目录

[注意 4](#_Toc9963)

[定义 5](#_Toc9431)

[STL 5](#_Toc30604)

[优先队列重载 5](#_Toc29936)

[set重载 6](#_Toc19308)

[动态开数组 6](#_Toc3690)

[set 8](#_Toc6858)

[map 9](#_Toc15248)

[unordered\_map 11](#_Toc30140)

[bitset 12](#_Toc26414)

[计算几何 13](#_Toc4778)

[zyx的计算几何 13](#_Toc12716)

[计算几何全家桶 22](#_Toc19830)

[自适应辛普森 30](#_Toc14362)

[数据结构 31](#_Toc1239)

[kruskal重构树 31](#_Toc25745)

[普通莫队 31](#_Toc28609)

[带修莫队 33](#_Toc6796)

[线段树合并分裂 35](#_Toc4807)

[主席树 36](#_Toc32223)

[LCT 37](#_Toc23693)

[Splay1 41](#_Toc31982)

[splay2 44](#_Toc14457)

[Treap 46](#_Toc31235)

[舞蹈链（多重覆盖） 47](#_Toc14935)

[舞蹈链（精确覆盖） 50](#_Toc1586)

[数论 52](#_Toc26894)

[lucas求组合数 52](#_Toc26211)

[扩展欧几里得求逆元 53](#_Toc20757)

[逆元线性递推 inv阶乘组合数 54](#_Toc11166)

[数学 54](#_Toc17433)

[一些范围 54](#_Toc23183)

[BSGS 54](#_Toc12009)

[扩展BSGS 56](#_Toc1114)

[二次剩余 57](#_Toc32631)

[卡特兰数 59](#_Toc769)

[快速幂 60](#_Toc9267)

[龟速乘快速幂（快速幂爆longlong 60](#_Toc24027)

[莫比乌斯反演 61](#_Toc14104)

[博弈 64](#_Toc16117)

[高精度GCD 65](#_Toc14865)

[高精度乘法（FFT） 68](#_Toc26711)

[高精度乘法（乘单精度 70](#_Toc14248)

[高精度乘法（朴素） 70](#_Toc20603)

[高精度除法（除单精度） 71](#_Toc29218)

[高精度除法（除高精度） 72](#_Toc1916)

[高精度加法 73](#_Toc23662)

[高精度减法 74](#_Toc21194)

[高精度阶乘 74](#_Toc13200)

[高精度进制转换 75](#_Toc7009)

[高精度幂 75](#_Toc1140)

[高精度平方根 77](#_Toc27351)

[高精度取模（对单精度） 81](#_Toc2381)

[欧拉筛 81](#_Toc21949)

[组合数（逆元线性递推 81](#_Toc25320)

[中国剩余定理 82](#_Toc26150)

[图论 83](#_Toc5532)

[有源汇上下界最大小流 83](#_Toc28503)

[树链剖分 85](#_Toc11970)

[虚树 88](#_Toc15328)

[spfa最短路及负环 90](#_Toc24276)

[二分图匹配（匈牙利） 91](#_Toc20925)

[强连通（kosaraju 93](#_Toc30119)

[强连通（tarjan 94](#_Toc9255)

[强连通（tarjan无vector 95](#_Toc22880)

[最大流 96](#_Toc2361)

[最大流（double） 98](#_Toc5645)

[最小费用最大流 100](#_Toc14758)

[树分治 101](#_Toc21259)

[拓扑排序 102](#_Toc4865)

[最近公共祖先（倍增） 103](#_Toc9377)

[最近公共祖先（线段树） 104](#_Toc22892)

[线性代数 105](#_Toc9083)

[高斯消元 105](#_Toc23470)

[矩阵行列式 107](#_Toc16613)

[线性基 107](#_Toc13562)

[线性基2 108](#_Toc17914)

[矩阵（加减乘快速幂 109](#_Toc495)

[稀疏矩阵乘法 110](#_Toc7454)

[杂项 111](#_Toc19098)

[快读 111](#_Toc16509)

[fread快读 111](#_Toc12671)

[模拟退火 111](#_Toc15725)

[整体二分 114](#_Toc14323)

[字符串 115](#_Toc31710)

[马拉车 115](#_Toc10836)

[AC自动机 116](#_Toc12860)

[KMP 117](#_Toc6875)

[KMP 2 118](#_Toc29037)

[Tire 119](#_Toc18458)

[后缀数组 120](#_Toc20955)

[可持久化字典树 121](#_Toc29804)

[对拍 121](#_Toc20903)

[windows环境下bat对拍 121](#_Toc7647)

## 注意

优先队列是大的在前面 如果要小的 要重载  
long long 二分答案的时候..精度 也有可能 爆int（?

哈希 自然溢出 yyds 双哈希  
输出限制..

匈牙利的复杂度常数非常小（..

递归爆栈 re

for i 进行计算的时候 （ i 开 long long ）

边界问题 各种01的特判

模 多模一点 都可以模（

char数组开小了也可能报错tle 和 wa （

图是否连通 是否重边 是否自环

读题！！与 或（

重点 重边

当保证n的总和不会很大，但数据组数可能很多的时候，注意初始化造成的tle问题（

pow() 的精度问题

unique erase 先排序

图论初始化！！  
没开longlong，中间有个判定条件爆了

他卡快排，由于答案不超过10000，可以计数排序

re：没有开longlong，（以为是dfs爆栈

## 定义

**()** (a,b)=1 最大公约数 即a，b互质

**|** 整除 a|b b%a==0

## STL

### 优先队列重载

priority\_queue<int, vector<int>, cmp>s;  
struct cmp{  
 bool operator()(const int &a,const int &b){  
 return a>b;   
 }   
};

### set重载

#include <bits/stdc++.h>  
using namespace std;  
#define l first  
#define r second   
struct cmp{  
 bool operator() (const pair <int, int> &a, const pair<int, int> &b) const{  
 int lena = a.r - a.l + 1;  
 int lenb = b.r - b.l + 1;  
 if(lena == lenb) return a.l < b.l;  
 return lena > lenb;  
 }  
};  
int main(){  
 ios :: sync\_with\_stdio(0); cin.tie(0); cout.tie(0);  
 int T;  
 cin >> T;  
 while(T -- ){  
 int n;  
 cin >> n;  
 set<pair<int, int>, cmp> segs;  
 segs.insert({0, n - 1});  
 vector<int> a(n);  
 for(int i = 1; i <= n; ++ i){  
 pair<int, int> cur = \*segs.begin();  
 segs.erase(segs.begin());  
 int id = (cur.l + cur.r) / 2;  
 a[id] = i;  
 if(cur.l < id) segs.insert({cur.l, id - 1});  
 if(id < cur.r) segs.insert({id + 1, cur.r});  
 }  
   
 for(auto it : a) cout << it << " ";  
 cout << endl;  
 }  
}

### 动态开数组

int a[15], n, m;  
cin >> n >> m;  
int (\*b)[m] = (int (\*)[m])a;

**new / delete**

#define M 10U  
#define N 20  
第一种，可以直接[][]访问。但是内存不连续，不是很推荐使用，除非M \ N都不确定  
//定义的时候  
int\*\* pNum;//以int为例  
pNum = new int\*[M];  
for(int i = 0;i < M;i ++){  
 pNum[i]=new int[N];  
}  
  
//删除的时候是  
for(int j = 0;j < M;j ++){  
 delete []pNum[i];  
}  
delete []pNum;

**malloc / free**

#include<stdio.h>   
#include<stdlib.h>   
   
int main() {   
 int \*\*a; //用二级指针动态申请二维数组   
 int i,j;   
 int m,n;   
 printf("请输入行数\n");   
 scanf("%d",&m);   
 printf("请输入列数\n");   
 scanf("%d",&n);   
 a=(int\*\*)malloc(sizeof(int\*)\*m);   
 for(i=0;i<m;i++)   
 a[i]=(int\*)malloc(sizeof(int)\*n);   
 for(i=0;i<m;i++) {  
 for(j=0;j<n;j++) {  
 printf("%p\n",&a[i][j]); //输出每个元素地址，每行的列与列之间的地址时连续的，行与行之间的地址不连续  
 }  
 }  
 for(i=0;i<m;i++)   
 free(a[i]);  
   
 free(a);   
 return 0;   
}

#include<stdio.h>   
#include<stdlib.h>   
   
int main()   
{   
 int i,j;   
 //申请一个3行2列的整型数组   
 int (\*a)[2]=(int(\*)[2])malloc(sizeof(int)\*3\*2);   
 for(i=0;i<3;i++) {  
 for(j=0;j<2;j++) {   
 printf("%p\n",&a[i][j]); //输出数组每个元素地址，每个元素的地址是连续的  
 }  
 }  
   
 free(a);  
 return 0;   
}

**vector**

//二维vector  
vector<vector <int> > ivec(m ,vector<int>(n)); //m\*n的二维vector  
   
//动态创建m\*n的二维vector  
//方法一：  
vector<vector <int> > ivec;  
ivec.resize(m);  
for(int i=0;i<m;i++)   
 ivec[i].resize(n);  
   
//方法二：  
vector<vector <int> > ivec;  
ivec.resize(m,vector<int>(n));

### set

**begin()** ,返回set容器的第一个元素

**end()** ,返回set容器的最后一个元素

**clear()** ,删除set容器中的所有的元素

**empty()**,判断set容器是否为空

**max\_size()** ,返回set容器可能包含的元素最大个数

**size()** ,返回当前set容器中的元素个数

**rbegin** ,返回的值和end()相同

**rend()**,返回的值和rbegin()相同

**count()** 用来查找set中某个某个键值出现的次数。

**equal\_range()** ，返回一对定位器，分别表示第一个大于或等于给定关键值的元素和 第一个大于给定关键值的元素，这个返回值是一个pair类型，如果这一对定位器中哪个返回失败，就会等于end()的值。

**erase(iterator)** ,删除定位器iterator指向的值

**erase(first,second)**,删除定位器first和second之间的值

**erase(key\_value)**,删除键值key\_value的值

**find()** ，返回给定值值得定位器，如果没找到则返回end()。

**insert(key\_value);** 将key*value插入到set中 ，返回值是pair<set::iterator,bool>，bool标志着插入是否成功，而iterator代表插入的位置，若key*value已经在set中，则iterator表示的key\_value在set中的位置。

**inset(first,second);**将定位器first到second之间的元素插入到set中，返回值是void.

**lower*bound(key*value)** ，返回第一个大于等于key\_value的定位器

**upper*bound(key*value)，**返回最后一个大于等于key\_value的定位器

### map

**插入操作**

使用[ ]进行单个插入

map<int, string> ID\_Name;  
// 如果已经存在键值2015，则会作赋值修改操作，如果没有则插入  
ID\_Name[2015] = "Tom";1234

使用insert进行单个和多个插入 (insert共有4个重载函数：

// 插入单个键值对，并返回插入位置和成功标志，插入位置已经存在值时，插入失败  
pair<iterator,bool> insert (const value\_type& val);  
//在指定位置插入，在不同位置插入效率是不一样的，因为涉及到重排  
iterator insert (const\_iterator position, const value\_type& val);  
// 插入多个  
void insert (InputIterator first, InputIterator last);  
//c++11开始支持，使用列表插入多个   
void insert (initializer\_list<value\_type> il);

**取值**

Map中元素取值主要有at和[ ]两种操作，at会作下标检查，而[]不会。

map<int, string> ID\_Name;  
//ID\_Name中没有关键字2016，使用[]取值会导致插入  
//因此，下面语句不会报错，但打印结果为空  
cout<<ID\_Name[2016].c\_str()<<endl;  
//使用at会进行关键字检查，因此下面语句会报错  
ID\_Name.at(2016) = "Bob";

**容量查询**

// 查询map是否为空  
bool empty();  
// 查询map中键值对的数量  
size\_t size();  
// 查询map所能包含的最大键值对数量，和系统和应用库有关。  
// 此外，这并不意味着用户一定可以存这么多，很可能还没达到就已经开辟内存失败了  
size\_t max\_size();  
// 查询关键字为key的元素的个数，在map里结果非0即1  
size\_t count( const Key& key ) const; //

**迭代器**

共有八个获取迭代器的函数：**begin, end, rbegin,rend** 以及对应的 **cbegin, cend, crbegin,crend**。

二者的区别在于，后者一定返回 const*iterator，而前者则根据map的类型返回iterator 或者 const*iterator。const情况下，不允许对值进行修改。如下面代码所示：

map<int,int>::iterator it;  
map<int,int> mmap;  
const map<int,int> const\_mmap;  
it = mmap.begin(); //iterator  
mmap.cbegin(); //const\_iterator  
const\_mmap.begin(); //const\_iterator  
const\_mmap.cbegin(); //const\_iterator123456789

返回的迭代器可以进行加减操作，此外，如果map为空，则 begin = end。

**删除**

// 删除迭代器指向位置的键值对，并返回一个指向下一元素的迭代器  
iterator erase( iterator pos )  
// 删除一定范围内的元素，并返回一个指向下一元素的迭代器  
iterator erase( const\_iterator first, const\_iterator last );  
// 根据Key来进行删除， 返回删除的元素数量，在map里结果非0即1  
size\_t erase( const key\_type& key );  
// 清空map，清空后的size为0  
void clear();

**交换**

// 就是两个map的内容互换  
void swap( map& other );

**顺序比较**

// 比较两个关键字在map中位置的先后  
key\_compare key\_comp() const;

**查找**

// 关键字查询，找到则返回指向该关键字的迭代器，否则返回指向end的迭代器  
// 根据map的类型，返回的迭代器为 iterator 或者 const\_iterator  
iterator find (const key\_type& k);  
const\_iterator find (const key\_type& k) const;

**操作符**

operator: == != < <= > >=  
**注意** 对于==运算符, 只有键值对以及顺序完全相等才算成立。

### unordered\_map

**查找元素是否存在**

若有unordered\_map <int, int> mp;查找x是否在map中  
 方法1: 若存在 mp.find(x)!=mp.end()  
 方法2: 若存在 mp.count(x)!=0123

**插入数据**

mp.insert(Map::value\_type(1,"Raoul"));1

**遍历map**

unordered\_map<key,T>::iterator it;  
 (\*it).first; //the key value  
 (\*it).second //the mapped value  
 for(unordered\_map<key,T>::iterator iter=mp.begin();iter!=mp.end();iter++)  
 cout<<"key value is"<<iter->first<<" the mapped value is "<< iter->second;  
 //也可以这样  
 for(auto& v : mp)  
 print v.first and v.second

### bitset

C++的 bitset 在 bitset 头文件中，它是一种类似数组的结构，它的每一个元素只能是０或１，每个元素仅用１bit空间。  
**bitset数组与vector数组区别**  
bitset声明数组:bitset<100> number[10]  
vector声明数组:vector number[10];  
**bitset<每个bitset元素的长度(没有占满前面全部自动补0)> 元素**  
**bitset内置转化函数：可将bitset转化为string,unsigned long,unsigned long long。**

#### 构造

bitset<4> bitset1;　　//无参构造，长度为４，默认每一位为０  
 bitset<8> bitset2(12);　　//长度为８，二进制保存，前面用０补充  
 string s = "100101";  
 bitset<10> bitset3(s);　　//长度为10，前面用０补充  
   
 char s2[] = "10101";  
 bitset<13> bitset4(s2);　　//长度为13，前面用０补充  
 cout << bitset1 << endl;　　//0000  
 cout << bitset2 << endl;　　//00001100  
 cout << bitset3 << endl;　　//0000100101  
 cout << bitset4 << endl;　　//0000000010101

#### 函数

bitset<8> foo ("10011011");  
 cout << foo.count() << endl;　　//5　　（count函数用来求bitset中1的位数，foo中共有５个１  
 cout << foo.size() << endl;　　 //8　　（size函数用来求bitset的大小，一共有８位  
 cout << foo.test(0) << endl;　　//true　　（test函数用来查下标处的元素是０还是１，并返回false或true，此处foo[0]为１，返回true  
 cout << foo.test(2) << endl;　　//false　　（同理，foo[2]为０，返回false  
 cout << foo.any() << endl;　　//true　　（any函数检查bitset中是否有１  
 cout << foo.none() << endl;　　//false　　（none函数检查bitset中是否没有１  
 cout << foo.all() << endl;　　//false　　（all函数检查bitset中是全部为１

[2019-2020 ICPC Asia Taipei-Hsinchu Regional Contest（H](https://blog.csdn.net/chitudexixi/article/details/109453360)

#### H

#include <bits/stdc++.h>  
#define ll long long  
using namespace std;  
int t,n,m;  
char str[1010];  
bitset<500> number[30];  
int main() {  
 ios::sync\_with\_stdio(false); cin.tie(0); cout.tie(0);  
 //freopen("test.in","r",stdin);  
 //freopen("test.out","w",stdout);  
 scanf("%d",&t);  
 while(t--)  
 {  
 scanf("%d %d",&n,&m);  
 for(int i=0;i<m;i++)  
 {  
 scanf("%s",str);  
 number[i]=bitset<500>(str);  
 }  
 int len=1<<m,ans=m+1;  
 for(int i=1;i<len;i++)  
 {  
 int t=i,s=0;  
 bitset<500> num(0);  
 for(int j=0;j<m&&t>0;j++)  
 {  
 if(t&1)   
 {  
 num=num|number[j];  
 s++;  
 }  
 t>>=1;  
 }  
 if(num.count()==n) ans=min(ans,s);  
 }  
 if(ans==m+1) printf("-1\n");  
 else printf("%d\n",ans);  
 }  
 return 0;  
}

## 计算几何

### zyx的计算几何

const double eps = 1e-9;  
const double PI = acos(-1.0);  
struct Line;  
struct Point {  
 double x, y;  
 Point() { x = y = 0; }  
 Point(const Line &a);  
 Point(const double &a, const double &b) : x(a), y(b) {}  
 Point operator+(const Point &a) const {  
 return {x + a.x, y + a.y};  
 }  
 Point operator-(const Point &a) const {  
 return {x - a.x, y - a.y};  
 }  
 Point operator\*(const double &a) const {  
 return {x \* a, y \* a};  
 }  
 Point operator/(const double &d) const {  
 return {x / d, y / d};  
 }  
 bool operator==(const Point &a) const {  
 return abs(x - a.x) + abs(y - a.y) < eps;  
 }  
 void standardize() {  
 \*this = \*this / sqrt(x \* x + y \* y);  
 }  
};  
Point normal(const Point &a) { return Point(-a.y, a.x); }  
double dist(const Point &a, const Point &b) {  
 return sqrt((a.x - b.x) \* (a.x - b.x) + (a.y - b.y) \* (a.y - b.y));  
}  
double dist2(const Point &a, const Point &b) {  
 return (a.x - b.x) \* (a.x - b.x) + (a.y - b.y) \* (a.y - b.y);  
}  
struct Line {  
 Point s, t;  
 Line() {}  
 Line(const Point &a, const Point &b) : s(a), t(b) {}  
};  
struct circle {  
 Point o;  
 double r;  
 circle() {}  
 circle(Point P, double R = 0) { o = P, r = R; }  
};  
double length(const Point &p) {  
 return sqrt(p.x \* p.x + p.y \* p.y);  
}  
double length(const Line &l) {  
 Point p(l);  
 return length(p);  
}  
Point::Point(const Line &a) { \*this = a.t - a.s; }  
istream &operator>>(istream &in, Point &a) {  
 in >> a.x >> a.y;  
 return in;  
}  
double dot(const Point &a, const Point &b) {  
 return a.x \* b.x + a.y \* b.y;  
}  
double det(const Point &a, const Point &b) {  
 return a.x \* b.y - a.y \* b.x;  
}  
int sgn(const double &x) { return fabs(x) < eps ? 0 : (x > 0 ? 1 : -1); }  
double sqr(const double &x) { return x \* x; }  
Point rotate(const Point &a, const double &ang) {  
 double x = cos(ang) \* a.x - sin(ang) \* a.y;  
 double y = sin(ang) \* a.x + cos(ang) \* a.y;  
 return {x, y};  
}  
//点在线段上 <=0 包含端点  
bool sp\_on(const Line &seg, const Point &p) {  
 Point a = seg.s, b = seg.t;  
 return !sgn(det(p - a, b - a)) && sgn(dot(p - a, p - b)) <= 0;  
}  
bool lp\_on(const Line &line, const Point &p) {  
 Point a = line.s, b = line.t;  
 return !sgn(det(p - a, b - a));  
}  
//等于不包含共线  
int andrew(Point \*point, Point \*convex, int n) {  
 sort(point, point + n, [](Point a, Point b) {  
 if (a.x != b.x) return a.x < b.x;  
 return a.y < b.y;  
 });  
 int top = 0;  
 for (int i = 0; i < n; i++) {  
 while ((top > 1) && det(convex[top - 1] - convex[top - 2], point[i] - convex[top - 1]) <= 0)  
 top--;  
 convex[top++] = point[i];  
 }  
 int tmp = top;  
 for (int i = n - 2; i >= 0; i--) {  
 while ((top > tmp) && det(convex[top - 1] - convex[top - 2], point[i] - convex[top - 1]) <= 0)  
 top--;  
 convex[top++] = point[i];  
 }  
 if (n > 1) top--;  
 return top;  
}  
double slope(const Point &a, const Point &b) {  
 return (a.y - b.y) / (a.x - b.x);  
}  
double slope(const Line &a) {  
 return slope(a.s, a.t);  
}  
Point ll\_intersection(const Line &a, const Line &b) {  
 double s1 = det(Point(a), b.s - a.s), s2 = det(Point(a), b.t - a.s);  
 return (b.s \* s2 - b.t \* s1) / (s2 - s1);  
}  
int ss\_cross(const Line &a, const Line &b, Point &p) {  
 int d1 = sgn(det(a.t - a.s, b.s - a.s));  
 int d2 = sgn(det(a.t - a.s, b.t - a.s));  
 int d3 = sgn(det(b.t - b.s, a.s - b.s));  
 int d4 = sgn(det(b.t - b.s, a.t - b.s));  
 if ((d1 ^ d2) == -2 && (d3 ^ d4) == -2) {  
 p = ll\_intersection(a, b);  
 return 1;  
 }  
 if (!d1 && sp\_on(a, b.s)) {  
 p = b.s;  
 return 2;  
 }  
 if (!d2 && sp\_on(a, b.t)) {  
 p = b.t;  
 return 2;  
 }  
 if (!d3 && sp\_on(b, a.s)) {  
 p = a.s;  
 return 2;  
 }  
 if (!d4 && sp\_on(b, a.t)) {  
 p = a.t;  
 return 2;  
 }  
 return 0;  
}  
Point project(const Line &l, const Point &p) {  
 Point base(l);  
 double r = dot(base, p - l.s) / sqr(length(base));  
 return l.s + (base \* r);  
}  
double sp\_dist(const Line &l, const Point &p) {  
 if (l.s == l.t) return dist(l.s, p);  
 Point x = p - l.s, y = p - l.t, z = l.t - l.s;  
 if (sgn(dot(x, z)) < 0)return length(x);//P距离A更近  
 if (sgn(dot(y, z)) > 0)return length(y);//P距离B更近  
 return abs(det(x, z) / length(z));//面积除以底边长  
}  
double lp\_dist(const Line &l, const Point &p) {  
 Point x = p - l.s, y = p - l.t, z = l.t - l.s;  
 return abs(det(x, z) / length(z));//面积除以底边长  
}  
int lc\_cross(const Line &l, const Point &a, const double &r, pair<Point, Point> &ans) {  
 int num = 0;  
 Point pr = project(l, a);  
 double dis = dist(pr, a);  
 double tmp = r \* r - dis \* dis;  
 if (sgn(tmp) == 1) num = 2;  
 else if (sgn(tmp) == 0) num = 1;  
 else return 0;  
 double base = sqrt(r \* r - dis \* dis);  
 Point e(l);  
 e.standardize();  
 e = e \* base;  
 ans = make\_pair(pr + e, pr - e);  
 return num;  
}  
int cc\_cross(const Point &c1, const double &r1, const Point &c2, const double &r2, pair<Point, Point> &ans) {  
 double x1 = c1.x, x2 = c2.x, y1 = c1.y, y2 = c2.y;  
 double d = length(c1 - c2);  
 if (sgn(fabs(r1 - r2) - d) > 0) return -1; //内含  
 if (sgn(r1 + r2 - d) < 0) return 0; //相离  
 double a = r1 \* (x1 - x2) \* 2, b = r1 \* (y1 - y2) \* 2, c = r2 \* r2 - r1 \* r1 - d \* d;  
 double p = a \* a + b \* b, q = -a \* c \* 2, r = c \* c - b \* b;  
 double cosa, sina, cosb, sinb;  
 //One Intersection  
 if (sgn(d - (r1 + r2)) == 0 || sgn(d - fabs(r1 - r2)) == 0) {  
 cosa = -q / p / 2;  
 sina = sqrt(1 - sqr(cosa));  
 Point p0(x1 + r1 \* cosa, y1 + r1 \* sina);  
 if (sgn(dist(p0, c2) - r2)) p0.y = y1 - r1 \* sina;  
 ans = pair<Point, Point>(p0, p0);  
 return 1;  
 }  
 //Two Intersections  
 double delta = sqrt(q \* q - p \* r \* 4);  
 cosa = (delta - q) / p / 2;  
 cosb = (-delta - q) / p / 2;  
 sina = sqrt(1 - sqr(cosa));  
 sinb = sqrt(1 - sqr(cosb));  
 Point p1(x1 + r1 \* cosa, y1 + r1 \* sina);  
 Point p2(x1 + r1 \* cosb, y1 + r1 \* sinb);  
 if (sgn(dist(p1, c2) - r2)) p1.y = y1 - r1 \* sina;  
 if (sgn(dist(p2, c2) - r2)) p2.y = y1 - r1 \* sinb;  
 if (p1 == p2) p1.y = y1 - r1 \* sina;  
 ans = pair<Point, Point>(p1, p2);  
 return 2;  
}  
Point lp\_sym(const Line &l, const Point &p) {  
 return p + (project(l, p) - p) \* 2;  
}  
double alpha(const Point &t1, const Point &t2) {  
 double theta;  
 theta = atan2((double) t2.y, (double) t2.x) - atan2((double) t1.y, (double) t1.x);  
 if (sgn(theta) < 0)  
 theta += 2.0 \* PI;  
 return theta;  
}  
int pip(const Point \*P, const int &n, const Point &a) {//【射线法】判断点A是否在任意多边形Poly以内  
 int cnt = 0;  
 int tmp;  
 for (int i = 1; i <= n; ++i) {  
 int j = i < n ? i + 1 : 1;  
 if (sp\_on(Line(P[i], P[j]), a))return 2;//点在多边形上  
 if (a.y >= min(P[i].y, P[j].y) && a.y < max(P[i].y, P[j].y))//纵坐标在该线段两端点之间  
 tmp = P[i].x + (a.y - P[i].y) / (P[j].y - P[i].y) \* (P[j].x - P[i].x), cnt += sgn(tmp - a.x) > 0;//交点在A右方  
 }  
 return cnt & 1;//穿过奇数次则在多边形以内  
}  
bool pip\_convex\_jud(const Point &a, const Point &L, const Point &R) {//判断AL是否在AR右边  
 return sgn(det(L - a, R - a)) > 0;//必须严格以内  
}  
bool pip\_convex(const Point \*P, const int &n, const Point &a) {//【二分法】判断点A是否在凸多边形Poly以内  
 //点按逆时针给出  
 if (pip\_convex\_jud(P[0], a, P[1]) || pip\_convex\_jud(P[0], P[n - 1], a)) return 0;//在P[0\_1]或P[0\_n-1]外  
 if (sp\_on(Line(P[0], P[1]), a) || sp\_on(Line(P[0], P[n - 1]), a)) return 2;//在P[0\_1]或P[0\_n-1]上  
 int l = 1, r = n - 2;  
 while (l < r) {//二分找到一个位置pos使得P[0]\_A在P[0\_pos],P[0\_(pos+1)]之间  
 int mid = (l + r + 1) >> 1;  
 if (pip\_convex\_jud(P[0], P[mid], a))l = mid;  
 else r = mid - 1;  
 }  
 if (pip\_convex\_jud(P[l], a, P[l + 1]))return 0;//在P[pos\_(pos+1)]外  
 if (sp\_on(Line(P[l], P[l + 1]), a))return 2;//在P[pos\_(pos+1)]上  
 return 1;  
}  
// 多边形是否包含线段  
// 因此我们可以先求出所有和线段相交的多边形的顶点，然后按照X-Y坐标排序(X坐标小的排在前面，对于X坐标相同的点，Y坐标小的排在前面，  
// 这种排序准则也是为了保证水平和垂直情况的判断正确)，这样相邻的两个点就是在线段上相邻的两交点，如果任意相邻两点的中点也在多边形内，  
// 则该线段一定在多边形内。  
int pp\_judge(Point \*A, int n, Point \*B, int m) {//【判断多边形A与多边形B是否相离】  
 for (int i1 = 1; i1 <= n; ++i1) {  
 int j1 = i1 < n ? i1 + 1 : 1;  
 for (int i2 = 1; i2 <= m; ++i2) {  
 int j2 = i2 < m ? i2 + 1 : 1;  
 Point tmp;  
 if (ss\_cross(Line(A[i1], A[j1]), Line(B[i2], B[j2]), tmp)) return 0;//两线段相交  
 if (pip(B, m, A[i1]) || pip(A, n, B[i2]))return 0;//点包含在内  
 }  
 }  
 return 1;  
}  
double area(Point \*P, int n) {//【任意多边形P的面积】  
 double S = 0;  
 for (int i = 1; i <= n; i++) S += det(P[i], P[i < n ? i + 1 : 1]);  
 return S / 2.0;  
}  
Line Q[N];  
int judge(Line L, Point a) { return sgn(det(a - L.s, L.t - L.s)) > 0; }//判断点a是否在直线L的右边  
int halfcut(Line \*L, int n, Point \*P) {//【半平面交】  
 sort(L, L + n, [](const Line &a, const Line &b) {  
 double d = atan2((a.t - a.s).y, (a.t - a.s).x) - atan2((b.t - b.s).y, (b.t - b.s).x);  
 return sgn(d) ? sgn(d) < 0 : judge(a, b.s);  
 });  
 int m = n;  
 n = 0;  
 for (int i = 0; i < m; ++i)  
 if (i == 0 || sgn(atan2(Point(L[i]).y, Point(L[i]).x) - atan2(Point(L[i - 1]).y, Point(L[i - 1]).x)))  
 L[n++] = L[i];  
 int h = 1, t = 0;  
 for (int i = 0; i < n; ++i) {  
 while (h < t && judge(L[i], ll\_intersection(Q[t], Q[t - 1]))) --t;//当队尾两个直线交点不是在直线L[i]上或者左边时就出队  
 while (h < t && judge(L[i], ll\_intersection(Q[h], Q[h + 1]))) ++h;//当队头两个直线交点不是在直线L[i]上或者左边时就出队  
 Q[++t] = L[i];  
 }  
 while (h < t && judge(Q[h], ll\_intersection(Q[t], Q[t - 1]))) --t;  
 while (h < t && judge(Q[t], ll\_intersection(Q[h], Q[h + 1]))) ++h;  
 n = 0;  
 for (int i = h; i <= t; ++i) {  
 P[n++] = ll\_intersection(Q[i], Q[i < t ? i + 1 : h]);  
 }  
 return n;  
}  
Point V1[N], V2[N];  
int mincowski(Point \*P1, int n, Point \*P2, int m, Point \*V) {//【闵可夫斯基和】求两个凸包{P1},{P2}的向量集合{V}={P1+P2}构成的凸包  
 for (int i = 0; i < n; ++i) V1[i] = P1[(i + 1) % n] - P1[i];  
 for (int i = 0; i < m; ++i) V2[i] = P2[(i + 1) % m] - P2[i];  
 int t = 0, i = 0, j = 0;  
 V[t++] = P1[0] + P2[0];  
 while (i < n && j < m) V[t] = V[t - 1] + (sgn(det(V1[i], V2[j])) > 0 ? V1[i++] : V2[j++]), t++;  
 while (i < n) V[t] = V[t - 1] + V1[i++], t++;  
 while (j < m) V[t] = V[t - 1] + V2[j++], t++;  
 return t;  
}  
circle getcircle(const Point &A, const Point &B, const Point &C) {//【三点确定一圆】向量垂心法  
 Point P1 = (A + B) \* 0.5, P2 = (A + C) \* 0.5;  
 Line R1 = Line(P1, P1 + normal(B - A));  
 Line R2 = Line(P2, P2 + normal(C - A));  
 circle O;  
 O.o = ll\_intersection(R1, R2);  
 O.r = length(A - O.o);  
 return O;  
}  
struct ConvexHull {  
 int op;  
 struct cmp {  
 bool operator()(const Point &a, const Point &b) const {  
 return sgn(a.x - b.x) < 0 || sgn(a.x - b.x) == 0 && sgn(a.y - b.y) < 0;  
 }  
 };  
 set<Point, cmp> s;  
 ConvexHull(int o) {  
 op = o;  
 s.clear();  
 }  
 inline int PIP(Point P) {  
 set<Point>::iterator it = s.lower\_bound(Point(P.x, -dinf));//找到第一个横坐标大于P的点  
 if (it == s.end())return 0;  
 if (sgn(it->x - P.x) == 0) return sgn((P.y - it->y) \* op) <= 0;//比较纵坐标大小  
 if (it == s.begin())return 0;  
 set<Point>::iterator j = it, k = it;  
 --j;  
 return sgn(det(P - \*j, \*k - \*j) \* op) >= 0;//看叉姬1  
 }  
 inline int judge(set<Point>::iterator it) {  
 set<Point>::iterator j = it, k = it;  
 if (j == s.begin())return 0;  
 --j;  
 if (++k == s.end())return 0;  
 return sgn(det(\*it - \*j, \*k - \*j) \* op) >= 0;//看叉姬  
 }  
 inline void insert(Point P) {  
 if (PIP(P))return;//如果点P已经在凸壳上或凸包里就不插入了  
 set<Point>::iterator tmp = s.lower\_bound(Point(P.x, -inf));  
 if (tmp != s.end() && sgn(tmp->x - P.x) == 0)s.erase(tmp);//特判横坐标相等的点要去掉  
 s.insert(P);  
 set<Point>::iterator it = s.find(P), p = it;  
 if (p != s.begin()) {  
 --p;  
 while (judge(p)) {  
 set<Point>::iterator temp = p--;  
 s.erase(temp);  
 }  
 }  
 if ((p = ++it) != s.end()) {  
 while (judge(p)) {  
 set<Point>::iterator temp = p++;  
 s.erase(temp);  
 }  
 }  
 }  
} up(1), down(-1);  
int PIC(circle C, Point a) { return sgn(length(a - C.o) - C.r) <= 0; }//判断点A是否在圆C内  
void Random(Point \*P, int n) { for (int i = 0; i < n; ++i)swap(P[i], P[(rand() + 1) % n]); }//随机一个排列  
circle min\_circle(Point \*P, int n) {//【求点集P的最小覆盖圆】 O(n)  
// random\_shuffle(P,P+n);  
 Random(P, n);  
 circle C = circle(P[0], 0);  
 for (int i = 1; i < n; ++i)  
 if (!PIC(C, P[i])) {  
 C = circle(P[i], 0);  
 for (int j = 0; j < i; ++j)  
 if (!PIC(C, P[j])) {  
 C.o = (P[i] + P[j]) \* 0.5, C.r = length(P[j] - C.o);  
 for (int k = 0; k < j; ++k) if (!PIC(C, P[k])) C = getcircle(P[i], P[j], P[k]);  
 }  
 }  
 return C;  
}

### 计算几何全家桶

#include <bits/stdc++.h>  
using namespace std;  
typedef long long ll;  
const ll N = 1 << 20;  
const ll mod = 1e9 + 7;  
const double dinf = 1e99;  
const int inf = 0x3f3f3f3f;  
const ll linf = 0x3f3f3f3f3f3f3f3f;  
const double eps = 1e-9;  
const double PI = acos(-1.0);  
struct Line;  
struct Point {  
 double x, y;  
 Point() { x = y = 0; }  
 Point(const Line &a);  
 Point(const double &a, const double &b) : x(a), y(b) {}  
 Point operator+(const Point &a) const {  
 return {x + a.x, y + a.y};  
 }  
 Point operator-(const Point &a) const {  
 return {x - a.x, y - a.y};  
 }  
 Point operator\*(const double &a) const {  
 return {x \* a, y \* a};  
 }  
 Point operator/(const double &d) const {  
 return {x / d, y / d};  
 }  
 bool operator==(const Point &a) const {  
 return abs(x - a.x) + abs(y - a.y) < eps;  
 }  
 void standardize() {  
 \*this = \*this / sqrt(x \* x + y \* y);  
 }  
};  
Point normal(const Point &a) { return Point(-a.y, a.x); }  
double dist(const Point &a, const Point &b) {  
 return sqrt((a.x - b.x) \* (a.x - b.x) + (a.y - b.y) \* (a.y - b.y));  
}  
double dist2(const Point &a, const Point &b) {  
 return (a.x - b.x) \* (a.x - b.x) + (a.y - b.y) \* (a.y - b.y);  
}  
struct Line {  
 Point s, t;  
 Line() {}  
 Line(const Point &a, const Point &b) : s(a), t(b) {}  
};  
struct circle {  
 Point o;  
 double r;  
 circle() {}  
 circle(Point P, double R = 0) { o = P, r = R; }  
};  
double length(const Point &p) {  
 return sqrt(p.x \* p.x + p.y \* p.y);  
}  
double length(const Line &l) {  
 Point p(l);  
 return length(p);  
}  
Point::Point(const Line &a) { \*this = a.t - a.s; }  
istream &operator>>(istream &in, Point &a) {  
 in >> a.x >> a.y;  
 return in;  
}  
double dot(const Point &a, const Point &b) {  
 return a.x \* b.x + a.y \* b.y;  
}  
double det(const Point &a, const Point &b) {  
 return a.x \* b.y - a.y \* b.x;  
}  
int sgn(const double &x) { return fabs(x) < eps ? 0 : (x > 0 ? 1 : -1); }  
double sqr(const double &x) { return x \* x; }  
Point rotate(const Point &a, const double &ang) {  
 double x = cos(ang) \* a.x - sin(ang) \* a.y;  
 double y = sin(ang) \* a.x + cos(ang) \* a.y;  
 return {x, y};  
}  
//点在线段上 <=0 包含端点  
bool sp\_on(const Line &seg, const Point &p) {  
 Point a = seg.s, b = seg.t;  
 return !sgn(det(p - a, b - a)) && sgn(dot(p - a, p - b)) <= 0;  
}  
bool lp\_on(const Line &line, const Point &p) {  
 Point a = line.s, b = line.t;  
 return !sgn(det(p - a, b - a));  
}  
//等于不包含共线  
int andrew(Point \*point, Point \*convex, int n) {  
 sort(point, point + n, [](Point a, Point b) {  
 if (a.x != b.x) return a.x < b.x;  
 return a.y < b.y;  
 });  
 int top = 0;  
 for (int i = 0; i < n; i++) {  
 while ((top > 1) && det(convex[top - 1] - convex[top - 2], point[i] - convex[top - 1]) <= 0)  
 top--;  
 convex[top++] = point[i];  
 }  
 int tmp = top;  
 for (int i = n - 2; i >= 0; i--) {  
 while ((top > tmp) && det(convex[top - 1] - convex[top - 2], point[i] - convex[top - 1]) <= 0)  
 top--;  
 convex[top++] = point[i];  
 }  
 if (n > 1) top--;  
 return top;  
}  
double slope(const Point &a, const Point &b) {  
 return (a.y - b.y) / (a.x - b.x);  
}  
double slope(const Line &a) {  
 return slope(a.s, a.t);  
}  
Point ll\_intersection(const Line &a, const Line &b) {  
 double s1 = det(Point(a), b.s - a.s), s2 = det(Point(a), b.t - a.s);  
 return (b.s \* s2 - b.t \* s1) / (s2 - s1);  
}  
int ss\_cross(const Line &a, const Line &b, Point &p) {  
 int d1 = sgn(det(a.t - a.s, b.s - a.s));  
 int d2 = sgn(det(a.t - a.s, b.t - a.s));  
 int d3 = sgn(det(b.t - b.s, a.s - b.s));  
 int d4 = sgn(det(b.t - b.s, a.t - b.s));  
 if ((d1 ^ d2) == -2 && (d3 ^ d4) == -2) {  
 p = ll\_intersection(a, b);  
 return 1;  
 }  
 if (!d1 && sp\_on(a, b.s)) {  
 p = b.s;  
 return 2;  
 }  
 if (!d2 && sp\_on(a, b.t)) {  
 p = b.t;  
 return 2;  
 }  
 if (!d3 && sp\_on(b, a.s)) {  
 p = a.s;  
 return 2;  
 }  
 if (!d4 && sp\_on(b, a.t)) {  
 p = a.t;  
 return 2;  
 }  
 return 0;  
}  
Point project(const Line &l, const Point &p) {  
 Point base(l);  
 double r = dot(base, p - l.s) / sqr(length(base));  
 return l.s + (base \* r);  
}  
double sp\_dist(const Line &l, const Point &p) {  
 if (l.s == l.t) return dist(l.s, p);  
 Point x = p - l.s, y = p - l.t, z = l.t - l.s;  
 if (sgn(dot(x, z)) < 0)return length(x);//P距离A更近  
 if (sgn(dot(y, z)) > 0)return length(y);//P距离B更近  
 return abs(det(x, z) / length(z));//面积除以底边长  
}  
double lp\_dist(const Line &l, const Point &p) {  
 Point x = p - l.s, y = p - l.t, z = l.t - l.s;  
 return abs(det(x, z) / length(z));//面积除以底边长  
}  
int lc\_cross(const Line &l, const Point &a, const double &r, pair<Point, Point> &ans) {  
 int num = 0;  
 Point pr = project(l, a);  
 double dis = dist(pr, a);  
 double tmp = r \* r - dis \* dis;  
 if (sgn(tmp) == 1) num = 2;  
 else if (sgn(tmp) == 0) num = 1;  
 else return 0;  
 double base = sqrt(r \* r - dis \* dis);  
 Point e(l);  
 e.standardize();  
 e = e \* base;  
 ans = make\_pair(pr + e, pr - e);  
 return num;  
}  
int cc\_cross(const Point &c1, const double &r1, const Point &c2, const double &r2, pair<Point, Point> &ans) {  
 double x1 = c1.x, x2 = c2.x, y1 = c1.y, y2 = c2.y;  
 double d = length(c1 - c2);  
 if (sgn(fabs(r1 - r2) - d) > 0) return -1; //内含  
 if (sgn(r1 + r2 - d) < 0) return 0; //相离  
 double a = r1 \* (x1 - x2) \* 2, b = r1 \* (y1 - y2) \* 2, c = r2 \* r2 - r1 \* r1 - d \* d;  
 double p = a \* a + b \* b, q = -a \* c \* 2, r = c \* c - b \* b;  
 double cosa, sina, cosb, sinb;  
 //One Intersection  
 if (sgn(d - (r1 + r2)) == 0 || sgn(d - fabs(r1 - r2)) == 0) {  
 cosa = -q / p / 2;  
 sina = sqrt(1 - sqr(cosa));  
 Point p0(x1 + r1 \* cosa, y1 + r1 \* sina);  
 if (sgn(dist(p0, c2) - r2)) p0.y = y1 - r1 \* sina;  
 ans = pair<Point, Point>(p0, p0);  
 return 1;  
 }  
 //Two Intersections  
 double delta = sqrt(q \* q - p \* r \* 4);  
 cosa = (delta - q) / p / 2;  
 cosb = (-delta - q) / p / 2;  
 sina = sqrt(1 - sqr(cosa));  
 sinb = sqrt(1 - sqr(cosb));  
 Point p1(x1 + r1 \* cosa, y1 + r1 \* sina);  
 Point p2(x1 + r1 \* cosb, y1 + r1 \* sinb);  
 if (sgn(dist(p1, c2) - r2)) p1.y = y1 - r1 \* sina;  
 if (sgn(dist(p2, c2) - r2)) p2.y = y1 - r1 \* sinb;  
 if (p1 == p2) p1.y = y1 - r1 \* sina;  
 ans = pair<Point, Point>(p1, p2);  
 return 2;  
}  
Point lp\_sym(const Line &l, const Point &p) {  
 return p + (project(l, p) - p) \* 2;  
}  
double alpha(const Point &t1, const Point &t2) {  
 double theta;  
 theta = atan2((double) t2.y, (double) t2.x) - atan2((double) t1.y, (double) t1.x);  
 if (sgn(theta) < 0)  
 theta += 2.0 \* PI;  
 return theta;  
}  
int pip(const Point \*P, const int &n, const Point &a) {//【射线法】判断点A是否在任意多边形Poly以内  
 int cnt = 0;  
 int tmp;  
 for (int i = 1; i <= n; ++i) {  
 int j = i < n ? i + 1 : 1;  
 if (sp\_on(Line(P[i], P[j]), a))return 2;//点在多边形上  
 if (a.y >= min(P[i].y, P[j].y) && a.y < max(P[i].y, P[j].y))//纵坐标在该线段两端点之间  
 tmp = P[i].x + (a.y - P[i].y) / (P[j].y - P[i].y) \* (P[j].x - P[i].x), cnt += sgn(tmp - a.x) > 0;//交点在A右方  
 }  
 return cnt & 1;//穿过奇数次则在多边形以内  
}  
bool pip\_convex\_jud(const Point &a, const Point &L, const Point &R) {//判断AL是否在AR右边  
 return sgn(det(L - a, R - a)) > 0;//必须严格以内  
}  
bool pip\_convex(const Point \*P, const int &n, const Point &a) {//【二分法】判断点A是否在凸多边形Poly以内  
 //点按逆时针给出  
 if (pip\_convex\_jud(P[0], a, P[1]) || pip\_convex\_jud(P[0], P[n - 1], a)) return 0;//在P[0\_1]或P[0\_n-1]外  
 if (sp\_on(Line(P[0], P[1]), a) || sp\_on(Line(P[0], P[n - 1]), a)) return 2;//在P[0\_1]或P[0\_n-1]上  
 int l = 1, r = n - 2;  
 while (l < r) {//二分找到一个位置pos使得P[0]\_A在P[0\_pos],P[0\_(pos+1)]之间  
 int mid = (l + r + 1) >> 1;  
 if (pip\_convex\_jud(P[0], P[mid], a))l = mid;  
 else r = mid - 1;  
 }  
 if (pip\_convex\_jud(P[l], a, P[l + 1]))return 0;//在P[pos\_(pos+1)]外  
 if (sp\_on(Line(P[l], P[l + 1]), a))return 2;//在P[pos\_(pos+1)]上  
 return 1;  
}  
// 多边形是否包含线段  
// 因此我们可以先求出所有和线段相交的多边形的顶点，然后按照X-Y坐标排序(X坐标小的排在前面，对于X坐标相同的点，Y坐标小的排在前面，  
// 这种排序准则也是为了保证水平和垂直情况的判断正确)，这样相邻的两个点就是在线段上相邻的两交点，如果任意相邻两点的中点也在多边形内，  
// 则该线段一定在多边形内。  
int pp\_judge(Point \*A, int n, Point \*B, int m) {//【判断多边形A与多边形B是否相离】  
 for (int i1 = 1; i1 <= n; ++i1) {  
 int j1 = i1 < n ? i1 + 1 : 1;  
 for (int i2 = 1; i2 <= m; ++i2) {  
 int j2 = i2 < m ? i2 + 1 : 1;  
 Point tmp;  
 if (ss\_cross(Line(A[i1], A[j1]), Line(B[i2], B[j2]), tmp)) return 0;//两线段相交  
 if (pip(B, m, A[i1]) || pip(A, n, B[i2]))return 0;//点包含在内  
 }  
 }  
 return 1;  
}  
double area(Point \*P, int n) {//【任意多边形P的面积】  
 double S = 0;  
 for (int i = 1; i <= n; i++) S += det(P[i], P[i < n ? i + 1 : 1]);  
 return S / 2.0;  
}  
Line Q[N];  
int judge(Line L, Point a) { return sgn(det(a - L.s, L.t - L.s)) > 0; }//判断点a是否在直线L的右边  
int halfcut(Line \*L, int n, Point \*P) {//【半平面交】  
 sort(L, L + n, [](const Line &a, const Line &b) {  
 double d = atan2((a.t - a.s).y, (a.t - a.s).x) - atan2((b.t - b.s).y, (b.t - b.s).x);  
 return sgn(d) ? sgn(d) < 0 : judge(a, b.s);  
 });  
 int m = n;  
 n = 0;  
 for (int i = 0; i < m; ++i)  
 if (i == 0 || sgn(atan2(Point(L[i]).y, Point(L[i]).x) - atan2(Point(L[i - 1]).y, Point(L[i - 1]).x)))  
 L[n++] = L[i];  
 int h = 1, t = 0;  
 for (int i = 0; i < n; ++i) {  
 while (h < t && judge(L[i], ll\_intersection(Q[t], Q[t - 1]))) --t;//当队尾两个直线交点不是在直线L[i]上或者左边时就出队  
 while (h < t && judge(L[i], ll\_intersection(Q[h], Q[h + 1]))) ++h;//当队头两个直线交点不是在直线L[i]上或者左边时就出队  
 Q[++t] = L[i];  
 }  
 while (h < t && judge(Q[h], ll\_intersection(Q[t], Q[t - 1]))) --t;  
 while (h < t && judge(Q[t], ll\_intersection(Q[h], Q[h + 1]))) ++h;  
 n = 0;  
 for (int i = h; i <= t; ++i) {  
 P[n++] = ll\_intersection(Q[i], Q[i < t ? i + 1 : h]);  
 }  
 return n;  
}  
Point V1[N], V2[N];  
int mincowski(Point \*P1, int n, Point \*P2, int m, Point \*V) {//【闵可夫斯基和】求两个凸包{P1},{P2}的向量集合{V}={P1+P2}构成的凸包  
 for (int i = 0; i < n; ++i) V1[i] = P1[(i + 1) % n] - P1[i];  
 for (int i = 0; i < m; ++i) V2[i] = P2[(i + 1) % m] - P2[i];  
 int t = 0, i = 0, j = 0;  
 V[t++] = P1[0] + P2[0];  
 while (i < n && j < m) V[t] = V[t - 1] + (sgn(det(V1[i], V2[j])) > 0 ? V1[i++] : V2[j++]), t++;  
 while (i < n) V[t] = V[t - 1] + V1[i++], t++;  
 while (j < m) V[t] = V[t - 1] + V2[j++], t++;  
 return t;  
}  
circle getcircle(const Point &A, const Point &B, const Point &C) {//【三点确定一圆】向量垂心法  
 Point P1 = (A + B) \* 0.5, P2 = (A + C) \* 0.5;  
 Line R1 = Line(P1, P1 + normal(B - A));  
 Line R2 = Line(P2, P2 + normal(C - A));  
 circle O;  
 O.o = ll\_intersection(R1, R2);  
 O.r = length(A - O.o);  
 return O;  
}  
struct ConvexHull {  
 int op;  
 struct cmp {  
 bool operator()(const Point &a, const Point &b) const {  
 return sgn(a.x - b.x) < 0 || sgn(a.x - b.x) == 0 && sgn(a.y - b.y) < 0;  
 }  
 };  
 set<Point, cmp> s;  
 ConvexHull(int o) {  
 op = o;  
 s.clear();  
 }  
 inline int PIP(Point P) {  
 set<Point>::iterator it = s.lower\_bound(Point(P.x, -dinf));//找到第一个横坐标大于P的点  
 if (it == s.end())return 0;  
 if (sgn(it->x - P.x) == 0) return sgn((P.y - it->y) \* op) <= 0;//比较纵坐标大小  
 if (it == s.begin())return 0;  
 set<Point>::iterator j = it, k = it;  
 --j;  
 return sgn(det(P - \*j, \*k - \*j) \* op) >= 0;//看叉姬1  
 }  
 inline int judge(set<Point>::iterator it) {  
 set<Point>::iterator j = it, k = it;  
 if (j == s.begin())return 0;  
 --j;  
 if (++k == s.end())return 0;  
 return sgn(det(\*it - \*j, \*k - \*j) \* op) >= 0;//看叉姬  
 }  
 inline void insert(Point P) {  
 if (PIP(P))return;//如果点P已经在凸壳上或凸包里就不插入了  
 set<Point>::iterator tmp = s.lower\_bound(Point(P.x, -inf));  
 if (tmp != s.end() && sgn(tmp->x - P.x) == 0)s.erase(tmp);//特判横坐标相等的点要去掉  
 s.insert(P);  
 set<Point>::iterator it = s.find(P), p = it;  
 if (p != s.begin()) {  
 --p;  
 while (judge(p)) {  
 set<Point>::iterator temp = p--;  
 s.erase(temp);  
 }  
 }  
 if ((p = ++it) != s.end()) {  
 while (judge(p)) {  
 set<Point>::iterator temp = p++;  
 s.erase(temp);  
 }  
 }  
 }  
} up(1), down(-1);  
int PIC(circle C, Point a) { return sgn(length(a - C.o) - C.r) <= 0; }//判断点A是否在圆C内  
void Random(Point \*P, int n) { for (int i = 0; i < n; ++i)swap(P[i], P[(rand() + 1) % n]); }//随机一个排列  
circle min\_circle(Point \*P, int n) {//【求点集P的最小覆盖圆】 O(n)  
// random\_shuffle(P,P+n);  
 Random(P, n);  
 circle C = circle(P[0], 0);  
 for (int i = 1; i < n; ++i)  
 if (!PIC(C, P[i])) {  
 C = circle(P[i], 0);  
 for (int j = 0; j < i; ++j)  
 if (!PIC(C, P[j])) {  
 C.o = (P[i] + P[j]) \* 0.5, C.r = length(P[j] - C.o);  
 for (int k = 0; k < j; ++k) if (!PIC(C, P[k])) C = getcircle(P[i], P[j], P[k]);  
 }  
 }  
 return C;  
}

### 自适应辛普森

double f(double x) {  
}  
double simpson(double l, double r) {  
 double mid = (l + r) / 2;  
 return (r - l) \* (f(l) + 4 \* f(mid) + f(r)) / 6; // 辛普森公式  
}  
double asr(double l, double r, double EPS, double ans) {  
 double mid = (l + r) / 2;  
 double fl = simpson(l, mid), fr = simpson(mid, r);  
 if (abs(fl + fr - ans) <= 15 \* EPS)  
 return fl + fr + (fl + fr - ans) / 15; // 足够相似的话就直接返回  
 return asr(l, mid, EPS / 2, fl) +  
 asr(mid, r, EPS / 2, fr); // 否则分割成两段递归求解  
}

## 数据结构

### kruskal重构树

int pa[N];  
void init(int n) {  
 for (int i = 0; i <= n; i++) {  
 pa[i] = i;  
 }  
}  
int find(int a) {  
 return pa[a] == a ? a : pa[a] = find(pa[a]);  
}  
int kruskal() {  
 int kcnt = n;  
 init(n);  
 sort(e + 1, e + 1 + m, [](edge a, edge b) { return a.l < b.l; });  
 for (int i = 1; i <= m; i++) {  
 int u = find(e[i].from);  
 int v = find(e[i].to);  
 if (u == v) continue;  
 w[++kcnt] = e[i].l;  
 pa[kcnt] = pa[u] = pa[v] = kcnt;  
 g[u].push\_back(kcnt);  
 g[v].push\_back(kcnt);  
 g[kcnt].push\_back(u);  
 g[kcnt].push\_back(v);  
 }  
 return kcnt;  
}  
cpp

### 普通莫队

#include <bits/stdc++.h>  
using namespace std;  
const int N = 1e6 + 10, M = 1e6 + 10;  
int a[N];  
struct node {   
 int id, l, r;  
} mp[M];  
int len;  
int ans[M], cnt[1000010];  
int getNum(int l) {  
 return l / len;  
}  
//左指针的分块，右指针的大小  
bool cmp (const node &a, const node & b) {  
 if(getNum(a.l) == getNum(b.l)) return a.r < b.r;  
 return a.l < b.l;  
}  
/\* 奇偶优化  
struct node {  
 int l, r, id;  
 bool operator<(const node &x) const {  
 if (l / unit != x.l / unit) return l < x.l;  
 if ((l / unit) & 1)  
 return r < x.r; // 注意这里和下面一行不能写小于（大于）等于  
 return r > x.r;  
 }  
};  
\*/  
void add(int x, int& res) {  
 if(cnt[x] == 0) res++;  
 cnt[x] ++;  
}  
void del(int x, int& res) {  
 cnt[x] --;  
 if(cnt[x] == 0) res --;  
}  
int main() {  
 ios::sync\_with\_stdio(0); cin.tie(0); cout.tie(0);  
   
 int n;  
 cin >> n;  
 for(int i = 1; i <= n; ++ i) {  
 cin >> a[i];  
 }  
 int m;  
 cin >> m;  
 len = sqrt((double)n \* n / m);  
 for(int i = 1; i <= m; ++ i) {  
 mp[i].id = i;  
 cin >> mp[i].l >> mp[i].r;  
 }  
 sort(mp + 1, mp + m + 1, cmp);  
   
 //离线处理询问   
 int res = 0, i = 0, j = 0;  
 for(int k = 1; k <= m; ++ k) {  
 int id = mp[k].id, l = mp[k].l, r = mp[k].r;  
 while(j < r) add(a[++j], res);  
 while(j > r) del(a[j--], res);  
 while(i < l) del(a[i++], res);  
 while(i > l) add(a[--i], res);  
 ans[id] = res;  
 }  
   
 for(int i = 1; i <= m; ++ i) {  
 cout << ans[i] << endl;  
 }  
 return 0;  
}

### 带修莫队

#include <bits/stdc++.h>  
using namespace std;  
const int N = 10010;  
int a[N], cnt[1000010], ans[N];  
int len, mq, mc;  
struct Query {  
 int id, l, r, t;  
} q[N];  
struct Modify {  
 int p, c;  
} c[N];  
int getNum(int x) {  
 return x / len;  
}  
// l所在块的编号，r所在块的编号，t升序  
bool cmp(const Query& a, const Query& b) {  
 if(getNum(a.l) == getNum(b.l) && getNum(a.r) == getNum(b.r)) {  
 return a.t < b.t;  
 }   
 if(getNum(a.l) == getNum(b.l)) return a.r < b.r;  
 return a.l < b.l;   
}  
void add(int x, int& res) {  
 if (!cnt[x]) res ++ ;  
 cnt[x] ++ ;  
}  
void del(int x, int& res) {  
 cnt[x] -- ;  
 if (!cnt[x]) res -- ;  
}  
  
int main() {  
 ios::sync\_with\_stdio(0); cin.tie(0); cout.tie(0);  
   
 int n, m;  
 cin >> n >> m;  
 char op;  
 int x, y;  
 for(int i = 1; i <= n; ++ i) {  
 cin >> a[i];  
 }  
 for(int i = 1; i <= m; ++ i) {  
 cin >> op >> x >> y;  
 if (op == 'Q') q[++ mq] = {mq, x, y, mc};  
 else c[ ++ mc] = {x, y};  
 }  
   
 ///  
 len = cbrt((double)n \* mc) + 1;  
 sort(q + 1, q + mq + 1, cmp);  
   
 int i = 1, j = 0, t = 0, res = 0;  
 for(int k = 1; k <= mq; ++ k) {  
 int id = q[k].id, l = q[k].l, r = q[k].r, tm = q[k].t;  
 while(j < r) add(a[++ j], res);  
 while(j > r) del(a[j --], res);  
 while(i < l) del(a[i ++], res);  
 while(i > l) add(a[-- i], res);  
 while(t < tm) {  
 ++ t;  
 if(c[t].p >= i && c[t].p <= j) {  
 del(a[c[t].p], res);  
 add(c[t].c, res);  
 }  
 swap(a[c[t].p], c[t].c);  
 }  
 while(t > tm) {  
 if(c[t].p >= i && c[t].p <= j) {  
 del(a[c[t].p], res);  
 add(c[t].c, res);  
 }  
 swap(a[c[t].p], c[t].c);  
 -- t;  
 }  
 ans[id] = res;  
 }  
   
 for(int i = 1; i <= mq; ++ i) {  
 cout << ans[i] << endl;  
 }  
}

### 线段树合并分裂

ll nodetot, recycnt, bac[N << 5], ch[N << 5][2], rt[N];  
ll val[N << 5];  
ll newnod() { return (recycnt ? bac[recycnt--] : ++nodetot); }  
void recyc(ll p) {  
 bac[++recycnt] = p, ch[p][0] = ch[p][1] = val[p] = 0;  
 return;  
}  
void pushdown(ll p) {  
}  
void pushup(ll p) {  
 val[p] = 0;  
 if (ch[p][0]) val[p] += val[ch[p][0]];  
 if (ch[p][1]) val[p] += val[ch[p][1]];  
}  
void modify(ll &p, ll l, ll r, ll pos, ll v) {  
 if (!p) { p = newnod(); }  
 if (l == r) {  
 val[p] += v;  
 return;  
 }  
 ll mid = (l + r) >> 1;  
// pushdown(p);  
 if (pos <= mid) { modify(ch[p][0], l, mid, pos, v); }  
 else { modify(ch[p][1], mid + 1, r, pos, v); }  
 pushup(p);  
 return;  
}  
ll query(ll p, ll l, ll r, ll xl, ll xr) {  
 if (xr < l || r < xl) { return 0; }  
 if (xl <= l && r <= xr) { return val[p]; }  
 ll mid = (l + r) >> 1;  
// pushdown(p);  
 return query(ch[p][0], l, mid, xl, xr) + query(ch[p][1], mid + 1, r, xl, xr);  
}  
ll kth(ll p, ll l, ll r, ll k) {  
 if (l == r) { return l; }  
 ll mid = (l + r) >> 1;  
// pushdown(p);  
 if (val[ch[p][0]] >= k) { return kth(ch[p][0], l, mid, k); }  
 else { return kth(ch[p][1], mid + 1, r, k - val[ch[p][0]]); }  
}  
ll merge(ll x, ll y, ll l, ll r) {  
 if (!x || !y) {  
 return x + y;  
 } // 只有一边有点，不用合并  
 ll p = newnod(); // 创建一个新结点 p  
 if (l == r) { // 边界（某些时候可以省略，见下面一个代码）  
 val[p] = val[x] + val[y];  
 return p;  
 }  
// pushdown(x), pushdown(y);  
 ll mid = (l + r) >> 1;  
 ch[p][0] = merge(ch[x][0], ch[y][0], l, mid);  
 ch[p][1] = merge(ch[x][1], ch[y][1], mid + 1, r);  
 recyc(x), recyc(y); // 垃圾回收  
 pushup(p); // pushup  
 return p;  
}  
void split(ll x, ll &y, ll k) {  
 if (x == 0) return;  
 y = newnod();  
 ll v = val[ch[x][0]];  
// pushdown(x);  
 if (k > v) { split(ch[x][1], ch[y][1], k - v); }  
 else { swap(ch[x][1], ch[y][1]); }  
 if (k < v) { split(ch[x][0], ch[y][0], k); }  
 val[y] = val[x] - k;  
 val[x] = k;  
 return;  
}

### 主席树

#include <bits/stdc++.h>  
using namespace std;  
typedef long long ll;  
const ll N = 1 << 20;  
ll ch[N << 5][2], rt[N], tot;  
ll val[N << 5];  
ll update(ll a, ll b) {  
 return a + b;  
}  
ll build(ll l, ll r) { // 建树  
 ll p = ++tot;  
 if (l == r) {  
 //初始化  
 val[p] = 0;  
 return p;  
 }  
 ll mid = (l + r) >> 1;  
 ch[p][0] = build(l, mid);  
 ch[p][1] = build(mid + 1, r);  
 val[p] = update(val[ch[p][0]], val[ch[p][1]]);  
 return p; // 返回该子树的根节点  
}  
ll modify(ll pre, ll l, ll r, ll pos, ll v) { // 插入操作  
 ll now = ++tot;  
 ch[now][0] = ch[pre][0], ch[now][1] = ch[pre][1];  
 if (l == r) {  
 val[now] = val[pre] + v;  
 return now;  
 }  
 ll mid = (l + r) >> 1;  
 if (pos <= mid)  
 ch[now][0] = modify(ch[now][0], l, mid, pos, v);  
 else  
 ch[now][1] = modify(ch[now][1], mid + 1, r, pos, v);  
 val[now] = update(val[ch[now][0]], val[ch[now][1]]);  
 return now;  
}  
ll kth(ll pre, ll now, ll l, ll r, ll k) { // 查询操作  
 ll mid = (l + r) >> 1;  
 ll x = val[ch[now][0]] - val[ch[pre][0]]; // 通过区间减法得到左儿子的信息  
 if (l == r) return l;  
 if (k <= x) // 说明在左儿子中  
 return kth(ch[pre][0], ch[now][0], l, mid, k);  
 else // 说明在右儿子中  
 return kth(ch[pre][1], ch[now][1], mid + 1, r, k - x);  
}  
ll query(ll pre, ll now, ll l, ll r, ll ql, ll qr) { // 查询操作  
 if (ql <= l && r <= qr) {  
 return val[now] - val[pre];  
 }  
 if (qr < l || r < ql) {  
 return 0;  
 }  
 ll mid = (l + r) >> 1;  
 ll lv = query(ch[pre][0], ch[now][0], l, mid, ql, qr);  
 ll rv = query(ch[pre][1], ch[now][1], mid + 1, r, ql, qr);  
 return update(lv, rv);  
}  
//修改查询记得用rt[]!!!

### LCT

ll ch[N][2], f[N], sum[N], val[N], tag[N], siz[N], siz2[N];  
  
inline void pushup(ll p) {  
 sum[p] = sum[ch[p][0]] ^ sum[ch[p][1]] ^ val[p];  
 siz[p] = siz[ch[p][0]] + siz[ch[p][1]] + 1 + siz2[p];  
}  
  
inline void pushdown(ll p) {  
 if (tag[p]) {  
 if (ch[p][0]) swap(ch[ch[p][0]][0], ch[ch[p][0]][1]), tag[ch[p][0]] ^= 1;  
 if (ch[p][1]) swap(ch[ch[p][1]][0], ch[ch[p][1]][1]), tag[ch[p][1]] ^= 1;  
 tag[p] = 0;  
 }  
}  
  
ll getch(ll x) { return ch[f[x]][1] == x; }  
  
bool isroot(ll x) { return ch[f[x]][0] != x && ch[f[x]][1] != x; }  
  
inline void rotate(ll x) {  
 ll y = f[x], z = f[y], k = getch(x);  
 if (!isroot(y)) ch[z][ch[z][1] == y] = x;  
 // 上面这句一定要写在前面，普通的Splay是不用的，因为 isRoot (后面会讲)  
 ch[y][k] = ch[x][!k], f[ch[x][!k]] = y;  
 ch[x][!k] = y, f[y] = x, f[x] = z;  
 pushup(y), pushup(x);  
}  
  
// 从上到下一层一层 pushDown 即可  
void update(ll p) {  
 if (!isroot(p)) update(f[p]);  
 pushdown(p);  
}  
  
inline void splay(ll x) {  
 update(x); // 马上就能看到啦。 在  
 // Splay之前要把旋转会经过的路径上的点都PushDown  
 for (ll fa; fa = f[x], !isroot(x); rotate(x)) {  
 if (!isroot(fa)) rotate(getch(fa) == getch(x) ? fa : x);  
 }  
}  
  
// 回顾一下代码  
inline void access(ll x) {  
 for (ll p = 0; x; p = x, x = f[x]) {  
 splay(x), siz2[x] += siz[ch[x][1]] - siz[p], ch[x][1] = p, pushup(x);  
 }  
}  
  
inline void makeroot(ll p) {  
 access(p);  
 splay(p);  
 swap(ch[p][0], ch[p][1]);  
 tag[p] ^= 1;  
}  
  
inline void split(ll a, ll b) {  
 makeroot(a);  
 access(b);  
 splay(b);  
}  
  
  
inline ll find(ll p) {  
 access(p), splay(p);  
 while (ch[p][0]) pushdown(p), p = ch[p][0];  
 splay(p);  
 return p;  
}  
  
inline void link(ll x, ll y) {  
 makeroot(y);  
 makeroot(x);  
 if (find(y) != x) {  
 f[x] = y;  
 siz2[y] += siz[x];  
 }  
}  
  
inline void cut(ll x, ll y) {  
 makeroot(x);  
 if (find(y) == x && f[y] == x) {  
 ch[x][1] = f[y] = 0;  
 pushup(x);  
 }  
}  
  
void init(int n) {  
 for (int i = 1; i <= n; i++) siz[i] = 1;  
}

### Splay1

#include <bits/stdc++.h>  
using namespace std;  
struct Splay {  
 static const int N = 100005;  
 int rt, tot, fa[N], ch[N][2], val[N], cnt[N], sz[N];  
 // rt=根编号，tot=总节点，fa=父节点编号，ch=左/右儿子编号，val=节点的值，cnt=权值出现次数，sz=子树大小  
 void maintain(int x) { //更新x节点字数大小  
 sz[x] = sz[ch[x][0]] + sz[ch[x][1]] + cnt[x];  
 }  
 bool get(int x) {  
 return x == ch[fa[x]][1];  
 } //返回节点是父亲的0/1-左/右儿子  
 void clear(int x) { //销毁节点x  
 ch[x][0] = ch[x][1] = fa[x] = val[x] = sz[x] = cnt[x] = 0;  
 }  
 void rotate(int x) { //旋转  
 int y = fa[x], z = fa[y], chk = get(x);  
 ch[y][chk] = ch[x][chk ^ 1];  
 fa[ch[x][chk ^ 1]] = y;  
 ch[x][chk ^ 1] = y;  
 fa[y] = x;  
 fa[x] = z;  
 if (z) ch[z][y == ch[z][1]] = x;  
 maintain(x);  
 maintain(y);  
 }  
 void splay(int x) { //将x节点移动到根  
 for (int f = fa[x]; f = fa[x], f; rotate(x))  
 if (fa[f]) rotate(get(x) == get(f) ? f : x);  
 rt = x;  
 }  
 void ins(int k) { //插入  
 if (!rt) {  
 val[++tot] = k;  
 cnt[tot]++;  
 rt = tot;  
 maintain(rt);  
 return;  
 }  
 int cnr = rt, f = 0;  
 while (1) {  
 if (val[cnr] == k) {  
 cnt[cnr]++;  
 maintain(cnr);  
 maintain(f);  
 splay(cnr);  
 break;  
 }  
 f = cnr;  
 cnr = ch[cnr][val[cnr] < k];  
 if (!cnr) {  
 val[++tot] = k;  
 cnt[tot]++;  
 fa[tot] = f;  
 ch[f][val[f] < k] = tot;  
 maintain(tot);  
 maintain(f);  
 splay(tot);  
 break;  
 }  
 }  
 }  
 int rk(int k) { // k权值的排名  
 int res = 0, cnr = rt;  
 while (1) {  
 if (k < val[cnr]) {  
 cnr = ch[cnr][0];  
 } else {  
 res += sz[ch[cnr][0]];  
 if (k == val[cnr]) {  
 splay(cnr);  
 return res + 1;  
 }  
 res += cnt[cnr];  
 cnr = ch[cnr][1];  
 }  
 }  
 }  
 int kth(int k) { //第k名的权值  
 int cnr = rt;  
 while (1) {  
 if (ch[cnr][0] && k <= sz[ch[cnr][0]]) {  
 cnr = ch[cnr][0];  
 } else {  
 k -= cnt[cnr] + sz[ch[cnr][0]];  
 if (k <= 0) {  
 splay(cnr);  
 return val[cnr];  
 }  
 cnr = ch[cnr][1];  
 }  
 }  
 }  
 int pre() { //前驱节点编号  
 int cnr = ch[rt][0];  
 while (ch[cnr][1]) cnr = ch[cnr][1];  
 splay(cnr);  
 return cnr;  
 } // 若需要得到前驱 tree.ins(x), printf("%d\n", tree.val[tree.pre()]),  
 // tree.del(x);  
 int nxt() { //后驱节点编号  
 int cnr = ch[rt][1];  
 while (ch[cnr][0]) cnr = ch[cnr][0];  
 splay(cnr);  
 return cnr;  
 } // 若需要得到后驱 tree.ins(x), printf("%d\n", tree.val[tree.pre()]),  
 // tree.del(x);  
 void del(int k) { //删除k值  
 rk(k);  
 if (cnt[rt] > 1) {  
 cnt[rt]--;  
 maintain(rt);  
 return;  
 }  
 if (!ch[rt][0] && !ch[rt][1]) {  
 clear(rt);  
 rt = 0;  
 return;  
 }  
 if (!ch[rt][0]) {  
 int cnr = rt;  
 rt = ch[rt][1];  
 fa[rt] = 0;  
 clear(cnr);  
 return;  
 }  
 if (!ch[rt][1]) {  
 int cnr = rt;  
 rt = ch[rt][0];  
 fa[rt] = 0;  
 clear(cnr);  
 return;  
 }  
 int cnr = rt;  
 int x = pre();  
 splay(x);  
 fa[ch[cnr][1]] = x;  
 ch[x][1] = ch[cnr][1];  
 clear(cnr);  
 maintain(rt);  
 }  
} tree;

### splay2

ll ch[N][2], f[N], sum[N], val[N], tag[N], siz[N];  
inline void pushup(ll p) {  
 sum[p] = sum[ch[p][0]] ^ sum[ch[p][1]] ^ val[p];  
 siz[p] = siz[ch[p][0]] + siz[ch[p][1]] + 1;  
}  
inline void pushdown(ll p) {  
 if (tag[p]) {  
 if (ch[p][0]) swap(ch[ch[p][0]][0], ch[ch[p][0]][1]), tag[ch[p][0]] ^= 1;  
 if (ch[p][1]) swap(ch[ch[p][1]][0], ch[ch[p][1]][1]), tag[ch[p][1]] ^= 1;  
 tag[p] = 0;  
 }  
}  
ll getch(ll x) { return ch[f[x]][1] == x; }  
bool isroot(ll x) { return ch[f[x]][0] != x && ch[f[x]][1] != x; }  
inline void rotate(ll x) {  
 ll y = f[x], z = f[y], k = getch(x);  
 if (!isroot(y)) ch[z][ch[z][1] == y] = x;  
 // 上面这句一定要写在前面，普通的Splay是不用的，因为 isRoot (后面会讲)  
 ch[y][k] = ch[x][!k], f[ch[x][!k]] = y;  
 ch[x][!k] = y, f[y] = x, f[x] = z;  
 pushup(y), pushup(x);  
}  
// 从上到下一层一层 pushDown 即可  
void update(ll p) {  
 if (!isroot(p)) update(f[p]);  
 pushdown(p);  
}  
inline void splay(ll x) {  
 update(x); // 马上就能看到啦。 在  
 // Splay之前要把旋转会经过的路径上的点都PushDown  
 for (ll fa; fa = f[x], !isroot(x); rotate(x)) {  
 if (!isroot(fa)) rotate(getch(fa) == getch(x) ? fa : x);  
 }  
}  
// 回顾一下代码  
inline void access(ll x) {  
 for (ll p = 0; x; p = x, x = f[x]) {  
 splay(x), ch[x][1] = p, pushup(x);  
 }  
}  
inline void makeroot(ll p) {  
 access(p);  
 splay(p);  
 swap(ch[p][0], ch[p][1]);  
 tag[p] ^= 1;  
}  
inline void split(ll a, ll b) {  
 makeroot(a);  
 access(b);  
 splay(b);  
}  
  
inline ll find(ll p) {  
 access(p), splay(p);  
 while (ch[p][0]) pushdown(p), p = ch[p][0];  
 splay(p);  
 return p;  
}  
inline void link(ll x, ll y) {  
 makeroot(x);  
 if (find(y) != x) f[x] = y;  
}  
inline void cut(ll x, ll y) {  
 makeroot(x);  
 if (find(y) == x && f[y] == x) {  
 ch[x][1] = f[y] = 0;  
 pushup(x);  
 }  
}

### Treap

#include <bits/stdc++.h>  
using namespace std;  
struct node {  
 node\* ch[2];  
 int r;  
 int v;  
 int cmp(int const& a) const {  
 if (v == a) return -a;  
 return a > v ? 1 : 0;  
 }  
};  
void rotate(node\*& a, int d) {  
 node\* k = a->ch[d ^ 1];  
 a->ch[d ^ 1] = k->ch[d];  
 k->ch[d] = a;  
 a = k;  
}  
void insert(node\*& a, int x) {  
 if (a == NULL) {  
 a = new node;  
 a->ch[0] = a->ch[1] = NULL;  
 a->v = x;  
 a->r = rand();  
 } else {  
 int d = a->cmp(x);  
 insert(a->ch[d], x);  
 if (a->ch[d]->r > a->r) rotate(a, d ^ 1);  
 }  
}  
void remove(node\*& a, int x) {  
 int d = a->cmp(x);  
 if (d == -1) {  
 if (a->ch[0] == NULL)  
 a = a->ch[1];  
 else if (a->ch[1] == NULL)  
 a = a->ch[0];  
 else {  
 int d2 = a->ch[1]->r > a->ch[0]->r ? 0 : 1;  
 rotate(a, d2);  
 remove(a->ch[d2], x);  
 }  
 } else {  
 remove(a->ch[d], x);  
 }  
}  
int find(node\*& a, int x) {  
 if (a == NULL)  
 return 0;  
 else if (a->v == x)  
 return 1;  
 else {  
 int d = a->cmp(x);  
 return find(a->ch[d], x);  
 }  
}  
int main() {  
 node\* a = NULL;  
 int k, l;  
 while (cin >> k >> l) {  
 if (k == 1)  
 insert(a, l);  
 else if (k == 2)  
 remove(a, l);  
 else {  
 cout << find(a, l) << endl;  
 }  
 }  
}

### 舞蹈链（多重覆盖）

#include <bits/stdc++.h>  
using namespace std;  
struct DLX {  
 static const int maxn = 1000; //列的上限  
 static const int maxr = 1000; //解的上限  
 static const int maxnode = 5000; //总结点数上限  
 static const int INF = 1000000000;  
 int n, sz;  
 int S[maxn];  
 int row[maxnode], col[maxnode];  
 int L[maxnode], R[maxnode], U[maxnode], D[maxnode];  
 int ansd, ans[maxr];  
 int vis[maxnode];  
 void init(int n) {  
 this->n = n;  
 //虚拟节点  
 for (int i = 0; i <= n; i++) {  
 U[i] = i;  
 D[i] = i;  
 L[i] = i - 1;  
 R[i] = i + 1;  
 }  
 R[n] = 0;  
 L[0] = n;  
 sz = n + 1;  
 memset(S, 0, sizeof(S));  
 }  
 void addRow(int r, vector<int> columns) {  
 int first = sz;  
 for (int i = 0; i < columns.size(); i++) {  
 int c = columns[i];  
 L[sz] = sz - 1;  
 R[sz] = sz + 1;  
 D[sz] = c;  
 U[sz] = U[c];  
 D[U[c]] = sz;  
 U[c] = sz;  
 row[sz] = r;  
 col[sz] = c;  
 S[c]++;  
 sz++;  
 }  
 R[sz - 1] = first;  
 L[first] = sz - 1;  
 }  
#define FOR(i, A, s) for (int i = A[s]; i != s; i = A[i])  
 void remove(int c) {  
 FOR(i, D, c) { L[R[i]] = L[i], R[L[i]] = R[i]; }  
 }  
 void restore(int c) {  
 FOR(i, U, c) { L[R[i]] = i, R[L[i]] = i; }  
 }  
 int f\_check() //精确覆盖区估算剪枝  
 {  
 /\*  
 强剪枝。这个  
 剪枝利用的思想是A\*搜索中的估价函数。即，对于当前的递归深度K下的矩阵，估计其最好情况下（即最少还需要多少步）才能出解。也就是，如果将能够覆盖当  
 前列的所有行全部选中，去掉这些行能够覆盖到的列，将这个操作作为一层深度。重复此操作直到所有列全部出解的深度是多少。如果当前深度加上这个估价函数返  
 回值，其和已然不能更优（也就是已经超过当前最优解），则直接返回，不必再搜。  
 \*/  
 int ret = 0;  
 FOR(c, R, 0) vis[c] = true;  
 FOR(c, R, 0)  
 if (vis[c]) {  
 ret++;  
 vis[c] = false;  
 FOR(i, D, c)  
 FOR(j, R, i) vis[col[j]] = false;  
 }  
 return ret;  
 }  
 // d为递归深度  
 void dfs(int d, vector<int>& v) {  
 if (d + f\_check() >= ansd) return;  
 if (R[0] == 0) {  
 if (d < ansd) {  
 ansd = d;  
 v.clear();  
 for (int i = 0; i < ansd; i++) {  
 v.push\_back(ans[i]);  
 }  
 } //找到解  
 return; //记录解的长度  
 }  
 //找到S最小的列c  
 int c = R[0];  
 FOR(i, R, 0)  
 if (S[i] < S[c])  
 c = i; //第一个未删除的列  
 //删除第c列  
 FOR(i, D, c) { //用结点i所在的行能覆盖的所有其他列  
 ans[d] = row[i];  
 remove(i);  
 FOR(j, R, i) remove(j); //删除结点i所在的能覆的所有其他列  
 dfs(d + 1, v);  
 FOR(j, L, i) restore(j);  
 restore(i); //恢复结点i所在的行能覆盖的所有其他列  
 } //恢复第c列  
 }  
 bool solve(vector<int>& v) {  
 v.clear();  
 ansd = INF;  
 dfs(0, v);  
 return !v.empty();  
 }  
};  
//使用时init初始化，vector中存入r行结点列表用addRow加行，solve(ans)后答案按行的选择在ans中  
DLX dlx;  
int main() {  
 int n, m;  
 cin >> n >> m;  
 dlx.init(m);  
 for (int i = 1; i <= n; i++) {  
 vector<int> v;  
 for (int j = 1; j <= m; j++) {  
 int a;  
 cin >> a;  
 if (a == 1) v.push\_back(j);  
 }  
 dlx.addRow(i, v);  
 }  
 vector<int> ans;  
 dlx.solve(ans);  
 for (int i = 0; i < ans.size(); i++) cout << ans[i];  
}

### 舞蹈链（精确覆盖）

#include <bits/stdc++.h>  
using namespace std;  
struct DLX {  
 static const int maxn = 1000; //列的上限  
 static const int maxr = 1000; //解的上限  
 static const int maxnode = 5000; //总结点数上限  
 int n, sz;  
 int S[maxn];  
 int row[maxnode], col[maxnode];  
 int L[maxnode], R[maxnode], U[maxnode], D[maxnode];  
 int ansd, ans[maxr];  
 void init(int n) {  
 this->n = n;  
 //虚拟节点  
 for (int i = 0; i <= n; i++) {  
 U[i] = i;  
 D[i] = i;  
 L[i] = i - 1;  
 R[i] = i + 1;  
 }  
 R[n] = 0;  
 L[0] = n;  
 sz = n + 1;  
 memset(S, 0, sizeof(S));  
 }  
 void addRow(int r, vector<int> columns) {  
 int first = sz;  
 for (int i = 0; i < columns.size(); i++) {  
 int c = columns[i];  
 L[sz] = sz - 1;  
 R[sz] = sz + 1;  
 D[sz] = c;  
 U[sz] = U[c];  
 D[U[c]] = sz;  
 U[c] = sz;  
 row[sz] = r;  
 col[sz] = c;  
 S[c]++;  
 sz++;  
 }  
 R[sz - 1] = first;  
 L[first] = sz - 1;  
 }  
#define FOR(i, A, s) for (int i = A[s]; i != s; i = A[i])  
 void remove(int c) {  
 L[R[c]] = L[c];  
 R[L[c]] = R[c];  
 FOR(i, D, c)  
 FOR(j, R, i) {  
 U[D[j]] = U[j];  
 D[U[j]] = D[j];  
 --S[col[j]];  
 }  
 }  
 void restore(int c) {  
 FOR(i, U, c)  
 FOR(j, L, i) {  
 ++S[col[j]];  
 U[D[j]] = j;  
 D[U[j]] = j;  
 }  
 L[R[c]] = c;  
 R[L[c]] = c;  
 }  
 // d为递归深度  
 bool dfs(int d) {  
 if (R[0] == 0) {  
 ansd = d; //找到解  
 return true; //记录解的长度  
 }  
 //找到S最小的列c  
 int c = R[0];  
 FOR(i, R, 0) if (S[i] < S[c]) c = i; //第一个未删除的列  
 remove(c); //删除第c列  
 FOR(i, D, c) { //用结点i所在的行能覆盖的所有其他列  
 ans[d] = row[i];  
 FOR(j, R, i) remove(col[j]); //删除结点i所在的能覆的所有其他列  
 if (dfs(d + 1)) return true;  
 FOR(j, L, i) restore(col[j]); //恢复结点i所在的行能覆盖的所有其他列  
 }  
 restore(c); //恢复第c列  
 return false;  
 }  
 bool solve(vector<int>& v) {  
 v.clear();  
 if (!dfs(0)) return false;  
 for (int i = 0; i < ansd; i++) v.push\_back(ans[i]);  
 return true;  
 }  
};  
//使用时init初始化，vector中存入r行结点列表用addRow加行，solve(ans)后答案按行的选择在ans中

## 数论

### lucas求组合数

#include <bits/stdc++.h>  
using namespace std;  
typedef long long ll;  
ll p;  
const int maxn = 1e5 + 10;  
ll qpow(ll x, ll n){  
 ll res = 1;  
 while(n){  
 if(n & 1) res = (res \* x) % p;  
 x = (x \* x) % p;  
 n >>= 1;  
 }  
   
 return res;  
}  
ll C(ll up, ll down){  
 if(up > down) return 0;  
 ll res = 1;  
// for(int i = up + 1; i <= down; ++ i){  
// res = (res \* i) % p;  
// }  
// for(int i = 1; i <= down - up; ++ i){  
// res = (res \* qpow(i, p - 2)) % p;   
// }  
 for(int i = 1, j = down; i <= up; ++ i, -- j){  
 res = (res \* j) % p;  
 res = (res \* qpow(i, p - 2)) % p;  
 }  
   
 return res;  
}  
  
ll lucas(ll up, ll down){  
 if(up < p && down < p) return C(up, down);  
 return C(up % p, down % p) \* lucas(up / p, down / p) % p;   
}  
int main(){  
 ios::sync\_with\_stdio(0); cin.tie(0); cout.tie(0);  
   
 int T;  
 cin >> T;  
 while (T --){  
 ll down, up;  
 cin >> down >> up >> p;  
   
 cout << lucas(up, down) % p << endl;  
 }  
   
 return 0;  
}

### 扩展欧几里得求逆元

typedef long long ll;  
void extgcd(ll a,ll b,ll& d,ll& x,ll& y){  
 if(!b){ d=a; x=1; y=0;}  
 else{ extgcd(b,a%b,d,y,x); y-=x\*(a/b); }  
}  
ll inverse(ll a,ll n){  
 ll d,x,y;  
 extgcd(a,n,d,x,y);  
 return d==1?(x+n)%n:-1;  
}

### 逆元线性递推 inv阶乘组合数

ll fac[maxn];// n!  
ll invfac[maxn]; // n!的inv  
ll invn[maxn]; //n的inv  
int init(){  
 int len=(int)(1e5+5);  
 fac[0]=fac[1]=invfac[0]=invfac[1]=invn[0]=invn[1]=1;  
 for(int i=2;i<=len;++i){  
 fac[i]=fac[i-1]\*i%mod;  
 invn[i]=(mod-mod/i)\*invn[mod%i]%mod;  
 invfac[i]=invfac[i-1]\*invn[i]%mod;  
 }  
}  
ll C(ll n,ll m){  
 if(n>m) return 0;  
 if(n<0 || m<0) return 0;  
 ll res=fac[m];  
 res=res\*invfac[m-n]%mod;  
 res=res\*invfac[n]%mod;  
 return res;  
}  
// 先init（）, C(n,m) n在上面..  
//改init里面的len（为maxn的长度 注意不要数组越界）

## 数学

### 一些范围

**1 ~ n 的质数个数**

**1 ~ 2e9 中拥有最多约数个数的数拥有的约数个数**

约1600

**n个不同的点可以构成棵不同的树**

### BSGS

求 (a,p) = 1的最小的t

对 建立hash表，枚举x看是否有解

#include <bits/stdc++.h>  
using namespace std;  
typedef long long ll;  
unordered\_map<int , int> mp;  
int bsgs(int a, int p, int b) {  
   
 if (1 % p == b % p) return 0; // 特判0是不是解  
 mp.clear();  
   
 int k = sqrt(p) + 1;  
   
 for(int i = 0, j = b % p; i < k; ++ i, j = (ll)j \* a % p) {  
 mp[j] = i;  
 }  
   
 int ak = 1;  
 for(int i = 0; i < k; ++i) {  
 ak = (ll)ak \* a % p;  
 }  
   
 for(int i = 1, j = ak % p; i <= k; ++ i, j = (ll)j \* ak % p) {  
 if(mp.count(j)) return (ll)i \* k - mp[j];  
 }  
   
 return -1;  
}  
int main() {  
 ios::sync\_with\_stdio(0);  
 cin.tie(0); cout.tie(0);  
   
 int a, p, b;  
 while(cin >> a >> p >> b, a | p | b) {  
 int res;  
 res = bsgs(a, p, b);  
 if(res == -1) {  
 cout << "No Solution\n";   
 }  
 else {  
 cout << res << endl;  
 }  
 }  
   
 return 0;  
}

### 扩展BSGS

求 的最小的t

当

无解

， 两边同时除以d，

#include <bits/stdc++.h>  
using namespace std;  
typedef long long ll;  
unordered\_map<ll, ll> mp;  
ll bsgs(ll a, ll p, ll b) {  
   
 if(1 % p == b % p) return 0; // 特判0是不是解  
 mp.clear();   
   
 ll k = sqrt(p) + 1;  
   
 for(ll i = 0, j = b % p; i < k; ++i, j = (ll)j \* a % p) {  
 mp[j] = i;  
 }  
   
 ll ak = 1;  
 for(ll i = 0; i < k; ++i) {  
 ak = (ll) ak \* a % p;  
 }  
   
 for(ll i = 1, j = ak % p;i <= k; ++i, j = (ll)j \* ak % p) {  
 if(mp.count(j)) return (ll) i \* k - mp[j];  
 }  
   
 return -1;  
}  
ll gcd(ll x, ll y) {  
 return x % y == 0 ? y : gcd(y, x % y);   
}  
void extgcd(ll a,ll b,ll& d,ll& x,ll& y){  
 if(!b){  
 d = a; x = 1; y = 0;  
 }  
 else{   
 extgcd(b, a%b, d, y, x);   
 y -= x \* (a / b);   
 }  
}  
ll inverse(ll a,ll n){  
 ll d,x,y;  
 extgcd(a,n,d,x,y);  
 return d == 1 ? (x + n) % n : -1;  
}  
int main() {  
 ll a, p, b;  
   
 while(cin >> a >> p >> b, a | p | b) {  
 ll d = gcd(a, p);  
 if(d == 1) {  
 ll res = bsgs(a, p, b);  
 if(res == -1) {  
 cout << "No Solution\n";  
 }  
 else {  
 cout << res << endl;  
 }  
 }  
 else {  
 if(b % d != 0) {  
 cout << "No Solution\n";  
 continue;  
 }  
 else {  
 p = p / d;  
 b = (b / d) \* inverse(a / d, p);  
 ll res = bsgs(a, p, b);  
 if(res == -1) {  
 cout << "No Solution\n";  
 }  
 else {  
 cout << res + 1 << endl;  
 }  
 }   
 }  
 }   
   
 return 0;  
}

### 二次剩余

**解的数量**

对于 能满足n是mod p的二次剩余的n一共有个（不包括0），非二次剩余为个

**勒让德符号**

**欧拉判别准则**

若n是二次剩余，当且仅当

若n是非二次剩余，当且仅当

**Cipolla**

找到一个数a满足是 **非二次剩余** ，至于为什么要找满足非二次剩余的数，在下文会给出解释。 这里通过生成随机数再检验的方法来实现，由于非二次剩余的数量为 ，接近 ，所以期望约 2 次就可以找到这个数。

建立一个＂复数域＂，并不是实际意义上的复数域，而是根据复数域的概念建立的一个类似的域。 在复数中 ，这里定义 ，于是就可以将所有的数表达为 的形式，这里的 和 都是模意义下的数，类似复数中的实部和虚部。

在有了 i和 a后可以直接得到答案， 的解为。

#include <bits/stdc++.h>  
using namespace std;  
typedef long long ll;  
int t;  
ll n, p;  
ll w;  
struct num { //建立一个复数域  
 ll x, y;  
};  
num mul(num a, num b, ll p) { //复数乘法  
 num ans = {0, 0};  
 ans.x = ((a.x \* b.x % p + a.y \* b.y % p \* w % p) % p + p) % p;  
 ans.y = ((a.x \* b.y % p + a.y \* b.x % p) % p + p) % p;  
 return ans;  
}  
ll binpow\_real(ll a, ll b, ll p) { //实部快速幂  
 ll ans = 1;  
 while (b) {  
 if (b & 1) ans = ans \* a % p;  
 a = a \* a % p;  
 b >>= 1;  
 }  
 return ans % p;  
}  
ll binpow\_imag(num a, ll b, ll p) { //虚部快速幂  
 num ans = {1, 0};  
 while (b) {  
 if (b & 1) ans = mul(ans, a, p);  
 a = mul(a, a, p);  
 b >>= 1;  
 }  
 return ans.x % p;  
}  
ll cipolla(ll n, ll p) {  
 n %= p;  
 if (p == 2) return n;  
 if (binpow\_real(n, (p - 1) / 2, p) == p - 1) return -1;  
 ll a;  
 while (1) { //生成随机数再检验找到满足非二次剩余的a  
 a = rand() % p;  
 w = ((a \* a % p - n) % p + p) % p;  
 if (binpow\_real(w, (p - 1) / 2, p) == p - 1) break;  
 }  
 num x = {a, 1};  
 return binpow\_imag(x, (p + 1) / 2, p);  
}

### 卡特兰数

卡特兰数1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012,...

超级卡特兰数1, 1, 3, 11, 45, 197, 903, 4279, 20793, 103049,...（从第0项开始）

大施罗德数(OEIS A006318)1, 2, 6, 22, 90, 394, 1806, 8558, 41586, 206098,...

超级卡特兰数的两倍（除第一项）

### 快速幂

ll qpow(ll a, ll b) {  
 ll ans = 1;  
 while (b) {  
 if (b & 1) ans = (ans \* a) % mod;  
 a = (a \* a) % mod;  
 b >>= 1;  
 }  
 return ans;  
}

### 龟速乘快速幂（快速幂爆longlong

#include <bits/stdc++.h>  
using namespace std;  
typedef long long ll;  
ll qmul(ll a, ll b, ll p) {  
 ll res = 0;  
 while(b) {  
 if(b & 1) res = (res + a) % p;  
 a = (a + a) % p;  
 b >>= 1;   
 }  
 return res;  
}  
ll qpow(ll x, ll n, ll p) {  
 ll res = 1;  
 while(n) {  
 if(n & 1) res = qmul(res, x, p);  
 x = qmul(x, x, p);  
 n >>= 1;  
 }  
 return res % p; // 1 0 1  
}  
int main() {  
 ll b, p, k;  
 cin >> b >> p >> k;  
 ll ans = qpow(b, p, k);  
 printf("%lld^%lld mod %lld=%lld", b, p, k, ans);  
   
 return 0;  
}

### 莫比乌斯反演

**莫比乌斯函数**

**n的所有约数的莫比乌斯的和**

**反演**

构造 使f(n)为目标，F(n)好求

**1**

求满足 且 gcd(x, y) = k 的xy的对数

#include <bits/stdc++.h>  
using namespace std;  
typedef long long ll;  
const int N = 50010;  
ll primes[N], mu[N], sum[N], cnt;  
bool st[N];  
void init() {  
 mu[1] = 1;  
   
 for(int i = 2; i < N; ++ i) {  
 if(!st[i]) {  
 primes[cnt ++] = i;  
 mu[i] = -1;  
 }  
   
 for(int j = 0; primes[j] \* i < N; ++ j) {  
 st[primes[j] \* i] = 1;  
 if(i % primes[j] == 0) break;  
 mu[primes[j] \* i] = -mu[i];   
 }  
 }  
   
 for(int i = 1; i < N; ++ i) {  
 sum[i] = sum[i - 1] + mu[i];  
 }  
}   
ll g(ll n, ll x) {  
 return n / (n / x);  
}  
ll f (int a, int b, int k) {  
 a = a / k, b = b / k;  
   
 ll res = 0;  
   
 ll n = min(a, b);  
   
 for(ll l = 1, r; l <= n; l = r + 1) {  
 r = min(n, min(g(a, l), g(b, l)));  
 res += (sum[r] - sum[l - 1]) \* (a / l) \* (b / l);  
 }  
   
 return res;  
}  
int main() {  
 ios::sync\_with\_stdio(0); cin.tie(0); cout.tie(0);  
   
 init();  
   
 int T;  
 cin >> T;  
 while(T --) {  
 int a, b, c, d, k;  
 cin >> a >> b >> c >> d >> k;  
 cout << f(b, d, k) - f(a - 1, d, k) - f(b, c - 1, k)   
 + f(a - 1, c - 1, k) << endl;  
 }   
   
 return 0;  
}

**2**

求

//

两次整数分块

#include <bits/stdc++.h>  
using namespace std;  
typedef long long ll;  
const int N = 50010;  
int primes[N], cnt, mu[N], sum[N], h[N];  
bool st[N];  
inline int g(int n, int x) {  
 return n / (n / x);  
}  
void init() {  
 mu[1] = 1;  
 for(int i = 2; i < N; ++i) {  
 if(!st[i]){  
 primes[cnt++] = i;  
 mu[i] = -1;  
 }  
 for(int j = 0; primes[j] \* i < N; ++j) {  
 st[primes[j] \* i] = 1;  
 if(i % primes[j] == 0) break;  
 mu[primes[j] \* i] = -mu[i];  
 }  
   
   
 }  
   
 for(int i = 1; i < N; ++ i) {  
 sum[i] = sum[i - 1] + mu[i];   
 }  
   
 for(int i = 1; i < N; ++i) {  
 for(int l = 1, r; l <= i; l = r + 1) {  
 r = min(i, g(i, l));  
 h[i] += (r - l + 1) \* (i / l);  
 }  
 }  
}  
int main() {  
 //ios::sync\_with\_stdio(0); cin.tie(0); cout.tie(0);   
 init();  
   
 int T;  
 scanf("%d", &T);  
 while(T--) {  
 int n, m;  
 scanf("%d %d", &n, &m);  
 ll res = 0;  
 int k = min(n, m);  
 for(int l = 1, r; l <= k; l = r + 1) {  
 r = min(k, min(g(n, l), g(m, l)));  
 res += (ll)(sum[r] - sum[l - 1]) \* h[n / l] \* h[m / l];  
 }  
 printf("%lld\n", res);  
 }  
   
 return 0;  
}

### 博弈

**SG定理：**

mex(minimal excludant)运算，表示最小的不属于这个集合的非负整数。例如mex{0,1,2,4}=3、mex{2,3,5}=0、mex{}=0。  
Sprague-Grundy定理（SG定理）：游戏和的SG函数等于各个游戏SG函数的Nim和。这样就可以将每一个子游戏分而治之，从而简化了问题。而Bouton定理就是Sprague-Grundy定理在Nim游戏中的直接应用，因为单堆的Nim游戏 SG函数满足 SG(x) = x。

**Nimk：**

普通的NIM游戏是在n堆石子中每次选一堆，取任意个石子，而NIMK游戏是在n堆石子中每次选择k堆，1<=k<=n，从这k堆中每堆里都取出任意数目的石子，取的石子数可以不同，其他规则相同。  
对于普通的NIM游戏，我们采取的是对每堆的SG值进行异或，异或其实就是对每一个SG值二进制位上的数求和然后模2，比如说3^5就是011+101=112，然后对每一位都模2就变成了110，所以3^5=6。而NIMK游戏和NIM游戏的区别就在于模的不是2，如果是取k堆，就模k+1，所以取1堆的普通NIM游戏是模2。当k=2时,3^5→011+101=112，对每一位都模3之后三位二进制位上对应的数仍然是1，1，2。那么当且仅当每一位二进制位上的数都是0的时候，先手必败，否则先手必胜。

**anti\_nim**

**描述**

和最普通的Nim游戏相同，不过是取走最后一个石子的人输。

**先手必胜条件**

以下两个条件满足其一即可：

1. 所有堆的石子个数=1，且异或和=0（其实这里就是有偶数堆的意思）。
2. 至少存在一堆石子个数>1，且异或和≠0。

### 高精度GCD

#include <bits/stdc++.h>  
using namespace std;  
string add(string a, string b) {  
 const int L = 1e5;  
 string ans;  
 int na[L] = {0}, nb[L] = {0};  
 int la = a.size(), lb = b.size();  
 for (int i = 0; i < la; i++) na[la - 1 - i] = a[i] - '0';  
 for (int i = 0; i < lb; i++) nb[lb - 1 - i] = b[i] - '0';  
 int lmax = la > lb ? la : lb;  
 for (int i = 0; i < lmax; i++)  
 na[i] += nb[i], na[i + 1] += na[i] / 10, na[i] %= 10;  
 if (na[lmax]) lmax++;  
 for (int i = lmax - 1; i >= 0; i--) ans += na[i] + '0';  
 return ans;  
}  
string mul(string a, string b) {  
 const int L = 1e5;  
 string s;  
 int na[L], nb[L], nc[L],  
 La = a.size(), Lb = b.size(); // na存储被乘数，nb存储乘数，nc存储积  
 fill(na, na + L, 0);  
 fill(nb, nb + L, 0);  
 fill(nc, nc + L, 0); //将na,nb,nc都置为0  
 for (int i = La - 1; i >= 0; i--)  
 na[La - i] =  
 a[i] - '0'; //将字符串表示的大整形数转成i整形数组表示的大整形数  
 for (int i = Lb - 1; i >= 0; i--) nb[Lb - i] = b[i] - '0';  
 for (int i = 1; i <= La; i++)  
 for (int j = 1; j <= Lb; j++)  
 nc[i + j - 1] +=  
 na[i] \*  
 nb[j]; // a的第i位乘以b的第j位为积的第i+j-1位（先不考虑进位）  
 for (int i = 1; i <= La + Lb; i++)  
 nc[i + 1] += nc[i] / 10, nc[i] %= 10; //统一处理进位  
 if (nc[La + Lb]) s += nc[La + Lb] + '0'; //判断第i+j位上的数字是不是0  
 for (int i = La + Lb - 1; i >= 1; i--)  
 s += nc[i] + '0'; //将整形数组转成字符串  
 return s;  
}  
int sub(int \*a, int \*b, int La, int Lb) {  
 if (La < Lb) return -1; //如果a小于b，则返回-1  
 if (La == Lb) {  
 for (int i = La - 1; i >= 0; i--)  
 if (a[i] > b[i])  
 break;  
 else if (a[i] < b[i])  
 return -1; //如果a小于b，则返回-1  
 }  
 for (int i = 0; i < La; i++) //高精度减法  
 {  
 a[i] -= b[i];  
 if (a[i] < 0) a[i] += 10, a[i + 1]--;  
 }  
 for (int i = La - 1; i >= 0; i--)  
 if (a[i]) return i + 1; //返回差的位数  
 return 0; //返回差的位数  
}  
string div(string n1, string n2,  
 int nn) // n1,n2是字符串表示的被除数，除数,nn是选择返回商还是余数  
{  
 const int L = 1e5;  
 string s, v; // s存商,v存余数  
 int a[L], b[L], r[L],  
 La = n1.size(), Lb = n2.size(), i,  
 tp = La; // a，b是整形数组表示被除数，除数，tp保存被除数的长度  
 fill(a, a + L, 0);  
 fill(b, b + L, 0);  
 fill(r, r + L, 0); //数组元素都置为0  
 for (i = La - 1; i >= 0; i--) a[La - 1 - i] = n1[i] - '0';  
 for (i = Lb - 1; i >= 0; i--) b[Lb - 1 - i] = n2[i] - '0';  
 if (La < Lb || (La == Lb && n1 < n2)) {  
 // cout<<0<<endl;  
 return n1;  
 } //如果a<b,则商为0，余数为被除数  
 int t = La - Lb; //除被数和除数的位数之差  
 for (int i = La - 1; i >= 0; i--) //将除数扩大10^t倍  
 if (i >= t)  
 b[i] = b[i - t];  
 else  
 b[i] = 0;  
 Lb = La;  
 for (int j = 0; j <= t; j++) {  
 int temp;  
 while ((temp = sub(a, b + j, La, Lb - j)) >=  
 0) //如果被除数比除数大继续减  
 {  
 La = temp;  
 r[t - j]++;  
 }  
 }  
 for (i = 0; i < L - 10; i++)  
 r[i + 1] += r[i] / 10, r[i] %= 10; //统一处理进位  
 while (!r[i]) i--; //将整形数组表示的商转化成字符串表示的  
 while (i >= 0) s += r[i--] + '0';  
 // cout<<s<<endl;  
 i = tp;  
 while (!a[i]) i--; //将整形数组表示的余数转化成字符串表示的</span>  
 while (i >= 0) v += a[i--] + '0';  
 if (v.empty()) v = "0";  
 // cout<<v<<endl;  
 if (nn == 1) return s;  
 if (nn == 2) return v;  
}  
bool judge(string s) //判断s是否为全0串  
{  
 for (int i = 0; i < s.size(); i++)  
 if (s[i] != '0') return false;  
 return true;  
}  
string gcd(string a, string b) //求最大公约数  
{  
 string t;  
 while (!judge(b)) //如果余数不为0，继续除  
 {  
 t = a; //保存被除数的值  
 a = b; //用除数替换被除数  
 b = div(t, b, 2); //用余数替换除数  
 }  
 return a;  
}  
//o(无法估计)

### 高精度乘法（FFT）

#include <bits/stdc++.h>  
using namespace std;  
#define L(x) (1 << (x))  
const double PI = acos(-1.0);  
const int Maxn = 133015;  
double ax[Maxn], ay[Maxn], bx[Maxn], by[Maxn];  
char sa[Maxn / 2], sb[Maxn / 2];  
int sum[Maxn];  
int x1[Maxn], x2[Maxn];  
int revv(int x, int bits) {  
 int ret = 0;  
 for (int i = 0; i < bits; i++) {  
 ret <<= 1;  
 ret |= x & 1;  
 x >>= 1;  
 }  
 return ret;  
}  
void fft(double\* a, double\* b, int n, bool rev) {  
 int bits = 0;  
 while (1 << bits < n) ++bits;  
 for (int i = 0; i < n; i++) {  
 int j = revv(i, bits);  
 if (i < j) swap(a[i], a[j]), swap(b[i], b[j]);  
 }  
 for (int len = 2; len <= n; len <<= 1) {  
 int half = len >> 1;  
 double wmx = cos(2 \* PI / len), wmy = sin(2 \* PI / len);  
 if (rev) wmy = -wmy;  
 for (int i = 0; i < n; i += len) {  
 double wx = 1, wy = 0;  
 for (int j = 0; j < half; j++) {  
 double cx = a[i + j], cy = b[i + j];  
 double dx = a[i + j + half], dy = b[i + j + half];  
 double ex = dx \* wx - dy \* wy, ey = dx \* wy + dy \* wx;  
 a[i + j] = cx + ex, b[i + j] = cy + ey;  
 a[i + j + half] = cx - ex, b[i + j + half] = cy - ey;  
 double wnx = wx \* wmx - wy \* wmy, wny = wx \* wmy + wy \* wmx;  
 wx = wnx, wy = wny;  
 }  
 }  
 }  
 if (rev) {  
 for (int i = 0; i < n; i++) a[i] /= n, b[i] /= n;  
 }  
}  
int solve(int a[], int na, int b[], int nb, int ans[]) {  
 int len = max(na, nb), ln;  
 for (ln = 0; L(ln) < len; ++ln)  
 ;  
 len = L(++ln);  
 for (int i = 0; i < len; ++i) {  
 if (i >= na)  
 ax[i] = 0, ay[i] = 0;  
 else  
 ax[i] = a[i], ay[i] = 0;  
 }  
 fft(ax, ay, len, 0);  
 for (int i = 0; i < len; ++i) {  
 if (i >= nb)  
 bx[i] = 0, by[i] = 0;  
 else  
 bx[i] = b[i], by[i] = 0;  
 }  
 fft(bx, by, len, 0);  
 for (int i = 0; i < len; ++i) {  
 double cx = ax[i] \* bx[i] - ay[i] \* by[i];  
 double cy = ax[i] \* by[i] + ay[i] \* bx[i];  
 ax[i] = cx, ay[i] = cy;  
 }  
 fft(ax, ay, len, 1);  
 for (int i = 0; i < len; ++i) ans[i] = (int)(ax[i] + 0.5);  
 return len;  
}  
string mul(string sa, string sb) {  
 int l1, l2, l;  
 int i;  
 string ans;  
 memset(sum, 0, sizeof(sum));  
 l1 = sa.size();  
 l2 = sb.size();  
 for (i = 0; i < l1; i++) x1[i] = sa[l1 - i - 1] - '0';  
 for (i = 0; i < l2; i++) x2[i] = sb[l2 - i - 1] - '0';  
 l = solve(x1, l1, x2, l2, sum);  
 for (i = 0; i < l || sum[i] >= 10; i++) // 进位  
 {  
 sum[i + 1] += sum[i] / 10;  
 sum[i] %= 10;  
 }  
 l = i;  
 while (sum[l] <= 0 && l > 0) l--; // 检索最高位  
 for (i = l; i >= 0; i--) ans += sum[i] + '0'; // 倒序输出  
 return ans;  
}  
int main() {  
 cin.sync\_with\_stdio(false);  
 string a, b;  
 while (cin >> a >> b) cout << mul(a, b) << endl;  
 return 0;  
}  
//o(nlogn)

### 高精度乘法（乘单精度

#include <bits/stdc++.h>  
using namespace std;  
string mul(string a, int b) //高精度a乘单精度b  
{  
 const int L = 100005;  
 int na[L];  
 string ans;  
 int La = a.size();  
 fill(na, na + L, 0);  
 for (int i = La - 1; i >= 0; i--) na[La - i - 1] = a[i] - '0';  
 int w = 0;  
 for (int i = 0; i < La; i++)  
 na[i] = na[i] \* b + w, w = na[i] / 10, na[i] = na[i] % 10;  
 while (w) na[La++] = w % 10, w /= 10;  
 La--;  
 while (La >= 0) ans += na[La--] + '0';  
 return ans;  
}  
//o(n)

### 高精度乘法（朴素）

#include <bits/stdc++.h>  
using namespace std;  
string mul(string a, string b) //高精度乘法a,b,均为非负整数  
{  
 const int L = 1e5;  
 string s;  
 int na[L], nb[L], nc[L],  
 La = a.size(), Lb = b.size(); // na存储被乘数，nb存储乘数，nc存储积  
 fill(na, na + L, 0);  
 fill(nb, nb + L, 0);  
 fill(nc, nc + L, 0); //将na,nb,nc都置为0  
 for (int i = La - 1; i >= 0; i--)  
 na[La - i] =  
 a[i] - '0'; //将字符串表示的大整形数转成i整形数组表示的大整形数  
 for (int i = Lb - 1; i >= 0; i--) nb[Lb - i] = b[i] - '0';  
 for (int i = 1; i <= La; i++)  
 for (int j = 1; j <= Lb; j++)  
 nc[i + j - 1] +=  
 na[i] \*  
 nb[j]; // a的第i位乘以b的第j位为积的第i+j-1位（先不考虑进位）  
 for (int i = 1; i <= La + Lb; i++)  
 nc[i + 1] += nc[i] / 10, nc[i] %= 10; //统一处理进位  
 if (nc[La + Lb]) s += nc[La + Lb] + '0'; //判断第i+j位上的数字是不是0  
 for (int i = La + Lb - 1; i >= 1; i--)  
 s += nc[i] + '0'; //将整形数组转成字符串  
 return s;  
}  
//o(n^2)

### 高精度除法（除单精度）

#include <bits/stdc++.h>  
using namespace std;  
string div(string a, int b) //高精度a除以单精度b  
{  
 string r, ans;  
 int d = 0;  
 if (a == "0") return a; //特判  
 for (int i = 0; i < a.size(); i++) {  
 r += (d \* 10 + a[i] - '0') / b + '0'; //求出商  
 d = (d \* 10 + (a[i] - '0')) % b; //求出余数  
 }  
 int p = 0;  
 for (int i = 0; i < r.size(); i++)  
 if (r[i] != '0') {  
 p = i;  
 break;  
 }  
 return r.substr(p);  
}  
//o(n)

### 高精度除法（除高精度）

#include <bits/stdc++.h>  
using namespace std;  
int sub(int \*a, int \*b, int La, int Lb) {  
 if (La < Lb) return -1; //如果a小于b，则返回-1  
 if (La == Lb) {  
 for (int i = La - 1; i >= 0; i--)  
 if (a[i] > b[i])  
 break;  
 else if (a[i] < b[i])  
 return -1; //如果a小于b，则返回-1  
 }  
 for (int i = 0; i < La; i++) //高精度减法  
 {  
 a[i] -= b[i];  
 if (a[i] < 0) a[i] += 10, a[i + 1]--;  
 }  
 for (int i = La - 1; i >= 0; i--)  
 if (a[i]) return i + 1; //返回差的位数  
 return 0; //返回差的位数  
}  
string div(string n1, string n2, int nn)  
// n1,n2是字符串表示的被除数，除数,nn是选择返回商还是余数  
{  
 const int L = 1e5;  
 string s, v; // s存商,v存余数  
 int a[L], b[L], r[L], La = n1.size(), Lb = n2.size(), i, tp = La;  
 // a，b是整形数组表示被除数，除数，tp保存被除数的长度  
 fill(a, a + L, 0);  
 fill(b, b + L, 0);  
 fill(r, r + L, 0); //数组元素都置为0  
 for (i = La - 1; i >= 0; i--) a[La - 1 - i] = n1[i] - '0';  
 for (i = Lb - 1; i >= 0; i--) b[Lb - 1 - i] = n2[i] - '0';  
 if (La < Lb || (La == Lb && n1 < n2)) {  
 // cout<<0<<endl;  
 return n1;  
 } //如果a<b,则商为0，余数为被除数  
 int t = La - Lb; //除被数和除数的位数之差  
 for (int i = La - 1; i >= 0; i--) //将除数扩大10^t倍  
 if (i >= t)  
 b[i] = b[i - t];  
 else  
 b[i] = 0;  
 Lb = La;  
 for (int j = 0; j <= t; j++) {  
 int temp;  
 while ((temp = sub(a, b + j, La, Lb - j)) >=  
 0) //如果被除数比除数大继续减  
 {  
 La = temp;  
 r[t - j]++;  
 }  
 }  
 for (i = 0; i < L - 10; i++)  
 r[i + 1] += r[i] / 10, r[i] %= 10; //统一处理进位  
 while (!r[i]) i--; //将整形数组表示的商转化成字符串表示的  
 while (i >= 0) s += r[i--] + '0';  
 // cout<<s<<endl;  
 i = tp;  
 while (!a[i]) i--; //将整形数组表示的余数转化成字符串表示的</span>  
 while (i >= 0) v += a[i--] + '0';  
 if (v.empty()) v = "0";  
 // cout<<v<<endl;  
 if (nn == 1) return s; //返回商  
 if (nn == 2) return v; //返回余数  
}  
//o(n^2)

### 高精度加法

#include <bits/stdc++.h>  
using namespace std;  
string add(string a, string b) //只限两个非负整数相加  
{  
 const int L = 1e5;  
 string ans;  
 int na[L] = {0}, nb[L] = {0};  
 int la = a.size(), lb = b.size();  
 for (int i = 0; i < la; i++) na[la - 1 - i] = a[i] - '0';  
 for (int i = 0; i < lb; i++) nb[lb - 1 - i] = b[i] - '0';  
 int lmax = la > lb ? la : lb;  
 for (int i = 0; i < lmax; i++)  
 na[i] += nb[i], na[i + 1] += na[i] / 10, na[i] %= 10;  
 if (na[lmax]) lmax++;  
 for (int i = lmax - 1; i >= 0; i--) ans += na[i] + '0';  
 return ans;  
}  
//o(n)

### 高精度减法

#include <bits/stdc++.h>  
using namespace std;  
string sub(string a, string b) //只限大的非负整数减小的非负整数  
{  
 const int L = 1e5;  
 string ans;  
 int na[L] = {0}, nb[L] = {0};  
 int la = a.size(), lb = b.size();  
 for (int i = 0; i < la; i++) na[la - 1 - i] = a[i] - '0';  
 for (int i = 0; i < lb; i++) nb[lb - 1 - i] = b[i] - '0';  
 int lmax = la > lb ? la : lb;  
 for (int i = 0; i < lmax; i++) {  
 na[i] -= nb[i];  
 if (na[i] < 0) na[i] += 10, na[i + 1]--;  
 }  
 while (!na[--lmax] && lmax > 0)  
 ;  
 lmax++;  
 for (int i = lmax - 1; i >= 0; i--) ans += na[i] + '0';  
 return ans;  
}  
//o(n)

### 高精度阶乘

#include <bits/stdc++.h>  
using namespace std;  
string fac(int n) {  
 const int L = 100005;  
 int a[L];  
 string ans;  
 if (n == 0) return "1";  
 fill(a, a + L, 0);  
 int s = 0, m = n;  
 while (m) a[++s] = m % 10, m /= 10;  
 for (int i = n - 1; i >= 2; i--) {  
 int w = 0;  
 for (int j = 1; j <= s; j++)  
 a[j] = a[j] \* i + w, w = a[j] / 10, a[j] = a[j] % 10;  
 while (w) a[++s] = w % 10, w /= 10;  
 }  
 while (!a[s]) s--;  
 while (s >= 1) ans += a[s--] + '0';  
 return ans;  
}  
//o(n^2)

### 高精度进制转换

#include <bits/stdc++.h>  
using namespace std;  
//将字符串表示的10进制大整数转换为m进制的大整数  
//并返回m进制大整数的字符串  
bool judge(string s) //判断串是否为全零串  
{  
 for (int i = 0; i < s.size(); i++)  
 if (s[i] != '0') return 1;  
 return 0;  
}  
string solve(  
 string s, int n,  
 int m) // n进制转m进制只限0-9进制，若涉及带字母的进制，稍作修改即可  
{  
 string r, ans;  
 int d = 0;  
 if (!judge(s)) return "0"; //特判  
 while (judge(s)) //被除数不为0则继续  
 {  
 for (int i = 0; i < s.size(); i++) {  
 r += (d \* n + s[i] - '0') / m + '0'; //求出商  
 d = (d \* n + (s[i] - '0')) % m; //求出余数  
 }  
 s = r; //把商赋给下一次的被除数  
 r = ""; //把商清空  
 ans += d + '0'; //加上进制转换后数字  
 d = 0; //清空余数  
 }  
 reverse(ans.begin(), ans.end()); //倒置下  
 return ans;  
}  
//o(n^2)

### 高精度幂

#include <bits/stdc++.h>  
#define L(x) (1 << (x))  
using namespace std;  
const double PI = acos(-1.0);  
const int Maxn = 133015;  
double ax[Maxn], ay[Maxn], bx[Maxn], by[Maxn];  
char sa[Maxn / 2], sb[Maxn / 2];  
int sum[Maxn];  
int x1[Maxn], x2[Maxn];  
int revv(int x, int bits) {  
 int ret = 0;  
 for (int i = 0; i < bits; i++) {  
 ret <<= 1;  
 ret |= x & 1;  
 x >>= 1;  
 }  
 return ret;  
}  
void fft(double\* a, double\* b, int n, bool rev) {  
 int bits = 0;  
 while (1 << bits < n) ++bits;  
 for (int i = 0; i < n; i++) {  
 int j = revv(i, bits);  
 if (i < j) swap(a[i], a[j]), swap(b[i], b[j]);  
 }  
 for (int len = 2; len <= n; len <<= 1) {  
 int half = len >> 1;  
 double wmx = cos(2 \* PI / len), wmy = sin(2 \* PI / len);  
 if (rev) wmy = -wmy;  
 for (int i = 0; i < n; i += len) {  
 double wx = 1, wy = 0;  
 for (int j = 0; j < half; j++) {  
 double cx = a[i + j], cy = b[i + j];  
 double dx = a[i + j + half], dy = b[i + j + half];  
 double ex = dx \* wx - dy \* wy, ey = dx \* wy + dy \* wx;  
 a[i + j] = cx + ex, b[i + j] = cy + ey;  
 a[i + j + half] = cx - ex, b[i + j + half] = cy - ey;  
 double wnx = wx \* wmx - wy \* wmy, wny = wx \* wmy + wy \* wmx;  
 wx = wnx, wy = wny;  
 }  
 }  
 }  
 if (rev) {  
 for (int i = 0; i < n; i++) a[i] /= n, b[i] /= n;  
 }  
}  
int solve(int a[], int na, int b[], int nb, int ans[]) {  
 int len = max(na, nb), ln;  
 for (ln = 0; L(ln) < len; ++ln)  
 ;  
 len = L(++ln);  
 for (int i = 0; i < len; ++i) {  
 if (i >= na)  
 ax[i] = 0, ay[i] = 0;  
 else  
 ax[i] = a[i], ay[i] = 0;  
 }  
 fft(ax, ay, len, 0);  
 for (int i = 0; i < len; ++i) {  
 if (i >= nb)  
 bx[i] = 0, by[i] = 0;  
 else  
 bx[i] = b[i], by[i] = 0;  
 }  
 fft(bx, by, len, 0);  
 for (int i = 0; i < len; ++i) {  
 double cx = ax[i] \* bx[i] - ay[i] \* by[i];  
 double cy = ax[i] \* by[i] + ay[i] \* bx[i];  
 ax[i] = cx, ay[i] = cy;  
 }  
 fft(ax, ay, len, 1);  
 for (int i = 0; i < len; ++i) ans[i] = (int)(ax[i] + 0.5);  
 return len;  
}  
string mul(string sa, string sb) {  
 int l1, l2, l;  
 int i;  
 string ans;  
 memset(sum, 0, sizeof(sum));  
 l1 = sa.size();  
 l2 = sb.size();  
 for (i = 0; i < l1; i++) x1[i] = sa[l1 - i - 1] - '0';  
 for (i = 0; i < l2; i++) x2[i] = sb[l2 - i - 1] - '0';  
 l = solve(x1, l1, x2, l2, sum);  
 for (i = 0; i < l || sum[i] >= 10; i++) // 进位  
 {  
 sum[i + 1] += sum[i] / 10;  
 sum[i] %= 10;  
 }  
 l = i;  
 while (sum[l] <= 0 && l > 0) l--; // 检索最高位  
 for (i = l; i >= 0; i--) ans += sum[i] + '0'; // 倒序输出  
 return ans;  
}  
string Pow(string a, int n) {  
 if (n == 1) return a;  
 if (n & 1) return mul(Pow(a, n - 1), a);  
 string ans = Pow(a, n / 2);  
 return mul(ans, ans);  
}  
//o(nlognlogm)

### 高精度平方根

#include <bits/stdc++.h>  
using namespace std;  
const int L = 2015;  
string add(string a, string b) //只限两个非负整数相加  
{  
 string ans;  
 int na[L] = {0}, nb[L] = {0};  
 int la = a.size(), lb = b.size();  
 for (int i = 0; i < la; i++) na[la - 1 - i] = a[i] - '0';  
 for (int i = 0; i < lb; i++) nb[lb - 1 - i] = b[i] - '0';  
 int lmax = la > lb ? la : lb;  
 for (int i = 0; i < lmax; i++)  
 na[i] += nb[i], na[i + 1] += na[i] / 10, na[i] %= 10;  
 if (na[lmax]) lmax++;  
 for (int i = lmax - 1; i >= 0; i--) ans += na[i] + '0';  
 return ans;  
}  
string sub(string a, string b) //只限大的非负整数减小的非负整数  
{  
 string ans;  
 int na[L] = {0}, nb[L] = {0};  
 int la = a.size(), lb = b.size();  
 for (int i = 0; i < la; i++) na[la - 1 - i] = a[i] - '0';  
 for (int i = 0; i < lb; i++) nb[lb - 1 - i] = b[i] - '0';  
 int lmax = la > lb ? la : lb;  
 for (int i = 0; i < lmax; i++) {  
 na[i] -= nb[i];  
 if (na[i] < 0) na[i] += 10, na[i + 1]--;  
 }  
 while (!na[--lmax] && lmax > 0)  
 ;  
 lmax++;  
 for (int i = lmax - 1; i >= 0; i--) ans += na[i] + '0';  
 return ans;  
}  
string mul(string a, string b) //高精度乘法a,b,均为非负整数  
{  
 string s;  
 int na[L], nb[L], nc[L],  
 La = a.size(), Lb = b.size(); // na存储被乘数，nb存储乘数，nc存储积  
 fill(na, na + L, 0);  
 fill(nb, nb + L, 0);  
 fill(nc, nc + L, 0); //将na,nb,nc都置为0  
 for (int i = La - 1; i >= 0; i--)  
 na[La - i] =  
 a[i] - '0'; //将字符串表示的大整形数转成i整形数组表示的大整形数  
 for (int i = Lb - 1; i >= 0; i--) nb[Lb - i] = b[i] - '0';  
 for (int i = 1; i <= La; i++)  
 for (int j = 1; j <= Lb; j++)  
 nc[i + j - 1] +=  
 na[i] \*  
 nb[j]; // a的第i位乘以b的第j位为积的第i+j-1位（先不考虑进位）  
 for (int i = 1; i <= La + Lb; i++)  
 nc[i + 1] += nc[i] / 10, nc[i] %= 10; //统一处理进位  
 if (nc[La + Lb]) s += nc[La + Lb] + '0'; //判断第i+j位上的数字是不是0  
 for (int i = La + Lb - 1; i >= 1; i--)  
 s += nc[i] + '0'; //将整形数组转成字符串  
 return s;  
}  
int sub(int \*a, int \*b, int La, int Lb) {  
 if (La < Lb) return -1; //如果a小于b，则返回-1  
 if (La == Lb) {  
 for (int i = La - 1; i >= 0; i--)  
 if (a[i] > b[i])  
 break;  
 else if (a[i] < b[i])  
 return -1; //如果a小于b，则返回-1  
 }  
 for (int i = 0; i < La; i++) //高精度减法  
 {  
 a[i] -= b[i];  
 if (a[i] < 0) a[i] += 10, a[i + 1]--;  
 }  
 for (int i = La - 1; i >= 0; i--)  
 if (a[i]) return i + 1; //返回差的位数  
 return 0; //返回差的位数  
}  
string div(string n1, string n2,  
 int nn) // n1,n2是字符串表示的被除数，除数,nn是选择返回商还是余数  
{  
 string s, v; // s存商,v存余数  
 int a[L], b[L], r[L],  
 La = n1.size(), Lb = n2.size(), i,  
 tp = La; // a，b是整形数组表示被除数，除数，tp保存被除数的长度  
 fill(a, a + L, 0);  
 fill(b, b + L, 0);  
 fill(r, r + L, 0); //数组元素都置为0  
 for (i = La - 1; i >= 0; i--) a[La - 1 - i] = n1[i] - '0';  
 for (i = Lb - 1; i >= 0; i--) b[Lb - 1 - i] = n2[i] - '0';  
 if (La < Lb || (La == Lb && n1 < n2)) {  
 // cout<<0<<endl;  
 return n1;  
 } //如果a<b,则商为0，余数为被除数  
 int t = La - Lb; //除被数和除数的位数之差  
 for (int i = La - 1; i >= 0; i--) //将除数扩大10^t倍  
 if (i >= t)  
 b[i] = b[i - t];  
 else  
 b[i] = 0;  
 Lb = La;  
 for (int j = 0; j <= t; j++) {  
 int temp;  
 while ((temp = sub(a, b + j, La, Lb - j)) >=  
 0) //如果被除数比除数大继续减  
 {  
 La = temp;  
 r[t - j]++;  
 }  
 }  
 for (i = 0; i < L - 10; i++)  
 r[i + 1] += r[i] / 10, r[i] %= 10; //统一处理进位  
 while (!r[i]) i--; //将整形数组表示的商转化成字符串表示的  
 while (i >= 0) s += r[i--] + '0';  
 // cout<<s<<endl;  
 i = tp;  
 while (!a[i]) i--; //将整形数组表示的余数转化成字符串表示的</span>  
 while (i >= 0) v += a[i--] + '0';  
 if (v.empty()) v = "0";  
 // cout<<v<<endl;  
 if (nn == 1) return s;  
 if (nn == 2) return v;  
}  
bool cmp(string a, string b) {  
 if (a.size() < b.size()) return 1; // a小于等于b返回真  
 if (a.size() == b.size() && a <= b) return 1;  
 return 0;  
}  
string DeletePreZero(string s) {  
 int i;  
 for (i = 0; i < s.size(); i++)  
 if (s[i] != '0') break;  
 return s.substr(i);  
}  
string BigInterSqrt(string n) {  
 n = DeletePreZero(n);  
 string l = "1", r = n, mid, ans;  
 while (cmp(l, r)) {  
 mid = div(add(l, r), "2", 1);  
 if (cmp(mul(mid, mid), n))  
 ans = mid, l = add(mid, "1");  
 else  
 r = sub(mid, "1");  
 }  
 return ans;  
}  
// o(n^3)

### 高精度取模（对单精度）

#include <bits/stdc++.h>  
using namespace std;  
int mod(string a,int b)//高精度a除以单精度b  
{  
 int d=0;  
 for(int i=0;i<a.size();i++) d=(d\*10+(a[i]-'0'))%b;//求出余数  
 return d;  
}  
//o(n)

### 欧拉筛

#include <bits/stdc++.h>  
using namespace std;  
typedef long long ll;  
const int N = 1000005;  
int phi[N], prime[N], cnt;  
bool st[N];  
void get\_eulers() {  
 phi[1] = 1;  
 for (int i = 2; i < N; i++) {  
 if (!st[i]) {  
 prime[cnt++] = i;  
 phi[i] = i - 1;  
 }  
 for (int j = 0; prime[j] \* i < N; j++) {  
 st[prime[j] \* i] = 1;  
 if (i % prime[j] == 0) {  
 phi[prime[j] \* i] = phi[i] \* prime[j];  
 break;  
 }  
 phi[prime[j] \* i] = phi[i] \* (prime[j] - 1);  
 }  
 }  
}  
int main() {  
 get\_eulers();  
 ll n;  
 cin >> n;  
 ll ans = 0;  
 for (int i = 1; i <= n; i++) ans += phi[i];  
 cout << ans;  
}

### 组合数（逆元线性递推

#include <bits/stdc++.h>  
using namespace std;  
typedef long long ll;  
const ll mod = 1e9 + 7;  
const ll maxn = 3e4 + 5;  
ll inv[maxn], fac[maxn];  
ll qpow(ll a, ll b) {  
 ll ans = 1;  
 while (b) {  
 if (b & 1) ans = (ans \* a) % mod;  
 a = (a \* a) % mod;  
 b >>= 1;  
 }  
 return ans;  
}  
ll c(ll n, ll m) {  
 if (n < 0 || m < 0 || n < m) return 0;  
 return fac[n] \* inv[n - m] % mod \* inv[m] % mod;  
}  
void init() {  
 fac[0] = 1;  
 for (int i = 1; i < maxn; i++) {  
 fac[i] = fac[i - 1] \* i % mod;  
 }  
 inv[maxn - 1] = qpow(fac[maxn - 1], mod - 2);  
 for (ll i = maxn - 2; i >= 0; i--) {  
 inv[i] = (inv[i + 1] \* (i + 1)) % mod;  
 }  
}

### 中国剩余定理

#include <bits/stdc++.h>  
using namespace std;  
typedef long long ll;  
const int maxn = 20;  
ll A[maxn], B[maxn];  
ll exgcd(ll a, ll b, ll & x, ll & y) {  
 if(b == 0) {  
 x = 1, y = 0;  
 return a;  
 }  
   
 ll d = exgcd(b, a % b, y, x);  
   
 y -= (a / b) \* x;  
   
 return d;   
}  
int main() {  
 int n;  
 cin >> n;  
 ll M = 1ll;  
 for(int i = 0; i < n; ++ i) {  
 cin >> A[i] >> B[i];  
 M = M \* A[i];  
 }  
   
 ll ans = 0;  
   
 ll x, y;  
   
 for(int i = 0; i < n; ++ i) {  
 ll Mi = M / A[i];  
 exgcd(Mi, A[i], x, y);  
 ans += B[i] \* Mi \* x;  
 }   
   
 cout << (ans % M + M) % M;  
   
}

## 图论

### 有源汇上下界最大小流

#include <bits/stdc++.h>  
using namespace std;  
typedef long long ll;  
struct Edge {  
 ll from, to, cap, flow, mn;  
 Edge(ll a, ll b, ll c, ll d, ll e) : from(a), to(b), cap(c), flow(d), mn(e) {}  
};  
ll n, m;  
struct Dinic {  
 static const ll maxn = 50010; // 点的大小，记得改  
 static const ll inf = 0x3f3f3f3f3f3f3f3f;  
 ll N, M, S, T;  
 vector<Edge> edges;  
 vector<ll> G[maxn];  
 bool vis[maxn];  
 ll d[maxn];  
 ll cur[maxn];  
 void AddEdge(ll from, ll to, ll cap, ll c) {  
 edges.push\_back(Edge(from, to, cap, 0, c));  
 edges.push\_back(Edge(to, from, 0, 0, c));  
 M = edges.size();  
 G[from].push\_back(M - 2);  
 G[to].push\_back(M - 1);  
 }  
 bool BFS() {  
 memset(vis, 0, sizeof(vis));  
 queue<ll> Q;  
 Q.push(S);  
 d[S] = 0;  
 vis[S] = 1;  
 while (!Q.empty()) {  
 ll x = Q.front();  
 Q.pop();  
 for (ll i = 0; i < G[x].size(); i++) {  
 Edge& e = edges[G[x][i]];  
 if (!vis[e.to] && e.cap > e.flow) {  
 vis[e.to] = 1;  
 d[e.to] = d[x] + 1;  
 Q.push(e.to);  
 }  
 }  
 }  
 return vis[T];  
 }  
 ll DFS(ll x, ll a) {  
 if (x == T || a == 0) return a;  
 ll flow = 0, f;  
 for (ll& i = cur[x]; i < G[x].size(); i++) {  
 Edge& e = edges[G[x][i]];  
 if (d[x] + 1 == d[e.to] &&  
 (f = DFS(e.to, min(a, e.cap - e.flow))) > 0) {  
 e.flow += f;  
 edges[G[x][i] ^ 1].flow -= f;  
 flow += f;  
 a -= f;  
 if (a == 0) break;  
 }  
 }  
 return flow;  
 }  
 void deleteEdge(ll u, ll v) {  
 ll siz = edges.size();  
 for(ll i = 0; i < siz; ++ i) {  
 if(edges[i].from == u && edges[i].to == v) {  
 edges[i].cap = edges[i].flow = 0;  
 edges[i ^ 1].cap = edges[i ^ 1].flow = 0;   
 break;   
 }  
 }  
 }  
 ll getValue() {  
 return edges[2 \* m].flow;  
 }   
 ll Maxflow(ll S, ll T) {  
 this->S = S, this->T = T;  
 ll flow = 0;  
 while (BFS()) {  
 memset(cur, 0, sizeof(cur));  
 flow += DFS(S, inf);  
 }  
 return flow;  
 }  
} MF;  
int main() {  
 ll s, t;  
 cin >> n >> m >> s >> t;  
 // n个点，m条边，给的源点汇点  
 ll mp[50010] = {0}; // 点的大小，记得改  
 for(ll i = 1; i <= m; ++ i) {  
 ll a, b, c, d; // 从a到b有一条下界c上界d的边  
 cin >> a >> b >> c >> d;  
 mp[b] += c;  
 mp[a] -= c;  
 MF.AddEdge(a, b, d - c, c);  
 }  
 MF.AddEdge(t, s, 1e18, 0); //  
 ll tot = 0;  
 for(ll i = 1; i <= n; ++ i) {  
 if(mp[i] > 0) {  
 tot += mp[i];  
 MF.AddEdge(0, i , mp[i], 0);  
 }  
 else {  
 MF.AddEdge(i, n + 1, -mp[i], 0);  
 }  
 }  
 if( MF.Maxflow(0, n + 1) != tot) {   
 cout << "No Solution" << endl;  
 }   
 else {  
 ll res = MF.getValue(); // 从t到s边的流量  
 MF.deleteEdge(t, s);  
 //cout << res + MF.Maxflow(s, t) << endl; // 最大流  
 cout << res - MF.Maxflow(t, s) << endl; // 最小流  
 }  
 return 0;  
}

### 树链剖分

ll fa[N], son[N], dep[N], siz[N], dfn[N], rnk[N], top[N];  
ll dfscnt;  
vector<ll> g[N];  
ll tree[N << 1];  
ll lazy[N << 1];  
void dfs1(ll u, ll f, ll d) {  
 son[u] = -1;  
 siz[u] = 1;  
 fa[u] = f;  
 dep[u] = d;  
 for (auto v:g[u]) {  
 if (v == f) continue;  
 dfs1(v, u, d + 1);  
 siz[u] += siz[v];  
 if (son[u] == -1 || siz[v] > siz[son[u]]) son[u] = v;  
 }  
}  
void dfs2(ll u, ll t) {  
 dfn[u] = ++dfscnt;  
 rnk[dfscnt] = u;  
 top[u] = t;  
 if (son[u] == -1) return;  
 dfs2(son[u], t);  
 for (auto v:g[u]) {  
 if (v == son[u] || v == fa[u]) continue;  
 dfs2(v, v);  
 }  
}  
ll lca(ll a, ll b) {  
 while (top[a] != top[b]) {  
 if (dep[top[a]] < dep[top[b]]) swap(a, b);  
 a = fa[top[a]];  
 }  
 return dep[a] < dep[b] ? a : b;  
}  
void init() {  
 for (ll i = 0; i < N; i++) g[i].clear();  
 for (ll i = 0; i < (N << 1); i++) {  
 tree[i] = 0;  
 lazy[i] = 0;  
 }  
 dfscnt = 0;  
}  
  
void pushdown(ll k, ll l, ll r) {  
 if (k >= N || lazy[k] == 0) return;  
 ll len = (r - l + 1) / 2;  
 tree[k << 1] = tree[k << 1] + len \* lazy[k];  
 tree[k << 1 | 1] = tree[k << 1 | 1] + len \* lazy[k];  
 lazy[k << 1] = lazy[k << 1] + lazy[k];  
 lazy[k << 1 | 1] = lazy[k << 1 | 1] + lazy[k];  
 lazy[k] = 0;  
}  
ll merge\_range(ll a, ll b) {  
 ll ans = a + b;  
 return ans;  
}  
void change\_range(ll k, ll l, ll r, ll ql, ll qr, ll x) {  
 if (r < ql || qr < l)return;  
 if (ql <= l && r <= qr) {  
 tree[k] = tree[k] + x \* (r - l + 1);  
 lazy[k] = lazy[k] + x;  
 return;  
 }  
 pushdown(k, l, r);  
 ll mid = (l + r) >> 1;  
 change\_range(k << 1, l, mid, ql, qr, x);  
 change\_range(k << 1 | 1, mid + 1, r, ql, qr, x);  
 tree[k] = merge\_range(tree[k << 1], tree[k << 1 | 1]);  
}  
ll query\_range(ll k, ll l, ll r, ll ql, ll qr) {  
 if (r < ql || qr < l)return 0;  
 if (ql <= l && r <= qr) {  
 return tree[k];  
 }  
 pushdown(k, l, r);  
 ll mid = (l + r) >> 1;  
 ll lq = query\_range(k << 1, l, mid, ql, qr);  
 ll rq = query\_range(k << 1 | 1, mid + 1, r, ql, qr);  
 return merge\_range(lq, rq);  
}  
ll query\_path(ll a, ll b) {  
 ll sum = 0;  
 while (top[a] != top[b]) {  
 if (dep[top[a]] < dep[top[b]]) swap(a, b);  
 sum = sum + query\_range(1, 1, N, dfn[top[a]], dfn[a]);  
 //dfn[top[a]]~dfn[a]  
 a = fa[top[a]];  
 }  
 if (dep[a] > dep[b]) swap(a, b);  
 //点权  
 sum = sum + query\_range(1, 1, N, dfn[a], dfn[b]);  
 //边权  
 //if (a != b) sum = sum + query\_range(1, 1, N, dfn[a] + 1, dfn[b]);  
 //dfn[a]~dfn[b],x  
 return sum;  
}  
void change\_path(ll a, ll b, ll x) {  
 while (top[a] != top[b]) {  
 if (dep[top[a]] < dep[top[b]]) swap(a, b);  
 change\_range(1, 1, N, dfn[top[a]], dfn[a], x);  
 //dfn[top[a]]~dfn[a]  
 a = fa[top[a]];  
 }  
 if (dep[a] > dep[b]) swap(a, b);  
 //点权  
 change\_range(1, 1, N, dfn[a], dfn[b], x);  
 //边权  
 //if (a != b) change\_range(1, 1, N, dfn[a] + 1, dfn[b], x);  
 //dfn[a]~dfn[b],x  
}

### 虚树

ll fa[N], son[N], dep[N], siz[N], dfn[N], rnk[N], top[N];  
ll dfscnt;  
vector<ll> g[N];  
ll mmin[N];  
void dfs1(ll u, ll f, ll d) {  
 son[u] = -1;  
 siz[u] = 1;  
 fa[u] = f;  
 dep[u] = d;  
 for (auto v:g[u]) {  
 if (v == f) continue;  
 dfs1(v, u, d + 1);  
 siz[u] += siz[v];  
 if (son[u] == -1 || siz[v] > siz[son[u]]) son[u] = v;  
 }  
}  
void dfs2(ll u, ll t) {  
 dfn[u] = ++dfscnt;  
 rnk[dfscnt] = u;  
 top[u] = t;  
 if (son[u] == -1) return;  
 dfs2(son[u], t);  
 for (auto v:g[u]) {  
 if (v == son[u] || v == fa[u]) continue;  
 dfs2(v, v);  
 }  
}  
ll lca(ll a, ll b) {  
 while (top[a] != top[b]) {  
 if (dep[top[a]] < dep[top[b]]) swap(a, b);  
 a = fa[top[a]];  
 }  
 return dep[a] < dep[b] ? a : b;  
}  
struct edge {  
 ll s, t, v;  
};  
edge e[N];  
vector<int> vg[N];  
int sta[N], tot;  
int h[N];  
void build(int \*H, int num) {  
 sort(H + 1, H + 1 + num, [](int a, int b) { return dfn[a] < dfn[b]; });  
 sta[tot = 1] = 1, vg[1].clear();// 1 号节点入栈，清空 1 号节点对应的邻接表，设置邻接表边数为 1  
 for (int i = 1, l; i <= num; ++i) {  
 if (H[i] == 1) continue; //如果 1 号节点是关键节点就不要重复添加  
 l = lca(H[i], sta[tot]); //计算当前节点与栈顶节点的 LCA  
 if (l != sta[tot]) { //如果 LCA 和栈顶元素不同，则说明当前节点不再当前栈所存的链上  
 while (dfn[l] < dfn[sta[tot - 1]]) {//当次大节点的 Dfs 序大于 LCA 的 Dfs 序  
 vg[sta[tot - 1]].push\_back(sta[tot]);  
 vg[sta[tot]].push\_back(sta[tot - 1]);  
 tot--;  
 } //把与当前节点所在的链不重合的链连接掉并且弹出  
 if (dfn[l] > dfn[sta[tot - 1]]) { //如果 LCA 不等于次大节点（这里的大于其实和不等于没有区别）  
 vg[l].clear();  
 vg[l].push\_back(sta[tot]);  
 vg[sta[tot]].push\_back(l);  
 sta[tot] = l;//说明 LCA 是第一次入栈，清空其邻接表，连边后弹出栈顶元素，并将 LCA 入栈  
 } else {  
 vg[l].push\_back(sta[tot]);  
 vg[sta[tot]].push\_back(l);  
 tot--; //说明 LCA 就是次大节点，直接弹出栈顶元素  
 }  
 }  
 vg[H[i]].clear();  
 sta[++tot] = H[i];  
 //当前节点必然是第一次入栈，清空邻接表并入栈  
 }  
 for (int i = 1; i < tot; ++i) {  
 vg[sta[i]].push\_back(sta[i + 1]);  
 vg[sta[i + 1]].push\_back(sta[i]);  
 } //剩余的最后一条链连接一下  
 return;  
}

### spfa最短路及负环

#include <bits/stdc++.h>  
using namespace std;  
typedef long long ll;  
const ll N = 1e6 + 10;  
struct edge {  
 ll to, v;  
 edge() {}  
 edge(ll a, ll b) : to(a), v(b) {}  
};  
edge e[N];  
ll h[N], vis[N], nxt[N], inque[N], tot, d[N];  
void init(ll n, ll m) {  
 tot = 0;  
 for (int i = 0; i <= n; i++) {  
 h[i] = vis[i] = inque[i] = 0;  
 }  
 for (int i = 0; i <= m; i++) {  
 nxt[i] = 0;  
 }  
}  
void addedge(ll a, ll b, ll v) {  
 nxt[++tot] = h[a];  
 e[tot] = edge(b, v);  
 h[a] = tot;  
}  
bool spfa(ll s, ll n) {  
 for (int i = 0; i <= n; i++) d[i] = INT\_MAX;  
 d[s] = 0;  
 queue<int> que;  
 que.push(s);  
 inque[s] = 1;  
 while (!que.empty()) {  
 int p = que.front();  
 que.pop();  
 vis[p] = 0;  
 for (int k = h[p]; k; k = nxt[k]) {  
 if (d[e[k].to] > d[p] + e[k].v) {  
 d[e[k].to] = d[p] + e[k].v;  
 if (!vis[e[k].to]) {  
 inque[e[k].to]++;  
 if (inque[e[k].to] > n) return 0;  
 vis[e[k].to] = 1;  
 que.push(e[k].to);  
 }  
 }  
 }  
 }  
 return 1;  
}  
int main() {  
 ios::sync\_with\_stdio(0), cin.tie(0), cout.tie(0);  
 int t;  
 cin >> t;  
 while (t--) {  
 int n, m;  
 cin >> n >> m;  
 init(n, m \* 2);  
 for (int i = 1; i <= m; i++) {  
 int a, b, c;  
 cin >> a >> b >> c;  
 addedge(a, b, c);  
 if (c >= 0) addedge(b, a, c);  
 }  
 if (spfa(1, n))  
 cout << "NO" << endl;  
 else  
 cout << "YES" << endl;  
 }  
}

### 二分图匹配（匈牙利）

//大量使用了memset，但常数貌似很小？HDU6808跑了998ms（限制5000ms），然而这个代int main()不是HDU6808的  
#include<bits/stdc++.h>  
using namespace std;  
const int maxn=505;// 最大点数  
const int inf=0x3f3f3f3f;// 距离初始值  
struct HK\_Hungary{//这个板子从1开始，0点不能用,nx为左边点数，ny为右边点数  
 int nx,ny;//左右顶点数量  
 vector<int>bmap[maxn];  
 int cx[maxn];//cx[i]表示左集合i顶点所匹配的右集合的顶点序号  
 int cy[maxn]; //cy[i]表示右集合i顶点所匹配的左集合的顶点序号  
 int dx[maxn];  
 int dy[maxn];  
 int dis;  
 bool bmask[maxn];  
 void init(int a,int b){  
 nx=a,ny=b;  
 for(int i=0;i<=nx;i++){  
 bmap[i].clear();  
 }  
 }  
 void add\_edge(int u,int v){  
 bmap[u].push\_back(v);  
 }  
 bool searchpath(){//寻找 增广路径  
 queue<int>Q;  
 dis=inf;  
 memset(dx,-1,sizeof(dx));  
 memset(dy,-1,sizeof(dy));  
 for(int i=1;i<=nx;i++){//cx[i]表示左集合i顶点所匹配的右集合的顶点序号  
 if(cx[i]==-1){//将未遍历的节点 入队 并初始化次节点距离为0  
 Q.push(i);  
 dx[i]=0;  
 }  
 }//广度搜索增广路径  
 while(!Q.empty()){  
 int u=Q.front();  
 Q.pop();  
 if(dx[u]>dis) break;//取右侧节点  
 for(int i=0;i<bmap[u].size();i++){  
 int v=bmap[u][i];//右侧节点的增广路径的距离  
 if(dy[v]==-1){  
 dy[v]=dx[u]+1;//v对应的距离 为u对应距离加1  
 if(cy[v]==-1)dis=dy[v];  
 else{  
 dx[cy[v]]=dy[v]+1;  
 Q.push(cy[v]);  
 }  
 }  
 }  
 }  
 return dis!=inf;  
 }  
 int findpath(int u){//寻找路径 深度搜索  
 for(int i=0;i<bmap[u].size();i++){  
 int v=bmap[u][i];//如果该点没有被遍历过 并且距离为上一节点+1  
 if(!bmask[v]&&dy[v]==dx[u]+1){//对该点染色  
 bmask[v]=1;  
 if(cy[v]!=-1&&dy[v]==dis)continue;  
 if(cy[v]==-1||findpath(cy[v])){  
 cy[v]=u;cx[u]=v;  
 return 1;  
 }  
 }  
 }  
 return 0;  
 }  
 int MaxMatch(){//得到最大匹配的数目  
 int res=0;  
 memset(cx,-1,sizeof(cx));  
 memset(cy,-1,sizeof(cy));  
 while(searchpath()){  
 memset(bmask,0,sizeof(bmask));  
 for(int i=1;i<=nx;i++){  
 if(cx[i]==-1){  
 res+=findpath(i);  
 }  
 }  
 }  
 return res;  
 }  
}HK;  
int main(){  
 int nn,n,m;  
 cin>>nn;  
 while(nn--){  
 scanf("%d%d",&n,&m);  
 HK.init(n,m);//左端点和右端点数量  
 for(int i=1;i<=n;i++){  
 int snum;  
 cin>>snum;  
 int v;  
 for(int j=1;j<=snum;j++){  
 cin>>v;  
 HK.add\_edge(i,v);//连边  
 }  
 }  
 cout<<HK.MaxMatch()<<endl;//求最大匹配  
 }  
 return 0;  
}

### 强连通（kosaraju

#include <bits/stdc++.h>  
using namespace std;  
struct SCC {  
 static const int MAXV = 100000;  
 int V;  
 vector<int> g[MAXV], rg[MAXV], vs;  
 bool used[MAXV];  
 int cmp[MAXV];  
 void add\_edge(int from, int to) {  
 g[from].push\_back(to);  
 rg[to].push\_back(from);  
 }  
 void dfs(int v) {  
 used[v] = 1;  
 for (int i = 0; i < g[v].size(); i++) {  
 if (!used[g[v][i]]) dfs(g[v][i]);  
 }  
 vs.push\_back(v);  
 }  
 void rdfs(int v, int k) {  
 used[v] = 1;  
 cmp[v] = k;  
 for (int i = 0; i < rg[v].size(); i++) {  
 if (!used[rg[v][i]]) rdfs(rg[v][i], k);  
 }  
 }  
 int solve() {  
 memset(used, 0, sizeof(used));  
 vs.clear();  
 for (int v = 1; v <= V; v++) {  
 if (!used[v]) dfs(v);  
 }  
 memset(used, 0, sizeof(used));  
 int k = 0;  
 for (int i = (int)vs.size() - 1; i >= 0; i--) {  
 if (!used[vs[i]]) rdfs(vs[i], ++k);  
 }  
 return k;  
 }  
 void init(int n) {  
 V = n;  
 vs.clear();  
 for (int i = 0; i < MAXV; i++) {  
 g[i].clear();  
 rg[i].clear();  
 used[i] = 0;  
 cmp[i] = 0;  
 }  
 }  
} scc;  
//记得调用init()

### 强连通（tarjan

#include <bits/stdc++.h>  
using namespace std;  
struct SCC {  
 static const int MAXN = 100000;  
 vector<int> g[MAXN];  
 int dfn[MAXN], lowlink[MAXN], sccno[MAXN], dfs\_clock, scc\_cnt;  
 stack<int> S;  
 void dfs(int u) {  
 dfn[u] = lowlink[u] = ++dfs\_clock;  
 S.push(u);  
 for (int i = 0; i < g[u].size(); i++) {  
 int v = g[u][i];  
 if (!dfn[v]) {  
 dfs(v);  
 lowlink[u] = min(lowlink[u], lowlink[v]);  
 } else if (!sccno[v]) {  
 lowlink[u] = min(lowlink[u], dfn[v]);  
 }  
 }  
 if (lowlink[u] == dfn[u]) {  
 ++scc\_cnt;  
 for (;;) {  
 int x = S.top();  
 S.pop();  
 sccno[x] = scc\_cnt;  
 if (x == u) break;  
 }  
 }  
 }  
 void solve(int n) {  
 dfs\_clock = scc\_cnt = 0;  
 memset(sccno, 0, sizeof(sccno));  
 memset(dfn, 0, sizeof(dfn));  
 memset(lowlink, 0, sizeof(lowlink));  
 for (int i = 1; i <= n; i++) {  
 if (!dfn[i]) dfs(i);  
 }  
 }  
} scc;  
// scc\_cnt为SCC计数器，sccno[i]为i所在SCC的编号  
// vector<int> g[MAXN]中加边  
//之后再补充init()

### 强连通（tarjan无vector

#include <bits/stdc++.h>  
using namespace std;  
struct SCC {  
 static const int MAXN = 5000;  
 static const int MAXM = 2000000;  
 int dfs\_clock, edge\_cnt = 1, scc\_cnt;  
 int head[MAXN];  
 int dfn[MAXN], lowlink[MAXN];  
 int sccno[MAXN];  
 stack<int> s;  
 struct edge {  
 int v, next;  
 } e[MAXM];  
 void add\_edge(int u, int v) {  
 e[edge\_cnt].v = v;  
 e[edge\_cnt].next = head[u];  
 head[u] = edge\_cnt++;  
 }  
 void tarjan(int u) {  
 int v;  
 dfn[u] = lowlink[u] = ++dfs\_clock; //每次dfs，u的次序号增加1  
 s.push(u); //将u入栈  
 for (int i = head[u]; i != -1; i = e[i].next) //访问从u出发的边  
 {  
 v = e[i].v;  
 if (!dfn[v]) //如果v没被处理过  
 {  
 tarjan(v); // dfs(v)  
 lowlink[u] = min(lowlink[u], lowlink[v]);  
 } else if (!sccno[v])  
 lowlink[u] = min(lowlink[u], dfn[v]);  
 }  
 if (dfn[u] == lowlink[u]) {  
 scc\_cnt++;  
 do {  
 v = s.top();  
 s.pop();  
 sccno[v] = scc\_cnt;  
 } while (u != v);  
 }  
 }  
 int find\_scc(int n) {  
 for (int i = 1; i <= n; i++)  
 if (!dfn[i]) tarjan(i);  
 return scc\_cnt;  
 }  
 void init() {  
 scc\_cnt = dfs\_clock = 0;  
 edge\_cnt = 1; //不用初始化e数组，省时间  
 while (!s.empty()) s.pop();  
 memset(head, -1, sizeof(head));  
 memset(sccno, 0, sizeof(sccno));  
 memset(dfn, 0, sizeof(dfn));  
 memset(lowlink, 0, sizeof(lowlink));  
 }  
} scc;

### 最大流

#include <bits/stdc++.h>  
using namespace std;  
typedef long long ll;  
struct Edge {  
 ll from, to, cap, flow;  
 Edge(ll a, ll b, ll c, ll d) : from(a), to(b), cap(c), flow(d) {}  
};  
struct Dinic {  
 static const ll maxn = 10000;  
 static const ll inf = 0x3f3f3f3f3f3f3f3f;  
 ll N, M, S, T;  
 vector<Edge> edges;  
 vector<ll> G[maxn];  
 bool vis[maxn];  
 ll d[maxn];  
 ll cur[maxn];  
 void AddEdge(ll from, ll to, ll cap) {  
 edges.push\_back(Edge(from, to, cap, 0));  
 edges.push\_back(Edge(to, from, 0, 0));  
 M = edges.size();  
 G[from].push\_back(M - 2);  
 G[to].push\_back(M - 1);  
 }  
 bool BFS() {  
 memset(vis, 0, sizeof(vis));  
 queue<ll> Q;  
 Q.push(S);  
 d[S] = 0;  
 vis[S] = 1;  
 while (!Q.empty()) {  
 ll x = Q.front();  
 Q.pop();  
 for (ll i = 0; i < G[x].size(); i++) {  
 Edge& e = edges[G[x][i]];  
 if (!vis[e.to] && e.cap > e.flow) {  
 vis[e.to] = 1;  
 d[e.to] = d[x] + 1;  
 Q.push(e.to);  
 }  
 }  
 }  
 return vis[T];  
 }  
 ll DFS(ll x, ll a) {  
 if (x == T || a == 0) return a;  
 ll flow = 0, f;  
 for (ll& i = cur[x]; i < G[x].size(); i++) {  
 Edge& e = edges[G[x][i]];  
 if (d[x] + 1 == d[e.to] &&  
 (f = DFS(e.to, min(a, e.cap - e.flow))) > 0) {  
 e.flow += f;  
 edges[G[x][i] ^ 1].flow -= f;  
 flow += f;  
 a -= f;  
 if (a == 0) break;  
 }  
 }  
 return flow;  
 }  
 ll Maxflow(ll S, ll T) {  
 this->S = S, this->T = T;  
 ll flow = 0;  
 while (BFS()) {  
 memset(cur, 0, sizeof(cur));  
 flow += DFS(S, inf);  
 }  
 return flow;  
 }  
} MF;

### 最大流（double）

#include <iostream>  
#include <cstring>  
#include <algorithm>  
using namespace std;  
struct Dinic {  
 static constexpr int N = 10010, M = 100010, INF = 1e8;  
 static constexpr double eps = 1e-8;   
// int n, m, S, T;  
 int S, T;  
 int h[N], e[M], ne[M], idx;  
 double f[M];  
 int q[N], d[N], cur[N]; // d 表示从源点开始走到该点的路径上所有边的容量的最小值   
   
 void AddEdge(int a, int b, double c)  
 {  
 e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx ++ ;  
 e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx ++ ;  
 }  
   
 bool bfs()  
 {  
 int hh = 0, tt = 0;  
 memset(d, -1, sizeof d);  
 q[0] = S, d[S] = 0, cur[S] = h[S];  
 while (hh <= tt)  
 {  
 int t = q[hh ++ ];  
 for (int i = h[t]; ~i; i = ne[i])  
 {  
 int ver = e[i];  
 if (d[ver] == -1 && f[i] > 0)  
 {  
 d[ver] = d[t] + 1;  
 cur[ver] = h[ver];  
 if (ver == T) return true;  
 q[ ++ tt] = ver;  
 }  
 }  
 }  
 return false;  
 }  
   
 double find(int u, double limit)  
 {  
 if (u == T) return limit;  
 double flow = 0;  
 for (int i = cur[u]; ~i && flow < limit; i = ne[i])  
 {  
 cur[u] = i;  
 int ver = e[i];  
 if (d[ver] == d[u] + 1 && f[i] > 0)  
 {  
 double t = find(ver, min(f[i], limit - flow));  
 if (t < eps) d[ver] = -1;  
 f[i] -= t, f[i ^ 1] += t, flow += t;  
 }  
 }  
 return flow;  
 }  
   
 double Maxflow(int S, int T)  
 {  
 this->S = S, this->T = T;  
 double r = 0, flow;  
 while (bfs()) while (flow = find(S, INF)) r += flow;  
 return r;  
 }   
 void init() ////////   
 {  
 memset(h, -1, sizeof h);  
 idx = 0;   
 }  
} MF;  
// ?èinit

### 最小费用最大流

#include <bits/stdc++.h>  
using namespace std;  
typedef long long ll;  
struct Edge {  
 ll from, to, cap, flow, cost;  
 Edge(ll u, ll v, ll c, ll f, ll w):from(u), to(v), cap(c), flow(f), cost(w) {}  
};  
struct MCMF {  
 static const ll maxn = 6000;  
 static const ll INF = 0x3f3f3f3f3f3f3f;  
 ll n, m;  
 vector<Edge> edges;  
 vector<ll> G[maxn];  
 ll inq[maxn];  
 ll d[maxn];  
 ll p[maxn];  
 ll a[maxn];  
 void init(ll n) {  
 this->n = n;  
 for (ll i = 1; i <= n; i++) G[i].clear();  
 edges.clear();  
 }  
 void add\_edge(ll from, ll to, ll cap, ll cost) {  
 from++,to++;//原板子无法使用0点，故修改  
 edges.push\_back(Edge(from, to, cap, 0, cost));  
 edges.push\_back(Edge(to, from, 0, 0, -cost));  
 m = edges.size();  
 G[from].push\_back(m - 2);  
 G[to].push\_back(m - 1);  
 }  
 bool BellmanFord(ll s, ll t, ll& flow, ll& cost) {  
 for (ll i = 1; i <= n; ++i) d[i] = INF;  
 memset(inq, 0, sizeof(inq));  
 d[s] = 0, inq[s] = 1, p[s] = 0, a[s] = INF;  
 queue<ll> Q;  
 Q.push(s);  
 while (!Q.empty()) {  
 ll u = Q.front();  
 Q.pop();  
 inq[u] = 0;  
 for (ll i = 0; i < G[u].size(); ++i) {  
 Edge& e = edges[G[u][i]];  
 if (e.cap > e.flow && d[e.to] > d[u] + e.cost) {  
 d[e.to] = d[u] + e.cost;  
 p[e.to] = G[u][i];  
 a[e.to] = min(a[u], e.cap - e.flow);  
 if (!inq[e.to]) {  
 Q.push(e.to);  
 inq[e.to] = 1;  
 }  
 }  
 }  
 }  
 if (d[t] == INF) return false;  
 flow += a[t];  
 cost += (ll)d[t] \* (ll)a[t];  
 for (ll u = t; u != s; u = edges[p[u]].from) {  
 edges[p[u]].flow += a[t];  
 edges[p[u] ^ 1].flow -= a[t];  
 }  
 return true;  
 }  
 //需要保证初始网络中没有负权圈  
 ll MincostMaxflow(ll s, ll t, ll& cost) {  
 s++,t++;//原板子无法使用0点，故修改  
 ll flow = 0;  
 cost = 0;  
 while (BellmanFord(s, t, flow, cost));  
 return flow;  
 }  
} mcmf; // 若固定流量k，增广时在flow+a>=k的时候只增广k-flow单位的流量，然后终止程序

### 树分治

#include <bits/stdc++.h>  
using namespace std;  
const int MAXN = 10005;  
const int INF = 1000000000;  
struct edge {  
 int to, length;  
 edge() {}  
 edge(int a, int b) : to(a), length(b) {}  
};  
int N, K;  
vector<edge> g[MAXN];  
bool centroid[MAXN];  
int subtree\_size[MAXN];  
int ans;  
//计算子树大小  
int compute\_subtree\_size(int v, int p) {  
 int c = 1;  
 for (int i = 0; i < g[v].size(); i++) {  
 int w = g[v][i].to;  
 if (w == p || centroid[w]) continue;  
 c += compute\_subtree\_size(w, v);  
 }  
 subtree\_size[v] = c;  
 return c;  
}  
//查找重心，t为连通分量大小  
// pair（最大子树顶点数，顶点编号）  
pair<int, int> search\_centroid(int v, int p, int t) {  
 pair<int, int> res = pair<int, int>(INF, -1);  
 int s = 1, m = 0;  
 for (int i = 0; i < g[v].size(); i++) {  
 int w = g[v][i].to;  
 if (w == p || centroid[w]) continue;  
 res = min(res, search\_centroid(w, v, t));  
 m = max(m, subtree\_size[w]);  
 s += subtree\_size[w];  
 }  
 m = max(m, t - s);  
 res = min(res, pair<int, int>(m, v));  
 return res;  
}  
void init() {  
 memset(centroid, 0, sizeof(centroid));  
 memset(subtree\_size, 0, sizeof(subtree\_size));  
 for (int i = 0; i <= N; i++) g[i].clear();  
 ans = 0;  
}  
int solve(int u) {  
 compute\_subtree\_size(u, -1);  
 int s = search\_centroid(u, -1, subtree\_size[u]).second;  
 centroid[s] = 1;  
 int ans;  
 for (int i = 0; i < g[s].size(); i++) {  
 int v = g[s][i].to;  
 if (centroid[v]) continue;  
 /\*solve()\*/  
 }  
 /\*do something\*/  
 centroid[s] = 0;  
 return ans;  
}

### 拓扑排序

#include <bits/stdc++.h>  
using namespace std;  
const int MAXN = 100000;  
int c[MAXN];  
int topo[MAXN], t, V;  
vector<int> g[MAXN];  
bool dfs(int u) {  
 c[u] = -1;  
 for (int i = 0; i < g[u].size(); i++) {  
 int v = g[u][i];  
 if (c[v] < 0)  
 return false;  
 else if (!c[v] && !dfs(v))  
 return false;  
 }  
 c[u] = 1;  
 topo[t--] = u;  
 return true;  
}  
bool toposort(int n) {  
 V = n;  
 t = n;  
 memset(c, 0, sizeof(c));  
 for (int u = 1; u <= V; u++)  
 if (!c[u] && !dfs(u)) return false;  
 return true;  
}

### 最近公共祖先（倍增）

#include <algorithm>  
#include <cstdio>  
#include <cstring>  
#include <iostream>  
using namespace std;  
const int MAX = 600000;  
struct edge {  
 int t, nex;  
} e[MAX << 1];  
int head[MAX], tot;  
int depth[MAX], fa[MAX][22], lg[MAX];  
void add\_edge(int x, int y) {  
 e[++tot].t = y;  
 e[tot].nex = head[x];  
 head[x] = tot;  
 e[++tot].t = x;  
 e[tot].nex = head[y];  
 head[y] = tot;  
}  
void dfs(int now, int fath) {  
 fa[now][0] = fath;  
 depth[now] = depth[fath] + 1;  
 for (int i = 1; i <= lg[depth[now]]; ++i)  
 fa[now][i] = fa[fa[now][i - 1]][i - 1];  
 for (int i = head[now]; i; i = e[i].nex)  
 if (e[i].t != fath) dfs(e[i].t, now);  
}  
int lca(int x, int y) {  
 if (depth[x] < depth[y]) swap(x, y);  
 while (depth[x] > depth[y]) x = fa[x][lg[depth[x] - depth[y]] - 1];  
 if (x == y) return x;  
 for (int k = lg[depth[x]] - 1; k >= 0; --k)  
 if (fa[x][k] != fa[y][k]) x = fa[x][k], y = fa[y][k];  
 return fa[x][0];  
}  
void init(int n, int root) {  
 for (int i = 1; i <= n; ++i) lg[i] = lg[i - 1] + (1 << lg[i - 1] == i);  
 dfs(root, 0);  
}

### 最近公共祖先（线段树）

#include <bits/stdc++.h>  
using namespace std;  
int n, m, root;  
const int MAX\_N = 500005;  
const int MAX = 1 << 20;  
vector<int> g[MAX\_N];  
vector<int> vs;  
pair<int, int> tree[MAX \* 2 + 10];  
int fir[MAX\_N];  
int fa[MAX\_N];  
int dep[MAX\_N];  
void dfs(int k, int p, int d) {  
 fa[k] = p;  
 dep[k] = d;  
 vs.push\_back(k);  
 for (int i = 0; i < g[k].size(); i++) {  
 if (g[k][i] != p) {  
 dfs(g[k][i], k, d + 1);  
 vs.push\_back(k);  
 }  
 }  
}  
void build(int k) {  
 if (k >= MAX) return;  
 build(k << 1);  
 build(k << 1 | 1);  
 tree[k] = min(tree[k << 1], tree[k << 1 | 1]);  
}  
pair<int, int> query(int k, int s, int e, int l, int r) {  
 if (e < l || r < s) return pair<int, int>(INT\_MAX, 0);  
 if (l <= s && e <= r) return tree[k];  
 return min(query(k << 1, s, (s + e) >> 1, l, r),  
 query(k << 1 | 1, ((s + e) >> 1) + 1, e, l, r));  
}  
void init() {  
 dfs(root, root, 0);  
 for (int i = 0; i < MAX \* 2 + 10; i++) tree[i] = pair<int, int>(INT\_MAX, 0);  
 for (int i = MAX; i < MAX + vs.size(); i++)  
 tree[i] = pair<int, int>(dep[vs[i - MAX]], vs[i - MAX]);  
 for (int i = 0; i < vs.size(); i++) {  
 if (fir[vs[i]] == 0) fir[vs[i]] = i + 1;  
 }  
 build(1);  
}  
int lca(int a, int b) {  
 return query(1, 1, MAX, min(fir[a], fir[b]), max(fir[a], fir[b])).second;  
}  
int main() {  
 scanf("%d%d%d", &n, &m, &root);  
 for (int i = 1; i < n; i++) {  
 int a, b;  
 scanf("%d%d", &a, &b);  
 g[a].push\_back(b);  
 g[b].push\_back(a);  
 }  
 init();  
 for (int i = 1; i <= m; i++) {  
 int a, b;  
 scanf("%d%d", &a, &b);  
 printf("%d\n", lca(a, b));  
 }  
}

## 线性代数

### 高斯消元

#include <iostream>  
#include <vector>  
using namespace std;  
const double eps = 1e-8;  
void sway(vector<double>& a, vector<double>& b) {  
 vector<double> s;  
 for (int i = 0; i < a.size(); i++) {  
 s.push\_back(a[i]);  
 }  
 a.clear();  
 for (int i = 0; i < b.size(); i++) {  
 a.push\_back(b[i]);  
 }  
 b.clear();  
 for (int i = 0; i < s.size(); i++) {  
 b.push\_back(s[i]);  
 }  
}  
vector<double> gauss\_jordan(const vector<vector<double> >& A,  
 const vector<double>& b) {  
 int n = A.size();  
 vector<vector<double> > B(n, vector<double>(n + 1));  
 for (int i = 0; i < n; i++)  
 for (int j = 0; j < n; j++) B[i][j] = A[i][j];  
 for (int i = 0; i < n; i++) B[i][n] = b[i];  
 for (int i = 0; i < n; i++) {  
 int pivot = i;  
 for (int j = i; j < n; j++) {  
 if (abs(B[j][i]) > abs(B[pivot][i])) pivot = j;  
 }  
 swap(B[i], B[pivot]);  
 if (abs(B[i][i]) < eps) return vector<double>();  
 for (int j = i + 1; j <= n; j++) B[i][j] /= B[i][i];  
 for (int j = 0; j < n; j++) {  
 if (i != j) {  
 for (int k = i + 1; k <= n; k++) B[j][k] -= B[j][i] \* B[i][k];  
 }  
 }  
 }  
 vector<double> x(n);  
 for (int i = 0; i < n; i++) x[i] = B[i][n];  
 return x;  
}  
int main() {  
 int n, m;  
 cin >> n >> m;  
 vector<vector<double> > mat(n, vector<double>(m));  
 for (int i = 0; i < n; i++) {  
 for (int j = 0; j < m; j++) {  
 cin >> mat[i][j];  
 }  
 }  
 vector<double> val(n);  
 for (int i = 0; i < n; i++) cin >> val[i];  
 vector<double> ans = gauss\_jordan(mat, val);  
 for (int i = 0; i < ans.size(); i++) cout << ans[i] << ' ';  
}

### 矩阵行列式

#include <bits/stdc++.h>  
using namespace std;  
typedef long long ll;  
const ll mod = 1e9 + 7;  
struct Matrix {  
 static const ll MAXN = 300;  
 ll a[MAXN][MAXN];  
 void init() { memset(a, 0, sizeof(a)); }  
 ll det(ll n) {  
 for (int i = 0; i < n; i++)  
 for (int j = 0; j < n; j++) a[i][j] = (a[i][j] + mod) % mod;  
 ll res = 1;  
 for (int i = 0; i < n; i++) {  
 if (!a[i][i]) {  
 bool flag = false;  
 for (int j = i + 1; j < n; j++) {  
 if (a[j][i]) {  
 flag = true;  
 for (int k = i; k < n; k++) {  
 swap(a[i][k], a[j][k]);  
 }  
 res = -res;  
 break;  
 }  
 }  
 if (!flag) return 0;  
 }  
 for (int j = i + 1; j < n; j++) {  
 while (a[j][i]) {  
 ll t = a[i][i] / a[j][i];  
 for (int k = i; k < n; k++) {  
 a[i][k] = (a[i][k] - t \* a[j][k]) % mod;  
 swap(a[i][k], a[j][k]);  
 }  
 res = -res;  
 }  
 }  
 res \*= a[i][i];  
 res %= mod;  
 }  
 return (res + mod) % mod;  
 }  
} mat;

### 线性基

//  
const int maxbit = 62; //maxbit不能太大  
struct L\_B{  
 ll lba[maxbit];  
 L\_B(){  
 memset(lba, 0, sizeof(lba));  
 }  
   
 void Insert(ll val){ //插入  
 for(int i = maxbit - 1; i >= 0; -- i) // 从高位向低位扫   
 if(val & (1ll << i)){ //   
 if(!lba[i]){  
 lba[i] = val;  
 break;  
 }  
 val ^= lba[i];  
 }  
 }  
};  
//对原集合的每个数val转为2进制，从高位向低位扫，对于当前位为1的，若lba[i]不存在就令lba[i]=x，否则令val=val`xor`lba[i]  
//使用： 直接insert   
// --------------线性基模板

### 线性基2

线性基 能表示的线性空间与原向量 能表示的线性空间等价

用高斯消元得到线性基

先输入数组a[] 中

int n, k;  
ll a[N];  
void getVec() {  
 k = 0;  
 for(int i = 62; i >= 0; -- i) {  
 for(int j = k; j < n; ++ j) {  
 if(a[j] >> i & 1) {  
 swap(a[j], a[k]);  
 break;  
 }  
 }  
 if(!(a[k] >> i & 1)) continue;  
 for(int j = 0; j < n; ++j) {  
 if(j != k && (a[j] >> i & 1)) {  
 a[j] ^= a[k];  
 }  
 }  
 ++k;  
 if(k == n) break;  
 }  
}

这里注意最后的线性基是a[]中从0到k-1个，在前的是**高位**

### 矩阵（加减乘快速幂

//矩阵类模板  
struct Matrix{  
 int n,m;  
 int a[maxn][maxm];  
 void clear(){  
 n=m=0;  
 memset(a,0,sizeof(a));  
 }  
 Matrix operator +(const Matrix &b) const {  
 Matrix tmp;  
 tmp.n=n;tmp.m=m;  
 for (int i=0;i<n;++i)  
 for(int j=0;j<m;++j)  
 tmp.a[i][j]=a[i][j]+b.a[i][j];  
 return tmp;  
 }  
 Matrix operator -(const Matrix &b)const{  
 Matrix tmp;  
 tmp.n=n;tmp.m=m;  
 for (int i=0;i<n;++i){  
 for(int j=0;j<m;++j)  
 tmp.a[i][j]=a[i][j]-b.a[i][j];  
 }  
   
 return tmp;  
 }  
 Matrix operator \* (const Matrix &b) const{  
 Matrix tmp;  
 tmp.clear();  
 tmp.n=n;tmp.m=b.m;  
 for (int i=0;i<n;++i)  
 for(int j=0;j<b.m;++j)  
 for (int k=0;k<m;++k){  
 tmp.a[i][j]+=a[i][k]\*b.a[k][j];  
 tmp.a[i][j]%=mod;  
 }  
 return tmp;  
 }  
 Matrix get(int x){//幂运算  
 Matrix E;  
 E.clear();  
 E.n=E.m=n;  
 for(int i=0;i<n;++i)  
 E.a[i][i]=1;  
 if(x==0) return E;  
 else if(x==1) return \*this;  
 Matrix tmp=get(x/2);  
 tmp=tmp\*tmp;  
 if(x%2) tmp=tmp\*(\*this);  
 return tmp;  
 }  
};  
//矩阵模板结束

### 稀疏矩阵乘法

struct Matrix{  
 int n,m;  
 int a[maxn][maxn];////  
 void clear(){  
 n=m=0;  
 memset(a,0,sizeof(a));  
 }  
 Matrix operator \* (const Matrix &b) const{  
 Matrix tmp;  
 tmp.clear();  
 tmp.n=n;tmp.m=b.m;  
 for (int k=0;k<m;++k){  
 for (int i=0;i<n;++i){  
 if(a[i][k]==0) continue;  
 for(int j=0;j<b.m;++j){  
 if(b.a[k][j]==0) continue;  
 tmp.a[i][j]+=a[i][k]\*b.a[k][j];  
 tmp.a[i][j]%=mod;  
 }   
 }   
 }  
 return tmp;  
 }  
};  
//稀疏矩阵乘法

## 杂项

### 快读

inline int read(){  
 int X=0,w=0;char ch=0;  
 while(!isdigit(ch)){w|=ch=='-';ch=getchar();}  
 while(isdigit(ch))X=(X<<3)+(X<<1)+(ch^48),ch=getchar();  
 return w?-X:X;  
}

### fread快读

#include <bits/stdc++.h>  
using namespace std;  
char next\_char() {  
 static char buf[1 << 20], \*first, \*last;  
 if(first == last) {  
 last = buf + fread(buf, 1, 1 << 20, stdin);  
 first = buf;  
 }  
 return first == last ? EOF : \*first ++;  
}  
inline int read(){  
 int x = 0, w = 0; char ch = 0;  
 while(!isdigit(ch)) {w |= ch == '-'; ch = next\_char(); }  
 while(isdigit(ch)) {x = (x << 3) + (x << 1) + (ch ^ 48), ch = next\_char(); }  
 return w ? -x : x;  
}  
int main(){  
 freopen("1.txt", "r", stdin); // 交代码的时候一定要去掉aaa   
 int T;  
 cin >> T;  
 while(T --){  
 int x = read();  
 cout << x << endl;  
 }  
}

### 模拟退火

“优化的随机算法”

连续函数找区间最优

// 找一个点，与平面中的n个点的距离和最近

//进行多次模拟退火避免局部最大值

#include <bits/stdc++.h>  
#include <ctime>  
using namespace std;  
const int maxn = 110;  
int n;  
#define x first  
#define y second  
typedef pair<double, double> PDD;  
PDD q[maxn];   
double ans = 1e8;  
double rand(double l, double r) {  
 return (double) rand() / RAND\_MAX \* (r - l) + l;   
}  
double getDist(PDD a, PDD b) {  
 double dx = a.x - b.x;  
 double dy = a.y - b.y;  
 return sqrt(dx \* dx + dy \* dy) ;  
}  
double calc(PDD p) {  
 double res = 0;  
 for(int i = 0; i < n; ++ i) {  
 res += getDist(q[i], p);  
 }  
 ans = min(ans, res);  
 return res;  
}  
double simulate\_anneal() {  
 PDD cur(rand(0, 10000), rand(0, 10000)); // 随机一个起点  
 for(double T = 1e4; T > 1e-4; T = T \* 0.99) { // 初始温度，末态温度，衰减系数，一般调整衰减系数0.999 0.95  
 PDD np(rand(cur.x - T, cur.x + T), rand(cur.y - T, cur.y + T)); // 随机新点  
 double delta = calc(np) - calc(cur);  
 if(exp(-delta / T) > rand(0, 1)) cur = np; //如果新点比现在的点更优，必过去，不然有一定概率过去  
 }  
}  
int main() {  
 cin >> n;  
 for(int i = 0; i < n; ++ i) {  
 cin >> q[i].x >> q[i].y;   
 }  
 while((double) clock() / CLOCKS\_PER\_SEC < 0.8) { // 卡时 // 或for（100）  
 simulate\_anneal();   
 }  
 cout << (int)(ans + 0.5) << endl;  
 return 0;  
}

// n个点带权费马点 // 平衡点||吊打XXX

//n个二维坐标点，带重物重量，找平衡点

//进行一次模拟退火，但是在局部最大值周围多次跳动（以提高精度

#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <ctime>  
const int N = 10005;  
int n, x[N], y[N], w[N];  
double ansx, ansy, dis;  
double Rand() { return (double)rand() / RAND\_MAX; }  
double calc(double xx, double yy) {  
 double res = 0;  
 for (int i = 1; i <= n; ++i) {  
 double dx = x[i] - xx, dy = y[i] - yy;  
 res += sqrt(dx \* dx + dy \* dy) \* w[i];  
 }  
 if (res < dis) dis = res, ansx = xx, ansy = yy;  
 return res;  
}  
void simulateAnneal() {  
 double t = 100000;  
 double nowx = ansx, nowy = ansy;  
 while (t > 0.001) {  
 double nxtx = nowx + t \* (Rand() \* 2 - 1);  
 double nxty = nowy + t \* (Rand() \* 2 - 1);  
 double delta = calc(nxtx, nxty) - calc(nowx, nowy);  
 if (exp(-delta / t) > Rand()) nowx = nxtx, nowy = nxty;  
 t \*= 0.97;  
 }  
 for (int i = 1; i <= 1000; ++i) {  
 double nxtx = ansx + t \* (Rand() \* 2 - 1);  
 double nxty = ansy + t \* (Rand() \* 2 - 1);  
 calc(nxtx, nxty);  
 }  
}  
int main() {  
 srand(time(0));  
 scanf("%d", &n);  
 for (int i = 1; i <= n; ++i) {  
 scanf("%d%d%d", &x[i], &y[i], &w[i]);  
 ansx += x[i], ansy += y[i];  
 }  
 ansx /= n, ansy /= n, dis = calc(ansx, ansy);  
 simulateAnneal();  
 printf("%.3lf %.3lf\n", ansx, ansy);  
 return 0;  
}

### 整体二分

ll bit[N];  
void add\_bit(ll k, ll a) {  
 while (k < N) {  
 bit[k] = bit[k] + a;  
 k += k & -k;  
 }  
}  
ll query\_bit(ll k) {  
 ll ans = 0;  
 while (k) {  
 ans = ans + bit[k];  
 k -= k & -k;  
 }  
 return ans;  
}  
struct node {  
 ll x, y, k, id, type;  
};  
node q[N], q1[N], q2[N];  
ll ans[N], now[N], tot, totx;  
void solve(ll l, ll r, ll ql, ll qr) {  
 if (ql > qr) return;  
 if (l == r) {  
 for (ll i = ql; i <= qr; i++) {  
 if (q[i].type == 2) {  
 ans[q[i].id] = l;  
 }  
 }  
 return;  
 }  
 ll mid = (l + r) >> 1;  
 ll cq1 = 0, cq2 = 0;  
 for (ll i = ql; i <= qr; i++) {  
 if (q[i].type == 1) {  
 if (q[i].y <= mid) {  
 add\_bit(q[i].x, q[i].k);  
 q1[++cq1] = q[i];  
 } else {  
 q2[++cq2] = q[i];  
 }  
 } else {  
 ll sum = query\_bit(q[i].y) - query\_bit(q[i].x - 1);  
 if (sum >= q[i].k) {  
 q1[++cq1] = q[i];  
 } else {  
 q2[++cq2] = q[i];  
 q2[cq2].k -= sum;  
 }  
 }  
 }  
 for (ll i = 1; i <= cq1; i++) if (q1[i].type == 1) add\_bit(q1[i].x, -q1[i].k);  
 for (ll i = 1; i <= cq1; i++) q[ql + i - 1] = q1[i];  
 for (ll i = 1; i <= cq2; i++) q[ql + cq1 + i - 1] = q2[i];  
 solve(l, mid, ql, ql + cq1 - 1);  
 solve(mid + 1, r, ql + cq1, qr);  
}  
void init() {  
 totx = 0;  
 tot = 0;  
 memset(bit, 0, sizeof bit);  
}

## 字符串

### 马拉车

#include <bits/stdc++.h>  
using namespace std;  
const int maxn = 100005;  
char s[maxn];  
char s\_new[maxn \* 2];  
int p[maxn \* 2];  
int Manacher(char\* a, int l) {  
 s\_new[0] = '$';  
 s\_new[1] = '#';  
 int len = 2;  
 for (int i = 0; i < l; i++) {  
 s\_new[len++] = a[i];  
 s\_new[len++] = '#';  
 }  
 s\_new[len] = '\0';  
 int id;  
 int mx = 0;  
 int mmax = 0;  
 for (int i = 1; i < len; i++) {  
 p[i] = i < mx ? min(p[2 \* id - i], mx - i) : 1;  
 while (s\_new[i + p[i]] == s\_new[i - p[i]]) p[i]++;  
 if (mx < i + p[i]) {  
 id = i;  
 mx = i + p[i];  
 }  
 mmax = max(mmax, p[i] - 1);  
 }  
 return mmax;  
}  
int main() {  
 cin >> s;  
 cout << Manacher(s, strlen(s));  
}

### AC自动机

#include <bits/stdc++.h>  
using namespace std;  
struct AC {  
 static const int maxnode = 200005;  
 static const int sigma\_size = 26;  
 char T[maxnode];  
 int ch[maxnode][sigma\_size];  
 int val[maxnode], fail[maxnode], last[maxnode];  
 int sz;  
 vector<pair<int, int> > ans;  
 void init() {  
 sz = 1;  
 memset(ch[0], 0, sizeof(ch[0]));  
 ans.clear();  
 }  
 int idx(const char &c) { return c - 'a'; }  
 void insert(string s, int v) {  
 int u = 0, n = s.length();  
 for (int i = 0; i < n; i++) {  
 int c = idx(s[i]);  
 if (!ch[u][c]) {  
 memset(ch[sz], 0, sizeof(ch[sz]));  
 val[sz] = 0;  
 ch[u][c] = sz++;  
 }  
 u = ch[u][c];  
 }  
 val[u] = v;  
 }  
 void get\_fail() {  
 queue<int> que;  
 fail[0] = 0;  
 for (int c = 0; c < sigma\_size; c++) {  
 int u = ch[0][c];  
 if (u) {  
 fail[u] = 0;  
 que.push(u);  
 last[u] = 0;  
 }  
 }  
 while (!que.empty()) {  
 int r = que.front();  
 que.pop();  
 for (int c = 0; c < sigma\_size; c++) {  
 int u = ch[r][c];  
 if (!u) continue;  
 que.push(u);  
 int v = fail[r];  
 while (v && !ch[v][c]) v = fail[v];  
 fail[u] = ch[v][c];  
 last[u] = val[fail[u]] ? fail[u] : last[fail[u]];  
 }  
 }  
 }  
 void print(int j) {  
 if (j) {  
 ans.push\_back(pair<int, int>(j, val[j]));  
 print(last[j]);  
 }  
 }  
 void find() {  
 int n = strlen(T);  
 int j = 0;  
 for (int i = 0; i < n; i++) {  
 int c = idx(T[i]);  
 while (j && !ch[j][c]) j = fail[j];  
 j = ch[j][c];  
 if (val[j])  
 print(j);  
 else if (last[j])  
 print(last[j]);  
 }  
 }  
} ac; //字符串下标从0开始

### KMP

//next数组等价于前缀函数  
#include<bits/stdc++.h>  
using namespace std;  
typedef long long ll;  
int kmp(char \*s1,int \*p1,char \*s2=0,int \*p2=0){//必须先求s1的next数组，即kmp(s1,p1);再kmp(s1,p1,s2,p2);  
 int n=strlen(s1);  
 if(p2==0){  
 p1[0]=0;  
 for(int i=1;s1[i]!='\0';i++){  
 int j=p1[i-1];  
 while(j>0&&s1[i]!=s1[j])j=p1[j-1];  
 if(s1[i]==s1[j])j++;  
 p1[i]=j;  
 }  
 }  
 else{  
 for(int i=0;s2[i]!='\0';i++){  
 int j=i==0?0:p2[i-1];  
 while(j>0&&s2[i]!=s1[j])j=p1[j-1];  
 if(s2[i]==s1[j])j++;  
 p2[i]=j;  
 if(j==n)return i-n+2;//返回位置  
 }  
 }  
 return 0;  
}  
int main(){  
 char s1[15],s2[105];  
 int p1[15],p2[105];  
 cin>>s1>>s2;  
 kmp(s1,p1);  
 cout<<kmp(s1,p1,s2,p2)<<endl;  
 return 0;  
}

### KMP 2

#include <bits/stdc++.h>  
using namespace std;  
struct KMP {  
 static const int MAXN = 1000010;  
 char T[MAXN], P[MAXN];  
 int fail[MAXN];  
 vector<int> ans;  
 void init() { ans.clear(); }  
 void get\_fail() {  
 int m = strlen(P);  
 fail[0] = fail[1] = 0;  
 for (int i = 1; i < m; i++) {  
 int j = fail[i];  
 while (j && P[i] != P[j]) j = fail[j];  
 fail[i + 1] = (P[i] == P[j] ? j + 1 : 0);  
 }  
 }  
 void find() {  
 int n = strlen(T), m = strlen(P);  
 get\_fail();  
 int j = 0;  
 for (int i = 0; i < n; i++) {  
 while (j && P[j] != T[i]) j = fail[j];  
 if (P[j] == T[i]) j++;  
 if (j == m) ans.push\_back(i - m + 1);  
 }  
 }  
} kmp; //P为模式串，下标从0开始，输入后直接调用find()

### Tire

#include <bits/stdc++.h>  
using namespace std;  
struct Trie {  
 static const int maxnode = 200005;  
 static const int sigma\_size = 26;  
 int ch[maxnode][sigma\_size];  
 int val[maxnode];  
 int sz;  
 Trie() {  
 sz = 1;  
 memset(ch[0], 0, sizeof(ch[0]));  
 }  
 int idx(const char &c) { return c - 'a'; }  
 void insert(string s, int v) {  
 int u = 0, n = s.length();  
 for (int i = 0; i < n; i++) {  
 int c = idx(s[i]);  
 if (!ch[u][c]) {  
 memset(ch[sz], 0, sizeof(ch[sz]));  
 val[sz] = 0;  
 ch[u][c] = sz++;  
 }  
 u = ch[u][c];  
 }  
 val[u] = v;  
 }  
 int find(string s) {  
 int u = 0, n = s.length();  
 for (int i = 0; i < n; i++) {  
 int c = idx(s[i]);  
 if (!ch[u][c]) return 0;  
 u = ch[u][c];  
 }  
 return val[u];  
 }  
} trie;

### 后缀数组

#include <bits/stdc++.h>  
using namespace std;  
struct SuffixArray {  
 static const int MAXN = 1100000;  
 char s[MAXN];  
 int sa[MAXN], t[MAXN], t1[MAXN], c[MAXN], ra[MAXN], height[MAXN], m;  
 inline void init() { memset(this, 0, sizeof(SuffixArray)); }  
 inline void get\_sa(int n) {  
 m = 256;  
 int \*x = t, \*y = t1;  
 for (int i = 1; i <= m; i++) c[i] = 0;  
 for (int i = 1; i <= n; i++) c[x[i] = s[i]]++;  
 for (int i = 1; i <= m; i++) c[i] += c[i - 1];  
 for (int i = n; i >= 1; i--) sa[c[x[i]]--] = i;  
 for (int k = 1; k <= n; k <<= 1) {  
 int p = 0;  
 for (int i = n - k + 1; i <= n; i++) y[++p] = i;  
 for (int i = 1; i <= n; i++)  
 if (sa[i] > k) y[++p] = sa[i] - k;  
 for (int i = 1; i <= m; i++) c[i] = 0;  
 for (int i = 1; i <= n; i++) c[x[y[i]]]++;  
 for (int i = 1; i <= m; i++) c[i] += c[i - 1];  
 for (int i = n; i >= 1; i--) sa[c[x[y[i]]]--] = y[i];  
 std::swap(x, y);  
 p = x[sa[1]] = 1;  
 for (int i = 2; i <= n; i++) {  
 x[sa[i]] = (y[sa[i - 1]] == y[sa[i]] &&  
 y[sa[i - 1] + k] == y[sa[i] + k])  
 ? p  
 : ++p;  
 }  
 if (p >= n) break;  
 m = p;  
 }  
 }  
 inline void get\_height(int n) {  
 int i, j, k = 0;  
 for (int i = 1; i <= n; i++) ra[sa[i]] = i;  
 for (int i = 1; i <= n; i++) {  
 if (k) k--;  
 int j = sa[ra[i] - 1];  
 while (s[i + k] == s[j + k]) k++;  
 height[ra[i]] = k;  
 }  
 }  
} SA; //字符串下标从一开始

### 可持久化字典树

struct Trie01 {  
 static const int maxnode = 2000005;  
 static const int sigma\_size = 2;  
 int ch[maxnode << 5][sigma\_size], val[maxnode << 5];  
 int rt[maxnode];  
 int sz;  
 Trie01() {  
 sz = 0;  
 memset(ch[0], 0, sizeof(ch[0]));  
 }  
 void insert(int &now, int pre, int v) {  
 now = ++sz;  
 for (int i = 30; i >= 0; i--) {  
 int k = ((v >> i) & 1);  
 ch[now][k] = ++sz;  
 ch[now][k ^ 1] = ch[pre][k ^ 1];  
 val[ch[now][k]] = val[ch[pre][k]] + 1;  
 now = ch[now][k];  
 pre = ch[pre][k];  
 }  
 }  
} trie;

## 对拍

### windows环境下bat对拍

@echo off  
:loop  
 dataa.exe > data.txt  
 biaocheng.exe < data.txt > ac.txt  
 A.exe < data.txt > test.txt  
 fc ac.txt test.txt  
 if not errorlevel 1 goto loop  
pause  
goto loop

**其中要改的部分（标红辽）**：

@echo off  
:loop  
 dataa.exe > data.txt  
 $\color{red}{biaocheng.exe}$ < data.txt > ac.txt  
 $\color{red}{A.exe}$ < data.txt > test.txt  
 fc ac.txt test.txt  
 if not errorlevel 1 goto loop  
pause  
goto loop

文件以.bat作为后缀

将三个程序（数据生成文件（dataa），标程或暴力代码（biaocheng）, 要看的代码（A））放在同一目录下，

记得加 freopen

随机数记得加srand((int)time(0));

随机数生成code

#include <iostream>  
#include <cstdlib>  
#include <ctime>  
using namespace std;  
int main(){  
 freopen("data.txt", "w", stdout);  
   
 srand((int)time(0));  
 int T = rand() % 100000;  
 cout << T << endl;  
   
 for (int i = 0; i < T; i++){  
 cout << rand() % 100;  
 }  
}

rand() 似乎只有三万多，需要更大的数的话要乘一下