}{p^{a_1}}}{\frac{m!}{p^{a_2}} * \frac{(n-m)!}{p^{a_3}}} * p^{a_1-a_2-a_3} \pmod{p^r}$$

其中 a_1, a_2, a_3 分别是 $n!$，$m!$, $(n-m)!$ 中 p 因子的幂次
不能直接求逆元，因为分母与模数不互质，只要把公因子提出来就好了，提出来之后就可以直接求逆元

扩展 Lucas 续

- 考虑计算
  $$\frac{n!}{p^a} \bmod p^k$$

其中 a 是 n! 中 p 的幂次
先看 $n! \bmod p^k$，
比如 n=19, p=3, k=2 时
$19! = 1*2*3*...*19 = (1*2*4*5*7*8*10*11*13*14*16*17*19)*3^6*6! = (1*2*4*5*7*8)^2*19*3^6*6!$
因为 $1*2*4*5*7*8 \equiv 10*11*13*14*16*17 \bmod 3^2$
即 $\prod_{i,(i,p)=1}^{p^k} i \equiv \prod_{i,(i,p)=1}^{p^k} (i + t*p^k) \pmod{p^k}$

$$n! \equiv p^{[\frac{n}{p}]} * [\frac{n}{p}]! * (\prod_{i,(i,p)=1}^{p^k} i)^{\frac{n}{p^k}} * (\prod_{i,(i,p)=1}^{n \bmod p^k} i) \pmod{p^k}$$

A. jwp 来发糖果了

内存限制：512 MiB

时间限制：1000 ms

题目描述

穷的叮当响的 jwp 去幼稚园做兼职，他决定每天订一些糖果发给小朋友们吃(虽然他已经很穷了，但他还是很想让小朋友开心)，他要做 k 天的老师，每天的学生数量是不一定的，但是为了方便，他每天订购的糖果数量是一定的，为了激励小朋友们的积极性，他决定每天表现最不好的 a 个小朋友分到的糖果要比其他人少，简单的来说，如果第 i 天有 A_i 个小朋友，他订购的糖果数量为 x，则 $x \bmod A_i = A_i - a$，(当然也有可能这 a 个小朋友分不到任何糖果)，jwp 想让你帮他求出他最少需要准备的糖果数量 x。

输入格式

第一行一个正整数 t，表示数据组数。

每一组数据第一行两个整数，分别为 k, a。

下面一行 k 个整数，表示每天来的小朋友的数量 A_i。

输出格式

输出 t 行，每一行一个整数，表示最小糖果数量 x。

数据范围与提示

$1 \le t \le 10^4$

$2 \le k \le 10$

$1 \le a < A_i \le 100$

解题思路

本题乍一看需要使用扩展中国剩余定理，但仔细分析发现其实没有必要。x 要满足的条件为 $x \equiv -a \pmod{A_i}$，变形得到 $x + a \equiv 0 \pmod{A_i}$，即 $x + a = \operatorname{lcm}(A_i)$。求出最小公倍数即可。

答案可能很大，故应使用 ullong 型存储答案。

代码

```
int t;
ullong gcd(ullong a, ullong b)
{
    return b ? gcd(b, a % b) : a;
}
int main()
{
    scanf("%d", &t);
    for (int i = 0; i < t; i++)
    {
        int k, a;
        ullong lcm = 1;
        scanf("%d%d", &k, &a);
```

```
        for (int j = 0; j < k; j++)
        {
            int aj;
            scanf("%d", &aj);
            lcm = (lcm / gcd(lcm, aj) * aj);
        }
        printf("%llu\n", lcm - a);
    }
    return 0;
}
```

B. 脸盲的 zzy 和 jwp

内存限制：512 MiB

时间限制：1000 ms

题目描述

一天，zzy 和 jwp 都在长度为 l 的环形的校园里跑步，他们朝着相同的方向跑，zzy 一开始在距离起点 x 米的地方，她每分钟能跑 a 米，jwp 一开始在距离起点 y 米的地方，他每分钟能跑 b 米，但是由于他们两个人都出奇的脸盲，只有当某一分钟结束时他们刚好跑到同一个位置，他们两个才能认出对方。请你帮忙计算，他们至少需要跑多久才能发现对方今天恰好也在跑步。

输入格式

五个整数，分别为 x,a,y,b,l。

输出格式

一个整数（如果无解的话，输出 -1）。

数据范围与提示

$1 \le x,y,a,b \le 10^8$

$1 \le L \le 10^9$

保证 x 不等于 y。

解题思路

设跑了 t 分钟后相遇，则满足 $x+at \equiv y+bt \pmod l$。不妨设 $a>b$，应用扩展欧几里得解方程 $(a-b)t \equiv y-x \pmod l$，并变换得到正整数解即可。

代码

```
llong x, a, y, b, l, dv, dx, ans, tp;
llong gcd(llong a, llong b)
{
    return b ? gcd(b, a % b) : a;
}
llong exgcd(llong a, llong b, llong &x, llong &y)
{
    if (!b) { x = 1; y = 0; return a; }
```

```
    llong d = exgcd(b, a % b, y, x);
    y = y - a / b * x;
    return d;
}
int main()
{
    scanf("%lld%lld%lld%lld%lld", &x, &a, &y, &b, &l);
    dv = a - b;
dx = y - x;
    if (dv < 0)
    {
        dv = -dv;
        dx = -dx;
    }
    llong gcds = exgcd(dv, l, ans, tp);
    if ((dx % gcds + gcds) % gcds != 0)
        goto badend;
    ans = dx / gcds * ans;
    ans = (ans % (l / gcds) + l / gcds) % (l / gcds);
    printf("%lld", ans);
    return 0;
badend:
    printf("-1");
    return 0;
}
```

C. zzy 数糖果

内存限制：512 MiB

时间限制：1000 ms

题目描述

一天，zzy 从 jwp 那里抢走了一大袋糖果，她想数清楚这些糖果到底有多少，无奈这些糖果真的是太多太多了（可怜的 jwp┬┬﹏┬┬），由于担心数不清楚，她决定每次数出 n 个来为一组，最后会剩下 a 个，由于组数太多了她根本记不住，只能记下来 n 和 a，她一共数了 m 次，记录下来了每一次的 n_i, a_i，现在她想知道根据自己记录的这些数据，能不能算出一共掠夺了 jwp 多少糖果。

输入格式

第一行一个正整数 T，表示数据组数。

每一组数据，第一行一个正整数 m。

下面 m 行，每行两个整数，分别为 n_i, a_i。

输出格式

T 行，每行一个数表示糖果数量，如果有多个解，输出最小的非负整数解，如果无解，输出 -1。

数据范围与提示

$1 \leq T \leq 200$

$1 \leq m \leq 8$

$1 \leq a_i < n_i \leq 50$

解题思路

扩展中国剩余定理 exCRT。

代码

```
int t;
llong gcd(llong a, llong b)
{
    return b ? gcd(b, a % b) : a;
}
llong exgcd(llong a, llong b, llong &x, llong &y)
{
    if (!b) { x = 1; y = 0; return a; }
    llong d = exgcd(b, a % b, y, x);
    y = y - a / b * x;
    return d;
}
inline llong ngmod(llong a, llong p)
{
    return ((a % p) + p) % p;
}
int main()
{
    scanf("%d", &t);
    for (int i = 0; i < t; i++)
    {
        llong tm, m, a, lcm = 1, x;
        scanf("%lld", &tm);
        for (int j = 0; j < tm; j++)
        {
            scanf("%lld%lld", &m, &a);
            if (j == 0)
            {
                x = a;
                lcm = m;
            }
            else
            {
                llong nt, tp, rgcd = exgcd(lcm, m, nt, tp);
                if (ngmod(a - x, m) % rgcd != 0)
                {
```

```
                    x = -1;
                    for (int jj = j + 1; jj < tm; jj++)
                        scanf("%lld%lld", &m, &a);
                    break;
                }
                nt = ngmod(a - x, m) / rgcd * nt;
                x += nt * lcm;
                lcm = lcm / rgcd * m;
                x = ngmod(x, lcm);
            }
        }
        printf("%lld\n", x);
    }
    return 0;
}
```

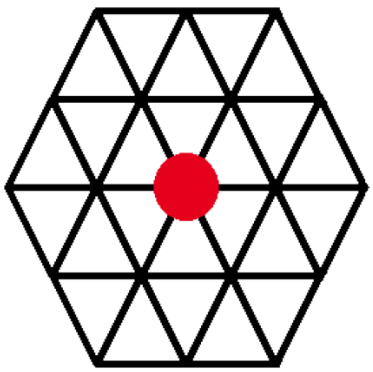
D. zzy 与飞行棋

内存限制：512 MiB

时间限制：1000 ms

题目描述

有一个边长为 n 的正六边形的飞行棋棋盘（如图），除中心点外，每个点都有一个飞行棋。zzy 也被变成了一枚飞行棋棋子并被放在了正六边形的中心。zzy 想知道自己能够看到周围的几个棋子？(内层的棋子会挡住共线的外层棋子，外层棋子不能被看到。)



输入格式

一个整数，n。

输出格式

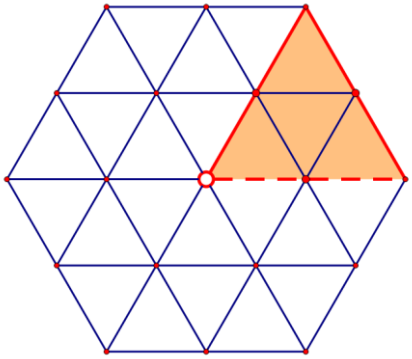
一个数，即 zzy 能看到的棋子个数。

数据范围与提示

$1 \le n \le 4 \cdot 10^5$

解题思路

由于棋盘有对称性，故只考虑下图阴影部分所示的 1/6 个棋盘中能看到的点即可。



设水平向右为 x 轴，左上方向为 y 轴，观察者所在位置为原点，边长为 1，这样每一个点都有一个坐标 (x,y)，且满足 $0<y\le x\le n$。对于所有的 $x=x_0$ 的点，其能被观测到的条件为 $\gcd(x_0,y)=1$，这样的点有 $\varphi(x_0)$ 个。所以在这个区域中，共有 $\sum_{x=1}^{n}\varphi(x)$ 个点能被观测到。使用欧拉筛求出 $\varphi(1)$ 至 $\varphi(n)$ 即可。整个棋盘共有 $6\sum_{x=1}^{n}\varphi(x)$ 个点能被观测，即为答案。

欧拉筛的原理：p 为质数，则满足：

1. $\varphi(p)=p-1$。
2. 若 $i\equiv 0\pmod p$，则 $\varphi(ip)=\varphi(i)\cdot p$。
3. 若 $i\not\equiv 0\pmod p$，则 $\varphi(ip)=\varphi(i)\cdot(p-1)$。

欧拉筛的时间复杂度为 $O(n)$。

代码

```
int phi[525000];
vector<int> ps;
int n;
llong ans = 0;
void Euler(int rge)
{
    phi[1] = 1;
    for (int i = 2; i <= rge; i++)
    {
        if (phi[i] == 0)
        {
            phi[i] = i - 1;
            ps.push_back(i);
        }
        for (int j = 0; j < ps.size() && ps.at(j) <= rge / i; j++)
        {
            if (i % ps.at(j) == 0)
            {
                phi[i * ps.at(j)] = phi[i] * ps.at(j);
                break;
            }
            else
                phi[i * ps.at(j)] = phi[i] * (ps.at(j) - 1);
```

```
        }
    }
}
int main()
{
    scanf("%d", &n);
    Euler(n);
    for (int i = 1; i <= n; i++)
        ans += phi[i];
    ans *= 6;
    printf("%lld", ans);
    return 0;
}
```

E. 谁来救救 jwp

内存限制：512 MiB

时间限制：1000 ms

题目描述

没有学过组合数学的 jwp 想自学一下，他随便拿出了一道题：有一个坐标网格，有一只青蛙，开始在 $(0,0)$ 点，它每次可以向右或者向上跳一步，问它跳到 (n,m) 点有多少种方案。由于方案可能很大很大，jwp 决定只求方案数对 p 取模之后的模数，p 可能为合数。然而他发现，这个问题他还不会做，为了不打消他学习组合数学的积极性，zzy 决定请你来帮他解决这个问题。

输入格式

第一行一个整数 T，表示数据组数。

下面 T 行，每行三个整数，n,m,p。

输出格式

T 行，每行一个整数。

数据范围与提示

$1 \le n,m \le 10^{18}$

$2 \le p \le 10^6$

解题思路

从 $(0,0)$ 点到 (n,m) 点需要向上跳 m 次，向右跳 m 次，总计跳 $(n+m)$ 次。方案数为 $(n+m)!/(n!\cdot m!) = C_{n+m}^{n}$。使用扩展 Lucas 进行计算即可。

代码

```
llong t, n, m, p;
llong exgcd(llong a, llong b, llong &x, llong &y)
{
    if (!b) { x = 1; y = 0; return a; }
    llong d = exgcd(b, a % b, y, x);
```

```
    y = y - a / b * x;
    return d;
}
llong qkpow(llong a, llong b, llong p)
{
    llong sum = 1;
    while (b)
    {
        if (b & 1)
            sum = (sum * a) % p;
        b >>= 1;
        a = (a * a) % p;
    }
    return sum;
}
llong fac(llong n, llong pi, llong pk)
{
    if (!n)
        return 1;
    llong res = 1;
    for (llong i = 2; i <= pk; ++i)
        if (i % pi)
            (res *= i) %= pk;
    res = qkpow(res, n / pk, pk);
    for (llong i = 2; i <= n%pk; ++i)
        if (i % pi)
            (res *= i) %= pk;
    return res * fac(n / pi, pi, pk) % pk;
}
llong inv(llong n, llong mod)
{
    llong x, y;
    exgcd(n, mod, x, y);
    return (x += mod) > mod ? x - mod : x;
}
llong CRT(llong b, llong mod, llong p)
{
    return b * inv(p / mod, mod) % p * (p / mod) % p;
}
llong C(llong n, llong m, llong pi, llong pk)
{
    llong up = fac(n, pi, pk),
        d1 = fac(m, pi, pk), d2 = fac(n - m, pi, pk);
    llong k = 0;
    for (llong i = n; i; i /= pi)
```

```
        k += i / pi;
    for (llong i = m; i; i /= pi)
        k -= i / pi;
    for (llong i = n - m; i; i /= pi)
        k -= i / pi;
    llong ans = (up * inv(d1, pk)) % pk;
    (ans *= inv(d2, pk)) %= pk;
    (ans *= qkpow(pi, k, pk)) %= pk;
    return ans;
}
llong exlucus(llong n, llong m, llong p)
{
    llong res = 0, tmp = p, pk;
    int lim = sqrt(p) + 5;
    for (int i = 2; i <= lim; ++i)
        if (tmp%i == 0)
        {
            pk = 1;
            while (tmp % i == 0)
                pk *= i, tmp /= i;
            (res += CRT(C(n, m, i, pk), pk, p)) %= p;
        }
    if (tmp > 1)
        (res += CRT(C(n, m, tmp, tmp), tmp, p)) %= p;
    return res;
}
int main()
{
    scanf("%lld", &t);
    for (int i = 0; i < t; i++)
    {
        scanf("%lld%lld%lld", &n, &m, &p);
        printf("%lld\n", exlucus(n + m, m, p));
    }
    return 0;
}
```

F. jwp 的游戏策划

内存限制：512 MiB

时间限制：1000 ms

题目描述

jwp 创造了一款新的游戏，这是一个世界观宏大的战争游戏，有 n 个角色职业，每个职业要占用 $cost_i$ 点的资源值，jwp 需要设定游戏的总资源值 all，使得至少有一组所有玩家选择职业的组合可以不加浪费的利用所有的资源值，即如果选择每个职业的玩家数量为 num_i，那么 $\prod_{i=1}^{n} cost_i \cdot num_i = a$，$num_i$ 至少有一组非负整数解，这里我们不虚要考虑玩家的总数量，

即玩家总数量可以是任意非负整数。由于技术手段的限制，all 有一个范围 $[all_l, all_r]$，现在 jwp 希望知道，在这个范围内有多少个可能的 all 值，使得他可以达成上述资源的不加浪费的分配。

输入格式

第一行三个整数 n, all_l, all_r。

下面一行 n 个整数 $cost_i$。

输出格式

一个整数，可能 all 值的数量

样例

样例输入

```
3 1 10
3 4 5
```

样例输出

```
8
```

数据范围与提示

$1 \le n \le 10$

$1 \le cost_i \le 10^6$

$1 \le all_l \le all_r \le 10^{12}$

Day 10

主题：杂题选讲

时间：2019 年 7 月 5 日 星期五

主要内容：

区间 dp

杂题选讲

题目：

+ A. JM 的浩然正气

区间dp

题目描述

在一个圆形操场的四周摆放N堆石子,现要将石子有次序地合并成一堆.规定每次只能选相邻的2堆合并成新的一堆,并将新的一堆的石子数,记为该次合并的得分。

试设计出1个算法,计算出将N堆石子合并成1堆的最小得分和最大得分.

输入输出格式

输入格式：

数据的第1行试正整数N,1≤N≤100,表示有N堆石子.第2行有N个数,分别表示每堆石子的个数.

输出格式：

输出共2行,第1行为最小得分,第2行为最大得分.

区间dp

- 贪心？

本题初看以为可以使用贪心法解决问题，但是事实上因为有必须相邻两堆才能合并这个条件在，用贪心法就无法保证每次都能找到所有堆中石子数最多的两堆。例如下面这个例子：

```
6
3 4 6 5 4 2
```

如果使用贪心法求最小得分，应该是如下的合并步骤：

```
第一次合并 3 4 6 5 4 2     2,3合并得分是5
第二次合并 5 4 6 5 4       5,4合并得分是9
第三次合并 9 6 5 4         5,4合并得分是9
第四次合并 9 6 9           9,6合并得分是15
第五次合并 15 9            15,9合并得分是24
总得分=5+9+9+15+24=62
```

但是如果采用如下合并方法，却可以得到比上面得分更少的方法：

```
第一次合并 3 4 6 5 4 2     3,4合并得分是7
第二次合并 7 6 5 4 2       7,6合并得分是13
第三次合并 13 5 4 2        4,2合并得分是6
第四次合并 13 5 6          5,6合并得分是11
第五次合并 13 11           13,11合并得分是24
总得分=7+13+6+11+24=61
```

区间dp

- dp
  - 用dp[i][j]表示区间i到j的花费最值
- 怎么递推？
  - 合并的是相邻的区间
  - 可以按照长度来枚举
  - dp[i][j]=min{dp[i][k]+dp[k+1][j]}+sum{a[i],a[j]}
    - ,i<=k<j
- 边界条件
  - dp[i][i]=i
- 复杂度
  - n^3

区间dp练习

- 给你一个由(和)组成的括号序列
- 问你最少添加几个括号，使得括号序列合法
- 合法就是能配对

区间dp练习

- 状态表示：dp[i][j]表示要将i,j内合法需要最少插入多少个括号.
- 边界情况：dp[i][i]=1
- 状态转移：dp[i][j]=min{dp[i][k]+dp[k+1][j]}
  if(check(i,j)) dp[i][j] = min(dp[i][j],dp[i+1][j-1])
- 按长度为第一顺序进行转移：类似于石子合并

区间dp练习

- dp[i][j]=min{dp[i][k]+dp[k+1][j]}
- if(check(i,j)) dp[i][j] = min(dp[i][j],dp[i+1][j-1])
- 为什么要check(i,j)?
- 比如对于一个序列(.....)
  - 左右括号既可能本身配对
  - 也可以分别配对

区间dp练习

```
memset( dp, MAX_INF, sizeof( dp ) );
for (int i = 1; i <= n; i++)
    dp[i][i] = 1;

for (int len = 2; len <= n; len++)
    for (int i = 1; i + len - 1 <= n; i++) {
        int j = i + len - 1;
        for (int k = i; k < j; ++k)
            dp[i][j] = min(dp[i][j], dp[i][k] + dp[k + 1][j]);
        if ((s[i] == '(' && s[j] == ')'))
            dp[i][j] = min(dp[i][j], j - i == 1 ? 0 : dp[i + 1][j - 1]);
    }
```

区间dp练习的思考

- 如果有多种括号(),[],{}，怎么做？
- 其实就是check的时候加一下就可以了

```
for (int len = 2; len <= n; len++)
    for (int i = 1; i + len - 1 <= n; i++) {
        int j = i + len - 1;
        for (int k = i; k < j; ++k)
            dp[i][j] = min(dp[i][j], dp[i][k] + dp[k + 1][j]);
        if ((s[i] == '(' && s[j] == ')'))
            dp[i][j] = min(dp[i][j], j - i == 1 ? 0 : dp[i + 1][j - 1]);
    }
```

计数问题

题目描述

给平面上的N个整点(x1,y1),(x2,y2),…,(xN,yN)，对每个点(xi,yi)求有多少个点(xj, yj), i≠j，满足xj≤xi且yj≤yi。

(1 ≤ N ≤ 5e5), (0 ≤ x, y ≤ 32000)

计数问题

解析

将点集以x坐标为第一关键字，y坐标为第二关键字排序，然后用树状数组维护y坐标。

依次对每个点的y坐标进行查询，查询后立即加入树状数组。由于点集是x坐标有序的，依次加入树状数组可以保证满足xj <= xi，且答案不会有任何遗漏。

注意，让树状数组的下标从1起比较简单。

甜点大师

题目描述

给你N个冰激凌球的大小B[i]，要求把它们组成若干个大小为K的冰激凌塔（即长度为K的序列），每个序列内满足2 * A[i-1] ≤ A[i]。求最多能组成多少个冰激凌塔。

1 ≤ N ≤ 3e5

1 ≤ K ≤ 30

1 ≤ Bi ≤ 1e9

甜点大师

解析

考虑贪心，从最小的冰激凌球开始，每次找最小的≥2Ai的冰激凌球堆在下层，取满K个为一个塔，取到所有球耗尽为止。

为什么这个思路是错的?

1 2 3 4　　K=2

甜点大师

解析

考虑一个简化的问题：

给你N个冰激凌球的大小B[i]，要求把它们组成若干个大小为K的冰激凌塔（即长度为K的序列），每个序列内满足2 * A[i-1] ≤ A[i]。给定一个正整数X，判断这些冰激凌球是否能组成至少X个冰激凌塔。

甜点大师

解析

贪心策略，最小的X个冰激凌球作为塔顶一定是最优的。

次小的X个满足条件的冰激凌球作为第二层。以此类推。如果在堆够X个冰激凌塔之前，冰激凌球就用完了，那么是不能的。

对于前面的问题，如何找到这个X?

甜点大师

解析

二分答案，判断是否能堆出X个，取最大的满足条件的X即可。

时间复杂度O(NlogN)。

有趣的思维题

Given a n*n matrix C_{ij} (1<=i,j<=n),We want to find a n*n matrix X_{ij} (1<=i,j<=n),which is 0 or 1.

Besides,X_{ij} meets the following conditions:

1.$X_{12}+X_{13}+...X_{1n}=1$
2.$X_{1n}+X_{2n}+...X_{n-1n}=1$
3.for each i (1<i<n), satisfies $\sum X_{ki}$ (1<=k<=n)=$\sum X_{ij}$ (1<=j<=n).

For example, if n=4,we can get the following equality:

$X_{12}+X_{13}+X_{14}=1$
$X_{14}+X_{24}+X_{34}=1$
$X_{12}+X_{22}+X_{32}+X_{42}=X_{21}+X_{22}+X_{23}+X_{24}$
$X_{13}+X_{23}+X_{33}+X_{43}=X_{31}+X_{32}+X_{33}+X_{34}$

Now ,we want to know the minimum of $\sum C_{ij}*X_{ij}$(1<=i,j<=n) you can get.

有趣的思维题

Sample Input

```
4
1 2 4 10
2 0 1 1
2 2 0 5
6 3 1 2
```

Sample Output

```
3
```

Hint

For sample, $X_{12}=X_{24}=1$,all other X_{ij} is 0.

Input

The input consists of multiple test cases (less than 35 case).
For each test case, the first line contains one integer n (1<n<=300).
The next n lines, for each lines, each of which contains n integers, illustrating the matrix C, The j-th integer on i-th line is C_{ij}(0<=C_{ij}<=100000).

Output

For each case, output the minimum of $\sum C_{ij}*X_{ij}$ you can get.

有趣的思维题

- 放在邻接矩阵上看？
- $X_{12}+X_{13}+...X_{1n}=1$
  - 1的出度为1
- $X_{1n}+X_{2n}+...X_{n-1n}=1$
  - -n的入度为1
- for each i (1<i<n), satisfies $\sum X_{ki}$ (1<=k<=n)=$\sum X_{ij}$ (1<=j<=n).
  - 其他每个点的入度和出度相同
- 本质就是1-n的最短路径

A. JM 的浩然正气

内存限制：512 MiB

时间限制：1000 ms

题目描述

余囚北庭，坐一土室。室广八尺，深可四寻。单扉低小，白间短窄，污下而幽暗。当此夏日，诸气萃然：雨潦四集，浮动床几，时则为水气；涂泥半朝，蒸沤历澜，时则为土气；乍晴暴热，风道四塞，时则为日气；檐阴薪爨，助长炎虐，时则为火气；仓腐寄顿，陈陈逼人，时则为米气；骈肩杂遝，腥臊汗垢，时则为人气；或圊溷、或毁尸、或腐鼠，恶气杂出，时则为秽气。叠是数气，当之者鲜不为厉。而予以孱弱，俯仰其间，於兹二年矣，幸而无恙，是殆有养致然尔。然亦安知所养何哉？孟子曰：「吾善养吾浩然之气。」彼气有七，吾气有一，以一敌七，吾何患焉！况浩然者，乃天地之正气也，作正气歌一首。

天地有正气，杂然赋流形。下则为河岳，上则为日星。
於人曰浩然，沛乎塞苍冥。皇路当清夷，含和吐明庭。
时穷节乃见，一一垂丹青。在齐太史简，在晋董狐笔。
在秦张良椎，在汉苏武节。为严将军头，为嵇侍中血。
为张睢阳齿，为颜常山舌。或为辽东帽，清操厉冰雪。
或为出师表，鬼神泣壮烈。或为渡江楫，慷慨吞胡羯。
或为击贼笏，逆竖头破裂。是气所磅礴，凛烈万古存。
当其贯日月，生死安足论。地维赖以立，天柱赖以尊。
三纲实系命，道义为之根。嗟予遘阳九，隶也实不力。
楚囚缨其冠，传车送穷北。鼎镬甘如饴，求之不可得。
阴房阗鬼火，春院闭天黑。牛骥同一皂，鸡栖凤凰食。
一朝蒙雾露，分作沟中瘠。如此再寒暑，百疠自辟易。
哀哉沮洳场，为我安乐国。岂有他缪巧，阴阳不能贼。
顾此耿耿存，仰视浮云白。悠悠我心悲，苍天曷有极。
哲人日已远，典刑在夙昔。风檐展书读，古道照颜色。

这是南宋诗人文天祥在狱中写的一首五言古诗《正气歌》，可是这和你 AC 这题有什么关系吗？你只需要输出诗中有多少个汉字即可，不包括序。

输出格式

输出一行一个整数表示答案。

解题思路

愣着干嘛？数啊！一共 300 个汉字。

代码

```cpp
int main()
{
    printf("3");
    cout << false << "0";
    return 0;
}
```

期末考试

时间：2019 年 7 月 6 日 星期六 8:30~11:00

题目：

+1	A. JM 的黑心商店	基础运算
+1	B. JM 的招摇撞骗	贪心
+	C. JM 的恶有恶报	二分查找
+	D. JM 的神庙逃亡	BFS
+	E. JM 的不卡常数	动态规划，区间 dp
+	F. JM 的赃物被盗	高维前缀和
−	G. JM 的月亮神树	
−	H. cyy 的养殖基地	

A. JM 的黑心商店

内存限制：512 MiB

时间限制：1000 ms

题目描述

黑心商人 JM 的黑心商店非常黑心，这家商店出售的糖果是一块钱一颗，但是如果你想买第二颗糖果，JM 就会把它的价格提高到两块钱，第三颗的价格是三块钱。也就是说，你买第 i 颗糖果的价格是 i 元。

今天 cyy 想吃糖了，但他非常吝啬，于是他把你抓去买糖。cyy 告诉你他想吃 n 颗糖，请计算你需要花多少钱才能买下这些糖。

输入格式

一行一个正整数 n 。

输出格式

一行一个正整数表示答案。

数据范围与提示

$1 \le n \le 10^9$

解题思路

总价钱为 $\sum_{i=1}^{n} i = n(n+1)/2$。答案最大可能为 $5 \cdot 10^{17}$，故应使用 llong 型存储答案。

代码

```
int main()
{
    llong a;
    scanf("%lld", &a);
    a = a * (a + 1) / 2;
    printf("%lld", a);
    return 0;
}
```

B. JM 的招摇撞骗

内存限制：512 MiB

时间限制：1000 ms

题目描述

JM 不仅经营黑心商店，还会在月黑风高的夜晚，打扮成一个派大星，出门到处骗人。今天晚上 jwp 和 zzy 在搬运浓芙膻泉，很不巧地遇到了派大星。派大星告诉他们，要想成为像 cyy 那么强的 dalao，就必须虔诚地供奉咕呐啦执杖之神，获得无上神力庇佑，就能吊打 cyy。

虽然 jwp 和 zzy 本来就是 dalao，但他们仍然被骗了（？）。JM 告诉他们，只要把两块五毛八一斤的浓芙膻泉交给他，他就可以帮两人将其献给咕咕咕……哦不，献给咕呐啦执杖之神。JM 带了一个水桶，容积为 V 升。jwp 和 zzy 搬着一箱子共 n 瓶不同品种的浓芙膻泉，第 i 瓶装了 a_i 升，且每升的价值为 b_i 。

为了尽可能打动咕呐啦执杖之神，jwp 和 zzy 决定尽可能地献出价值更高的浓芙膻泉。而浓芙膻泉是一种神奇的东西，即使不同种类的浓芙膻泉混合在一起也没关系，所以他们只需要往水桶里灌到满就好了。他们想知道他们最多能献出价值多高的浓芙膻泉。

输入格式

第一行两个正整数 V 和 n 。

接下来 n 行，每行两个正整数 a_i, b_i 。

输出格式

输出一行一个正整数表示答案。

数据范围与提示

$1 \leq V \leq 10^9$

$1 \leq n \leq 2 \cdot 10^5$

$1 \leq a_i \leq 10^9$

$1 \leq b_i \leq 10^9$

解题思路

贪心，尽可能选择单位价值最高的浓芙膻泉即可。

代码

```
llong v, n, wal = 0;
vector<prdd> nfsqs;
int main()
{
    scanf("%lld%lld", &v, &n);
    for (int i = 0; i < n; i++)
    {
        int a, b;
        scanf("%d%d", &a, &b);
        nfsqs.push_back(prdd(b, a));
    }
    sort(nfsqs.begin(), nfsqs.end());
    for (int i = n - 1; i >= 0; i--)
    {
        if (nfsqs.at(i).second <= v)
        {
            v -= llong(nfsqs.at(i).second);
            wal += 1ll * nfsqs.at(i).first * nfsqs.at(i).second;
        }
        else
        {
            wal += 1ll * nfsqs.at(i).first * v;
            break;
        }
```

```
    }
    printf("%lld", wal);
    return 0;
}
```

C. JM 的恶有恶报

内存限制：512 MiB

时间限制：1000 ms

题目描述

由于 JM 太黑心了，长期坑钱，坑了许多人的钱，今天 tyx 和 cty 实在是忍无可忍，带了一大群寒域爷过来把 JM 的商店拆了。JM 吓得赶紧从密道逃走了，可是 tyx 和 cty 并不打算放过他，他们带着寒域爷们追了进去。可是想不到 JM 竟然如此狡猾，密道出入口事先被安放了炸弹，JM 引爆炸弹摧毁了道路，将众人困在了地下。

众人发现，想要挖开通路回到地面上恐怕并不容易，需要很多天，但他们身上的食物并不充足。除了 tyx 和 cty 是 julao 不需要进食以外，这一大群共 n 个寒域爷都是要吃饭的。大家凑了凑，发现身上一共带了 m 种不同的食物，第 i 种食物有 c_i 份。

公平起见，tyx 和 cty 决定每个寒域爷每天吃 1 份食物。但是寒域爷非常挑食（？），如果某个寒域爷第一天吃了某种类型的食物，那么接下来他就必须一直吃这种类型的食物。此外寒域爷还非常脆弱，只要一天没饭吃就会 GG。

现在 tyx 和 cty 想知道，在合理安排的情况下，寒域爷们最多能坚持多少天没有任何一人 GG，这样他们可以决定他们应该以多快的速度挖地道，毕竟这是很累人的事儿。

输入格式

第一行两个正整数 n, m 。

接下来一行 m 个正整数，表示每种食物的份数。

输出格式

输出一行一个非负整数表示答案。

数据范围与提示

$1 \le n \le 10^5$

$1 \le m \le 10^5$

$1 \le c_i \le 10^9$

对于样例 1，让一个寒域爷吃第一种食物、一个寒域爷吃第二种食物、两个寒域爷吃第三种食物，可以吃 5 天。

对于样例 2，第一天就会有寒域爷 GG。

解题思路

假设能支撑 x 天，则第 i 种食物能支撑 $\lfloor c_i/x \rfloor$ 个寒域爷活下去，则一共可以支撑的寒域爷数量为 $\sum_{i=1}^{n} \lfloor c_i/x \rfloor$。二分查找最小的 x 满足 $\sum_{i=1}^{n} \lfloor c_i/x \rfloor \ge n$ 即可。

代码

```
int n, m;
vector<int> cs;
```

```cpp
bool check(int d)
{
    int top = 0;
    for (int i = 0; i < m; i++)
        top += cs.at(i) / d;
    if (top >= n)
        return true;
    return false;
}
void sch(int l, int r)
{
    if (r - l <= 1)
    {
        printf("%d", l);
        return;
    }
    int mid = (l + r) >> 1;
    if (check(mid))
        sch(mid, r);
    else
        sch(l, mid);
}
int main()
{
    scanf("%d%d", &n, &m);
    for (int i = 0; i < m; i++)
    {
        int mi;
        scanf("%d", &mi);
        cs.push_back(mi);
    }
    sch(0, 1000000099);
    return 0;
}
```

D. JM 的神庙逃亡

内存限制：512 MiB

时间限制：1000 ms

题目描述

虽然寒域爷们带了充足的食物，可他们没带水呀（滑稽），所以他们还是不幸 GG 了。为了给他的三千佳丽寒域爷们报仇，后宫王 qz 抄起了他祖传的 40 米长刀，杀进了 JM 的神庙。他知道 JM 修的密道的出口一定是通往那里的。

神庙中有 n 个房间，这些房间的编号为 $1,2,\ldots,n$，一共有 m 条长度相等的**双向**通路连接这些房间。后宫王 qz 进入神庙后先到了 a 号房间，他身具火眼金睛，一眼就看穿了 JM 躲

在 b 号房间，怒火冲天的 qz 想以最快的速度捉拿 JM，请你告诉他，他至少需要经过多少个房间才能抵达 JM 的所在。

输入格式

第一行四个正整数 n 和 m 。

接下来 m 行，每行两个正整数 u,v ，表示有一条从 u 号房间到 v 号房间的**双向**通路。

输入保证不存在两条通路是重叠的，且保证 $u \neq v$ 。

输出格式

输出一个正整数，表示最少需要经过的房间数，包括起点和终点。如果 qz 无论如何也抓不到 JM，则输出 -1 。

数据范围与提示

$1 \leq n,m \leq 2 \cdot 10^5$

解题思路

BFS。

代码

```
int n, m, a, b, ans = 1;
bool vis[270000];
vector<int> ways[270000];
vector<int> bfs, tmp;
int main()
{
    scanf("%d%d%d%d", &n, &m, &a, &b);
    for (int i = 0; i < m; i++)
    {
        int u, v;
        scanf("%d%d", &u, &v);
        ways[u].push_back(v);
        ways[v].push_back(u);
    }
    bfs.push_back(a);
    while (!bfs.empty())
    {
        ans++;
        for (int i = 0; i < bfs.size(); i++)
        {
            int ni = bfs.at(i);
            for (int j = 0; j < ways[ni].size(); j++)
            {
                int nj = ways[ni].at(j);
                if (!vis[nj])
                {
```

```
                    if (nj == b)
                        goto end;
                    tmp.push_back(nj);
                    vis[nj] = true;
                }
            }
        }
        swap(bfs, tmp);
        tmp.clear();
    }
    printf("-1");
    return 0;
end:
    printf("%d", ans);
    return 0;
}
```

E. JM 的不卡常数

内存限制：512 MiB

时间限制：1000 ms

题目描述

我做此题发自真心。 ——Mr. JM. Liu

就在大家以为 JM 跑不掉的时候，他遇到了一个名为不卡常的人，不卡常说，只要 JM 说出他的幸运数字，也就是不卡常数，他就帮助 JM 逃离。

JM 面前有一串以左圆括号"("，右圆括号")"，左方括号"["，右方括号"]"组成的序列，而且定义：

(1)空序列是合法的；

(2)如果一个序列 S 是合法的，那么 (S) 和 $[S]$ 都是合法的；

(3)如果序列 S 和 T 是合法的，那么序列 ST 也是合法的。

定义不卡常数为最少填充几个括号，以使得括号序列合法。

你能帮帮善良可爱又软萌的小 jm 嘛 quq

输入格式

输入一行一个非空字符串 S，仅由四种括号组成。

输出格式

输出一行一个非负整数，表示最少填充几个括号可以使输入的括号序列合法。

数据范围与提示

$1 \le |S| \le 100$

解题思路

考虑区间 dp，从第 i 个到第 j 个字符组成的子串需要填充的最少括号数 $dp(i,j)$，其可以从中间任意位置 k 拆成的两个子串合并而来，即 $dp(i,j)=dp(i,k)+dp(k+1,j)$，取其中最小值，即 $dp(i,j)=\min_{k=i}^{j}\big(dp(i,k)+dp(k+1,j)\big)$。若第 i 个字符与第 j 个字符配对，则 $dp(i,j)=\min\big(dp(i+1,j-1),dp(i,j)\big)$。

代码

```
string str;
int dp[128][128];
inline bool check_1(int i, int j)
{
    return str.at(i) == '(' && str.at(j) == ')';
}
inline bool check_2(int i, int j)
{
    return str.at(i) == '[' && str.at(j) == ']';
}
inline bool check(int i, int j)
{
    return check_1(i, j) || check_2(i, j);
}
int main()
{
    cin.sync_with_stdio(false);
    cin.tie(0);
    cin >> str;
    for (int i = str.length() - 1; i >= 0 ; i--)
        for (int j = i; j < str.length(); j++)
        {
            if (i == j)
            {
                dp[i][i] = 1;
                continue;
            }
            dp[i][j] = 2147483647;
            for (int k = i; k <= j; k++)
                dp[i][j] = min(dp[i][j], dp[i][k] + dp[k + 1][j]);
            if (check(i, j))
                dp[i][j] = min(dp[i][j], dp[i + 1][j - 1]);
        }
    printf("%d", dp[0][str.length() - 1]);
    return 0;
}
```

F. JM 的赃物被盗

内存限制：512 MiB

时间限制：1000 ms

题目描述

就在JM逃亡的时候，渣男mzj挖进了JM的秘密宝库，发现了一个装满不明液体的 $a \times b \times c$ 的长方体，里面装的正是他从 jwp 和 zzy 那里骗来的浓芙膻泉，看来要便宜 mzj 了。

渣男 mzj 把这个长方体抱了回去，并用机器分析它的成分。如果你还记得的话，这里面混合了多种价值不同的浓芙膻泉，而这些神奇的浓芙膻泉有些相容有些不相容，总之就是每一单位液体的价值都是不一样的，我们把位于 (i,j,k) 处的液体的价值记作 $cost[i][j][k]$ 。长方体中的坐标下标都是从 1 起的。

因为 mzj 的这个机器是 qz 造的，此时它突然给出 q 次询问，每次给出一个子长方体的左上角的坐标 (x_1,y_1,z_1) 和右下角的坐标 (x_2,y_2,z_2) 来表示一个子长方体，问你这个子长方体中的液体价值之和是多少。如果 mzj 不能及时回答，他就会被这台机器抓去喂 qz 吃。mzj 不想被 qz 吃掉，所以你必须帮助他，不然他就会把你带走一起被 qz 吃掉。

输入格式

第一行三个正整数 a,b,c ，表示长方体的体积。

接下来 a 个块，每块 b 行，每行 c 个整数，表示长方体中每格液体的价值。

接下来一行一个正整数 q ，表示询问次数。

接下来 q 行，每行六个正整数 x_1,y_1,z_1,x_2,y_2,z_2 ，表示长方体的左上角和右下角的坐标。

输入保证：

$1 \le x_1 \le x_2 \le a$

$1 \le y_1 \le y_2 \le b$

$1 \le z_1 \le z_2 \le c$

输出格式

输出 q 行，每行一个正整数，表示对应询问的答案。

数据范围与提示

$1 \le a,b,c \le 125$

$|cost[i][j][k]| \le 100$

$1 \le q \le 10^5$

本题 I/O 量较大，请使用 scanf/printf，避免使用 cin/cout 导致 TLE。

解题思路

高维前缀和，设 $sum[i][j][k] = \sum\{cost[x][y][z] | x \le i, y \le j, z \le k\}$，则询问的值为

$$ans\begin{pmatrix}x_1,y_1,z_1\\x_2,y_2,z_2\end{pmatrix} = (d_0 - d_1 + d_2 - d_3)\begin{pmatrix}x_1,y_1,z_1\\x_2,y_2,z_2\end{pmatrix}$$

$$d_0 = sum[x_2][y_2][z_2]$$

$$d_1 = sum[x_1'][y_2][z_2] + sum[x_2][y_1'][z_2] + sum[x_2][y_2][z_1']$$

$$d_2 = sum[x_1'][y_1'][z_2] + sum[x_1'][y_2][z_1'] + sum[x_2][y_1'][z_1']$$

$$d_3 = sum[x_1'][y_1'][z_1']$$

$$x_1' = x_1 - 1, y_1' = y_1 - 1, z_1' = z_1 - 1$$

由 $cost[i][j][k] = ans\binom{i,j,k}{i,j,k}$ 可以递推得到 $sum[i][j][k]$ 的值：

$$sum[i][j][k] = d_0\binom{i,j,k}{i,j,k} = (d_1 - d_2 + d_3)\binom{i,j,k}{i,j,k} + cost[i][j][k]$$

代码

```
int a, b, c, q;
int cst[128][128][128], sum[128][128][128];
int get(int i, int j, int k)
{
    if (i <= 0 || j <= 0 || k <= 0)
        return 0;
    if (sum[i][j][k] < 2147483647)
        return sum[i][j][k];
    int ip = i - 1, jp = j - 1, kp = k - 1,
        d_1 = get(ip, j, k) + get(i, jp, k) + get(i, j, kp),
        d_2 = get(ip, jp, k) + get(ip, j, kp) + get(i, jp, kp);
    return sum[i][j][k] = d_1 - d_2 + get(ip, jp, kp) + cst[i][j][k];
}
int main()
{
    scanf("%d%d%d", &a, &b, &c);
    for (int i = 1; i <= a; i++)
        for (int j = 1; j <= b; j++)
            for (int k = 1; k <= c; k++)
            {
                scanf("%d", &cst[i][j][k]);
                sum[i][j][k] = 2147483647;
            }
    scanf("%d", &q);
    for (int i = 0; i < q; i++)
    {
        int x, y, z, sa, sb, sc;
        scanf("%d%d%d%d%d%d", &sa, &sb, &sc, &x, &y, &z);
        sa--; sb--; sc--;
        int d_1 = get(sa,y,z) + get(x,sb,z) + get(x,y,sc),
            d_2 = get(sa,sb,z) + get(sa,y,sc) + get(x,sb,sc);
        int ans = get(x,y,z) - d_1 + d_2 - get(sa,sb,sc);
        printf("%d\n", ans);
    }
    return 0;
}
```

G. JM 的月亮神树

内存限制：512 MiB

时间限制：2000 ms

题目描述

JM 今天过得实在是太惨了，他肥肠不爽，因此他搬出了他祖传的月亮神树。月亮神树的光辉照耀 XJTUACM，每分钟都可以从所有被照射的人的手中夺取他 1%的财产。JM 觉得他很快就能赚得盆满钵满。然而月亮神树不久后突然失控了，不仅不把它夺来的财产交给 JM，甚至开始夺取 JM 的财产，把 JM 气疯了，赶紧关闭了月亮神树。

因为月亮神树没运行多久就被关掉了，所以大家并没有损失多少，迫于月亮神树的威压，并没有人去理它。可是 wzk 昨天刚中了彩票，赚了 100000000000￥，转眼就被剥夺了很多。因为 wzk 太 rich 了，所以他的损失格外大。为了夺回财产，无敌的 wzk 决定消灭月亮神树。

月亮神树的构造非常神奇，它的枝杈交错纵横，树上甚至存在环路，可以视为一个无向带权连通图的结构。月亮神树有一个核心节点，记为 s 。要想消灭月亮神树，必须找到月亮神树的严格最不科学生成树，这是它的弱点，这样就能将其一举摧毁。

定义：一个图 G 的不科学生成树是 G 的一棵子树，在这棵子树上，从核心节点 s 到任意一个节点 u 的最短路径长度，和在原图上是等长的。其中节点 s 是月亮神树的核心节点。

定义：一个图 G 的最不科学生成树是 G 的所有不科学生成树中，边权和最小的一棵树。

定义：一个图 G 的严格最不科学生成树是 G 的所有最不科学生成树中，有序边序列的字典序最小的一棵树。

定义：一个图 G 的有序边序列是指，将图上所有边按编号从小到大排序后得到的编号序列。当然，子图子树也适用。

现在给定月亮神树，即给定一个 n 个点 m 条边的无向带权连通图，点的编号为 $1,2,\ldots,n$ ，边的编号为 $1,2,\ldots,m$ ，给定核心节点的编号 s ，求其严格最不科学生成树。

输入格式

第一行三个正整数 n,m,s 。

接下来 m 行，第 i 行三个正整数 u,v,w ，表示编号为 i 的边。

输入保证图是连通的，保证图上不含重边和自环。

输出格式

第一行两个正整数 cnt 和 sum ，用空格隔开，其中 snt 表示严格最不科学生成树的边的个数， sum 表示严格最不科学生成树的边权和。

接下来一行 cnt 个正整数，用空格隔开，为树上所有边的编号，按编号从小到大输出。

数据范围与提示

$1 \le n,m \le 3 \cdot 10^5$

$1 \le w_i \le 10^9$

解题思路

存储 s 到 i 的最短路上以 i 为终点的边，若 s 到最短路更小的所有点的最短路均不变，选择其中任意一条边均保证 s 到 i 的最短路不变，故选择边权最小、编号最小的边即可。根据数学归纳法可知，除 s 外的所有点均按上述方法选择一条边即可保证所有点的最短路不变，选择的边数固定为 $n-1$。

代码

```
struct tprdd
{
    llong first;
    int second, third, forth;
};
struct gter_tprdd
{
    bool operator() (const tprdd &x, const tprdd &y)
    {
        return x.first > y.first;
    };
};
int n, m, s;
llong sum;
bool vis[500000];
llong dis[500000];
vector<tprdd> pic[500000];
priority_queue<int, vector<int>, greater<int>> ans;
priority_queue<prdd, vector<prdd>, greater<prdd>> frm[500000];
void dijkstra(int fr)
{
    for (int i = 0; i <= n; i++)
        dis[i] = llinf;
    priority_queue<tprdd, vector<tprdd>, gter_tprdd> mins;
    mins.push(tprdd{ 0, fr });
    while (!mins.empty())
    {
        tprdd nm = mins.top();
        mins.pop();
        if (nm.first > dis[nm.second])
            continue;
        frm[nm.second].push(prdd(nm.third, nm.forth));
        if (vis[nm.second])
            continue;
        vis[nm.second] = true;
        for (auto p : pic[nm.second])
            if (dis[p.second] >= nm.first + p.third)
            {
                tprdd np = p;
                np.first = dis[p.second] = nm.first + p.third;
                mins.push(np);
            }
    }
}
```

```cpp
int main()
{
    scanf("%d%d%d", &n, &m, &s);
    for (int i = 1; i <= m; i++)
    {
        int u, v, w;
        scanf("%d%d%d", &u, &v, &w);
        pic[u].push_back(tprdd{ 0, v, w, i });
        pic[v].push_back(tprdd{ 0, u, w, i });
    }
    dijkstra(s);
    for (int i = 1; i <= n; i++)
        if (i != s)
        {
            sum += frm[i].top().first;
            ans.push(frm[i].top().second);
        }
    printf("%d %lld\n", n - 1, sum);
    for (int i = 0; i < n - 1; i++)
    {
        printf("%d ", ans.top());
        ans.pop();
    }
    return 0;
}
```

H. cyy 的养殖基地

内存限制：512 MiB

时间限制：1000 ms

题目描述

无敌的 wzk 毁灭了月亮神树，连空间都被他打爆了，想不到竟然意外地发现了隐藏在异度空间中的 cyy 的秘密养殖基地。cyy 对此感到非常愤怒，因为基地暴露意味着他要交税了，他把 wzk 绑了起来，如果 wzk 回答不出他的神仙题，就会被他拿去喂 qz。

养殖基地里有一个长度为 L 的圆环，有 n 只蚂蚁在圆环上爬行，速度均为 1。若两只蚂蚁在爬行过程中相遇，则二者均调头爬行。定义坐标从 0 开始逆时针数，编号为 i 的蚂蚁的初始坐标为 a_i，方向为 w_i，其中 $w_i = 0/1$ 表示逆时针/顺时针。蚂蚁坐标是随编号单调递增的，求 T 秒后每只蚂蚁位置。快从暴徒 cyy 的手中救救无敌的 wzk 吧！

输入格式

第一行三个整数 n, L, T。

接下来 n 行，每行两个整数 a_i, w_i。

输入保证蚂蚁坐标是随编号单调递增的，且 $w_i \in \{0, 1\}$。

输出格式

输出 n 行，第 i 行一个数表示编号为 i 的蚂蚁最后的坐标。

数据范围与提示

$2 \le n \le 10^5$

$2 \le L \le 10^9$

$2 \le T \le 10^9$

$0 \le a_1 < a_2 < \cdots < a_n < L$

$w_i \in \{0, 1\}$

解题思路

首先不考虑编号，由于蚂蚁速度均为 1，发现两蚂蚁相遇后调头与否并无区别，最终统计所有蚂蚁的位置时结果相同。若考虑编号，则调头等价于交换两蚂蚁的编号。若按调头考虑，则容易发现所有蚂蚁的相对位置均按编号排列，即在 $\bmod n$ 意义下，i 号蚂蚁的上一个蚂蚁一定是 $i-1$ 号蚂蚁，下一个蚂蚁一定是 $i+1$ 号蚂蚁。若按交换编号考虑，则很容易得出所有蚂蚁的最终具体位置。综上：若能得出任意一个蚂蚁的位置和编号，便可以为所有位置的蚂蚁得出编号。考虑跟踪最开始的 1 号蚂蚁，按交换编号考虑，很容易得到其最终位置。若其逆时针前进，则其只能与编号比其大 1 的蚂蚁交换编号，即交换一次编号 $+1$；反之编号 -1。计算所有其他蚂蚁与其相遇的次数，就能得出其最终编号。之后就能得到所有蚂蚁的位置和编号。

注意，若为逆时针运动，且最终位置有两个点，另一个点编号要小于跟踪的点；否则，同位置的另一个点编号要大于跟踪的点。

代码

```
int n, l, t;
int fa, fw, rfw;
llong dnm;
vector<int> ant[2], anss;
inline llong nmod(llong x, llong p)
{
    return ((x % p) + p) % p;
}
int main()
{
    scanf("%d%d%d", &n, &l, &t);
    for (int i = 0; i < n; i++)
    {
        int a, w;
        scanf("%d%d", &a, &w);
        ant[w].push_back(a);
        if (i == 0)
            fa = a, fw = w, rfw = w ^ 1;
    }
    for (int i = 0; i < ant[rfw].size(); i++)
    {
```

```
        int iant = ant[rfw][i], tp = 2 * t,
            fmt = nmod((iant - fa) * (1 - fw * 2), l);
        if (fmt <= tp)
            dnm += 1, tp -= fmt;
        dnm += tp / l;
    }
    for (int w = 0; w < 2; w++)
        for (int i = 0; i < ant[w].size(); i++)
            anss.push_back(nmod(t * (1 - w * 2) + ant[w][i], l));
    sort(anss.begin(), anss.end());
    int ftp = nmod(fa + t * (1 - fw * 2), l),
        i = lower_bound(anss.begin(), anss.end(), ftp) - anss.begin();
    if (anss[nmod(i - fw * 2 + 1, n)] == anss[i])
        i = nmod(i - fw * 2 + 1, n);
    for (llong j = 0; j < n; j++)
        printf("%lld\n", anss[nmod(j + i - dnm * (1 - fw * 2), n)]);
    return 0;
}
```