

板子汇总

板子汇总

注意

定义

STL

优先队列重载

set重载

动态开数组

set

map

unordered_map

bitset

构造

函数

H

计算几何

几何的一些定理

zyx的计算几何

计算几何全家桶

自适应辛普森

数据结构

仙人掌

CDQ

kruskal重构树

普通莫队

带修莫队

回滚莫队

线段树合并分裂

主席树

LCT

Splay1

splay2

Treap

舞蹈链（多重覆盖）

舞蹈链（精确覆盖）

数论

lucas求组合数

扩展欧几里得求逆元

逆元线性递推 inv阶乘组合数

数学

一些范围

勾股数/圆上格点数

exgcd

Pollard_Rho+Miller-Rabin

FFT

BSGS

扩展BSGS

二次剩余

卡特兰数

快速幂

龟速乘快速幂（快速幂爆longlong

莫比乌斯反演

博弈

- 高精度GCD
- 高精度乘法 (FFT)
- 高精度乘法 (乘单精度)
- 高精度乘法 (朴素)
- 高精度除法 (除单精度)
- 高精度除法 (除高精度)
- 高精度加法
- 高精度减法
- 高精度阶乘
- 高精度进制转换
- 高精度幂
- 高精度平方根
- 高精度取模 (对单精度)
- 欧拉筛
- 组合数 (逆元线性递推)
- 中国剩余定理

图论

- 有源汇上下界最大小流
- 树链剖分
- 虚树
- spfa最短路及负环
- 二分图匹配 (匈牙利)
- 强连通 (kosaraju)
- 强连通 (tarjan)
- 强连通 (tarjan无vector)
- 最大流
- 最大流 (double)
- 最小费用最大流
- 树分治
- 拓扑排序
- 最近公共祖先 (倍增)
- 最近公共祖先 (线段树)

线性代数

- 高斯消元
- 矩阵行列式
- 线性基
- 线性基2
- 矩阵 (加减乘快速幂)
- 稀疏矩阵乘法

杂项

- mt19937
- 洗牌算法
- 快读
- fread快读
- 朝鲜大哥快读
- 模拟退火
- 整体二分

字符串

- 马拉车
- AC自动机
- KMP
- KMP 2
- Tire
- 后缀数组
- 可持久化字典树

对拍

- windows环境下bat对拍

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

$\gcd(a,b)=1$ 最大公约数 即a, b互质

| 整除 $a|b$ $b\%a==0$

STL

优先队列重载

```
1 priority_queue<int, vector<int>, cmp>s;  
2  
3 struct cmp{  
4     bool operator()(const int &a,const int &b){  
5         return a>b;  
6     }  
7 };
```

set重载

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  #define l first
5  #define r second
6
7  struct cmp{
8      bool operator() (const pair<int, int> &a, const pair<int, int> &b)
9      const{
10         int lena = a.r - a.l + 1;
11         int lenb = b.r - b.l + 1;
12         if(lena == lenb) return a.l < b.l;
13         return lena > lenb;
14     }
15 };
16
17 int main(){
18     ios :: sync_with_stdio(0); cin.tie(0); cout.tie(0);
19     int T;
20     cin >> T;
21     while(T -- ){
22         int n;
23         cin >> n;
24         set<pair<int, int>, cmp> segs;
25         segs.insert({0, n - 1});
26         vector<int> a(n);
27         for(int i = 1; i <= n; ++ i){
28             pair<int, int> cur = *segs.begin();
29             segs.erase(segs.begin());
30             int id = (cur.l + cur.r) / 2;
31             a[id] = i;
32             if(cur.l < id) segs.insert({cur.l, id - 1});
33             if(id < cur.r) segs.insert({id + 1, cur.r});
34         }
35         for(auto it : a) cout << it << " ";
36         cout << endl;
37     }
38 }
```

动态开数组

```
1  int a[15], n, m;
2  cin >> n >> m;
3  int (*b)[m] = (int (*)[m])a;
```

new / delete

```
1  #define M 100
2
3  #define N 20
```

```

4
5 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
6
7  //定义的时候
8
9  int** pNum;//以int为例
10
11  pNum = new int*[M];
12
13  for(int i = 0;i < M;i ++){
14      pNum[i]=new int[N];
15  }
16
17
18  //删除的时候是
19
20  for(int j = 0;j < M;j ++){
21      delete []pNum[i];
22  }
23
24  delete []pNum;

```

malloc / free

```

1  #include<stdio.h>
2  #include<stdlib.h>
3
4  int main() {
5      int **a;  //用二级指针动态申请二维数组
6      int i,j;
7      int m,n;
8      printf("请输入行数\n");
9      scanf("%d",&m);
10     printf("请输入列数\n");
11     scanf("%d",&n);
12     a=(int**)malloc(sizeof(int*)*m);
13     for(i=0;i<m;i++)
14         a[i]=(int*)malloc(sizeof(int)*n);
15     for(i=0;i<m;i++) {
16         for(j=0;j<n;j++) {
17             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
18             //时连续的，行与行之间的地址不连续
19         }
20     }
21     free(a[i]);
22
23     free(a);
24     return 0;
25 }

```

```

1  #include<stdio.h>
2  #include<stdlib.h>
3

```

```

4  int main()
5  {
6      int i,j;
7      //申请一个3行2列的整型数组
8      int (*a)[2]=(int(*)[2])malloc(sizeof(int)*3*2);
9      for(i=0;i<3;i++) {
10         for(j=0;j<2;j++) {
11             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
12             的
13         }
14     }
15     free(a);
16     return 0;
17 }

```

vector

```

1  //二维vector
2  vector<vector<int>> ivec(m ,vector<int>(n)); //m*n的二维vector
3
4  //动态创建m*n的二维vector
5  //方法一：
6  vector<vector<int>> ivec;
7  ivec.resize(m);
8  for(int i=0;i<m;i++)
9      ivec[i].resize(n);
10
11 //方法二：
12 vector<vector<int>> ivec;
13 ivec.resize(m,vector<int>(n));
14

```

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键值的元素和 第一个大于给定键值的元素，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

```
1 map<int, string> ID_Name;
2
3 // 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
4 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:

```
1 // 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
2 pair<iterator,bool> insert (const value_type& val);
3
4 //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
5 iterator insert (const_iterator position, const value_type& val);
6
7 // 插入多个
8 void insert (InputIterator first, InputIterator last);
9
10 //c++11开始支持, 使用列表插入多个
11 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。

```
1 map<int, string> ID_Name;
2
3 //ID_Name中没有关键字2016, 使用[]取值会导致插入
4 //因此, 下面语句不会报错, 但打印结果为空
5 cout<<ID_Name[2016].c_str()<<endl;
6
7 //使用at会进行关键字检查, 因此下面语句会报错
8 ID_Name.at(2016) = "Bob";
```

容量查询

```

1 // 查询map是否为空
2 bool empty();
3
4 // 查询map中键值对的数量
5 size_t size();
6
7 // 查询map所能包含的最大键值对数量，和系统和应用库有关。
8 // 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
9 size_t max_size();
10
11 // 查询关键字为key的元素的个数，在map里结果非0即1
12 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：

```

1 map<int,int>::iterator it;
2 map<int,int> mmap;
3 const map<int,int> const_mmap;
4
5 it = mmap.begin(); //iterator
6 mmap.cbegin(); //const_iterator
7
8 const_mmap.begin(); //const_iterator
9 const_mmap.cbegin(); //const_iterator 123456789

```

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

```

1 // 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
2 iterator erase( iterator pos )
3
4 // 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
5 iterator erase( const_iterator first, const_iterator last );
6
7 // 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
8 size_t erase( const key_type& key );
9
10 // 清空map，清空后的size为0
11 void clear();

```

交换

```

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

```

顺序比较

```

1 // 比较两个关键字在map中位置的先后
2 key_compare key_comp() const;

```


查找

```
1 // 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
2 // 根据map的类型，返回的迭代器为 iterator 或者 const_iterator
3 iterator find (const key_type& k);
4 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)!=0

插入数据

```
1 mp.insert(Map::value_type(1, "Raoul"));
```

遍历map

```
1 unordered_map<key, T>::iterator it;
2 (*it).first; //the key value
3 (*it).second //the mapped value
4 for(unordered_map<key, T>::iterator iter=mp.begin(); iter!=mp.end(); iter++)
5     cout<<"key value is"<<iter->first<<" the mapped value is "<< iter-
6     >second;
7 //也可以这样
8 for(auto& v : mp)
9     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 1 bit 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

```
1    bitset<4> bitset1;    //无参构造，长度为4，默认每一位为0
2
3    bitset<8> bitset2(12);    //长度为8，二进制保存，前面用0补充
4
5    string s = "100101";
6    bitset<10> bitset3(s);    //长度为10，前面用0补充
7
8    char s2[] = "10101";
9    bitset<13> bitset4(s2);    //长度为13，前面用0补充
10
11    cout << bitset1 << endl;    //0000
12    cout << bitset2 << endl;    //00001100
13    cout << bitset3 << endl;    //0000100101
14    cout << bitset4 << endl;    //0000000010101
```

函数

```
1    bitset<8> foo ("10011011");
2
3    cout << foo.count() << endl;    //5    (count函数用来求bitset中1的位数，foo
    中共有5个1
4    cout << foo.size() << endl;    //8    (size函数用来求bitset的大小，一共有8
    位
5
6    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
7    cout << foo.test(2) << endl;    //false    (同理，foo[2]为0，返回false
8
9    cout << foo.any() << endl;    //true    (any函数检查bitset中是否有1
10    cout << foo.none() << endl;    //false    (none函数检查bitset中是否没有1
11    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](#)

H

```
1    #include <bits/stdc++.h>
2    #define ll long long
3    using namespace std;
4    int t,n,m;
5    char str[1010];
6    bitset<500> number[30];
7    int main() {
8        ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
9        //freopen("test.in","r",stdin);
10       //freopen("test.out","w",stdout);
11       scanf("%d",&t);
12       while(t--)
13       {
14           scanf("%d %d",&n,&m);
15           for(int i=0;i<m;i++)
16           {
```

```

17         scanf("%s",str);
18         number[i]=bitset<500>(str);
19     }
20     int len=1<<m,ans=m+1;
21     for(int i=1;i<len;i++)
22     {
23         int t=i,s=0;
24         bitset<500> num(0);
25         for(int j=0;j<m&& t>0;j++)
26         {
27             if(t&1)
28             {
29                 num=num|number[j];
30                 s++;
31             }
32             t>>=1;
33         }
34         if(num.count()==n) ans=min(ans,s);
35     }
36     if(ans==m+1) printf("-1\n");
37     else printf("%d\n",ans);
38 }
39 return 0;
40 }
41

```

计算几何

几何的一些定理

多面体欧拉定理

多面体欧拉定理是指对于简单多面体，其各维对象数总满足一定的数学关系，在三维空间中多面体欧拉定理可表示为：

“顶点数-棱长数+表面数=2”。

简单多面体即表面经过连续变形可以变为球面的多面体。

zyx的计算几何

```

1  const double eps = 1e-9;
2  const double PI = acos(-1.0);
3
4  struct Line;
5
6  struct Point {
7      double x, y;
8
9      Point() { x = y = 0; }
10
11     Point(const Line &a);
12
13     Point(const double &a, const double &b) : x(a), y(b) {}
14
15     Point operator+(const Point &a) const {

```

```

16         return {x + a.x, y + a.y};
17     }
18
19     Point operator-(const Point &a) const {
20         return {x - a.x, y - a.y};
21     }
22
23     Point operator*(const double &a) const {
24         return {x * a, y * a};
25     }
26
27     Point operator/(const double &d) const {
28         return {x / d, y / d};
29     }
30
31     bool operator==(const Point &a) const {
32         return abs(x - a.x) + abs(y - a.y) < eps;
33     }
34
35     void standardize() {
36         *this = *this / sqrt(x * x + y * y);
37     }
38 };
39
40 Point normal(const Point &a) { return Point(-a.y, a.x); }
41
42 double dist(const Point &a, const Point &b) {
43     return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
44 }
45
46 double dist2(const Point &a, const Point &b) {
47     return (a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y);
48 }
49
50 struct Line {
51     Point s, t;
52
53     Line() {}
54
55     Line(const Point &a, const Point &b) : s(a), t(b) {}
56
57 };
58
59 struct circle {
60     Point o;
61     double r;
62
63     circle() {}
64
65     circle(Point P, double R = 0) { o = P, r = R; }
66 };
67
68 double length(const Point &p) {
69     return sqrt(p.x * p.x + p.y * p.y);
70 }
71
72 double length(const Line &l) {
73     Point p(l);

```

```

74     return length(p);
75 }
76
77 Point::Point(const Line &a) { *this = a.t - a.s; }
78
79 istream &operator>>(istream &in, Point &a) {
80     in >> a.x >> a.y;
81     return in;
82 }
83
84 double dot(const Point &a, const Point &b) {
85     return a.x * b.x + a.y * b.y;
86 }
87
88 double det(const Point &a, const Point &b) {
89     return a.x * b.y - a.y * b.x;
90 }
91
92 int sgn(const double &x) { return fabs(x) < eps ? 0 : (x > 0 ? 1 : -1); }
93
94 double sqr(const double &x) { return x * x; }
95
96 Point rotate(const Point &a, const double &ang) {
97     double x = cos(ang) * a.x - sin(ang) * a.y;
98     double y = sin(ang) * a.x + cos(ang) * a.y;
99     return {x, y};
100 }
101
102 //点在线段上 <=0 包含端点
103 bool sp_on(const Line &seg, const Point &p) {
104     Point a = seg.s, b = seg.t;
105     return !sgn(det(p - a, b - a)) && sgn(dot(p - a, p - b)) <= 0;
106 }
107
108 bool lp_on(const Line &line, const Point &p) {
109     Point a = line.s, b = line.t;
110     return !sgn(det(p - a, b - a));
111 }
112
113 //等于不包含共线
114 int andrew(Point *point, Point *convex, int n) {
115     sort(point, point + n, [](Point a, Point b) {
116         if (a.x != b.x) return a.x < b.x;
117         return a.y < b.y;
118     });
119     int top = 0;
120     for (int i = 0; i < n; i++) {
121         while ((top > 1) && det(convex[top - 1] - convex[top - 2], point[i]
- convex[top - 1]) <= 0)
122             top--;
123         convex[top++] = point[i];
124     }
125     int tmp = top;
126     for (int i = n - 2; i >= 0; i--) {
127         while ((top > tmp) && det(convex[top - 1] - convex[top - 2],
point[i] - convex[top - 1]) <= 0)
128             top--;
129         convex[top++] = point[i];

```

```

130     }
131     if (n > 1) top--;
132     return top;
133 }
134
135 double slope(const Point &a, const Point &b) {
136     return (a.y - b.y) / (a.x - b.x);
137 }
138
139 double slope(const Line &a) {
140     return slope(a.s, a.t);
141 }
142
143 Point ll_intersection(const Line &a, const Line &b) {
144     double s1 = det(Point(a), b.s - a.s), s2 = det(Point(a), b.t - a.s);
145     return (b.s * s2 - b.t * s1) / (s2 - s1);
146 }
147
148 int ss_cross(const Line &a, const Line &b, Point &p) {
149     int d1 = sgn(det(a.t - a.s, b.s - a.s));
150     int d2 = sgn(det(a.t - a.s, b.t - a.s));
151     int d3 = sgn(det(b.t - b.s, a.s - b.s));
152     int d4 = sgn(det(b.t - b.s, a.t - b.s));
153     if ((d1 ^ d2) == -2 && (d3 ^ d4) == -2) {
154         p = ll_intersection(a, b);
155         return 1;
156     }
157     if (!d1 && sp_on(a, b.s)) {
158         p = b.s;
159         return 2;
160     }
161     if (!d2 && sp_on(a, b.t)) {
162         p = b.t;
163         return 2;
164     }
165     if (!d3 && sp_on(b, a.s)) {
166         p = a.s;
167         return 2;
168     }
169     if (!d4 && sp_on(b, a.t)) {
170         p = a.t;
171         return 2;
172     }
173     return 0;
174 }
175
176 Point project(const Line &l, const Point &p) {
177     Point base(l);
178     double r = dot(base, p - l.s) / sqr(length(base));
179     return l.s + (base * r);
180 }
181
182 double sp_dist(const Line &l, const Point &p) {
183     if (l.s == l.t) return dist(l.s, p);
184     Point x = p - l.s, y = p - l.t, z = l.t - l.s;
185     if (sgn(dot(x, z)) < 0) return length(x); //P距离A更近
186     if (sgn(dot(y, z)) > 0) return length(y); //P距离B更近
187     return abs(det(x, z) / length(z)); //面积除以底边长

```

```

188 }
189
190 double lp_dist(const Line &l, const Point &p) {
191     Point x = p - l.s, y = p - l.t, z = l.t - l.s;
192     return abs(det(x, z) / length(z)); //面积除以底边长
193 }
194
195 int lc_cross(const Line &l, const Point &a, const double &r, pair<Point,
Point> &ans) {
196     int num = 0;
197     Point pr = project(l, a);
198     double dis = dist(pr, a);
199     double tmp = r * r - dis * dis;
200     if (sgn(tmp) == 1) num = 2;
201     else if (sgn(tmp) == 0) num = 1;
202     else return 0;
203     double base = sqrt(r * r - dis * dis);
204     Point e(l);
205     e.standardize();
206     e = e * base;
207     ans = make_pair(pr + e, pr - e);
208     return num;
209 }
210
211 int cc_cross(const Point &c1, const double &r1, const Point &c2, const
double &r2, pair<Point, Point> &ans) {
212     double x1 = c1.x, x2 = c2.x, y1 = c1.y, y2 = c2.y;
213     double d = length(c1 - c2);
214     if (sgn(fabs(r1 - r2) - d) > 0) return -1; //内含
215     if (sgn(r1 + r2 - d) < 0) return 0; //相离
216     double a = r1 * (x1 - x2) * 2, b = r1 * (y1 - y2) * 2, c = r2 * r2 - r1
* r1 - d * d;
217     double p = a * a + b * b, q = -a * c * 2, r = c * c - b * b;
218
219     double cosa, sina, cosb, sinb;
220     //One Intersection
221     if (sgn(d - (r1 + r2)) == 0 || sgn(d - fabs(r1 - r2)) == 0) {
222         cosa = -q / p / 2;
223         sina = sqrt(1 - sqr(cosa));
224         Point p0(x1 + r1 * cosa, y1 + r1 * sina);
225         if (sgn(dist(p0, c2) - r2)) p0.y = y1 - r1 * sina;
226         ans = pair<Point, Point>(p0, p0);
227         return 1;
228     }
229     //Two Intersections
230     double delta = sqrt(q * q - p * r * 4);
231     cosa = (delta - q) / p / 2;
232     cosb = (-delta - q) / p / 2;
233     sina = sqrt(1 - sqr(cosa));
234     sinb = sqrt(1 - sqr(cosb));
235     Point p1(x1 + r1 * cosa, y1 + r1 * sina);
236     Point p2(x1 + r1 * cosb, y1 + r1 * sinb);
237     if (sgn(dist(p1, c2) - r2)) p1.y = y1 - r1 * sina;
238     if (sgn(dist(p2, c2) - r2)) p2.y = y1 - r1 * sinb;
239     if (p1 == p2) p1.y = y1 - r1 * sina;
240     ans = pair<Point, Point>(p1, p2);
241     return 2;
242 }

```

```

243
244 Point lp_sym(const Line &l, const Point &p) {
245     return p + (project(l, p) - p) * 2;
246 }
247
248 double alpha(const Point &t1, const Point &t2) {
249     double theta;
250     theta = atan2((double) t2.y, (double) t2.x) - atan2((double) t1.y,
251 (double) t1.x);
252     if (sgn(theta) < 0)
253         theta += 2.0 * PI;
254     return theta;
255 }
256
257 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
258     int cnt = 0;
259     int tmp;
260     for (int i = 1; i <= n; ++i) {
261         int j = i < n ? i + 1 : 1;
262         if (sp_on(Line(P[i], P[j]), a)) return 2; // 点在多边形上
263         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
264             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
265     }
266     return cnt & 1; // 穿过奇数次则在多边形以内
267 }
268
269 bool pip_convex_jud(const Point &a, const Point &L, const Point &R) { // 判断
    AL是否在AR右边
270     return sgn(det(L - a, R - a)) > 0; // 必须严格以内
271 }
272
273 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
274     // 点按逆时针给出
275     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
276     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
277     int l = 1, r = n - 2;
278     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
279         int mid = (l + r + 1) >> 1;
280         if (pip_convex_jud(P[0], P[mid], a)) l = mid;
281         else r = mid - 1;
282     }
283     if (pip_convex_jud(P[l], a, P[l + 1])) return 0; // 在P[pos_(pos+1)]外
284     if (sp_on(Line(P[l], P[l + 1]), a)) return 2; // 在P[pos_(pos+1)]上
285     return 1;
286 }
287 // 多边形是否包含线段
288 // 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，
289 // 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，
290 // 则该线段一定在多边形内。

```



```

291 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】
292     for (int i1 = 1; i1 <= n; ++i1) {
293         int j1 = i1 < n ? i1 + 1 : 1;
294         for (int i2 = 1; i2 <= m; ++i2) {
295             int j2 = i2 < m ? i2 + 1 : 1;
296             Point tmp;
297             if (ss_cross(Line(A[i1], A[j1]), Line(B[i2], B[j2]), tmp))
return 0; // 两线段相交
298             if (pip(B, m, A[i1]) || pip(A, n, B[i2])) return 0; // 点包含在内
299         }
300     }
301     return 1;
302 }
303
304 double area(Point *P, int n) { // 【任意多边形P的面积】
305     double S = 0;
306     for (int i = 1; i <= n; i++) S += det(P[i], P[i < n ? i + 1 : 1]);
307     return S / 2.0;
308 }
309
310 Line Q[N];
311
312 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
313 int halfcut(Line *L, int n, Point *P) { // 【半平面交】
314     sort(L, L + n, [](const Line &a, const Line &b) {
315         double d = atan2((a.t - a.s).y, (a.t - a.s).x) - atan2((b.t -
b.s).y, (b.t - b.s).x);
316         return sgn(d) ? sgn(d) < 0 : judge(a, b.s);
317     });
318
319     int m = n;
320     n = 0;
321     for (int i = 0; i < m; ++i)
322         if (i == 0 || sgn(atan2(Point(L[i]).y, Point(L[i]).x) -
atan2(Point(L[i - 1]).y, Point(L[i - 1]).x)))
323             L[n++] = L[i];
324     int h = 1, t = 0;
325     for (int i = 0; i < n; ++i) {
326         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
327         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
328         Q[++t] = L[i];
329     }
330
331     while (h < t && judge(Q[h], ll_intersection(Q[t], Q[t - 1]))) --t;
332     while (h < t && judge(Q[t], ll_intersection(Q[h], Q[h + 1]))) ++h;
333     n = 0;
334     for (int i = h; i <= t; ++i) {
335         P[n++] = ll_intersection(Q[i], Q[i < t ? i + 1 : h]);
336     }
337     return n;
338 }
339
340 Point v1[N], v2[N];
341

```

```

342 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
343     for (int i = 0; i < n; ++i) V1[i] = P1[(i + 1) % n] - P1[i];
344     for (int i = 0; i < m; ++i) V2[i] = P2[(i + 1) % m] - P2[i];
345     int t = 0, i = 0, j = 0;
346     V[t++] = P1[0] + P2[0];
347     while (i < n && j < m) V[t] = V[t - 1] + (sgn(det(V1[i], V2[j])) > 0 ?
V1[i++] : V2[j++]), t++;
348     while (i < n) V[t] = V[t - 1] + V1[i++], t++;
349     while (j < m) V[t] = V[t - 1] + V2[j++], t++;
350     return t;
351 }
352
353 circle getcircle(const Point &A, const Point &B, const Point &C) { // 【三点确
    定一圆】向量垂心法
354     Point P1 = (A + B) * 0.5, P2 = (A + C) * 0.5;
355     Line R1 = Line(P1, P1 + normal(B - A));
356     Line R2 = Line(P2, P2 + normal(C - A));
357     circle o;
358     o.o = ll_intersection(R1, R2);
359     o.r = length(A - o.o);
360     return o;
361 }
362
363 struct ConvexHull {
364
365     int op;
366
367     struct cmp {
368         bool operator()(const Point &a, const Point &b) const {
369             return sgn(a.x - b.x) < 0 || sgn(a.x - b.x) == 0 && sgn(a.y -
b.y) < 0;
370         }
371     };
372
373     set<Point, cmp> s;
374
375     ConvexHull(int o) {
376         op = o;
377         s.clear();
378     }
379
380     inline int PIP(Point P) {
381         set<Point>::iterator it = s.lower_bound(Point(P.x, -dinf)); // 找到第
    一个横坐标大于P的点
382         if (it == s.end()) return 0;
383         if (sgn(it->x - P.x) == 0) return sgn((P.y - it->y) * op) <= 0; // 比
    较纵坐标大小
384         if (it == s.begin()) return 0;
385         set<Point>::iterator j = it, k = it;
386         --j;
387         return sgn(det(P - *j, *k - *j) * op) >= 0; // 看叉姬1
388     }
389
390     inline int judge(set<Point>::iterator it) {
391         set<Point>::iterator j = it, k = it;
392         if (j == s.begin()) return 0;
393         --j;

```

```

394         if (++k == s.end()) return 0;
395         return sgn(det(*it - *j, *k - *j) * op) >= 0; //看叉姬
396     }
397
398     inline void insert(Point P) {
399         if (PIP(P)) return; //如果点P已经在凸壳上或凸包里就不插入了
400         set<Point>::iterator tmp = s.lower_bound(Point(P.x, -inf));
401         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
402         s.insert(P);
403         set<Point>::iterator it = s.find(P), p = it;
404         if (p != s.begin()) {
405             --p;
406             while (judge(p)) {
407                 set<Point>::iterator temp = p--;
408                 s.erase(temp);
409             }
410         }
411         if ((p = ++it) != s.end()) {
412             while (judge(p)) {
413                 set<Point>::iterator temp = p++;
414                 s.erase(temp);
415             }
416         }
417     }
418 } up(1), down(-1);
419
420 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
421 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
422 circle min_circle(Point *P, int n) { //【求点集P的最小覆盖圆】 o(n)
423     // random_shuffle(P, P+n);
424     Random(P, n);
425     circle C = circle(P[0], 0);
426     for (int i = 1; i < n; ++i)
427         if (!PIC(C, P[i])) {
428             C = circle(P[i], 0);
429             for (int j = 0; j < i; ++j)
430                 if (!PIC(C, P[j])) {
431                     C.o = (P[i] + P[j]) * 0.5, C.r = length(P[j] - C.o);
432                     for (int k = 0; k < j; ++k) if (!PIC(C, P[k])) C =
                        getcircle(P[i], P[j], P[k]);
433                 }
434             }
435     return C;
436 }
437

```

计算几何全家桶

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4  typedef long long ll;
5  const ll N = 1 << 20;

```

```

6  const ll mod = 1e9 + 7;
7  const double dinf = 1e99;
8  const int inf = 0x3f3f3f3f;
9  const ll linf = 0x3f3f3f3f3f3f3f3f;
10
11  const double eps = 1e-9;
12  const double PI = acos(-1.0);
13
14  struct Line;
15
16  struct Point {
17      double x, y;
18
19      Point() { x = y = 0; }
20
21      Point(const Line &a);
22
23      Point(const double &a, const double &b) : x(a), y(b) {}
24
25      Point operator+(const Point &a) const {
26          return {x + a.x, y + a.y};
27      }
28
29      Point operator-(const Point &a) const {
30          return {x - a.x, y - a.y};
31      }
32
33      Point operator*(const double &a) const {
34          return {x * a, y * a};
35      }
36
37      Point operator/(const double &d) const {
38          return {x / d, y / d};
39      }
40
41      bool operator==(const Point &a) const {
42          return abs(x - a.x) + abs(y - a.y) < eps;
43      }
44
45      void standardize() {
46          *this = *this / sqrt(x * x + y * y);
47      }
48  };
49
50  Point normal(const Point &a) { return Point(-a.y, a.x); }
51
52  double dist(const Point &a, const Point &b) {
53      return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
54  }
55
56  double dist2(const Point &a, const Point &b) {
57      return (a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y);
58  }
59
60  struct Line {
61      Point s, t;
62
63      Line() {}

```

```

64
65     Line(const Point &a, const Point &b) : s(a), t(b) {}
66
67 };
68
69 struct circle {
70     Point o;
71     double r;
72
73     circle() {}
74
75     circle(Point P, double R = 0) { o = P, r = R; }
76 };
77
78 double length(const Point &p) {
79     return sqrt(p.x * p.x + p.y * p.y);
80 }
81
82 double length(const Line &l) {
83     Point p(l);
84     return length(p);
85 }
86
87 Point::Point(const Line &a) { *this = a.t - a.s; }
88
89 istream &operator>>(istream &in, Point &a) {
90     in >> a.x >> a.y;
91     return in;
92 }
93
94 double dot(const Point &a, const Point &b) {
95     return a.x * b.x + a.y * b.y;
96 }
97
98 double det(const Point &a, const Point &b) {
99     return a.x * b.y - a.y * b.x;
100 }
101
102 int sgn(const double &x) { return fabs(x) < eps ? 0 : (x > 0 ? 1 : -1); }
103
104 double sqr(const double &x) { return x * x; }
105
106 Point rotate(const Point &a, const double &ang) {
107     double x = cos(ang) * a.x - sin(ang) * a.y;
108     double y = sin(ang) * a.x + cos(ang) * a.y;
109     return {x, y};
110 }
111
112 //点在线段上 <=0 包含端点
113 bool sp_on(const Line &seg, const Point &p) {
114     Point a = seg.s, b = seg.t;
115     return !sgn(det(p - a, b - a)) && sgn(dot(p - a, p - b)) <= 0;
116 }
117
118 bool lp_on(const Line &line, const Point &p) {
119     Point a = line.s, b = line.t;
120     return !sgn(det(p - a, b - a));
121 }

```

```

122
123 //等于不包含共线
124 int andrew(Point *point, Point *convex, int n) {
125     sort(point, point + n, [](Point a, Point b) {
126         if (a.x != b.x) return a.x < b.x;
127         return a.y < b.y;
128     });
129     int top = 0;
130     for (int i = 0; i < n; i++) {
131         while ((top > 1) && det(convex[top - 1] - convex[top - 2], point[i]
- convex[top - 1]) <= 0)
132             top--;
133         convex[top++] = point[i];
134     }
135     int tmp = top;
136     for (int i = n - 2; i >= 0; i--) {
137         while ((top > tmp) && det(convex[top - 1] - convex[top - 2],
point[i] - convex[top - 1]) <= 0)
138             top--;
139         convex[top++] = point[i];
140     }
141     if (n > 1) top--;
142     return top;
143 }
144
145 double slope(const Point &a, const Point &b) {
146     return (a.y - b.y) / (a.x - b.x);
147 }
148
149 double slope(const Line &a) {
150     return slope(a.s, a.t);
151 }
152
153 Point ll_intersection(const Line &a, const Line &b) {
154     double s1 = det(Point(a), b.s - a.s), s2 = det(Point(a), b.t - a.s);
155     return (b.s * s2 - b.t * s1) / (s2 - s1);
156 }
157
158 int ss_cross(const Line &a, const Line &b, Point &p) {
159     int d1 = sgn(det(a.t - a.s, b.s - a.s));
160     int d2 = sgn(det(a.t - a.s, b.t - a.s));
161     int d3 = sgn(det(b.t - b.s, a.s - b.s));
162     int d4 = sgn(det(b.t - b.s, a.t - b.s));
163     if ((d1 ^ d2) == -2 && (d3 ^ d4) == -2) {
164         p = ll_intersection(a, b);
165         return 1;
166     }
167     if (!d1 && sp_on(a, b.s)) {
168         p = b.s;
169         return 2;
170     }
171     if (!d2 && sp_on(a, b.t)) {
172         p = b.t;
173         return 2;
174     }
175     if (!d3 && sp_on(b, a.s)) {
176         p = a.s;
177         return 2;

```

```

178     }
179     if (!d4 && sp_on(b, a.t)) {
180         p = a.t;
181         return 2;
182     }
183     return 0;
184 }
185
186 Point project(const Line &l, const Point &p) {
187     Point base(l);
188     double r = dot(base, p - l.s) / sqr(length(base));
189     return l.s + (base * r);
190 }
191
192 double sp_dist(const Line &l, const Point &p) {
193     if (l.s == l.t) return dist(l.s, p);
194     Point x = p - l.s, y = p - l.t, z = l.t - l.s;
195     if (sgn(dot(x, z)) < 0) return length(x); //P距离A更近
196     if (sgn(dot(y, z)) > 0) return length(y); //P距离B更近
197     return abs(det(x, z) / length(z)); //面积除以底边长
198 }
199
200 double lp_dist(const Line &l, const Point &p) {
201     Point x = p - l.s, y = p - l.t, z = l.t - l.s;
202     return abs(det(x, z) / length(z)); //面积除以底边长
203 }
204
205 int lc_cross(const Line &l, const Point &a, const double &r, pair<Point,
Point> &ans) {
206     int num = 0;
207     Point pr = project(l, a);
208     double dis = dist(pr, a);
209     double tmp = r * r - dis * dis;
210     if (sgn(tmp) == 1) num = 2;
211     else if (sgn(tmp) == 0) num = 1;
212     else return 0;
213     double base = sqrt(r * r - dis * dis);
214     Point e(l);
215     e.standardize();
216     e = e * base;
217     ans = make_pair(pr + e, pr - e);
218     return num;
219 }
220
221 int cc_cross(const Point &c1, const double &r1, const Point &c2, const
double &r2, pair<Point, Point> &ans) {
222     double x1 = c1.x, x2 = c2.x, y1 = c1.y, y2 = c2.y;
223     double d = length(c1 - c2);
224     if (sgn(fabs(r1 - r2) - d) > 0) return -1; //内含
225     if (sgn(r1 + r2 - d) < 0) return 0; //相离
226     double a = r1 * (x1 - x2) * 2, b = r1 * (y1 - y2) * 2, c = r2 * r2 - r1
* r1 - d * d;
227     double p = a * a + b * b, q = -a * c * 2, r = c * c - b * b;
228
229     double cosa, sina, cosb, sinb;
230     //One Intersection
231     if (sgn(d - (r1 + r2)) == 0 || sgn(d - fabs(r1 - r2)) == 0) {
232         cosa = -q / p / 2;

```

```

233     sina = sqrt(1 - sqr(cosa));
234     Point p0(x1 + r1 * cosa, y1 + r1 * sina);
235     if (sgn(dist(p0, c2) - r2)) p0.y = y1 - r1 * sina;
236     ans = pair<Point, Point>(p0, p0);
237     return 1;
238 }
239 //Two Intersections
240 double delta = sqrt(q * q - p * r * 4);
241 cosa = (delta - q) / p / 2;
242 cosb = (-delta - q) / p / 2;
243 sina = sqrt(1 - sqr(cosa));
244 sinb = sqrt(1 - sqr(cosb));
245 Point p1(x1 + r1 * cosa, y1 + r1 * sina);
246 Point p2(x1 + r1 * cosb, y1 + r1 * sinb);
247 if (sgn(dist(p1, c2) - r2)) p1.y = y1 - r1 * sina;
248 if (sgn(dist(p2, c2) - r2)) p2.y = y1 - r1 * sinb;
249 if (p1 == p2) p1.y = y1 - r1 * sina;
250 ans = pair<Point, Point>(p1, p2);
251 return 2;
252 }
253
254 Point lp_sym(const Line &l, const Point &p) {
255     return p + (project(l, p) - p) * 2;
256 }
257
258 double alpha(const Point &t1, const Point &t2) {
259     double theta;
260     theta = atan2((double) t2.y, (double) t2.x) - atan2((double) t1.y,
261 (double) t1.x);
262     if (sgn(theta) < 0)
263         theta += 2.0 * PI;
264     return theta;
265 }
266
267 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
268     int cnt = 0;
269     int tmp;
270     for (int i = 1; i <= n; ++i) {
271         int j = i < n ? i + 1 : 1;
272         if (sp_on(Line(P[i], P[j]), a)) return 2; //点在多边形上
273         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
274             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
275     }
276     return cnt & 1; //穿过奇数次则在多边形以内
277 }
278
279 bool pip_convex_jud(const Point &a, const Point &L, const Point &R) { //判断
    AL是否在AR右边
280     return sgn(det(L - a, R - a)) > 0; //必须严格以内
281 }
282
283 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
284     //点按逆时针给出

```



```

284     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
285     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
286     int l = 1, r = n - 2;
287     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
288         int mid = (l + r + 1) >> 1;
289         if (pip_convex_jud(P[0], P[mid], a)) l = mid;
290         else r = mid - 1;
291     }
292     if (pip_convex_jud(P[1], a, P[l + 1])) return 0; //在P[pos_(pos+1)]外
293     if (sp_on(Line(P[1], P[l + 1]), a)) return 2; //在P[pos_(pos+1)]上
294     return 1;
295 }
296 // 多边形是否包含线段
297 // 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，
298 // 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，
299 // 则该线段一定在多边形内。
300
301 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】
302     for (int i1 = 1; i1 <= n; ++i1) {
303         int j1 = i1 < n ? i1 + 1 : 1;
304         for (int i2 = 1; i2 <= m; ++i2) {
305             int j2 = i2 < m ? i2 + 1 : 1;
306             Point tmp;
307             if (ss_cross(Line(A[i1], A[j1]), Line(B[i2], B[j2]), tmp))
return 0; //两线段相交
308             if (pip(B, m, A[i1]) || pip(A, n, B[i2])) return 0; //点包含在内
309         }
310     }
311     return 1;
312 }
313
314 double area(Point *P, int n) { // 【任意多边形P的面积】
315     double s = 0;
316     for (int i = 1; i <= n; i++) s += det(P[i], P[i < n ? i + 1 : 1]);
317     return s / 2.0;
318 }
319
320 Line Q[N];
321
322 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
323 int halfcut(Line *L, int n, Point *P) { // 【半平面交】
324     sort(L, L + n, [](const Line &a, const Line &b) {
325         double d = atan2((a.t - a.s).y, (a.t - a.s).x) - atan2((b.t -
b.s).y, (b.t - b.s).x);
326         return sgn(d) ? sgn(d) < 0 : judge(a, b.s);
327     });
328
329     int m = n;
330     n = 0;
331     for (int i = 0; i < m; ++i)
332         if (i == 0 || sgn(atan2(Point(L[i]).y, Point(L[i]).x) -
atan2(Point(L[i - 1]).y, Point(L[i - 1]).x)))

```

```

333         L[n++] = L[i];
334     int h = 1, t = 0;
335     for (int i = 0; i < n; ++i) {
336         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
337         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
338         Q[++t] = L[i];
339
340     }
341     while (h < t && judge(Q[h], ll_intersection(Q[t], Q[t - 1]))) --t;
342     while (h < t && judge(Q[t], ll_intersection(Q[h], Q[h + 1]))) ++h;
343     n = 0;
344     for (int i = h; i <= t; ++i) {
345         P[n++] = ll_intersection(Q[i], Q[i < t ? i + 1 : h]);
346     }
347     return n;
348 }
349
350 Point v1[N], v2[N];
351
352 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
353     for (int i = 0; i < n; ++i) v1[i] = P1[(i + 1) % n] - P1[i];
354     for (int i = 0; i < m; ++i) v2[i] = P2[(i + 1) % m] - P2[i];
355     int t = 0, i = 0, j = 0;
356     v[t++] = P1[0] + P2[0];
357     while (i < n && j < m) v[t] = v[t - 1] + (sgn(det(v1[i], v2[j]))) > 0 ?
v1[i++] : v2[j++], t++;
358     while (i < n) v[t] = v[t - 1] + v1[i++], t++;
359     while (j < m) v[t] = v[t - 1] + v2[j++], t++;
360     return t;
361 }
362
363 circle getcircle(const Point &A, const Point &B, const Point &C) { // 【三点确
定一圆】向量垂心法
364     Point P1 = (A + B) * 0.5, P2 = (A + C) * 0.5;
365     Line R1 = Line(P1, P1 + normal(B - A));
366     Line R2 = Line(P2, P2 + normal(C - A));
367     circle o;
368     o.o = ll_intersection(R1, R2);
369     o.r = length(A - o.o);
370     return o;
371 }
372
373 struct ConvexHull {
374
375     int op;
376
377     struct cmp {
378         bool operator()(const Point &a, const Point &b) const {
379             return sgn(a.x - b.x) < 0 || sgn(a.x - b.x) == 0 && sgn(a.y -
b.y) < 0;
380         }
381     };
382
383     set<Point, cmp> s;
384

```

```

385     ConvexHull(int o) {
386         op = o;
387         s.clear();
388     }
389
390     inline int PIP(Point P) {
391         set<Point>::iterator it = s.lower_bound(Point(P.x, -dinf)); //找到第
一个横坐标大于P的点
392         if (it == s.end()) return 0;
393         if (sgn(it->x - P.x) == 0) return sgn((P.y - it->y) * op) <= 0; //比
较纵坐标大小
394         if (it == s.begin()) return 0;
395         set<Point>::iterator j = it, k = it;
396         --j;
397         return sgn(det(P - *j, *k - *j) * op) >= 0; //看叉姬1
398     }
399
400     inline int judge(set<Point>::iterator it) {
401         set<Point>::iterator j = it, k = it;
402         if (j == s.begin()) return 0;
403         --j;
404         if (++k == s.end()) return 0;
405         return sgn(det(*it - *j, *k - *j) * op) >= 0; //看叉姬
406     }
407
408     inline void insert(Point P) {
409         if (PIP(P)) return; //如果点P已经在凸壳上或凸包里就不插入了
410         set<Point>::iterator tmp = s.lower_bound(Point(P.x, -inf));
411         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
412         s.insert(P);
413         set<Point>::iterator it = s.find(P), p = it;
414         if (p != s.begin()) {
415             --p;
416             while (judge(p)) {
417                 set<Point>::iterator temp = p--;
418                 s.erase(temp);
419             }
420         }
421         if ((p = ++it) != s.end()) {
422             while (judge(p)) {
423                 set<Point>::iterator temp = p++;
424                 s.erase(temp);
425             }
426         }
427     }
428 } up(1), down(-1);
429
430 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
431 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
432 circle min_circle(Point *P, int n) { //【求点集P的最小覆盖圆】 O(n)
433 // random_shuffle(P, P+n);
434 Random(P, n);
435 circle C = circle(P[0], 0);
436 for (int i = 1; i < n; ++i)
437     if (!PIC(C, P[i])) {

```

```

438         C = circle(P[i], 0);
439         for (int j = 0; j < i; ++j)
440             if (!PIC(C, P[j])) {
441                 C.o = (P[i] + P[j]) * 0.5, C.r = length(P[j] - C.o);
442                 for (int k = 0; k < j; ++k) if (!PIC(C, P[k])) C =
getcircle(P[i], P[j], P[k]);
443             }
444         }
445         return C;
446     }
447

```

自适应辛普森

```

1  double f(double x) {
2  }
3
4  double simpson(double l, double r) {
5      double mid = (l + r) / 2;
6      return (r - l) * (f(l) + 4 * f(mid) + f(r)) / 6; // 辛普森公式
7  }
8
9  double asr(double l, double r, double EPS, double ans) {
10     double mid = (l + r) / 2;
11     double fl = simpson(l, mid), fr = simpson(mid, r);
12     if (abs(fl + fr - ans) <= 15 * EPS)
13         return fl + fr + (fl + fr - ans) / 15; // 足够相似的话就直接返回
14     return asr(l, mid, EPS / 2, fl) +
15           asr(mid, r, EPS / 2, fr); // 否则分割成两段递归求解
16 }
17

```

数据结构

仙人掌

```

1  /*
2   仙人掌:任意一条边至多只出现在一条简单回路的无向连通图称为仙人掌。
3   转化为圆方树, 然后根据树的算法来做一些问题, 注意区分圆点和方点
4   这题:求带环(环和环之间无公共边)无向图两点间的最短路径
5   */
6
7  #include <iostream>
8  #include <cstring>
9  #include <algorithm>
10
11  using namespace std;
12
13  const int N = 12010, M = N * 3;
14
15  int n, m, Q, new_n;
16  int h1[N], h2[N], e[M], w[M], ne[M], idx;

```

```

17 int dfn[N], low[N], cnt;
18 int s[N], stot[N], fu[N], fw[N];
19 int fa[N][14], depth[N], d[N];
20 int A, B;
21
22 void add(int h[], int a, int b, int c)
23 {
24     e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx ++ ;
25 }
26
27 void build_circle(int x, int y, int z)
28 {
29     int sum = z;
30     for (int k = y; k != x; k = fu[k])
31     {
32         s[k] = sum;
33         sum += fw[k];
34     }
35     s[x] = stot[x] = sum;
36     add(h2, x, ++ new_n, 0);
37     for (int k = y; k != x; k = fu[k])
38     {
39         stot[k] = sum;
40         add(h2, new_n, k, min(s[k], sum - s[k]));
41     }
42 }
43
44 void tarjan(int u, int from)
45 {
46     dfn[u] = low[u] = ++ cnt;
47     for (int i = h1[u]; ~i; i = ne[i])
48     {
49         int j = e[i];
50         if (!dfn[j])
51         {
52             fu[j] = u, fw[j] = w[i];
53             tarjan(j, i);
54             low[u] = min(low[u], low[j]);
55             if (dfn[u] < low[j]) add(h2, u, j, w[i]);
56         }
57         else if (i != (from ^ 1)) low[u] = min(low[u], dfn[j]);
58     }
59     for (int i = h1[u]; ~i; i = ne[i])
60     {
61         int j = e[i];
62         if (dfn[u] < dfn[j] && fu[j] != u)
63             build_circle(u, j, w[i]);
64     }
65 }
66
67 void dfs_lca(int u, int father)
68 {
69     depth[u] = depth[father] + 1;
70     fa[u][0] = father;
71     for (int k = 1; k <= 13; k ++ )
72         fa[u][k] = fa[fa[u][k - 1]][k - 1];
73     for (int i = h2[u]; ~i; i = ne[i])
74     {

```

```

75     int j = e[i];
76     d[j] = d[u] + w[i];
77     dfs_lca(j, u);
78 }
79 }
80
81 int lca(int a, int b)
82 {
83     if (depth[a] < depth[b]) swap(a, b);
84     for (int k = 13; k >= 0; k -- )
85         if (depth[fa[a][k]] >= depth[b])
86             a = fa[a][k];
87     if (a == b) return a;
88     for (int k = 13; k >= 0; k -- )
89         if (fa[a][k] != fa[b][k])
90             {
91                 a = fa[a][k];
92                 b = fa[b][k];
93             }
94     A = a, B = b;
95     return fa[a][0];
96 }
97
98 int main()
99 {
100     scanf("%d%d%d", &n, &m, &Q);
101     new_n = n;
102     memset(h1, -1, sizeof h1);
103     memset(h2, -1, sizeof h2);
104     while (m -- )
105     {
106         int a, b, c;
107         scanf("%d%d%d", &a, &b, &c);
108         add(h1, a, b, c), add(h1, b, a, c);
109     }
110     tarjan(1, -1);
111     dfs_lca(1, 0);
112
113     while (Q -- )
114     {
115         int a, b;
116         scanf("%d%d", &a, &b);
117         int p = lca(a, b);
118         if (p <= n) printf("%d\n", d[a] + d[b] - d[p] * 2);
119         else
120             {
121                 int da = d[a] - d[A], db = d[b] - d[B];
122                 int l = abs(s[A] - s[B]);
123                 int dm = min(l, stot[A] - l);
124                 printf("%d\n", da + dm + db);
125             }
126     }
127
128     return 0;
129 }
130

```

CDQ

```
1  /*
2  处理三维偏序问题，
3  每个node的三维不能完全相等，完全相等的话加权做
4  */
5
6  #include <iostream>
7  #include <cstring>
8  #include <algorithm>
9
10 using namespace std;
11
12 const int N = 100010, M = 200010;
13
14 int n, m;
15 struct Data
16 {
17     int a, b, c, s, res;
18
19     bool operator< (const Data& t) const
20     {
21         if (a != t.a) return a < t.a;
22         if (b != t.b) return b < t.b;
23         return c < t.c;
24     }
25     bool operator== (const Data& t) const
26     {
27         return a == t.a && b == t.b && c == t.c;
28     }
29 }q[N], w[N];
30 int tr[M], ans[N];
31
32 int lowbit(int x)
33 {
34     return x & -x;
35 }
36
37 void add(int x, int v)
38 {
39     for (int i = x; i < M; i += lowbit(i)) tr[i] += v;
40 }
41
42 int query(int x)
43 {
44     int res = 0;
45     for (int i = x; i; i -= lowbit(i)) res += tr[i];
46     return res;
47 }
48
49 void merge_sort(int l, int r)
50 {
51     if (l >= r) return;
52     int mid = l + r >> 1;
53     merge_sort(l, mid), merge_sort(mid + 1, r);
```

```

54     int i = 1, j = mid + 1, k = 0;
55     while (i <= mid && j <= r)
56         if (q[i].b <= q[j].b) add(q[i].c, q[i].s), w[k ++ ] = q[i ++ ];
57         else q[j].res += query(q[j].c), w[k ++ ] = q[j ++ ];
58     while (i <= mid) add(q[i].c, q[i].s), w[k ++ ] = q[i ++ ];
59     while (j <= r) q[j].res += query(q[j].c), w[k ++ ] = q[j ++ ];
60     for (i = 1; i <= mid; i ++ ) add(q[i].c, -q[i].s);
61     for (i = 1, j = 0; j < k; i ++, j ++ ) q[i] = w[j];
62 }
63
64 int main()
65 {
66     scanf("%d%d", &n, &m);
67     for (int i = 0; i < n; i ++ )
68     {
69         int a, b, c;
70         scanf("%d%d%d", &a, &b, &c);
71         q[i] = {a, b, c, 1};
72     }
73     sort(q, q + n);
74
75     int k = 1;
76     for (int i = 1; i < n; i ++ )
77         if (q[i] == q[k - 1]) q[k - 1].s ++ ;
78         else q[k ++ ] = q[i];
79
80     merge_sort(0, k - 1);
81     for (int i = 0; i < k; i ++ )
82         ans[q[i].res + q[i].s - 1] += q[i].s;
83
84     for (int i = 0; i < n; i ++ ) printf("%d\n", ans[i]);
85
86     return 0;
87 }
88

```

kruskal重构树

```

1  int pa[N];
2
3  void init(int n) {
4      for (int i = 0; i <= n; i++) {
5          pa[i] = i;
6      }
7  }
8
9  int find(int a) {
10     return pa[a] == a ? a : pa[a] = find(pa[a]);
11 }
12
13 struct edge {
14     int from, to, l;
15 };
16
17 int w[N];

```



```

18 edge e[N];
19 vector<int> g[N];
20
21 int kruskal(int n, int m) {
22     int kcmt = n;
23     init(n);
24     sort(e + 1, e + 1 + m, [](edge a, edge b) { return a.l < b.l; });
25     for (int i = 1; i <= m; i++) {
26         int u = find(e[i].from);
27         int v = find(e[i].to);
28         if (u == v) continue;
29         w[++kcmt] = e[i].l;
30         pa[kcmt] = pa[u] = pa[v] = kcmt;
31         g[u].push_back(kcmt);
32         g[v].push_back(kcmt);
33         g[kcmt].push_back(u);
34         g[kcmt].push_back(v);
35     }
36     return kcmt;
37 }
38

```

普通莫队

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const int N = 1e6 + 10, M = 1e6 + 10;
5  int a[N];
6
7  struct node {
8      int id, l, r;
9  } mp[M];
10
11 int len;
12 int ans[M], cnt[1000010];
13
14 int getNum(int l) {
15     return l / len;
16 }
17
18 //左指针的分块, 右指针的大小
19 bool cmp (const node &a, const node &b) {
20     if(getNum(a.l) == getNum(b.l)) return a.r < b.r;
21     return a.l < b.l;
22 }
23 /* 奇偶优化
24 struct node {
25     int l, r, id;
26     bool operator<(const node &x) const {
27         if (l / unit != x.l / unit) return l < x.l;
28         if ((l / unit) & 1)
29             return r < x.r; // 注意这里和下面一行不能写小于（大于）等于
30         return r > x.r;
31     }
32 };

```

```

33  */
34
35  void add(int x, int& res) {
36      if(cnt[x] == 0) res++;
37      cnt[x] ++;
38  }
39
40  void del(int x, int& res) {
41      cnt[x] --;
42      if(cnt[x] == 0) res --;
43  }
44
45  int main() {
46      ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
47
48      int n;
49      cin >> n;
50      for(int i = 1; i <= n; ++ i) {
51          cin >> a[i];
52      }
53      int m;
54      cin >> m;
55      len = sqrt((double)n * n / m);
56      for(int i = 1; i <= m; ++ i) {
57          mp[i].id = i;
58          cin >> mp[i].l >> mp[i].r;
59      }
60      sort(mp + 1, mp + m + 1, cmp);
61
62      //离线处理询问
63      int res = 0, i = 0, j = 0;
64      for(int k = 1; k <= m; ++ k) {
65          int id = mp[k].id, l = mp[k].l, r = mp[k].r;
66          while(j < r) add(a[++j], res);
67          while(j > r) del(a[j--], res);
68          while(i < l) del(a[i++], res);
69          while(i > l) add(a[--i], res);
70          ans[id] = res;
71      }
72
73      for(int i = 1; i <= m; ++ i) {
74          cout << ans[i] << endl;
75      }
76      return 0;
77  }
78

```

带修莫队

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const int N = 10010;
5
6  int a[N], cnt[1000010], ans[N];
7

```

```

8  int len, mq, mc;
9
10 struct Query {
11     int id, l, r, t;
12 } q[N];
13
14 struct Modify {
15     int p, c;
16 } c[N];
17
18 int getNum(int x) {
19     return x / len;
20 }
21
22 // l所在块的编号, r所在块的编号, t升序
23
24 bool cmp(const Query& a, const Query& b) {
25     if(getNum(a.l) == getNum(b.l) && getNum(a.r) == getNum(b.r)) {
26         return a.t < b.t;
27     }
28     if(getNum(a.l) == getNum(b.l)) return a.r < b.r;
29     return a.l < b.l;
30 }
31
32 void add(int x, int& res) {
33     if (!cnt[x]) res ++ ;
34     cnt[x] ++ ;
35 }
36
37 void del(int x, int& res) {
38     cnt[x] -- ;
39     if (!cnt[x]) res -- ;
40 }
41
42
43 int main() {
44     ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
45
46     int n, m;
47     cin >> n >> m;
48     char op;
49     int x, y;
50     for(int i = 1; i <= n; ++ i) {
51         cin >> a[i];
52     }
53     for(int i = 1; i <= m; ++ i) {
54         cin >> op >> x >> y;
55         if (op == 'Q') q[++ mq] = {mq, x, y, mc};
56         else c[ ++ mc] = {x, y};
57     }
58
59     ///
60     len = cbrt((double)n * mc) + 1;
61     sort(q + 1, q + mq + 1, cmp);
62
63     int i = 1, j = 0, t = 0, res = 0;
64     for(int k = 1; k <= mq; ++ k) {
65         int id = q[k].id, l = q[k].l, r = q[k].r, tm = q[k].t;

```

```

66     while(j < r) add(a[++ j], res);
67     while(j > r) del(a[j --], res);
68     while(i < l) del(a[i ++], res);
69     while(i > l) add(a[-- i], res);
70     while(t < tm) {
71         ++ t;
72         if(c[t].p >= i && c[t].p <= j) {
73             del(a[c[t].p], res);
74             add(c[t].c, res);
75         }
76         swap(a[c[t].p], c[t].c);
77     }
78     while(t > tm) {
79         if(c[t].p >= i && c[t].p <= j) {
80             del(a[c[t].p], res);
81             add(c[t].c, res);
82         }
83         swap(a[c[t].p], c[t].c);
84         -- t;
85     }
86     ans[id] = res;
87 }
88
89 for(int i = 1; i <= mq; ++ i) {
90     cout << ans[i] << endl;
91 }
92 }
93

```

回滚莫队

```

1  /*
2   离线，询问按左端点升序为第一关键字，右端点升序为第二关键字
3   对于都在块内的点直接暴力，否则跨块：
4   若当前左端点所属的块与上一个不同，则将左端点初始为当前块的右端点+1，右端点初始为当前块的右
   端点
5   左端点每次暴力，右端点单调
6   */
7
8  #include <iostream>
9  #include <cstring>
10 #include <cstdio>
11 #include <algorithm>
12 #include <cmath>
13 #include <vector>
14
15 using namespace std;
16
17 typedef long long LL;
18 const int N = 100010;
19
20 int n, m, len;
21 int w[N], cnt[N];
22 LL ans[N];
23 struct Query
24 {

```

```

25     int id, l, r;
26 }q[N];
27 vector<int> nums;
28
29 int get(int x)
30 {
31     return x / len;
32 }
33
34 bool cmp(const Query& a, const Query& b)
35 {
36     int i = get(a.l), j = get(b.l);
37     if (i != j) return i < j;
38     return a.r < b.r;
39 }
40
41 void add(int x, LL& res)
42 {
43     cnt[x] ++ ;
44     res = max(res, (LL)cnt[x] * nums[x]);
45 }
46
47 int main()
48 {
49     scanf("%d%d", &n, &m);
50     len = sqrt(n);
51     for (int i = 1; i <= n; i ++ ) scanf("%d", &w[i]),
nums.push_back(w[i]);
52     sort(nums.begin(), nums.end());
53     nums.erase(unique(nums.begin(), nums.end()), nums.end());
54     for (int i = 1; i <= n; i ++ )
55         w[i] = lower_bound(nums.begin(), nums.end(), w[i]) - nums.begin();
56
57     for (int i = 0; i < m; i ++ )
58     {
59         int l, r;
60         scanf("%d%d", &l, &r);
61         q[i] = {i, l, r};
62     }
63     sort(q, q + m, cmp);
64
65     for (int x = 0; x < m;)
66     {
67         int y = x;
68         while (y < m && get(q[y].l) == get(q[x].l)) y ++ ;
69         int right = get(q[x].l) * len + len - 1;
70
71         // 暴力求块内的询问
72         while (x < y && q[x].r <= right)
73         {
74             LL res = 0;
75             int id = q[x].id, l = q[x].l, r = q[x].r;
76             for (int k = l; k <= r; k ++ ) add(w[k], res);
77             ans[id] = res;
78             for (int k = l; k <= r; k ++ ) cnt[w[k]] -- ;
79             x ++ ;
80         }
81

```

```

82 // 求块外的询问
83 LL res = 0;
84 int i = right, j = right + 1;
85 while (x < y)
86 {
87     int id = q[x].id, l = q[x].l, r = q[x].r;
88     while (i < r) add(w[ ++ i], res);
89     LL backup = res;
90     while (j > l) add(w[ -- j], res);
91     ans[id] = res;
92     while (j < right + 1) cnt[w[j ++ ]] -- ;
93     res = backup;
94     x ++ ;
95 }
96 memset(cnt, 0, sizeof cnt);
97 }
98
99 for (int i = 0; i < m; i ++ ) printf("%lld\n", ans[i]);
100 return 0;
101 }
102

```

线段树合并分裂

```

1  ll nodetot, recycnt, bac[N << 5], ch[N << 5][2], rt[N];
2  ll val[N << 5];
3
4  ll newnod() { return (recycnt ? bac[recycnt--] : ++nodetot); }
5
6  void recyc(ll p) {
7      bac[++recycnt] = p, ch[p][0] = ch[p][1] = val[p] = 0;
8      return;
9  }
10
11 void pushdown(ll p) {
12
13 }
14
15 void pushup(ll p) {
16     val[p] = 0;
17     if (ch[p][0]) val[p] += val[ch[p][0]];
18     if (ch[p][1]) val[p] += val[ch[p][1]];
19 }
20
21 void modify(ll &p, ll l, ll r, ll pos, ll v) {
22     if (!p) { p = newnod(); }
23     if (l == r) {
24         val[p] += v;
25         return;
26     }
27     ll mid = (l + r) >> 1;

```

```

28 //    pushdown(p);
29     if (pos <= mid) { modify(ch[p][0], l, mid, pos, v); }
30     else { modify(ch[p][1], mid + 1, r, pos, v); }
31     pushup(p);
32     return;
33 }
34
35 ll query(ll p, ll l, ll r, ll x1, ll xr) {
36     if (xr < l || r < x1) { return 0; }
37     if (x1 <= l && r <= xr) { return val[p]; }
38     ll mid = (l + r) >> 1;
39     //    pushdown(p);
40     return query(ch[p][0], l, mid, x1, xr) + query(ch[p][1], mid + 1, r, x1,
41     xr);
42 }
43
44 ll kth(ll p, ll l, ll r, ll k) {
45     if (l == r) { return l; }
46     ll mid = (l + r) >> 1;
47     //    pushdown(p);
48     if (val[ch[p][0]] >= k) { return kth(ch[p][0], l, mid, k); }
49     else { return kth(ch[p][1], mid + 1, r, k - val[ch[p][0]]); }
50 }
51
52 ll merge(ll x, ll y, ll l, ll r) {
53     if (!x || !y) {
54         return x + y;
55     } // 只有一边有点，不用合并
56     ll p = newnod(); // 创建一个新结点 p
57     if (l == r) { // 边界（某些时候可以省略，见下面一个代码）
58         val[p] = val[x] + val[y];
59         return p;
60     }
61     //    pushdown(x), pushdown(y);
62     ll mid = (l + r) >> 1;
63     ch[p][0] = merge(ch[x][0], ch[y][0], l, mid);
64     ch[p][1] = merge(ch[x][1], ch[y][1], mid + 1, r);
65     recyc(x), recyc(y); // 垃圾回收
66     pushup(p); // pushup
67     return p;
68 }
69
70 void split(ll x, ll &y, ll k) {
71     if (x == 0) return;
72     y = newnod();
73     ll v = val[ch[x][0]];
74     //    pushdown(x);
75     if (k > v) { split(ch[x][1], ch[y][1], k - v); }
76     else { swap(ch[x][1], ch[y][1]); }
77     if (k < v) { split(ch[x][0], ch[y][0], k); }
78     val[y] = val[x] - k;
79     val[x] = k;
80     return;
81 }

```

主席树

```
1  #include <bits/stdc++.h>
2
3  using namespace std;
4  typedef long long ll;
5  const ll N = 1 << 20;
6
7  ll ch[N << 5][2], rt[N], tot;
8  ll val[N << 5];
9
10 ll update(ll a, ll b) {
11     return a + b;
12 }
13
14 ll build(ll l, ll r) { // 建树
15     ll p = ++tot;
16     if (l == r) {
17         //初始化
18         val[p] = 0;
19         return p;
20     }
21     ll mid = (l + r) >> 1;
22     ch[p][0] = build(l, mid);
23     ch[p][1] = build(mid + 1, r);
24     val[p] = update(val[ch[p][0]], val[ch[p][1]]);
25     return p; // 返回该子树的根节点
26 }
27
28 ll modify(ll pre, ll l, ll r, ll pos, ll v) { // 插入操作
29     ll now = ++tot;
30     ch[now][0] = ch[pre][0], ch[now][1] = ch[pre][1];
31     if (l == r) {
32         val[now] = val[pre] + v;
33         return now;
34     }
35     ll mid = (l + r) >> 1;
36     if (pos <= mid)
37         ch[now][0] = modify(ch[now][0], l, mid, pos, v);
38     else
39         ch[now][1] = modify(ch[now][1], mid + 1, r, pos, v);
40     val[now] = update(val[ch[now][0]], val[ch[now][1]]);
41     return now;
42 }
43
44 ll kth(ll pre, ll now, ll l, ll r, ll k) { // 查询操作
45     ll mid = (l + r) >> 1;
46     ll x = val[ch[now][0]] - val[ch[pre][0]]; // 通过区间减法得到左儿子的信息
47     if (l == r) return l;
48     if (k <= x) // 说明在左儿子中
49         return kth(ch[pre][0], ch[now][0], l, mid, k);
50     else // 说明在右儿子中
51         return kth(ch[pre][1], ch[now][1], mid + 1, r, k - x);
52 }
53
54 ll query(ll pre, ll now, ll l, ll r, ll ql, ll qr) { // 查询操作
55     if (ql <= l && r <= qr) {
```



```

56         return val[now] - val[pre];
57     }
58     if (qr < 1 || r < ql) {
59         return 0;
60     }
61     ll mid = (l + r) >> 1;
62     ll lv = query(ch[pre][0], ch[now][0], l, mid, ql, qr);
63     ll rv = query(ch[pre][1], ch[now][1], mid + 1, r, ql, qr);
64     return update(lv, rv);
65 }
66 //修改查询记得用rt[0]!!!
67

```

LCT

```

1  ll ch[N][2], f[N], sum[N], val[N], tag[N], siz[N], siz2[N];
2
3  inline void pushup(ll p) {
4      sum[p] = sum[ch[p][0]] ^ sum[ch[p][1]] ^ val[p];
5      siz[p] = siz[ch[p][0]] + siz[ch[p][1]] + 1 + siz2[p];
6  }
7
8  inline void pushdown(ll p) {
9      if (tag[p]) {
10         if (ch[p][0]) swap(ch[ch[p][0]][0], ch[ch[p][0]][1]), tag[ch[p][0]]
11         ^= 1;
12         if (ch[p][1]) swap(ch[ch[p][1]][0], ch[ch[p][1]][1]), tag[ch[p][1]]
13         ^= 1;
14         tag[p] = 0;
15     }
16 }
17
18 ll getch(ll x) { return ch[f[x]][1] == x; }
19
20 bool isroot(ll x) { return ch[f[x]][0] != x && ch[f[x]][1] != x; }
21
22 inline void rotate(ll x) {
23     ll y = f[x], z = f[y], k = getch(x);
24     if (!isroot(y)) ch[z][ch[z][1] == y] = x;
25     // 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)
26     ch[y][k] = ch[x][!k], f[ch[x][!k]] = y;
27     ch[x][!k] = y, f[y] = x, f[x] = z;
28     pushup(y), pushup(x);
29 }
30
31 // 从上到下一层一层 pushDown 即可
32 void update(ll p) {
33     if (!isroot(p)) update(f[p]);
34     pushdown(p);
35 }
36
37 inline void splay(ll x) {
38     update(x); // 马上就能看到啦。 在
39     // Splay之前要把旋转会经过的路径上的点都PushDown
40     for (ll fa; fa = f[x], !isroot(x); rotate(x)) {
41
42     }
43 }
44

```

```

39         if (!isroot(fa)) rotate(getch(fa) == getch(x) ? fa : x);
40     }
41 }
42
43 // 回顾一下代码
44 inline void access(ll x) {
45     for (ll p = 0; x; p = x, x = f[x]) {
46         splay(x), siz2[x] += siz[ch[x][1]] - siz[p], ch[x][1] = p,
pushup(x);
47     }
48 }
49
50 inline void makeroot(ll p) {
51     access(p);
52     splay(p);
53     swap(ch[p][0], ch[p][1]);
54     tag[p] ^= 1;
55 }
56
57 inline void split(ll a, ll b) {
58     makeroot(a);
59     access(b);
60     splay(b);
61 }
62
63
64 inline ll find(ll p) {
65     access(p), splay(p);
66     while (ch[p][0]) pushdown(p), p = ch[p][0];
67     splay(p);
68     return p;
69 }
70
71 inline void link(ll x, ll y) {
72     makeroot(y);
73     makeroot(x);
74     if (find(y) != x) {
75         f[x] = y;
76         siz2[y] += siz[x];
77     }
78 }
79
80 inline void cut(ll x, ll y) {
81     makeroot(x);
82     if (find(y) == x && f[y] == x) {
83         ch[x][1] = f[y] = 0;
84         pushup(x);
85     }
86 }
87
88 void init(int n) {
89     for (int i = 1; i <= n; i++) siz[i] = 1;
90 }
91

```

Splay1

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  struct Splay {
5      static const int N = 100005;
6      int rt, tot, fa[N], ch[N][2], val[N], cnt[N], sz[N];
7      // 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
8      void maintain(int x) { //更新x节点字数大小
9          sz[x] = sz[ch[x][0]] + sz[ch[x][1]] + cnt[x];
10     }
11
12     bool get(int x) {
13         return x == ch[fa[x]][1];
14     } //返回节点是父亲的0/1-左/右儿子
15
16     void clear(int x) { //销毁节点x
17         ch[x][0] = ch[x][1] = fa[x] = val[x] = sz[x] = cnt[x] = 0;
18     }
19
20     void rotate(int x) { //旋转
21         int y = fa[x], z = fa[y], chk = get(x);
22         ch[y][chk] = ch[x][chk ^ 1];
23         fa[ch[x][chk ^ 1]] = y;
24         ch[x][chk ^ 1] = y;
25         fa[y] = x;
26         fa[x] = z;
27         if (z) ch[z][y == ch[z][1]] = x;
28         maintain(x);
29         maintain(y);
30     }
31
32     void splay(int x) { //将x节点移动到根
33         for (int f = fa[x]; f = fa[x], f; rotate(x))
34             if (fa[f]) rotate(get(x) == get(f) ? f : x);
35         rt = x;
36     }
37
38     void ins(int k) { //插入
39         if (!rt) {
40             val[++tot] = k;
41             cnt[tot]++;
42             rt = tot;
43             maintain(rt);
44             return;
45         }
46         int cnr = rt, f = 0;
47         while (1) {
48             if (val[cnr] == k) {
49                 cnt[cnr]++;
50                 maintain(cnr);
51                 maintain(f);
52                 splay(cnr);
53                 break;
54             }
```

```

55         f = cnr;
56         cnr = ch[cnr][val[cnr] < k];
57         if (!cnr) {
58             val[++tot] = k;
59             cnt[tot]++;
60             fa[tot] = f;
61             ch[f][val[f] < k] = tot;
62             maintain(tot);
63             maintain(f);
64             splay(tot);
65             break;
66         }
67     }
68 }
69
70 int rk(int k) { // k权值的排名
71     int res = 0, cnr = rt;
72     while (1) {
73         if (k < val[cnr]) {
74             cnr = ch[cnr][0];
75         } else {
76             res += sz[ch[cnr][0]];
77             if (k == val[cnr]) {
78                 splay(cnr);
79                 return res + 1;
80             }
81             res += cnt[cnr];
82             cnr = ch[cnr][1];
83         }
84     }
85 }
86
87 int kth(int k) { //第k名的权值
88     int cnr = rt;
89     while (1) {
90         if (ch[cnr][0] && k <= sz[ch[cnr][0]]) {
91             cnr = ch[cnr][0];
92         } else {
93             k -= cnt[cnr] + sz[ch[cnr][0]];
94             if (k <= 0) {
95                 splay(cnr);
96                 return val[cnr];
97             }
98             cnr = ch[cnr][1];
99         }
100     }
101 }
102
103 int pre() { //前驱节点编号
104     int cnr = ch[rt][0];
105     while (ch[cnr][1]) cnr = ch[cnr][1];
106     splay(cnr);
107     return cnr;
108 } // 若需要得到前驱 tree.ins(x), printf("%d\n", tree.val[tree.pre()]),
109 // tree.del(x);
110
111 int nxt() { //后驱节点编号
112     int cnr = ch[rt][1];

```

```

113     while (ch[cnr][0]) cnr = ch[cnr][0];
114     splay(cnr);
115     return cnr;
116 } // 若需要得到后驱 tree.ins(x), printf("%d\n", tree.val[tree.pre()]),
117 // tree.del(x);
118
119 void del(int k) { //删除k值
120     rk(k);
121     if (cnt[rt] > 1) {
122         cnt[rt]--;
123         maintain(rt);
124         return;
125     }
126     if (!ch[rt][0] && !ch[rt][1]) {
127         clear(rt);
128         rt = 0;
129         return;
130     }
131     if (!ch[rt][0]) {
132         int cnr = rt;
133         rt = ch[rt][1];
134         fa[rt] = 0;
135         clear(cnr);
136         return;
137     }
138     if (!ch[rt][1]) {
139         int cnr = rt;
140         rt = ch[rt][0];
141         fa[rt] = 0;
142         clear(cnr);
143         return;
144     }
145     int cnr = rt;
146     int x = pre();
147     splay(x);
148     fa[ch[cnr][1]] = x;
149     ch[x][1] = ch[cnr][1];
150     clear(cnr);
151     maintain(rt);
152 }
153 } tree;

```

splay2

```

1  ll ch[N][2], f[N], sum[N], val[N], tag[N], siz[N];
2
3  inline void pushup(ll p) {
4      sum[p] = sum[ch[p][0]] ^ sum[ch[p][1]] ^ val[p];
5      siz[p] = siz[ch[p][0]] + siz[ch[p][1]] + 1;
6  }
7
8  inline void pushdown(ll p) {
9      if (tag[p]) {
10         if (ch[p][0]) swap(ch[ch[p][0]][0], ch[ch[p][0]][1]), tag[ch[p][0]]
            ^= 1;

```

```

11         if (ch[p][1]) swap(ch[ch[p][1]][0], ch[ch[p][1]][1]), tag[ch[p][1]]
    ^= 1;
12         tag[p] = 0;
13     }
14 }
15
16 ll getch(ll x) { return ch[f[x]][1] == x; }
17
18 bool isroot(ll x) { return ch[f[x]][0] != x && ch[f[x]][1] != x; }
19
20 inline void rotate(ll x) {
21     ll y = f[x], z = f[y], k = getch(x);
22     if (!isroot(y)) ch[z][ch[z][1] == y] = x;
23     // 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)
24     ch[y][k] = ch[x][!k], f[ch[x][!k]] = y;
25     ch[x][!k] = y, f[y] = x, f[x] = z;
26     pushup(y), pushup(x);
27 }
28
29 // 从上到下一层一层 pushDown 即可
30 void update(ll p) {
31     if (!isroot(p)) update(f[p]);
32     pushdown(p);
33 }
34
35 inline void splay(ll x) {
36     update(x); // 马上就能看到啦。 在
37     // Splay之前要把旋转会经过的路径上的点都PushDown
38     for (ll fa; fa = f[x], !isroot(x); rotate(x)) {
39         if (!isroot(fa)) rotate(getch(fa) == getch(x) ? fa : x);
40     }
41 }
42
43 // 回顾一下代码
44 inline void access(ll x) {
45     for (ll p = 0; x; p = x, x = f[x]) {
46         splay(x), ch[x][1] = p, pushup(x);
47     }
48 }
49
50 inline void makeroot(ll p) {
51     access(p);
52     splay(p);
53     swap(ch[p][0], ch[p][1]);
54     tag[p] ^= 1;
55 }
56
57 inline void split(ll a, ll b) {
58     makeroot(a);
59     access(b);
60     splay(b);
61 }
62
63
64 inline ll find(ll p) {
65     access(p), splay(p);
66     while (ch[p][0]) pushdown(p), p = ch[p][0];
67     splay(p);

```

```

68     return p;
69 }
70
71 inline void link(ll x, ll y) {
72     makeroot(x);
73     if (find(y) != x) f[x] = y;
74 }
75
76 inline void cut(ll x, ll y) {
77     makeroot(x);
78     if (find(y) == x && f[y] == x) {
79         ch[x][1] = f[y] = 0;
80         pushup(x);
81     }
82 }
83

```

Treap

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  struct node {
4      node* ch[2];
5      int r;
6      int v;
7      int cmp(int const& a) const {
8          if (v == a) return -a;
9          return a > v ? 1 : 0;
10     }
11 };
12 void rotate(node*& a, int d) {
13     node* k = a->ch[d ^ 1];
14     a->ch[d ^ 1] = k->ch[d];
15     k->ch[d] = a;
16     a = k;
17 }
18 void insert(node*& a, int x) {
19     if (a == NULL) {
20         a = new node;
21         a->ch[0] = a->ch[1] = NULL;
22         a->v = x;
23         a->r = rand();
24     } else {
25         int d = a->cmp(x);
26         insert(a->ch[d], x);
27         if (a->ch[d]->r > a->r) rotate(a, d ^ 1);
28     }
29 }
30 void remove(node*& a, int x) {
31     int d = a->cmp(x);
32     if (d == -1) {
33         if (a->ch[0] == NULL)
34             a = a->ch[1];

```

```

35         else if (a->ch[1] == NULL)
36             a = a->ch[0];
37         else {
38             int d2 = a->ch[1]->r > a->ch[0]->r ? 0 : 1;
39             rotate(a, d2);
40             remove(a->ch[d2], x);
41         }
42     } else {
43         remove(a->ch[d], x);
44     }
45 }
46 int find(node*& a, int x) {
47     if (a == NULL)
48         return 0;
49     else if (a->v == x)
50         return 1;
51     else {
52         int d = a->cmp(x);
53         return find(a->ch[d], x);
54     }
55 }
56 int main() {
57     node* a = NULL;
58     int k, l;
59     while (cin >> k >> l) {
60         if (k == 1)
61             insert(a, l);
62         else if (k == 2)
63             remove(a, l);
64         else {
65             cout << find(a, l) << endl;
66         }
67     }
68 }

```

舞蹈链（多重覆盖）

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  struct DLX {
4      static const int maxn = 1000;    //列的上限
5      static const int maxr = 1000;    //解的上限
6      static const int maxnode = 5000; //总结点数上限
7      static const int INF = 1000000000;
8      int n, sz;
9      int S[maxn];
10
11     int row[maxnode], col[maxnode];
12     int L[maxnode], R[maxnode], U[maxnode], D[maxnode];
13
14     int ansd, ans[maxr];
15
16     int vis[maxnode];
17
18     void init(int n) {
19         this->n = n;

```



```

20
21 //虚拟节点
22 for (int i = 0; i <= n; i++) {
23     U[i] = i;
24     D[i] = i;
25     L[i] = i - 1;
26     R[i] = i + 1;
27 }
28 R[n] = 0;
29 L[0] = n;
30
31 sz = n + 1;
32 memset(S, 0, sizeof(S));
33 }
34
35 void addRow(int r, vector<int> columns) {
36     int first = sz;
37     for (int i = 0; i < columns.size(); i++) {
38         int c = columns[i];
39         L[sz] = sz - 1;
40         R[sz] = sz + 1;
41         D[sz] = c;
42         U[sz] = U[c];
43         D[U[c]] = sz;
44         U[c] = sz;
45         row[sz] = r;
46         col[sz] = c;
47         S[c]++;
48         sz++;
49     }
50     R[sz - 1] = first;
51     L[first] = sz - 1;
52 }
53 #define FOR(i, A, s) for (int i = A[s]; i != s; i = A[i])
54 void remove(int c) {
55     FOR(i, D, c) { L[R[i]] = L[i], R[L[i]] = R[i]; }
56 }
57
58 void restore(int c) {
59     FOR(i, U, c) { L[R[i]] = i, R[L[i]] = i; }
60 }
61 int f_check() //精确覆盖区估算剪枝
62 {
63     /*
64     强剪枝。这个
65     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
66     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
67     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
68     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
69     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。
70     */
71
72     int ret = 0;
73     FOR(c, R, 0) vis[c] = true;
74     FOR(c, R, 0)
75         if (vis[c]) {
76             ret++;
77             vis[c] = false;

```

```

76         FOR(i, D, c)
77             FOR(j, R, i) vis[col[j]] = false;
78     }
79     return ret;
80 }
81 // d为递归深度
82 void dfs(int d, vector<int>& v) {
83     if (d + f_check() >= ansd) return;
84     if (R[0] == 0) {
85         if (d < ansd) {
86             ansd = d;
87             v.clear();
88             for (int i = 0; i < ansd; i++) {
89                 v.push_back(ans[i]);
90             }
91         } //找到解
92         return; //记录解的长度
93     }
94
95     //找到S最小的列c
96     int c = R[0];
97     FOR(i, R, 0)
98         if (S[i] < S[c])
99             c = i; //第一个未删除的列
100                //删除第c列
101     FOR(i, D, c) { //用结点i所在的行能覆盖的所有其他列
102         ans[d] = row[i];
103         remove(i);
104         FOR(j, R, i) remove(j); //删除结点i所在的能覆的所有其他列
105         dfs(d + 1, v);
106         FOR(j, L, i) restore(j);
107         restore(i); //恢复结点i所在的行能覆盖的所有其他列
108     } //恢复第c列
109 }
110
111 bool solve(vector<int>& v) {
112     v.clear();
113     ansd = INF;
114     dfs(0, v);
115     return !v.empty();
116 }
117 };
118 //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
119 DLX dlx;
120 int main() {
121     int n, m;
122     cin >> n >> m;
123     dlx.init(m);
124     for (int i = 1; i <= n; i++) {
125         vector<int> v;
126         for (int j = 1; j <= m; j++) {
127             int a;
128             cin >> a;
129             if (a == 1) v.push_back(j);
130         }
131         dlx.addRow(i, v);
132     }

```

```

133     vector<int> ans;
134     dlx.solve(ans);
135     for (int i = 0; i < ans.size(); i++) cout << ans[i];
136 }
137

```

舞蹈链（精确覆盖）

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  struct DLX {
4      static const int maxn = 1000;    //列的上限
5      static const int maxr = 1000;    //解的上限
6      static const int maxnode = 5000; //总结点数上限
7      int n, sz;
8      int S[maxn];
9
10     int row[maxnode], col[maxnode];
11     int L[maxnode], R[maxnode], U[maxnode], D[maxnode];
12
13     int ansd, ans[maxr];
14
15     void init(int n) {
16         this->n = n;
17
18         //虚拟节点
19         for (int i = 0; i <= n; i++) {
20             U[i] = i;
21             D[i] = i;
22             L[i] = i - 1;
23             R[i] = i + 1;
24         }
25         R[n] = 0;
26         L[0] = n;
27
28         sz = n + 1;
29         memset(S, 0, sizeof(S));
30     }
31
32     void addRow(int r, vector<int> columns) {
33         int first = sz;
34         for (int i = 0; i < columns.size(); i++) {
35             int c = columns[i];
36             L[sz] = sz - 1;
37             R[sz] = sz + 1;
38             D[sz] = c;
39             U[sz] = U[c];
40             D[U[c]] = sz;
41             U[c] = sz;
42             row[sz] = r;
43             col[sz] = c;
44             S[c]++;
45             sz++;
46         }
47         R[sz - 1] = first;
48         L[first] = sz - 1;

```

```

49     }
50 #define FOR(i, A, s) for (int i = A[s]; i != s; i = A[i])
51     void remove(int c) {
52         L[R[c]] = L[c];
53         R[L[c]] = R[c];
54         FOR(i, D, c)
55             FOR(j, R, i) {
56                 U[D[j]] = U[j];
57                 D[U[j]] = D[j];
58                 --S[col[j]];
59             }
60     }
61
62     void restore(int c) {
63         FOR(i, U, c)
64             FOR(j, L, i) {
65                 ++S[col[j]];
66                 U[D[j]] = j;
67                 D[U[j]] = j;
68             }
69         L[R[c]] = c;
70         R[L[c]] = c;
71     }
72
73     // d为递归深度
74     bool dfs(int d) {
75         if (R[0] == 0) {
76             ansd = d;    //找到解
77             return true; //记录解的长度
78         }
79
80         //找到S最小的列c
81         int c = R[0];
82         FOR(i, R, 0) if (S[i] < S[c]) c = i; //第一个未删除的列
83
84         remove(c);    //删除第c列
85         FOR(i, D, c) { //用结点i所在的行能覆盖的所有其他列
86             ans[d] = row[i];
87             FOR(j, R, i) remove(col[j]); //删除结点i所在的能覆的所有其他列
88             if (dfs(d + 1)) return true;
89             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
90         }
91         restore(c); //恢复第c列
92
93         return false;
94     }
95
96     bool solve(vector<int>& v) {
97         v.clear();
98         if (!dfs(0)) return false;
99         for (int i = 0; i < ansd; i++) v.push_back(ans[i]);
100         return true;
101     }
102 };
103 //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
104

```

数论

lucas求组合数

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  typedef long long ll;
5
6  ll p;
7
8  const int maxn = 1e5 + 10;
9
10 ll qpow(ll x, ll n){
11     ll res = 1;
12     while(n){
13         if(n & 1) res = (res * x) % p;
14         x = (x * x) % p;
15         n >>= 1;
16     }
17
18     return res;
19 }
20
21 ll c(ll up, ll down){
22     if(up > down) return 0;
23     ll res = 1;
24
25     // for(int i = up + 1; i <= down; ++ i){
26     //     res = (res * i) % p;
27     // }
28     // for(int i = 1; i <= down - up; ++ i){
29     //     res = (res * qpow(i, p - 2)) % p;
30     // }
31
32     for(int i = 1, j = down; i <= up; ++ i, -- j){
33         res = (res * j) % p;
34         res = (res * qpow(i, p - 2)) % p;
35     }
36
37     return res;
38 }
39
40
41 ll lucas(ll up, ll down){
42     if(up < p && down < p) return c(up, down);
43     return c(up % p, down % p) * lucas(up / p, down / p) % p;
44 }
45
46 int main(){
47     ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
48
49     int T;
```

```

50     cin >> T;
51     while (T --){
52         ll down, up;
53         cin >> down >> up >> p;
54
55         cout << lucas(up, down) % p << endl;
56     }
57
58     return 0;
59 }

```

扩展欧几里得求逆元

```

1  typedef long long ll;
2
3  void extgcd(ll a, ll b, ll& d, ll& x, ll& y){
4      if(!b){ d=a; x=1; y=0;}
5      else{ extgcd(b, a%b, d, y, x); y-=x*(a/b); }
6  }
7
8  ll inverse(ll a, ll n){
9      ll d, x, y;
10     extgcd(a, n, d, x, y);
11     return d==1?(x+n)%n:-1;
12 }

```

逆元线性递推 inv阶乘组合数

```

1  ll fac[N]; // n!
2  ll invfac[N]; // n!的inv
3  ll invn[N]; // n的inv
4
5  inline void init() {
6      fac[0] = fac[1] = invfac[0] = invfac[1] = invn[0] = invn[1] = 1;
7      for (int i = 2; i < N; ++i) {
8          fac[i] = fac[i - 1] * i % mod;
9          invn[i] = (mod - mod / i) * invn[mod % i] % mod;
10         invfac[i] = invfac[i - 1] * invn[i] % mod;
11     }
12 }
13
14 ll C(ll up, ll down) {
15     if (up > down) return 0;
16     if (up < 0 || down < 0) return 0;
17     ll res = fac[down];
18     res = res * invfac[down - up] % mod;
19     res = res * invfac[up] % mod;
20     return res;
21 }
22
23 //先init
24

```

数学

一些范围

1 ~ n 的质数个数

$$\frac{n}{\ln n}$$

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 n^{n-2} 棵不同的树

勾股数/圆上格点数

勾股数

$$a^2 + b^2 = c^2$$

1.任何一个勾股数(a,b,c)内的三个数同时乘以一个正整数n得到的新数组(na, nb, nc)仍然是勾股数,

于是找abc互质的勾股数

一, 当a为大于1的奇数 $2n+1$ 时, $b = 2n^2 + 2n$, $c = 2n^2 + 2n + 1$

(把a拆成两个连续的自然数)

二, 当a为大于4的偶数 $2n$ 时, $b = n^2 - 1$, $c = n^2 + 1$

(只想得到互质的数的话: $a=4n$, $b = 4n^2 - 1$, $c = 4n^2 + 1$)

公式1

$$a=2mnt$$

$$b= (m^2-n^2) t$$

$$c= (m^2+n^2) t$$

(t是倍数)

完全公式

$$a=m, b=(m^2/k-k)/2, c=(m^2/k+k)/2 \text{ ①}$$

其中 $m \geq 3$

1 当m确定为任意一个 ≥ 3 的奇数时, $k=\{1, m^2 \text{的所有小于} m \text{的因子}\}$

2 当m确定为任意一个 ≥ 4 的偶数时, $k=\{m^2/2 \text{的所有小于} m \text{的偶数因子}\}$

高斯整数/高斯素数

二维平面转化为复数平面,

$4n+1$ 的素数, 都能分解成高斯素数, $4n+3$ 的素数, 他们本身就是高斯素数, 2特殊

(乘以1, -1, i, -i 四个

半径为 \sqrt{n} 的圆上的格点数, 先将n分解质因数, 对每个不是高斯素数的数分解成共轭的高斯素数, 分配数比指数多1, 指数是偶数的话, 有一种方法分配, 不然就没有格点

$2 = (1 + i)(1 - i)$ ，但是这对数格点数没有影响，因为要乘 $-i$ 。

$$\text{引入 } f(x) = \begin{cases} 1, x \text{ 为素数 } x = 4n + 1 \\ -1, x \text{ 为素数 } x = 4n + 3 \\ 0, x \text{ 为偶数} \end{cases}$$

它是一个周期函数，同时是一个积性函数，

再来看这个问题，

$$\begin{aligned} 45 &= 3^2 \times 5 \\ \text{半径为 } \sqrt{45} \text{ 圆上格点数问题} &= 4 \times (f(1) + f(3) + f(3^2)) \times (f(1) + f(5)) \\ &= 4 \times (f(1) + f(3) + f(5) + f(9) + f(15) + f(45)) \end{aligned}$$

最后转化为45的所有约数

$$f(x) = \begin{cases} 1, x \text{ 为素数 } x = 4n + 1 \\ -1, x \text{ 为素数 } x = 4n + 3 \\ 0, x \text{ 为偶数} \end{cases}$$

$$\text{半径为 } \sqrt{n} \text{ 的圆上的格点数 (二维坐标轴 } xy \text{ 都为整数的点) 是 } 4 \times \sum_{d|n} f(d)$$

exgcd

```
1  ll ex_gcd(ll a, ll b, ll &x, ll &y) {
2      if (b == 0) {
3          x = 1;
4          y = 0;
5          return a;
6      }
7      ll d = ex_gcd(b, a % b, x, y);
8      ll temp = x;
9      x = y;
10     y = temp - a / b * y;
11     return d;
12 }
13
```

Pollard_Rho+Miller-Rabin

```
1  typedef long long ll;
2  namespace Miller_Rabin {
3      const ll Pcnt = 12;
4      const ll p[Pcnt] = {2, 3, 5, 7, 11, 13, 17, 19, 61, 2333, 4567, 24251};
5
6      ll pow(ll a, ll b, ll p) {
7          ll ans = 1;
8          for (; b; a = (__int128) a * a % p, b >>= 1) if (b & 1) ans =
9              (__int128) ans * a % p;
10         return ans;
11     }
```



```

12 bool check(ll x, ll p) {
13     if (x % p == 0 || pow(p % x, x - 1, x) ^ 1) return true;
14     ll t, k = x - 1;
15     while ((k ^ 1) & 1) {
16         t = pow(p % x, k >= 1, x);
17         if (t ^ 1 && t ^ x - 1) return true;
18         if (!(t ^ x - 1)) return false;
19     }
20     return false;
21 }
22
23 inline bool MR(ll x) { //用这个
24     if (x < 2) return false;
25     for (int i = 0; i ^ Pcnt; ++i) {
26         if (!(x ^ p[i])) return true;
27         if (check(x, p[i])) return false;
28     }
29     return true;
30 }
31 }
32 namespace Pollard_Rho {
33 #define Rand(x) (ll)rand()*rand()%(x)+1
34
35 ll gcd(const ll a, const ll b) { return b ? gcd(b, a % b) : a; }
36
37 ll mul(const ll x, const ll y, const ll x) {
38     ll k = (1.0L * x * y) / (1.0L * x) - 1, t = (__int128) x * y -
39     (__int128) k * x;
40     while (t < 0) t += x;
41     return t;
42 }
43
44 ll PR(const ll x, const ll y) {
45     int t = 0, k = 1;
46     ll v0 = Rand(x - 1), v = v0, d, s = 1;
47     while (true) {
48         v = (mul(v, v, x) + y) % x, s = mul(s, abs(v - v0), x);
49         if (!(v ^ v0) || !s) return x;
50         if (++t == k) {
51             if ((d = gcd(s, x)) ^ 1) return d;
52             v0 = v, k <= 1;
53         }
54     }
55 }
56
57 void Resolve(ll x, ll &ans) {
58     if (!(x ^ 1) || x <= ans) return;
59     if (Miller_Rabin::MR(x)) {
60         if (ans < x) ans = x;
61         return;
62     }
63     ll y = x;
64     while ((y = PR(x, Rand(x))) == x);
65     while (!(x % y)) x /= y;
66     Resolve(x, ans);
67     Resolve(y, ans);
68 }

```

```

69     long long check(ll x) { //用这个，素数返回本身
70         ll ans = 0;
71         Resolve(x, ans);
72         return ans;
73     }
74 }
75

```

FFT

```

1  #include <iostream>
2  #include <cstring>
3  #include <algorithm>
4  #include <cmath>
5
6  using namespace std;
7
8  const int N = 300010;
9  const double PI = acos(-1);
10
11 int n, m;
12 struct Complex
13 {
14     double x, y;
15     Complex operator+ (const Complex& t) const
16     {
17         return {x + t.x, y + t.y};
18     }
19     Complex operator- (const Complex& t) const
20     {
21         return {x - t.x, y - t.y};
22     }
23     Complex operator* (const Complex& t) const
24     {
25         return {x * t.x - y * t.y, x * t.y + y * t.x};
26     }
27 }a[N], b[N];
28 int rev[N], bit, tot;
29
30 void fft(Complex a[], int inv)
31 {
32     for (int i = 0; i < tot; i++)
33         if (i < rev[i])
34             swap(a[i], a[rev[i]]);
35     for (int mid = 1; mid < tot; mid <= 1)
36     {
37         auto w1 = Complex({cos(PI / mid), inv * sin(PI / mid)});
38         for (int i = 0; i < tot; i += mid * 2)
39         {
40             auto wk = Complex({1, 0});
41             for (int j = 0; j < mid; j++, wk = wk * w1)
42             {
43                 auto x = a[i + j], y = wk * a[i + j + mid];

```

```

44         a[i + j] = x + y, a[i + j + mid] = x - y;
45     }
46 }
47 }
48 }
49
50 int main()
51 {
52     scanf("%d%d", &n, &m);
53     for (int i = 0; i <= n; i ++ ) scanf("%lf", &a[i].x);
54     for (int i = 0; i <= m; i ++ ) scanf("%lf", &b[i].x);
55     while ((1 << bit) < n + m + 1) bit ++;
56     tot = 1 << bit;
57     for (int i = 0; i < tot; i ++ )
58         rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (bit - 1));
59     fft(a, 1), fft(b, 1);
60     for (int i = 0; i < tot; i ++ ) a[i] = a[i] * b[i];
61     fft(a, -1);
62     for (int i = 0; i <= n + m; i ++ )
63         printf("%d ", (int)(a[i].x / tot + 0.5));
64
65     return 0;
66 }
67
68 作者: yxc
69 链接: https://www.acwing.com/activity/content/code/content/664840/
70 来源: AcWing
71 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
72

```

BSGS

求 $a^t \equiv b \pmod{p}$ ($a, p = 1$ 的最小的 t)

$t = x \times k - y, x \in [1, k], y \in [0, k - 1]$

$t \in [1, k^2]$

$a^k x \equiv b \times a^y \pmod{p}$

对 $b \times a^y$ 建立hash表, 枚举 x 看是否有解

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  typedef long long ll;
5
6  unordered_map<int, int> mp;
7
8  int bsgs(int a, int p, int b) {
9
10     if (1 % p == b % p) return 0; // 特判0是不是解
11     mp.clear();
12
13     int k = sqrt(p) + 1;

```

```

14
15     for(int i = 0, j = b % p; i < k; ++ i, j = (ll)j * a % p) {
16         mp[j] = i;
17     }
18
19     int ak = 1;
20     for(int i = 0; i < k; ++i) {
21         ak = (ll)ak * a % p;
22     }
23
24     for(int i = 1, j = ak % p; i <= k; ++ i, j = (ll)j * ak % p) {
25         if(mp.count(j)) return (ll)i * k - mp[j];
26     }
27
28     return -1;
29 }
30
31 int main() {
32     ios::sync_with_stdio(0);
33     cin.tie(0); cout.tie(0);
34
35     int a, p, b;
36     while(cin >> a >> p >> b, a | p | b) {
37         int res;
38         res = bsgs(a, p, b);
39         if(res == -1) {
40             cout << "No solution\n";
41         }
42         else {
43             cout << res << endl;
44         }
45     }
46
47     return 0;
48 }

```

扩展BSGS

求 $a^t \equiv b \pmod{p}$ 的最小的 t

当 $(a, p) \neq 1$

$(a, p) = d \nmid b$ 无解

$a^t \equiv b \pmod{p}$, $a^t + kp = b$ 两边同时除以 d , $\frac{a}{d}a^{t-1} + k\frac{p}{d} = \frac{b}{d}$

$a^{t-1} \equiv \frac{b}{d}(\frac{a}{d})^{-1}$

$t' = t - 1, p' = \frac{p}{d}, b' = \frac{b}{d}(\frac{a}{d})^{-1}$

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  typedef long long ll;
5
6  unordered_map<ll, ll> mp;
7
8  ll bsgs(ll a, ll p, ll b) {

```

```

9
10     if(1 % p == b % p) return 0; // 特判0是不是解
11     mp.clear();
12
13     ll k = sqrt(p) + 1;
14
15     for(ll i = 0, j = b % p; i < k; ++i, j = (ll)j * a % p) {
16         mp[j] = i;
17     }
18
19     ll ak = 1;
20     for(ll i = 0; i < k; ++i) {
21         ak = (ll) ak * a % p;
22     }
23
24     for(ll i = 1, j = ak % p; i <= k; ++i, j = (ll)j * ak % p) {
25         if(mp.count(j)) return (ll) i * k - mp[j];
26     }
27
28     return -1;
29 }
30
31 ll gcd(ll x, ll y) {
32     return x % y == 0 ? y : gcd(y, x % y);
33 }
34
35 void extgcd(ll a, ll b, ll& d, ll& x, ll& y) {
36     if(!b) {
37         d = a; x = 1; y = 0;
38     }
39     else {
40         extgcd(b, a % b, d, y, x);
41         y -= x * (a / b);
42     }
43 }
44
45 ll inverse(ll a, ll n) {
46     ll d, x, y;
47     extgcd(a, n, d, x, y);
48     return d == 1 ? (x + n) % n : -1;
49 }
50
51
52 int main() {
53     ll a, p, b;
54
55     while(cin >> a >> p >> b, a | p | b) {
56         ll d = gcd(a, p);
57         if(d == 1) {
58             ll res = bsgs(a, p, b);
59             if(res == -1) {
60                 cout << "No Solution\n";
61             }
62             else {
63                 cout << res << endl;
64             }
65         }
66         else {

```

```

67         if(b % d != 0) {
68             cout << "No Solution\n";
69             continue;
70         }
71         else {
72             p = p / d;
73             b = (b / d) * inverse(a / d, p);
74             ll res = bsgs(a, p, b);
75             if(res == -1) {
76                 cout << "No Solution\n";
77             }
78             else {
79                 cout << res + 1 << endl;
80             }
81         }
82     }
83 }
84
85 return 0;
86 }

```

二次剩余

解的数量

对于 $x^2 \equiv n \pmod{p}$ 能满足 n 是 $\text{mod } p$ 的二次剩余的 n 一共有 $\frac{p-1}{2}$ 个 (不包括 0), 非二次剩余为 $\frac{p-1}{2}$ 个

勒让德符号

$$\left(\frac{n}{p}\right) = \begin{cases} 1, p \nmid n, n \text{ 是 } p \text{ 的二次剩余} \\ -1, p \nmid n, n \text{ 不是 } p \text{ 的二次剩余} \\ 0, p \mid n \end{cases}$$

欧拉判别准则

$$\left(\frac{n}{p}\right) \equiv n^{\frac{p-1}{2}} \pmod{p}$$

若 n 是二次剩余, 当且仅当 $n^{\frac{p-1}{2}} \equiv 1 \pmod{p}$

若 n 是非二次剩余, 当且仅当 $n^{\frac{p-1}{2}} \equiv -1 \pmod{p}$

Cipolla

找到一个数 a 满足 $a^2 - n$ 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 $\frac{p-1}{2}$, 接近 $\frac{p}{2}$, 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 $i^2 = -1$, 这里定义 $i^2 = a^2 - n$, 于是就可以将所有的数表达为 $A + Bi$ 的形式, 这里的 A 和 B 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, $x^2 \equiv n \pmod{p}$ 的解为 $(a + i)^{\frac{p+1}{2}}$ 。

```

1  #include <bits/stdc++.h>
2  using namespace std;
3

```

```

4  typedef long long ll;
5  int t;
6  ll n, p;
7  ll w;
8
9  struct num {    //建立一个复数域
10
11      ll x, y;
12  };
13
14  num mul(num a, num b, ll p) {    //复数乘法
15      num ans = {0, 0};
16      ans.x = ((a.x * b.x % p + a.y * b.y % p * w % p) % p + p) % p;
17      ans.y = ((a.x * b.y % p + a.y * b.x % p) % p + p) % p;
18      return ans;
19  }
20
21  ll binpow_real(ll a, ll b, ll p) {    //实部快速幂
22      ll ans = 1;
23      while (b) {
24          if (b & 1) ans = ans * a % p;
25          a = a * a % p;
26          b >>= 1;
27      }
28      return ans % p;
29  }
30
31  ll binpow_imag(num a, ll b, ll p) {    //虚部快速幂
32      num ans = {1, 0};
33      while (b) {
34          if (b & 1) ans = mul(ans, a, p);
35          a = mul(a, a, p);
36          b >>= 1;
37      }
38      return ans.x % p;
39  }
40
41  ll cipolla(ll n, ll p) {
42      n %= p;
43      if (p == 2) return n;
44      if (binpow_real(n, (p - 1) / 2, p) == p - 1) return -1;
45      ll a;
46      while (1) {    //生成随机数再检验找到满足非二次剩余的a
47          a = rand() % p;
48          w = ((a * a % p - n) % p + p) % p;
49          if (binpow_real(w, (p - 1) / 2, p) == p - 1) break;
50      }
51      num x = {a, 1};
52      return binpow_imag(x, (p + 1) / 2, p);
53  }

```

卡特兰数

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

$$C_n = \frac{1}{n+1} C_{2n}^n = C_{2n}^n - C_{2n}^{n-1}$$

$$C_n = \frac{1}{n+1} \sum_{i=0}^n (C_n^i)^2$$

$$C_n = \frac{4n-2}{n+1} C_{n-1} (C_0 = 1)$$

$$C_{n+1} = \sum_{i=0}^n C_i C_{n-i} (C_0 = 1)$$

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

$$F_n * (n + 1) = (6 * n - 3) * F_{n-1} - (n - 2) * F_{n-2}$$

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

```
1  ll qpow(ll a, ll b) {
2      ll ans = 1;
3      while (b) {
4          if (b & 1) ans = (ans * a) % mod;
5          a = (a * a) % mod;
6          b >>= 1;
7      }
8      return ans;
9  }
```

龟速乘快速幂（快速幂爆longlong

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  typedef long long ll;
5
6  ll qmul(ll a, ll b, ll p) {
7      ll res = 0;
8      while(b) {
9          if(b & 1) res = (res + a) % p;
10         a = (a + a) % p;
11         b >>= 1;
12     }
13     return res;
14 }
15
16 ll qpow(ll x, ll n, ll p) {
17     ll res = 1;
18     while(n) {
19         if(n & 1) res = qmul(res, x, p);
```



```

20     x = qmul(x, x, p);
21     n >>= 1;
22 }
23 return res % p; // 1 0 1
24 }
25
26 int main() {
27     ll b, p, k;
28     cin >> b >> p >> k;
29     ll ans = qpow(b, p, k);
30     printf("%lld^%lld mod %lld=%lld", b, p, k, ans);
31
32     return 0;
33 }

```

莫比乌斯反演

莫比乌斯函数

对 n 进行因数分解: $n = P_1^{\alpha_1} P_2^{\alpha_2} \dots P_k^{\alpha_k}$, 则 $\mu(n) = \begin{cases} 1, & n = 1 \\ 0, & \forall \alpha_i \geq 2 \\ \pm 1, & (-1)^k \end{cases}$

n 的所有约数的莫比乌斯的和

$$S(n) = \sum_{d|n} \mu(d) = \begin{cases} 1, & n = 1 \\ 0, & else \end{cases}$$

反演

(一般不用) 1. 若 $F(n) = \sum_{d|n} f(d)$, 则 $f(n) = \sum_{d|n} \mu(d) F(\frac{n}{d})$

(√) 2. 若 $F(n) = \sum_{n|d} f(d)$, 则 $f(n) = \sum_{n|d} \mu(\frac{d}{n}) F(d)$

构造 $F(n)$ 和 $f(n)$ 使 $f(n)$ 为目标, $F(n)$ 好求

1

求满足 $a \leq x \leq b, c \leq y \leq d$ 且 $\gcd(x, y) = k$ 的 xy 的对数

$F(n) = \gcd(x, y) = n$ 的倍数的 xy 的对数

$f(n) = \gcd(x, y) = n$ 的 xy 的对数

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  typedef long long ll;
5
6  const int N = 50010;
7

```

```

8  ll primes[N], mu[N], sum[N], cnt;
9  bool st[N];
10
11 void init() {
12     mu[1] = 1;
13
14     for(int i = 2; i < N; ++ i) {
15         if(!st[i]) {
16             primes[cnt ++] = i;
17             mu[i] = -1;
18         }
19
20         for(int j = 0; primes[j] * i < N; ++ j) {
21             st[primes[j] * i] = 1;
22             if(i % primes[j] == 0) break;
23             mu[primes[j] * i] = -mu[i];
24         }
25     }
26
27     for(int i = 1; i < N; ++ i) {
28         sum[i] = sum[i - 1] + mu[i];
29     }
30 }
31
32 ll g(ll n, ll x) {
33     return n / (n / x);
34 }
35
36 ll f (int a, int b, int k) {
37     a = a / k, b = b / k;
38
39     ll res = 0;
40
41     ll n = min(a, b);
42
43     for(ll l = 1, r; l <= n; l = r + 1) {
44         r = min(n, min(g(a, l), g(b, l)));
45         res += (sum[r] - sum[l - 1]) * (a / l) * (b / l);
46     }
47
48     return res;
49 }
50
51 int main() {
52     ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
53
54     init();
55
56     int T;
57     cin >> T;
58     while(T --) {
59         int a, b, c, d, k;
60         cin >> a >> b >> c >> d >> k;
61         cout << f(b, d, k) - f(a - 1, d, k) - f(b, c - 1, k)
62             + f(a - 1, c - 1, k) << endl;
63     }
64
65     return 0;

```

2

求 $\sum_{i=1}^N \sum_{j=1}^M d(ij)$

// $d(ij) = \sum_{x|i} \sum_{y|j} [(x, y) = 1]$

$F(n) = \sum_{i=1}^N \sum_{j=1}^M \sum_{x|i} \sum_{y|j} [n|(x, y)]$

$f(n) = \sum_{i=1}^N \sum_{j=1}^M \sum_{x|i} \sum_{y|j} [(x, y) = n]$

$F(n) = \sum_{i=1}^N \sum_{j=1}^M \sum_{x|i} \sum_{y|j} [n|(x, y)] = \sum_{x=1}^N \sum_{y=1}^M \lfloor \frac{N}{x} \rfloor \lfloor \frac{M}{y} \rfloor [n|(x, y)] = \sum_{x'=1}^{\frac{N}{n}} \sum_{y'=1}^{\frac{M}{n}} \lfloor \frac{N}{x'n} \rfloor \lfloor \frac{M}{y'n} \rfloor$

两次整数分块

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  typedef long long ll;
5  const int N = 50010;
6
7  int primes[N], cnt, mu[N], sum[N], h[N];
8  bool st[N];
9
10 inline int g(int n, int x) {
11     return n / (n / x);
12 }
13
14 void init() {
15     mu[1] = 1;
16     for(int i = 2; i < N; ++i) {
17         if(!st[i]){
18             primes[cnt++] = i;
19             mu[i] = -1;
20         }
21         for(int j = 0; primes[j] * i < N; ++j) {
22             st[primes[j] * i] = 1;
23             if(i % primes[j] == 0) break;
24             mu[primes[j] * i] = -mu[i];
25         }
26     }
27
28     for(int i = 1; i < N; ++i) {
29         sum[i] = sum[i - 1] + mu[i];
30     }
31
32     for(int i = 1; i < N; ++i) {
33         for(int l = 1, r; l <= i; l = r + 1) {
34             r = min(i, g(i, l));
35             h[i] += (r - l + 1) * (i / l);
36         }
37     }
38 }
39
40 int main() {
41
42     int main() {

```

```

43 //ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
44 init();
45
46 int T;
47 scanf("%d", &T);
48 while(T--) {
49     int n, m;
50     scanf("%d %d", &n, &m);
51     ll res = 0;
52     int k = min(n, m);
53     for(int l = 1, r; l <= k; l = r + 1) {
54         r = min(k, min(g(n, l), g(m, l)));
55         res += (ll)(sum[r] - sum[l - 1]) * h[n / l] * h[m / l];
56     }
57     printf("%lld\n", res);
58 }
59
60 return 0;
61 }

```

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 $\text{mex}\{0,1,2,4\}=3$ 、 $\text{mex}\{2,3,5\}=0$ 、 $\text{mex}\{\}=0$ 。

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 $\text{SG}(x) = x$ 。

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， $1 \leq k \leq n$ ，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， $3^5 \rightarrow 011+101=112$ ，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 $\neq 0$ 。

高精度GCD

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  string add(string a, string b) {
4      const int L = 1e5;
5      string ans;
6      int na[L] = {0}, nb[L] = {0};
7      int la = a.size(), lb = b.size();
8      for (int i = 0; i < la; i++) na[la - 1 - i] = a[i] - '0';
9      for (int i = 0; i < lb; i++) nb[lb - 1 - i] = b[i] - '0';
10     int lmax = la > lb ? la : lb;
11     for (int i = 0; i < lmax; i++)
12         na[i] += nb[i], na[i + 1] += na[i] / 10, na[i] %= 10;
13     if (na[lmax]) lmax++;
14     for (int i = lmax - 1; i >= 0; i--) ans += na[i] + '0';
15     return ans;
16 }
17 string mul(string a, string b) {
18     const int L = 1e5;
19     string s;
20     int na[L], nb[L], nc[L],
21         La = a.size(), Lb = b.size(); // na存储被乘数, nb存储乘数, nc存储积
22     fill(na, na + L, 0);
23     fill(nb, nb + L, 0);
24     fill(nc, nc + L, 0); //将na,nb,nc都置为0
25     for (int i = La - 1; i >= 0; i--)
26         na[La - i] =
27             a[i] - '0'; //将字符串表示的大整数数组转成i整形数组表示的大整数数组
28     for (int i = Lb - 1; i >= 0; i--) nb[Lb - i] = b[i] - '0';
29     for (int i = 1; i <= La; i++)
30         for (int j = 1; j <= Lb; j++)
31             nc[i + j - 1] +=
32                 na[i] *
33                 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）
34     for (int i = 1; i <= La + Lb; i++)
35         nc[i + 1] += nc[i] / 10, nc[i] %= 10; //统一处理进位
36     if (nc[La + Lb]) s += nc[La + Lb] + '0'; //判断第i+j位上的数字是不是0
37     for (int i = La + Lb - 1; i >= 1; i--)
38         s += nc[i] + '0'; //将整形数组转成字符串
39     return s;
40 }
41 int sub(int *a, int *b, int La, int Lb) {
42     if (La < Lb) return -1; //如果a小于b, 则返回-1
43     if (La == Lb) {
44         for (int i = La - 1; i >= 0; i--)
45             if (a[i] > b[i])
46                 break;
47             else if (a[i] < b[i])
48                 return -1; //如果a小于b, 则返回-1
49     }
50     for (int i = 0; i < La; i++) //高精度减法
51     {
52         a[i] -= b[i];
53         if (a[i] < 0) a[i] += 10, a[i + 1]--;
```

```

54     }
55     for (int i = La - 1; i >= 0; i--)
56         if (a[i]) return i + 1; //返回差的位数
57     return 0; //返回差的位数
58 }
59 string div(string n1, string n2,
60           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
61 {
62     const int L = 1e5;
63     string s, v; // s存商,v存余数
64     int a[L], b[L], r[L],
65         La = n1.size(), Lb = n2.size(), i,
66         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
67     fill(a, a + L, 0);
68     fill(b, b + L, 0);
69     fill(r, r + L, 0); //数组元素都置为0
70     for (i = La - 1; i >= 0; i--) a[La - 1 - i] = n1[i] - '0';
71     for (i = Lb - 1; i >= 0; i--) b[Lb - 1 - i] = n2[i] - '0';
72     if (La < Lb || (La == Lb && n1 < n2)) {
73         // cout<<0<<endl;
74         return n1;
75     } //如果a<b,则商为0, 余数为被除数
76     int t = La - Lb; //除被数和除数的位数之差
77     for (int i = La - 1; i >= 0; i--) //将除数扩大10^t倍
78         if (i >= t)
79             b[i] = b[i - t];
80         else
81             b[i] = 0;
82     Lb = La;
83     for (int j = 0; j <= t; j++) {
84         int temp;
85         while ((temp = sub(a, b + j, La, Lb - j)) >=
86             0) //如果被除数比除数大继续减
87         {
88             La = temp;
89             r[t - j]++;
90         }
91     }
92     for (i = 0; i < L - 10; i++)
93         r[i + 1] += r[i] / 10, r[i] %= 10; //统一处理进位
94     while (!r[i]) i--; //将整形数组表示的商转化成字符串表示的
95     while (i >= 0) s += r[i--] + '0';
96     // cout<<s<<endl;
97     i = tp;
98     while (!a[i]) i--; //将整形数组表示的余数转化成字符串表示的
99     while (i >= 0) v += a[i--] + '0';
100    if (v.empty()) v = "0";
101    // cout<<v<<endl;
102    if (nn == 1) return s;
103    if (nn == 2) return v;
104 }
105 bool judge(string s) //判断s是否为全0串
106 {
107     for (int i = 0; i < s.size(); i++)
108         if (s[i] != '0') return false;
109     return true;
110 }
111 string gcd(string a, string b) //求最大公约数

```

```

112 {
113     string t;
114     while (!judge(b)) //如果余数不为0，继续除
115     {
116         t = a; //保存被除数的值
117         a = b; //用除数替换被除数
118         b = div(t, b, 2); //用余数替换除数
119     }
120     return a;
121 }
122
123 //o(无法估计)
124

```

高精度乘法 (FFT)

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  #define L(x) (1 << (x))
4  const double PI = acos(-1.0);
5  const int Maxn = 133015;
6  double ax[Maxn], ay[Maxn], bx[Maxn], by[Maxn];
7  char sa[Maxn / 2], sb[Maxn / 2];
8  int sum[Maxn];
9  int x1[Maxn], x2[Maxn];
10 int revv(int x, int bits) {
11     int ret = 0;
12     for (int i = 0; i < bits; i++) {
13         ret <<= 1;
14         ret |= x & 1;
15         x >>= 1;
16     }
17     return ret;
18 }
19 void fft(double* a, double* b, int n, bool rev) {
20     int bits = 0;
21     while (1 << bits < n) ++bits;
22     for (int i = 0; i < n; i++) {
23         int j = revv(i, bits);
24         if (i < j) swap(a[i], a[j]), swap(b[i], b[j]);
25     }
26     for (int len = 2; len <= n; len <<= 1) {
27         int half = len >> 1;
28         double wmx = cos(2 * PI / len), wmy = sin(2 * PI / len);
29         if (rev) wmy = -wmy;
30         for (int i = 0; i < n; i += len) {
31             double wx = 1, wy = 0;
32             for (int j = 0; j < half; j++) {
33                 double cx = a[i + j], cy = b[i + j];
34                 double dx = a[i + j + half], dy = b[i + j + half];
35                 double ex = dx * wx - dy * wy, ey = dx * wy + dy * wx;
36                 a[i + j] = cx + ex, b[i + j] = cy + ey;
37                 a[i + j + half] = cx - ex, b[i + j + half] = cy - ey;
38                 double wnx = wx * wmx - wy * wmy, wny = wx * wmy + wy *
39                 wx = wnx, wy = wny;
40             }
41         }
42     }
43 }

```

```

41     }
42 }
43 if (rev) {
44     for (int i = 0; i < n; i++) a[i] /= n, b[i] /= n;
45 }
46 }
47 int solve(int a[], int na, int b[], int nb, int ans[]) {
48     int len = max(na, nb), ln;
49     for (ln = 0; L(ln) < len; ++ln)
50         ;
51     len = L(++ln);
52     for (int i = 0; i < len; ++i) {
53         if (i >= na)
54             ax[i] = 0, ay[i] = 0;
55         else
56             ax[i] = a[i], ay[i] = 0;
57     }
58     fft(ax, ay, len, 0);
59     for (int i = 0; i < len; ++i) {
60         if (i >= nb)
61             bx[i] = 0, by[i] = 0;
62         else
63             bx[i] = b[i], by[i] = 0;
64     }
65     fft(bx, by, len, 0);
66     for (int i = 0; i < len; ++i) {
67         double cx = ax[i] * bx[i] - ay[i] * by[i];
68         double cy = ax[i] * by[i] + ay[i] * bx[i];
69         ax[i] = cx, ay[i] = cy;
70     }
71     fft(ax, ay, len, 1);
72     for (int i = 0; i < len; ++i) ans[i] = (int)(ax[i] + 0.5);
73     return len;
74 }
75 string mul(string sa, string sb) {
76     int l1, l2, l;
77     int i;
78     string ans;
79     memset(sum, 0, sizeof(sum));
80     l1 = sa.size();
81     l2 = sb.size();
82     for (i = 0; i < l1; i++) x1[i] = sa[l1 - i - 1] - '0';
83     for (i = 0; i < l2; i++) x2[i] = sb[l2 - i - 1] - '0';
84     l = solve(x1, l1, x2, l2, sum);
85     for (i = 0; i < l || sum[i] >= 10; i++) // 进位
86     {
87         sum[i + 1] += sum[i] / 10;
88         sum[i] %= 10;
89     }
90     l = i;
91     while (sum[l] <= 0 && l > 0) l--; // 检索最高位
92     for (i = l; i >= 0; i--) ans += sum[i] + '0'; // 倒序输出
93     return ans;
94 }
95 int main() {
96     cin.sync_with_stdio(false);
97     string a, b;
98     while (cin >> a >> b) cout << mul(a, b) << endl;

```



```

99     return 0;
100 }
101
102 //o(nlogn)

```

高精度乘法（乘单精度）

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  string mul(string a, int b) //高精度a乘单精度b
4  {
5      const int L = 100005;
6      int na[L];
7      string ans;
8      int La = a.size();
9      fill(na, na + L, 0);
10     for (int i = La - 1; i >= 0; i--) na[La - i - 1] = a[i] - '0';
11     int w = 0;
12     for (int i = 0; i < La; i++)
13         na[i] = na[i] * b + w, w = na[i] / 10, na[i] = na[i] % 10;
14     while (w) na[La++] = w % 10, w /= 10;
15     La--;
16     while (La >= 0) ans += na[La--] + '0';
17     return ans;
18 }
19
20 //o(n)

```

高精度乘法（朴素）

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  string mul(string a, string b) //高精度乘法a,b,均为非负整数
4  {
5      const int L = 1e5;
6      string s;
7      int na[L], nb[L], nc[L],
8          La = a.size(), Lb = b.size(); // na存储被乘数, nb存储乘数, nc存储积
9      fill(na, na + L, 0);
10     fill(nb, nb + L, 0);
11     fill(nc, nc + L, 0); //将na,nb,nc都置为0
12     for (int i = La - 1; i >= 0; i--)
13         na[La - i] =
14             a[i] - '0'; //将字符串表示的大整数转成i整形数组表示的大整数
15     for (int i = Lb - 1; i >= 0; i--) nb[Lb - i] = b[i] - '0';
16     for (int i = 1; i <= La; i++)
17         for (int j = 1; j <= Lb; j++)
18             nc[i + j - 1] +=
19                 na[i] *
20                 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）
21     for (int i = 1; i <= La + Lb; i++)
22         nc[i + 1] += nc[i] / 10, nc[i] %= 10; //统一处理进位
23     if (nc[La + Lb]) s += nc[La + Lb] + '0'; //判断第i+j位上的数字是不是0
24     for (int i = La + Lb - 1; i >= 1; i--)
25         s += nc[i] + '0'; //将整形数组转成字符串
26     return s;

```

```

27 }
28
29 //o(n^2)

```

高精度除法（除单精度）

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  string div(string a, int b) //高精度a除以单精度b
4  {
5      string r, ans;
6      int d = 0;
7      if (a == "0") return a; //特判
8      for (int i = 0; i < a.size(); i++) {
9          r += (d * 10 + a[i] - '0') / b + '0'; //求出商
10         d = (d * 10 + (a[i] - '0')) % b; //求出余数
11     }
12     int p = 0;
13     for (int i = 0; i < r.size(); i++)
14         if (r[i] != '0') {
15             p = i;
16             break;
17         }
18     return r.substr(p);
19 }
20
21 //o(n)
22

```

高精度除法（除高精度）

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  int sub(int *a, int *b, int La, int Lb) {
4      if (La < Lb) return -1; //如果a小于b，则返回-1
5      if (La == Lb) {
6          for (int i = La - 1; i >= 0; i--)
7              if (a[i] > b[i])
8                  break;
9              else if (a[i] < b[i])
10                 return -1; //如果a小于b，则返回-1
11     }
12     for (int i = 0; i < La; i++) //高精度减法
13     {
14         a[i] -= b[i];
15         if (a[i] < 0) a[i] += 10, a[i + 1]--;
16     }
17     for (int i = La - 1; i >= 0; i--)
18         if (a[i]) return i + 1; //返回差的位数
19     return 0; //返回差的位数
20 }
21 string div(string n1, string n2, int nn)
22 // 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
23 {

```

```

24     const int L = 1e5;
25     string s, v; // s存商,v存余数
26     int a[L], b[L], r[L], La = n1.size(), Lb = n2.size(), i, tp = La;
27     // a, b是整形数组表示被除数, 除数, tp保存被除数的长度
28     fill(a, a + L, 0);
29     fill(b, b + L, 0);
30     fill(r, r + L, 0); //数组元素都置为0
31     for (i = La - 1; i >= 0; i--) a[La - 1 - i] = n1[i] - '0';
32     for (i = Lb - 1; i >= 0; i--) b[Lb - 1 - i] = n2[i] - '0';
33     if (La < Lb || (La == Lb && n1 < n2)) {
34         // cout<<0<<endl;
35         return n1;
36     } //如果a<b,则商为0, 余数为被除数
37     int t = La - Lb; //除被数和除数的位数之差
38     for (int i = La - 1; i >= 0; i--) //将除数扩大10^t倍
39         if (i >= t)
40             b[i] = b[i - t];
41         else
42             b[i] = 0;
43     Lb = La;
44     for (int j = 0; j <= t; j++) {
45         int temp;
46         while ((temp = sub(a, b + j, La, Lb - j)) >=
47             0) //如果被除数比除数大继续减
48         {
49             La = temp;
50             r[t - j]++;
51         }
52     }
53     for (i = 0; i < L - 10; i++)
54         r[i + 1] += r[i] / 10, r[i] %= 10; //统一处理进位
55     while (!r[i]) i--; //将整形数组表示的商转化成字符串表示的
56     while (i >= 0) s += r[i--] + '0';
57     // cout<<s<<endl;
58     i = tp;
59     while (!a[i]) i--; //将整形数组表示的余数转化成字符串表示的
60     while (i >= 0) v += a[i--] + '0';
61     if (v.empty()) v = "0";
62     // cout<<v<<endl;
63     if (nn == 1) return s; //返回商
64     if (nn == 2) return v; //返回余数
65 }
66
67 //o(n^2)
68

```

高精度加法

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  string add(string a, string b) //只限两个非负整数相加
4  {
5      const int L = 1e5;
6      string ans;
7      int na[L] = {0}, nb[L] = {0};
8      int la = a.size(), lb = b.size();

```

```

9     for (int i = 0; i < la; i++) na[la - 1 - i] = a[i] - '0';
10    for (int i = 0; i < lb; i++) nb[lb - 1 - i] = b[i] - '0';
11    int lmax = la > lb ? la : lb;
12    for (int i = 0; i < lmax; i++)
13        na[i] += nb[i], na[i + 1] += na[i] / 10, na[i] %= 10;
14    if (na[lmax]) lmax++;
15    for (int i = lmax - 1; i >= 0; i--) ans += na[i] + '0';
16    return ans;
17 }
18
19 //o(n)
20

```

高精度减法

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  string sub(string a, string b) //只限大的非负整数减小的非负整数
4  {
5      const int L = 1e5;
6      string ans;
7      int na[L] = {0}, nb[L] = {0};
8      int la = a.size(), lb = b.size();
9      for (int i = 0; i < la; i++) na[la - 1 - i] = a[i] - '0';
10     for (int i = 0; i < lb; i++) nb[lb - 1 - i] = b[i] - '0';
11     int lmax = la > lb ? la : lb;
12     for (int i = 0; i < lmax; i++) {
13         na[i] -= nb[i];
14         if (na[i] < 0) na[i] += 10, na[i + 1]--;
15     }
16     while (!na[--lmax] && lmax > 0)
17         ;
18     lmax++;
19     for (int i = lmax - 1; i >= 0; i--) ans += na[i] + '0';
20     return ans;
21 }
22
23 //o(n)
24

```

高精度阶乘

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  string fac(int n) {
4      const int L = 100005;
5      int a[L];
6      string ans;
7      if (n == 0) return "1";
8      fill(a, a + L, 0);
9      int s = 0, m = n;
10     while (m) a[++s] = m % 10, m /= 10;
11     for (int i = n - 1; i >= 2; i--) {
12         int w = 0;

```

```

13         for (int j = 1; j <= s; j++)
14             a[j] = a[j] * i + w, w = a[j] / 10, a[j] = a[j] % 10;
15         while (w) a[++s] = w % 10, w /= 10;
16     }
17     while (!a[s]) s--;
18     while (s >= 1) ans += a[s--] + '0';
19     return ans;
20 }
21
22 //o(n^2)
23

```

高精度进制转换

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  //将字符串表示的10进制大整数转换为m进制的大整数
4  //并返回m进制大整数的字符串
5  bool judge(string s) //判断串是否为全零串
6  {
7      for (int i = 0; i < s.size(); i++)
8          if (s[i] != '0') return 1;
9      return 0;
10 }
11 string solve(
12     string s, int n,
13     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
14 {
15     string r, ans;
16     int d = 0;
17     if (!judge(s)) return "0"; //特判
18     while (judge(s)) //被除数不为0则继续
19     {
20         for (int i = 0; i < s.size(); i++) {
21             r += (d * n + s[i] - '0') / m + '0'; //求出商
22             d = (d * n + (s[i] - '0')) % m; //求出余数
23         }
24         s = r; //把商赋给下一次的被除数
25         r = ""; //把商清空
26         ans += d + '0'; //加上进制转换后数字
27         d = 0; //清空余数
28     }
29     reverse(ans.begin(), ans.end()); //倒置下
30     return ans;
31 }
32
33 //o(n^2)
34

```

高精度幂

```
1  #include <bits/stdc++.h>
2  #define L(x) (1 << (x))
3  using namespace std;
4  const double PI = acos(-1.0);
5  const int Maxn = 133015;
6  double ax[Maxn], ay[Maxn], bx[Maxn], by[Maxn];
7  char sa[Maxn / 2], sb[Maxn / 2];
8  int sum[Maxn];
9  int x1[Maxn], x2[Maxn];
10 int revv(int x, int bits) {
11     int ret = 0;
12     for (int i = 0; i < bits; i++) {
13         ret <= 1;
14         ret |= x & 1;
15         x >>= 1;
16     }
17     return ret;
18 }
19 void fft(double* a, double* b, int n, bool rev) {
20     int bits = 0;
21     while (1 << bits < n) ++bits;
22     for (int i = 0; i < n; i++) {
23         int j = revv(i, bits);
24         if (i < j) swap(a[i], a[j]), swap(b[i], b[j]);
25     }
26     for (int len = 2; len <= n; len <= 1) {
27         int half = len >> 1;
28         double wmx = cos(2 * PI / len), wmy = sin(2 * PI / len);
29         if (rev) wmy = -wmy;
30         for (int i = 0; i < n; i += len) {
31             double wx = 1, wy = 0;
32             for (int j = 0; j < half; j++) {
33                 double cx = a[i + j], cy = b[i + j];
34                 double dx = a[i + j + half], dy = b[i + j + half];
35                 double ex = dx * wx - dy * wy, ey = dx * wy + dy * wx;
36                 a[i + j] = cx + ex, b[i + j] = cy + ey;
37                 a[i + j + half] = cx - ex, b[i + j + half] = cy - ey;
38                 double wnx = wx * wmx - wy * wmy, wny = wx * wmy + wy *
wnx;
39                 wx = wnx, wy = wny;
40             }
41         }
42     }
43     if (rev) {
44         for (int i = 0; i < n; i++) a[i] /= n, b[i] /= n;
45     }
46 }
47 int solve(int a[], int na, int b[], int nb, int ans[]) {
48     int len = max(na, nb), ln;
49     for (ln = 0; L(ln) < len; ++ln)
50         ;
51     len = L(++ln);
52     for (int i = 0; i < len; ++i) {
53         if (i >= na)
54             ax[i] = 0, ay[i] = 0;
```

```

55         else
56             ax[i] = a[i], ay[i] = 0;
57     }
58     fft(ax, ay, len, 0);
59     for (int i = 0; i < len; ++i) {
60         if (i >= nb)
61             bx[i] = 0, by[i] = 0;
62         else
63             bx[i] = b[i], by[i] = 0;
64     }
65     fft(bx, by, len, 0);
66     for (int i = 0; i < len; ++i) {
67         double cx = ax[i] * bx[i] - ay[i] * by[i];
68         double cy = ax[i] * by[i] + ay[i] * bx[i];
69         ax[i] = cx, ay[i] = cy;
70     }
71     fft(ax, ay, len, 1);
72     for (int i = 0; i < len; ++i) ans[i] = (int)(ax[i] + 0.5);
73     return len;
74 }
75 string mul(string sa, string sb) {
76     int l1, l2, l;
77     int i;
78     string ans;
79     memset(sum, 0, sizeof(sum));
80     l1 = sa.size();
81     l2 = sb.size();
82     for (i = 0; i < l1; i++) x1[i] = sa[l1 - i - 1] - '0';
83     for (i = 0; i < l2; i++) x2[i] = sb[l2 - i - 1] - '0';
84     l = solve(x1, l1, x2, l2, sum);
85     for (i = 0; i < l || sum[i] >= 10; i++) // 进位
86     {
87         sum[i + 1] += sum[i] / 10;
88         sum[i] %= 10;
89     }
90     l = i;
91     while (sum[l] <= 0 && l > 0) l--; // 检索最高位
92     for (i = l; i >= 0; i--) ans += sum[i] + '0'; // 倒序输出
93     return ans;
94 }
95 string Pow(string a, int n) {
96     if (n == 1) return a;
97     if (n & 1) return mul(Pow(a, n - 1), a);
98     string ans = Pow(a, n / 2);
99     return mul(ans, ans);
100 }
101
102 //o(nlognlogm)

```

高精度平方根

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  const int L = 2015;
4  string add(string a, string b) //只限两个非负整数相加
5  {
6      string ans;

```

```

7     int na[L] = {0}, nb[L] = {0};
8     int la = a.size(), lb = b.size();
9     for (int i = 0; i < la; i++) na[la - 1 - i] = a[i] - '0';
10    for (int i = 0; i < lb; i++) nb[lb - 1 - i] = b[i] - '0';
11    int lmax = la > lb ? la : lb;
12    for (int i = 0; i < lmax; i++)
13        na[i] += nb[i], na[i + 1] += na[i] / 10, na[i] %= 10;
14    if (na[lmax]) lmax++;
15    for (int i = lmax - 1; i >= 0; i--) ans += na[i] + '0';
16    return ans;
17 }
18 string sub(string a, string b) //只限大的非负整数减小的非负整数
19 {
20     string ans;
21     int na[L], nb[L] = {0};
22     int la = a.size(), lb = b.size();
23     for (int i = 0; i < la; i++) na[la - 1 - i] = a[i] - '0';
24     for (int i = 0; i < lb; i++) nb[lb - 1 - i] = b[i] - '0';
25     int lmax = la > lb ? la : lb;
26     for (int i = 0; i < lmax; i++) {
27         na[i] -= nb[i];
28         if (na[i] < 0) na[i] += 10, na[i + 1]--;
29     }
30     while (!na[--lmax] && lmax > 0)
31         ;
32     lmax++;
33     for (int i = lmax - 1; i >= 0; i--) ans += na[i] + '0';
34     return ans;
35 }
36 string mul(string a, string b) //高精度乘法a,b,均为非负整数
37 {
38     string s;
39     int na[L], nb[L], nc[L],
40         La = a.size(), Lb = b.size(); // na存储被乘数, nb存储乘数, nc存储积
41     fill(na, na + L, 0);
42     fill(nb, nb + L, 0);
43     fill(nc, nc + L, 0); //将na,nb,nc都置为0
44     for (int i = La - 1; i >= 0; i--)
45         na[La - i] =
46             a[i] - '0'; //将字符串表示的大整数数组转成i整数数组表示的大整数数
47     for (int i = Lb - 1; i >= 0; i--) nb[Lb - i] = b[i] - '0';
48     for (int i = 1; i <= La; i++)
49         for (int j = 1; j <= Lb; j++)
50             nc[i + j - 1] +=
51                 na[i] *
52                 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)
53     for (int i = 1; i <= La + Lb; i++)
54         nc[i + 1] += nc[i] / 10, nc[i] %= 10; //统一处理进位
55     if (nc[La + Lb]) s += nc[La + Lb] + '0'; //判断第i+j位上的数字是不是0
56     for (int i = La + Lb - 1; i >= 1; i--)
57         s += nc[i] + '0'; //将整数数组转成字符串
58     return s;
59 }
60 int sub(int *a, int *b, int La, int Lb) {
61     if (La < Lb) return -1; //如果a小于b, 则返回-1
62     if (La == Lb) {
63         for (int i = La - 1; i >= 0; i--)
64             if (a[i] > b[i])

```



```

65         break;
66     else if (a[i] < b[i])
67         return -1; //如果a小于b, 则返回-1
68 }
69 for (int i = 0; i < La; i++) //高精度减法
70 {
71     a[i] -= b[i];
72     if (a[i] < 0) a[i] += 10, a[i + 1]--;
73 }
74 for (int i = La - 1; i >= 0; i--)
75     if (a[i]) return i + 1; //返回差的位数
76 return 0; //返回差的位数
77 }
78 string div(string n1, string n2,
79     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
80 {
81     string s, v; // s存商,v存余数
82     int a[L], b[L], r[L],
83         La = n1.size(), Lb = n2.size(), i,
84         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
85     fill(a, a + L, 0);
86     fill(b, b + L, 0);
87     fill(r, r + L, 0); //数组元素都置为0
88     for (i = La - 1; i >= 0; i--) a[La - 1 - i] = n1[i] - '0';
89     for (i = Lb - 1; i >= 0; i--) b[Lb - 1 - i] = n2[i] - '0';
90     if (La < Lb || (La == Lb && n1 < n2)) {
91         // cout<<0<<endl;
92         return n1;
93     } //如果a<b,则商为0, 余数为被除数
94     int t = La - Lb; //除被数和除数的位数之差
95     for (int i = La - 1; i >= 0; i--) //将除数扩大10^t倍
96         if (i >= t)
97             b[i] = b[i - t];
98         else
99             b[i] = 0;
100     Lb = La;
101     for (int j = 0; j <= t; j++) {
102         int temp;
103         while ((temp = sub(a, b + j, La, Lb - j)) >=
104             0) //如果被除数比除数大继续减
105         {
106             La = temp;
107             r[t - j]++;
108         }
109     }
110     for (i = 0; i < L - 10; i++)
111         r[i + 1] += r[i] / 10, r[i] %= 10; //统一处理进位
112     while (!r[i]) i--; //将整形数组表示的商转化成字符串表示的
113     while (i >= 0) s += r[i--] + '0';
114     // cout<<s<<endl;
115     i = tp;
116     while (!a[i]) i--; //将整形数组表示的余数转化成字符串表示的
117     while (i >= 0) v += a[i--] + '0';
118     if (v.empty()) v = "0";
119     // cout<<v<<endl;
120     if (nn == 1) return s;
121     if (nn == 2) return v;
122 }

```

```

123 bool cmp(string a, string b) {
124     if (a.size() < b.size()) return 1; // a小于等于b返回真
125     if (a.size() == b.size() && a <= b) return 1;
126     return 0;
127 }
128 string DeletePreZero(string s) {
129     int i;
130     for (i = 0; i < s.size(); i++)
131         if (s[i] != '0') break;
132     return s.substr(i);
133 }
134
135 string BigInterSqrt(string n) {
136     n = DeletePreZero(n);
137     string l = "1", r = n, mid, ans;
138     while (cmp(l, r)) {
139         mid = div(add(l, r), "2", 1);
140         if (cmp(mul(mid, mid), n))
141             ans = mid, l = add(mid, "1");
142         else
143             r = sub(mid, "1");
144     }
145     return ans;
146 }
147
148 // o(n^3)
149

```

高精度取模（对单精度）

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  int mod(string a,int b)//高精度a除以单精度b
4  {
5      int d=0;
6      for(int i=0;i<a.size();i++) d=(d*10+(a[i]-'0'))%b;//求出余数
7      return d;
8  }
9
10 //o(n)

```

欧拉筛

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  const int N = 1000005;
5  int phi[N], prime[N], cnt;
6  bool st[N];
7
8  void get_eulers() {
9      phi[1] = 1;
10     for (int i = 2; i < N; i++) {
11         if (!st[i]) {
12             prime[cnt++] = i;
13             phi[i] = i - 1;

```

```

14     }
15     for (int j = 0; prime[j] * i < N; j++) {
16         st[prime[j] * i] = 1;
17         if (i % prime[j] == 0) {
18             phi[prime[j] * i] = phi[i] * prime[j];
19             break;
20         }
21         phi[prime[j] * i] = phi[i] * (prime[j] - 1);
22     }
23 }
24 }
25
26 int main() {
27     get_eulers();
28     ll n;
29     cin >> n;
30     ll ans = 0;
31     for (int i = 1; i <= n; i++) ans += phi[i];
32     cout << ans;
33 }

```

组合数（逆元线性递推

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4  typedef long long ll;
5  const ll mod = 1e9 + 7;
6  const ll maxn = 3e4 + 5;
7  ll inv[maxn], fac[maxn];
8
9  ll qpow(ll a, ll b) {
10     ll ans = 1;
11     while (b) {
12         if (b & 1) ans = (ans * a) % mod;
13         a = (a * a) % mod;
14         b >>= 1;
15     }
16     return ans;
17 }
18
19 ll c(ll n, ll m) {
20     if (n < 0 || m < 0 || n < m) return 0;
21     return fac[n] * inv[n - m] % mod * inv[m] % mod;
22 }
23
24 void init() {
25     fac[0] = 1;
26     for (int i = 1; i < maxn; i++) {
27         fac[i] = fac[i - 1] * i % mod;
28     }
29     inv[maxn - 1] = qpow(fac[maxn - 1], mod - 2);
30     for (ll i = maxn - 2; i >= 0; i--) {
31         inv[i] = (inv[i + 1] * (i + 1)) % mod;
32     }
33 }
34

```

中国剩余定理

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  typedef long long ll;
5
6  const int maxn = 20;
7
8  ll A[maxn], B[maxn];
9
10 ll exgcd(ll a, ll b, ll & x, ll & y) {
11     if(b == 0) {
12         x = 1, y = 0;
13         return a;
14     }
15
16     ll d = exgcd(b, a % b, y, x);
17
18     y -= (a / b) * x;
19
20     return d;
21 }
22
23 int main() {
24     int n;
25     cin >> n;
26     ll M = 1ll;
27     for(int i = 0; i < n; ++ i) {
28         cin >> A[i] >> B[i];
29         M = M * A[i];
30     }
31
32     ll ans = 0;
33
34     ll x, y;
35
36     for(int i = 0; i < n; ++ i) {
37         ll Mi = M / A[i];
38         exgcd(Mi, A[i], x, y);
39         ans += B[i] * Mi * x;
40     }
41
42     cout << (ans % M + M) % M;
43
44 }
```

图论

有源汇上下界最大小流

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4
5  struct Edge {
6      ll from, to, cap, flow, mn;
7      Edge(ll a, ll b, ll c, ll d, ll e) : from(a), to(b), cap(c), flow(d),
mn(e) {}
8  };
9
10 ll n, m;
11
12 struct Dinic {
13     static const ll maxn = 50010; // 点的大小, 记得改
14     static const ll inf = 0x3f3f3f3f3f3f3f3f;
15     ll N, M, S, T;
16     vector<Edge> edges;
17     vector<ll> G[maxn];
18     bool vis[maxn];
19     ll d[maxn];
20     ll cur[maxn];
21
22     void AddEdge(ll from, ll to, ll cap, ll c) {
23         edges.push_back(Edge(from, to, cap, 0, c));
24         edges.push_back(Edge(to, from, 0, 0, c));
25         M = edges.size();
26         G[from].push_back(M - 2);
27         G[to].push_back(M - 1);
28     }
29
30     bool BFS() {
31         memset(vis, 0, sizeof(vis));
32         queue<ll> Q;
33         Q.push(S);
34         d[S] = 0;
35         vis[S] = 1;
36         while (!Q.empty()) {
37             ll x = Q.front();
38             Q.pop();
39             for (ll i = 0; i < G[x].size(); i++) {
40                 Edge& e = edges[G[x][i]];
41                 if (!vis[e.to] && e.cap > e.flow) {
42                     vis[e.to] = 1;
43                     d[e.to] = d[x] + 1;
44                     Q.push(e.to);
45                 }
46             }
47         }
48         return vis[T];
49     }
50
51     ll DFS(ll x, ll a) {
52         if (x == T || a == 0) return a;
53         ll flow = 0, f;
54         for (ll& i = cur[x]; i < G[x].size(); i++) {
```

```

55         Edge& e = edges[G[x][i]];
56         if (d[x] + 1 == d[e.to] &&
57             (f = DFS(e.to, min(a, e.cap - e.flow))) > 0) {
58             e.flow += f;
59             edges[G[x][i] ^ 1].flow -= f;
60             flow += f;
61             a -= f;
62             if (a == 0) break;
63         }
64     }
65     return flow;
66 }
67
68 void deleteEdge(int u, int v) {
69     int siz = edges.size();
70     for(int i = 0; i < siz; ++ i) {
71         if(edges[i].from == u && edges[i].to == v) {
72             edges[i].cap = edges[i].flow = 0;
73             edges[i ^ 1].cap = edges[i ^ 1].flow = 0;
74             break;
75         }
76     }
77 }
78
79 }
80
81 int getValue() {
82     return edges[2 * m].flow;
83 }
84
85 int Maxflow(int S, int T) {
86     this->S = S, this->T = T;
87     int flow = 0;
88     while (BFS()) {
89         memset(cur, 0, sizeof(cur));
90         flow += DFS(S, inf);
91     }
92     return flow;
93 }
94 } MF;
95
96 int main() {
97     int s, t;
98     cin >> n >> m >> s >> t;
99     // n个点, m条边, 给的源点汇点
100
101     int mp[50010] = {0}; // 点的大小, 记得改
102     for(int i = 1; i <= m; ++ i) {
103         int a, b, c, d; // 从a到b有一条下界c上界d的边
104         cin >> a >> b >> c >> d;
105         mp[b] += c;
106         mp[a] -= c;
107         MF.AddEdge(a, b, d - c, c);
108     }
109     MF.AddEdge(t, s, 1e18, 0); //
110     int tot = 0;
111     for(int i = 1; i <= n; ++ i) {
112         if(mp[i] > 0) {

```

```

113         tot += mp[i];
114         MF.AddEdge(0, i, mp[i], 0);
115     }
116     else {
117         MF.AddEdge(i, n + 1, -mp[i], 0);
118     }
119 }
120
121 if( MF.Maxflow(0, n + 1) != tot) {
122     cout << "No Solution" << endl;
123 }
124 else {
125     ll res = MF.getValue(); // 从t到s边的流量
126     MF.deleteEdge(t, s);
127     //cout << res + MF.Maxflow(s, t) << endl; // 最大流
128     cout << res - MF.Maxflow(t, s) << endl; // 最小流
129 }
130
131 return 0;
132 }
133

```

树链剖分

```

1  ll fa[N], son[N], dep[N], siz[N], dfn[N], rnk[N], top[N];
2  ll dfscnt;
3  vector<ll> g[N];
4  ll tree[N << 1];
5  ll lazy[N << 1];
6
7  void dfs1(ll u, ll f, ll d) {
8      son[u] = -1;
9      siz[u] = 1;
10     fa[u] = f;
11     dep[u] = d;
12     for (auto v:g[u]) {
13         if (v == f) continue;
14         dfs1(v, u, d + 1);
15         siz[u] += siz[v];
16         if (son[u] == -1 || siz[v] > siz[son[u]]) son[u] = v;
17     }
18 }
19
20 void dfs2(ll u, ll t) {
21     dfn[u] = ++dfscnt;
22     rnk[dfscnt] = u;
23     top[u] = t;
24     if (son[u] == -1) return;
25     dfs2(son[u], t);
26     for (auto v:g[u]) {
27         if (v == son[u] || v == fa[u]) continue;
28         dfs2(v, v);
29     }
30 }
31
32 ll lca(ll a, ll b) {

```

```

33     while (top[a] != top[b]) {
34         if (dep[top[a]] < dep[top[b]]) swap(a, b);
35         a = fa[top[a]];
36     }
37     return dep[a] < dep[b] ? a : b;
38 }
39
40 void init() {
41     for (ll i = 0; i < N; i++) g[i].clear();
42     for (ll i = 0; i < (N << 1); i++) {
43         tree[i] = 0;
44         lazy[i] = 0;
45     }
46     dfscnt = 0;
47 }
48
49
50 void pushdown(ll k, ll l, ll r) {
51     if (k >= N || lazy[k] == 0) return;
52     ll len = (r - l + 1) / 2;
53     tree[k << 1] = tree[k << 1] + len * lazy[k];
54     tree[k << 1 | 1] = tree[k << 1 | 1] + len * lazy[k];
55     lazy[k << 1] = lazy[k << 1] + lazy[k];
56     lazy[k << 1 | 1] = lazy[k << 1 | 1] + lazy[k];
57     lazy[k] = 0;
58 }
59
60 ll merge_range(ll a, ll b) {
61     ll ans = a + b;
62     return ans;
63 }
64
65 void change_range(ll k, ll l, ll r, ll ql, ll qr, ll x) {
66     if (r < ql || qr < l) return;
67     if (ql <= l && r <= qr) {
68         tree[k] = tree[k] + x * (r - l + 1);
69         lazy[k] = lazy[k] + x;
70         return;
71     }
72     pushdown(k, l, r);
73     ll mid = (l + r) >> 1;
74     change_range(k << 1, l, mid, ql, qr, x);
75     change_range(k << 1 | 1, mid + 1, r, ql, qr, x);
76     tree[k] = merge_range(tree[k << 1], tree[k << 1 | 1]);
77 }
78
79 ll query_range(ll k, ll l, ll r, ll ql, ll qr) {
80     if (r < ql || qr < l) return 0;
81     if (ql <= l && r <= qr) {
82         return tree[k];
83     }
84     pushdown(k, l, r);
85     ll mid = (l + r) >> 1;
86     ll lq = query_range(k << 1, l, mid, ql, qr);
87     ll rq = query_range(k << 1 | 1, mid + 1, r, ql, qr);
88     return merge_range(lq, rq);
89 }
90

```



```

91 11 query_path(11 a, 11 b) {
92     11 sum = 0;
93     while (top[a] != top[b]) {
94         if (dep[top[a]] < dep[top[b]]) swap(a, b);
95         sum = sum + query_range(1, 1, N, dfn[top[a]], dfn[a]);
96         //dfn[top[a]]~dfn[a]
97         a = fa[top[a]];
98     }
99     if (dep[a] > dep[b]) swap(a, b);
100    //点权
101    sum = sum + query_range(1, 1, N, dfn[a], dfn[b]);
102    //边权
103    //if (a != b) sum = sum + query_range(1, 1, N, dfn[a] + 1, dfn[b]);
104    //dfn[a]~dfn[b],x
105    return sum;
106 }
107
108 void change_path(11 a, 11 b, 11 x) {
109     while (top[a] != top[b]) {
110         if (dep[top[a]] < dep[top[b]]) swap(a, b);
111         change_range(1, 1, N, dfn[top[a]], dfn[a], x);
112         //dfn[top[a]]~dfn[a]
113         a = fa[top[a]];
114     }
115     if (dep[a] > dep[b]) swap(a, b);
116    //点权
117    change_range(1, 1, N, dfn[a], dfn[b], x);
118    //边权
119    //if (a != b) change_range(1, 1, N, dfn[a] + 1, dfn[b], x);
120    //dfn[a]~dfn[b],x
121 }
122
123

```

虚树

```

1  11 fa[N], son[N], dep[N], siz[N], dfn[N], rnk[N], top[N];
2  11 dfscnt;
3  vector<11> g[N];
4  11 mmin[N];
5
6  void dfs1(11 u, 11 f, 11 d) {
7      son[u] = -1;
8      siz[u] = 1;
9      fa[u] = f;
10     dep[u] = d;
11     for (auto v:g[u]) {
12         if (v == f) continue;
13         dfs1(v, u, d + 1);
14         siz[u] += siz[v];
15         if (son[u] == -1 || siz[v] > siz[son[u]]) son[u] = v;
16     }
17 }
18
19 void dfs2(11 u, 11 t) {

```

```

20     dfn[u] = ++dfscnt;
21     rnk[dfscnt] = u;
22     top[u] = t;
23     if (son[u] == -1) return;
24     dfs2(son[u], t);
25     for (auto v:g[u]) {
26         if (v == son[u] || v == fa[u]) continue;
27         dfs2(v, v);
28     }
29 }
30
31 ll lca(ll a, ll b) {
32     while (top[a] != top[b]) {
33         if (dep[top[a]] < dep[top[b]]) swap(a, b);
34         a = fa[top[a]];
35     }
36     return dep[a] < dep[b] ? a : b;
37 }
38
39 struct edge {
40     ll s, t, v;
41 };
42 edge e[N];
43
44 vector<int> vg[N];
45 int sta[N], tot;
46 int h[N];
47
48 void build(int *H, int num) {
49     sort(H + 1, H + 1 + num, [](int a, int b) { return dfn[a] < dfn[b]; });
50     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
51     for (int i = 1, l; i <= num; ++i) {
52         if (H[i] == 1) continue; // 如果 1 号节点是关键节点就不要重复添加
53         l = lca(H[i], sta[tot]); // 计算当前节点与栈顶节点的 LCA
54         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
55             while (dfn[l] < dfn[sta[tot - 1]]) { // 当次大节点的 Dfs 序大于 LCA
的 Dfs 序
56                 vg[sta[tot - 1]].push_back(sta[tot]);
57                 vg[sta[tot]].push_back(sta[tot - 1]);
58                 tot--;
59             } // 把与当前节点所在的链不重合的链连接掉并且弹出
60             if (dfn[l] > dfn[sta[tot - 1]]) { // 如果 LCA 不等于次大节点 (这里的大
于其实和不等于是没有区别)
61                 vg[1].clear();
62                 vg[1].push_back(sta[tot]);
63                 vg[sta[tot]].push_back(l);
64                 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
65             } else {
66                 vg[1].push_back(sta[tot]);
67                 vg[sta[tot]].push_back(l);
68                 tot--; // 说明 LCA 就是次大节点, 直接弹出栈顶元素
69             }
70         }
71         vg[H[i]].clear();
72         sta[++tot] = H[i];

```

```

73         //当前节点必然是第一次入栈，清空邻接表并入栈
74     }
75     for (int i = 1; i < tot; ++i) {
76         vg[sta[i]].push_back(sta[i + 1]);
77         vg[sta[i + 1]].push_back(sta[i]);
78     } //剩余的最后一条链连接一下
79     return;
80 }
81

```

spfa最短路及负环

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  const int N = 1 << 20;
5  struct edge {
6      ll to, len;
7  };
8
9  vector<edge> g[N];
10 ll d[N], cnt[N], vis[N];
11
12 bool spfa(ll s, ll n) {
13     queue<int> que;
14     for (int i = 1; i <= n; i++) { //防止不连通，全加进去
15         que.push(i);
16         vis[i] = 1;
17     }
18     while (!que.empty()) {
19         ll p = que.front();
20         que.pop();
21         vis[p] = 0;
22         for (auto x:g[p]) {
23             if (d[x.to] > d[p] + x.len) {
24                 d[x.to] = d[p] + x.len;
25                 cnt[x.to] = cnt[p] + 1;
26                 if (!vis[x.to]) {
27                     if (cnt[x.to] > n) return 0;
28                     vis[x.to] = 1;
29                     que.push(x.to);
30                 }
31             }
32         }
33     }
34     return 1;
35 }
36

```

二分图匹配（匈牙利）

```
1 //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
2 #include<bits/stdc++.h>
3 using namespace std;
4
5 const int maxn=505;// 最大点数
6 const int inf=0x3f3f3f3f;// 距离初始值
7 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
8     int nx,ny;//左右顶点数量
9     vector<int>bmap[maxn];
10    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
11    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
12    int dx[maxn];
13    int dy[maxn];
14    int dis;
15    bool bmask[maxn];
16    void init(int a,int b){
17        nx=a,ny=b;
18        for(int i=0;i<=nx;i++){
19            bmap[i].clear();
20        }
21    }
22    void add_edge(int u,int v){
23        bmap[u].push_back(v);
24    }
25    bool searchpath(){//寻找 增广路径
26        queue<int>Q;
27        dis=inf;
28        memset(dx,-1,sizeof(dx));
29        memset(dy,-1,sizeof(dy));
30        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
31            if(cx[i]==-1){//将未遍历的节点 入队 并初始化次节点距离为0
32                Q.push(i);
33                dx[i]=0;
34            }
35        }//广度搜索增广路径
36        while(!Q.empty()){
37            int u=Q.front();
38            Q.pop();
39            if(dx[u]>dis) break;//取右侧节点
40            for(int i=0;i<bmap[u].size();i++){
41                int v=bmap[u][i];//右侧节点的增广路径的距离
42                if(dy[v]==-1){
43                    dy[v]=dx[u]+1;//v对应的距离 为u对应距离加1
44                    if(cy[v]==-1)dis=dy[v];
45                    else{
46                        dx[cy[v]]=dy[v]+1;
47                        Q.push(cy[v]);
48                    }
49                }
50            }
51        }
52        return dis!=inf;
53    }
54    int findpath(int u){//寻找路径 深度搜索
```

```

55         for(int i=0;i<bmap[u].size();i++){
56             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
57             if(!bmask[v]&&dy[v]==dx[u]+1){//对该点染色
58                 bmask[v]=1;
59                 if(cy[v]!=-1&&dy[v]==dis)continue;
60                 if(cy[v]==-1||findpath(cy[v])){
61                     cy[v]=u;cx[u]=v;
62                     return 1;
63                 }
64             }
65         }
66         return 0;
67     }
68     int MaxMatch(){//得到最大匹配的数目
69         int res=0;
70         memset(cx,-1,sizeof(cx));
71         memset(cy,-1,sizeof(cy));
72         while(searchpath()){
73             memset(bmask,0,sizeof(bmask));
74             for(int i=1;i<=nx;i++){
75                 if(cx[i]==-1){
76                     res+=findpath(i);
77                 }
78             }
79         }
80         return res;
81     }
82 }HK;
83
84 int main(){
85     int nn,n,m;
86     cin>>nn;
87     while(nn--){
88         scanf("%d%d",&n,&m);
89         HK.init(n,m);//左端点和右端点数量
90         for(int i=1;i<=n;i++){
91             int snum;
92             cin>>snum;
93             int v;
94             for(int j=1;j<=snum;j++){
95                 cin>>v;
96                 HK.add_edge(i,v);//连边
97             }
98         }
99         cout<<HK.MaxMatch()<<endl;//求最大匹配
100     }
101     return 0;
102 }

```

强连通 (kosaraju)

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  struct SCC {
4      static const int MAXV = 100000;
5      int v;

```

```

6     vector<int> g[MAXV], rg[MAXV], vs;
7     bool used[MAXV];
8     int cmp[MAXV];
9
10    void add_edge(int from, int to) {
11        g[from].push_back(to);
12        rg[to].push_back(from);
13    }
14
15    void dfs(int v) {
16        used[v] = 1;
17        for (int i = 0; i < g[v].size(); i++) {
18            if (!used[g[v][i]]) dfs(g[v][i]);
19        }
20        vs.push_back(v);
21    }
22
23    void rdfs(int v, int k) {
24        used[v] = 1;
25        cmp[v] = k;
26        for (int i = 0; i < rg[v].size(); i++) {
27            if (!used[rg[v][i]]) rdfs(rg[v][i], k);
28        }
29    }
30
31    int solve() {
32        memset(used, 0, sizeof(used));
33        vs.clear();
34        for (int v = 1; v <= V; v++) {
35            if (!used[v]) dfs(v);
36        }
37        memset(used, 0, sizeof(used));
38        int k = 0;
39        for (int i = (int)vs.size() - 1; i >= 0; i--) {
40            if (!used[vs[i]]) rdfs(vs[i], ++k);
41        }
42        return k;
43    }
44
45    void init(int n) {
46        V = n;
47        vs.clear();
48        for (int i = 0; i < MAXV; i++) {
49            g[i].clear();
50            rg[i].clear();
51            used[i] = 0;
52            cmp[i] = 0;
53        }
54    }
55
56    } scc;
57
58    //记得调用init()
59

```

强连通 (tarjan)

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  struct SCC {
5      static const int MAXN = 100000;
6      vector<int> g[MAXN];
7      int dfn[MAXN], lowlink[MAXN], sccno[MAXN], dfs_clock, scc_cnt;
8      stack<int> S;
9
10     void dfs(int u) {
11         dfn[u] = lowlink[u] = ++dfs_clock;
12         S.push(u);
13         for (int i = 0; i < g[u].size(); i++) {
14             int v = g[u][i];
15             if (!dfn[v]) {
16                 dfs(v);
17                 lowlink[u] = min(lowlink[u], lowlink[v]);
18             } else if (!sccno[v]) {
19                 lowlink[u] = min(lowlink[u], dfn[v]);
20             }
21         }
22         if (lowlink[u] == dfn[u]) {
23             ++scc_cnt;
24             for (;;) {
25                 int x = S.top();
26                 S.pop();
27                 sccno[x] = scc_cnt;
28                 if (x == u) break;
29             }
30         }
31     }
32
33     void solve(int n) {
34         dfs_clock = scc_cnt = 0;
35         memset(sccno, 0, sizeof(sccno));
36         memset(dfn, 0, sizeof(dfn));
37         memset(lowlink, 0, sizeof(lowlink));
38         for (int i = 1; i <= n; i++) {
39             if (!dfn[i]) dfs(i);
40         }
41     }
42 } scc;
43
44 // scc_cnt为SCC计数器, sccno[i]为i所在SCC的编号
45 // vector<int> g[MAXN]中加边
46 //之后再补充init()
47
```

强连通 (tarjan无vector)

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  struct SCC {
4      static const int MAXN = 5000;
5      static const int MAXM = 2000000;
6      int dfs_clock, edge_cnt = 1, scc_cnt;
7      int head[MAXN];
8      int dfn[MAXN], lowlink[MAXN];
9      int sccno[MAXN];
10     stack<int> s;
11
12     struct edge {
13         int v, next;
14     } e[MAXM];
15
16     void add_edge(int u, int v) {
17         e[edge_cnt].v = v;
18         e[edge_cnt].next = head[u];
19         head[u] = edge_cnt++;
20     }
21
22     void tarjan(int u) {
23         int v;
24         dfn[u] = lowlink[u] = ++dfs_clock; //每次dfs, u的次序号增加1
25         s.push(u); //将u入栈
26         for (int i = head[u]; i != -1; i = e[i].next) //访问从u出发的边
27         {
28             v = e[i].v;
29             if (!dfn[v]) //如果v没被处理过
30             {
31                 tarjan(v); // dfs(v)
32                 lowlink[u] = min(lowlink[u], lowlink[v]);
33             } else if (!sccno[v])
34                 lowlink[u] = min(lowlink[u], dfn[v]);
35         }
36         if (dfn[u] == lowlink[u]) {
37             scc_cnt++;
38             do {
39                 v = s.top();
40                 s.pop();
41                 sccno[v] = scc_cnt;
42             } while (u != v);
43         }
44     }
45
46     int find_scc(int n) {
47         for (int i = 1; i <= n; i++)
48             if (!dfn[i]) tarjan(i);
49         return scc_cnt;
50     }
51
52     void init() {
53         scc_cnt = dfs_clock = 0;
54         edge_cnt = 1; //不用初始化e数组, 省时间
55         while (!s.empty()) s.pop();
56     }
```



```

56     memset(head, -1, sizeof(head));
57     memset(sccno, 0, sizeof(sccno));
58     memset(dfn, 0, sizeof(dfn));
59     memset(lowlink, 0, sizeof(lowlink));
60 }
61 } scc;

```

最大流

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4
5  struct Edge {
6      ll from, to, cap, flow;
7      Edge(ll a, ll b, ll c, ll d) : from(a), to(b), cap(c), flow(d) {}
8  };
9
10 struct Dinic {
11     static const ll maxn = 10000;
12     static const ll inf = 0x3f3f3f3f3f3f3f3f;
13     ll N, M, S, T;
14     vector<Edge> edges;
15     vector<ll> G[maxn];
16     bool vis[maxn];
17     ll d[maxn];
18     ll cur[maxn];
19
20     void AddEdge(ll from, ll to, ll cap) {
21         edges.push_back(Edge(from, to, cap, 0));
22         edges.push_back(Edge(to, from, 0, 0));
23         M = edges.size();
24         G[from].push_back(M - 2);
25         G[to].push_back(M - 1);
26     }
27
28     bool BFS() {
29         memset(vis, 0, sizeof(vis));
30         queue<ll> Q;
31         Q.push(S);
32         d[S] = 0;
33         vis[S] = 1;
34         while (!Q.empty()) {
35             ll x = Q.front();
36             Q.pop();
37             for (ll i = 0; i < G[x].size(); i++) {
38                 Edge& e = edges[G[x][i]];
39                 if (!vis[e.to] && e.cap > e.flow) {
40                     vis[e.to] = 1;
41                     d[e.to] = d[x] + 1;
42                     Q.push(e.to);
43                 }
44             }
45         }
46         return vis[T];
47     }

```

```

48
49     ll DFS(ll x, ll a) {
50         if (x == T || a == 0) return a;
51         ll flow = 0, f;
52         for (ll& i = cur[x]; i < G[x].size(); i++) {
53             Edge& e = edges[G[x][i]];
54             if (d[x] + 1 == d[e.to] &&
55                 (f = DFS(e.to, min(a, e.cap - e.flow))) > 0) {
56                 e.flow += f;
57                 edges[G[x][i] ^ 1].flow -= f;
58                 flow += f;
59                 a -= f;
60                 if (a == 0) break;
61             }
62         }
63         return flow;
64     }
65
66     ll Maxflow(ll S, ll T) {
67         this->S = S, this->T = T;
68         ll flow = 0;
69         while (BFS()) {
70             memset(cur, 0, sizeof(cur));
71             flow += DFS(S, inf);
72         }
73         return flow;
74     }
75 } MF;
76
77 //有源汇上下界最大流，跑完可行流后，s-t的最大流即为答案
78
79 //有源汇上下届最小流，不连无穷边，s-t跑最大流，再加上t-s无穷边，再跑最大流，无穷边流量为答案
80
81 //最大权闭合子图
82 //构造一个新的流网络，建一个源点s和汇点t，从s向原图中所有点权为正数的点建一条容量为点权的边，
83 //从点权为负数的点向t建一条容量为点权绝对值的边，原图中各点建的边都建成容量为正无穷的边。
84 //然后求从s到t的最小割，再用所有点权为正的权值之和减去最小割，就是我们要求的最大权值和了。
85
86 //最大密度子图
87 //01分数规划
88 //addedge(S,v,m),addedge(E,1),addedge(v,T,2*g-deg(v)+m)
89 //h(g)=n*m-maxflow(S,T)
90
91

```

最大流 (double)

```

1  #include <iostream>
2  #include <cstring>
3  #include <algorithm>
4
5  using namespace std;
6
7  struct Dinic {

```

```

8     static constexpr int N = 10010, M = 100010, INF = 1e8;
9     static constexpr double eps = 1e-8;
10    // int n, m, S, T;
11    int S, T;
12    int h[N], e[M], ne[M], idx;
13    double f[M];
14    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
15
16    void AddEdge(int a, int b, double c)
17    {
18        e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx ++ ;
19        e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx ++ ;
20    }
21
22    bool bfs()
23    {
24        int hh = 0, tt = 0;
25        memset(d, -1, sizeof d);
26        q[0] = S, d[S] = 0, cur[S] = h[S];
27        while (hh <= tt)
28        {
29            int t = q[hh ++ ];
30            for (int i = h[t]; ~i; i = ne[i])
31            {
32                int ver = e[i];
33                if (d[ver] == -1 && f[i] > 0)
34                {
35                    d[ver] = d[t] + 1;
36                    cur[ver] = h[ver];
37                    if (ver == T) return true;
38                    q[ ++ tt] = ver;
39                }
40            }
41        }
42        return false;
43    }
44
45    double find(int u, double limit)
46    {
47        if (u == T) return limit;
48        double flow = 0;
49        for (int i = cur[u]; ~i && flow < limit; i = ne[i])
50        {
51            cur[u] = i;
52            int ver = e[i];
53            if (d[ver] == d[u] + 1 && f[i] > 0)
54            {
55                double t = find(ver, min(f[i], limit - flow));
56                if (t < eps) d[ver] = -1;
57                f[i] -= t, f[i ^ 1] += t, flow += t;
58            }
59        }
60        return flow;
61    }
62
63    double Maxflow(int S, int T)
64    {
65        this->S = S, this->T = T;

```

```

66         double r = 0, flow;
67         while (bfs()) while (flow = find(S, INF)) r += flow;
68         return r;
69     }
70     void init() //////////
71     {
72         memset(h, -1, sizeof h);
73         idx = 0;
74     }
75 } MF;
76
77 // ?èinit
78
79

```

最小费用最大流

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4
5  struct Edge {
6      ll from, to, cap, flow, cost;
7      Edge(ll u, ll v, ll c, ll f, ll w):from(u), to(v), cap(c), flow(f),
8      cost(w) {}
9  };
10
11 struct MCMF {
12     static const ll maxn = 6000;
13     static const ll INF = 0x3f3f3f3f3f3f3f;
14     ll n, m;
15     vector<Edge> edges;
16     vector<ll> G[maxn];
17     ll inq[maxn];
18     ll d[maxn];
19     ll p[maxn];
20     ll a[maxn];
21
22     void init(ll n) {
23         this->n = n;
24         for (ll i = 1; i <= n; i++) G[i].clear();
25         edges.clear();
26     }
27
28     void add_edge(ll from, ll to, ll cap, ll cost) {
29         from++, to++; //原板子无法使用0点，故修改
30         edges.push_back(Edge(from, to, cap, 0, cost));
31         edges.push_back(Edge(to, from, 0, 0, -cost));
32         m = edges.size();
33         G[from].push_back(m - 2);
34         G[to].push_back(m - 1);
35     }
36 }

```

```

35
36 bool BellmanFord(int s, int t, int& flow, int& cost) {
37     for (int i = 1; i <= n; ++i) d[i] = INF;
38     memset(inq, 0, sizeof(inq));
39     d[s] = 0, inq[s] = 1, p[s] = 0, a[s] = INF;
40     queue<int> Q;
41     Q.push(s);
42     while (!Q.empty()) {
43         int u = Q.front();
44         Q.pop();
45         inq[u] = 0;
46         for (int i = 0; i < G[u].size(); ++i) {
47             Edge& e = edges[G[u][i]];
48             if (e.cap > e.flow && d[e.to] > d[u] + e.cost) {
49                 d[e.to] = d[u] + e.cost;
50                 p[e.to] = G[u][i];
51                 a[e.to] = min(a[u], e.cap - e.flow);
52                 if (!inq[e.to]) {
53                     Q.push(e.to);
54                     inq[e.to] = 1;
55                 }
56             }
57         }
58     }
59     if (d[t] == INF) return false;
60     flow += a[t];
61     cost += (ll)d[t] * (ll)a[t];
62     for (int u = t; u != s; u = edges[p[u]].from) {
63         edges[p[u]].flow += a[t];
64         edges[p[u] ^ 1].flow -= a[t];
65     }
66     return true;
67 }
68
69 //需要保证初始网络中没有负权圈
70 int MincostMaxflow(int s, int t, int& cost) {
71     s++, t++; //原板子无法使用0点, 故修改
72     int flow = 0;
73     cost = 0;
74     while (BellmanFord(s, t, flow, cost));
75     return flow;
76 }
77 } 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
78 //下标从0开始
79
80

```

树分治

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int MAXN = 10005;
4 const int INF = 1000000000;
5 struct edge {
6     int to, length;
7     edge() {}

```

```

8     edge(int a, int b) : to(a), length(b) {}
9 };
10
11
12 vector<edge> g[MAXN];
13
14 bool centroid[MAXN];
15 int subtree_size[MAXN];
16
17 int ans;
18
19 //计算子树大小
20 int compute_subtree_size(int v, int p) {
21     int c = 1;
22     for (int i = 0; i < g[v].size(); i++) {
23         int w = g[v][i].to;
24         if (w == p || centroid[w]) continue;
25         c += compute_subtree_size(w, v);
26     }
27     subtree_size[v] = c;
28     return c;
29 }
30
31 //查找重心, t为连通分量大小
32 // pair (最大子树顶点数, 顶点编号)
33 pair<int, int> search_centroid(int v, int p, int t) {
34     pair<int, int> res = pair<int, int>(INF, -1);
35     int s = 1, m = 0;
36     for (int i = 0; i < g[v].size(); i++) {
37         int w = g[v][i].to;
38         if (w == p || centroid[w]) continue;
39         res = min(res, search_centroid(w, v, t));
40         m = max(m, subtree_size[w]);
41         s += subtree_size[w];
42     }
43     m = max(m, t - s);
44     res = min(res, pair<int, int>(m, v));
45     return res;
46 }
47
48 void init(int n) {
49     memset(centroid, 0, sizeof(centroid));
50     memset(subtree_size, 0, sizeof(subtree_size));
51     for (int i = 0; i <= n; i++) g[i].clear();
52     ans = 0;
53 }
54
55 int solve(int u) {
56     compute_subtree_size(u, -1);
57     int s = search_centroid(u, -1, subtree_size[u]).second;
58     centroid[s] = 1;
59     for (int i = 0; i < g[s].size(); i++) {
60         int v = g[s][i].to;
61         if (centroid[v]) continue;
62         /*solve()*/
63     }
64     /*do something*/
65     centroid[s] = 0;

```

```
66     return ans;
67 }
68
```

拓扑排序

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  const int MAXN = 100000;
4
5  int c[MAXN];
6  int topo[MAXN], t, v;
7  vector<int> g[MAXN];
8
9  bool dfs(int u) {
10     c[u] = -1;
11     for (int i = 0; i < g[u].size(); i++) {
12         int v = g[u][i];
13         if (c[v] < 0)
14             return false;
15         else if (!c[v] && !dfs(v))
16             return false;
17     }
18     c[u] = 1;
19     topo[t--] = u;
20     return true;
21 }
22
23 bool toposort(int n) {
24     v = n;
25     t = n;
26     memset(c, 0, sizeof(c));
27     for (int u = 1; u <= v; u++)
28         if (!c[u] && !dfs(u)) return false;
29     return true;
30 }
31
```

最近公共祖先（倍增）

```
1  #include <algorithm>
2  #include <cstdio>
3  #include <cstring>
4  #include <iostream>
5  using namespace std;
6  const int MAX = 600000;
7
8  struct edge {
9     int t, nex;
10 } e[MAX << 1];
11 int head[MAX], tot;
12
13 int depth[MAX], fa[MAX][22], lg[MAX];
```

```

14
15 void add_edge(int x, int y) {
16     e[++tot].t = y;
17     e[tot].nex = head[x];
18     head[x] = tot;
19
20     e[++tot].t = x;
21     e[tot].nex = head[y];
22     head[y] = tot;
23 }
24
25 void dfs(int now, int fath) {
26     fa[now][0] = fath;
27     depth[now] = depth[fath] + 1;
28     for (int i = 1; i <= lg[depth[now]]; ++i)
29         fa[now][i] = fa[fa[now][i - 1]][i - 1];
30     for (int i = head[now]; i; i = e[i].nex)
31         if (e[i].t != fath) dfs(e[i].t, now);
32 }
33
34 int lca(int x, int y) {
35     if (depth[x] < depth[y]) swap(x, y);
36     while (depth[x] > depth[y]) x = fa[x][lg[depth[x]] - depth[y] - 1];
37     if (x == y) return x;
38     for (int k = lg[depth[x]] - 1; k >= 0; --k)
39         if (fa[x][k] != fa[y][k]) x = fa[x][k], y = fa[y][k];
40     return fa[x][0];
41 }
42
43 void init(int n, int root) {
44     for (int i = 1; i <= n; ++i) lg[i] = lg[i - 1] + (1 << lg[i - 1] == i);
45     dfs(root, 0);
46 }
47

```

最近公共祖先（线段树）

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  int n, m, root;
4  const int MAX_N = 500005;
5  const int MAX = 1 << 20;
6  vector<int> g[MAX_N];
7  vector<int> vs;
8  pair<int, int> tree[MAX * 2 + 10];
9  int fir[MAX_N];
10 int fa[MAX_N];
11 int dep[MAX_N];
12 void dfs(int k, int p, int d) {
13     fa[k] = p;
14     dep[k] = d;
15     vs.push_back(k);
16     for (int i = 0; i < g[k].size(); i++) {
17         if (g[k][i] != p) {
18             dfs(g[k][i], k, d + 1);
19             vs.push_back(k);
20         }
21     }
22 }

```



```

21     }
22 }
23 void build(int k) {
24     if (k >= MAX) return;
25     build(k << 1);
26     build(k << 1 | 1);
27     tree[k] = min(tree[k << 1], tree[k << 1 | 1]);
28 }
29 pair<int, int> query(int k, int s, int e, int l, int r) {
30     if (e < l || r < s) return pair<int, int>(INT_MAX, 0);
31     if (l <= s && e <= r) return tree[k];
32     return min(query(k << 1, s, (s + e) >> 1, l, r),
33               query(k << 1 | 1, ((s + e) >> 1) + 1, e, l, r));
34 }
35 void init() {
36     dfs(root, root, 0);
37     for (int i = 0; i < MAX * 2 + 10; i++) tree[i] = pair<int, int>(INT_MAX,
0);
38     for (int i = MAX; i < MAX + vs.size(); i++)
39         tree[i] = pair<int, int>(dep[vs[i - MAX]], vs[i - MAX]);
40     for (int i = 0; i < vs.size(); i++) {
41         if (fir[vs[i]] == 0) fir[vs[i]] = i + 1;
42     }
43     build(1);
44 }
45 int lca(int a, int b) {
46     return query(1, 1, MAX, min(fir[a], fir[b]), max(fir[a],
fir[b])).second;
47 }
48 int main() {
49     scanf("%d%d%d", &n, &m, &root);
50     for (int i = 1; i < n; i++) {
51         int a, b;
52         scanf("%d%d", &a, &b);
53         g[a].push_back(b);
54         g[b].push_back(a);
55     }
56     init();
57     for (int i = 1; i <= m; i++) {
58         int a, b;
59         scanf("%d%d", &a, &b);
60         printf("%d\n", lca(a, b));
61     }
62 }
63

```

线性代数

高斯消元

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4  const double eps = 1e-8;
5  void sway(vector<double>& a, vector<double>& b) {
6      vector<double> s;
7      for (int i = 0; i < a.size(); i++) {
8          s.push_back(a[i]);
9      }
10     a.clear();
11     for (int i = 0; i < b.size(); i++) {
12         a.push_back(b[i]);
13     }
14     b.clear();
15     for (int i = 0; i < s.size(); i++) {
16         b.push_back(s[i]);
17     }
18 }
19 vector<double> gauss_jordan(const vector<vector<double> >& A,
20                             const vector<double>& b) {
21     int n = A.size();
22     vector<vector<double> > B(n, vector<double>(n + 1));
23     for (int i = 0; i < n; i++)
24         for (int j = 0; j < n; j++) B[i][j] = A[i][j];
25     for (int i = 0; i < n; i++) B[i][n] = b[i];
26
27     for (int i = 0; i < n; i++) {
28         int pivot = i;
29         for (int j = i; j < n; j++) {
30             if (abs(B[j][i]) > abs(B[pivot][i])) pivot = j;
31         }
32         swap(B[i], B[pivot]);
33         if (abs(B[i][i]) < eps) return vector<double>();
34         for (int j = i + 1; j <= n; j++) B[i][j] /= B[i][i];
35         for (int j = 0; j < n; j++) {
36             if (i != j) {
37                 for (int k = i + 1; k <= n; k++) B[j][k] -= B[j][i] * B[i][
38 [k];
39             }
40         }
41     }
42     vector<double> x(n);
43     for (int i = 0; i < n; i++) x[i] = B[i][n];
44     return x;
45 }
46 int main() {
47     int n, m;
48     cin >> n >> m;
49     vector<vector<double> > mat(n, vector<double>(m));
50     for (int i = 0; i < n; i++) {
51         for (int j = 0; j < m; j++) {
52             cin >> mat[i][j];
53         }
54     }
55     vector<double> val(n);
```

```

55     for (int i = 0; i < n; i++) cin >> val[i];
56     vector<double> ans = gauss_jordan(mat, val);
57     for (int i = 0; i < ans.size(); i++) cout << ans[i] << ' ';
58 }
59

```

矩阵行列式

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  const ll mod = 1e9 + 7;
5  struct Matrix {
6      static const ll MAXN = 300;
7      ll a[MAXN][MAXN];
8
9      void init() { memset(a, 0, sizeof(a)); }
10
11     ll det(ll n) {
12         for (int i = 0; i < n; i++)
13             for (int j = 0; j < n; j++) a[i][j] = (a[i][j] + mod) % mod;
14         ll res = 1;
15         for (int i = 0; i < n; i++) {
16             if (!a[i][i]) {
17                 bool flag = false;
18                 for (int j = i + 1; j < n; j++) {
19                     if (a[j][i]) {
20                         flag = true;
21                         for (int k = i; k < n; k++) {
22                             swap(a[i][k], a[j][k]);
23                         }
24                         res = -res;
25                         break;
26                     }
27                 }
28                 if (!flag) return 0;
29             }
30
31             for (int j = i + 1; j < n; j++) {
32                 while (a[j][i]) {
33                     ll t = a[i][i] / a[j][i];
34                     for (int k = i; k < n; k++) {
35                         a[i][k] = (a[i][k] - t * a[j][k]) % mod;
36                         swap(a[i][k], a[j][k]);
37                     }
38                     res = -res;
39                 }
40             }
41             res *= a[i][i];
42             res %= mod;
43         }
44         return (res + mod) % mod;
45     }
46 } mat;
47

```

线性基

```
1 //
2
3 const int maxbit = 62;      //maxbit不能太大
4
5 struct L_B{
6     ll lba[maxbit];
7     L_B(){
8         memset(lba, 0, sizeof(lba));
9     }
10
11     void Insert(ll val){      //插入
12         for(int i = maxbit - 1; i >= 0; -- i) // 从高位向低位扫
13             if(val & (1ll << i)){ //
14                 if(!lba[i]){
15                     lba[i] = val;
16                     break;
17                 }
18                 val ^= lba[i];
19             }
20     }
21 };
22 //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
23 //lba[i]=x，否则令val=val`xor`lba[i]
24 //使用： 直接insert
25 // -----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

```
1 int n, k;
2 ll a[N];
3
4 void getVec() {
5     k = 0;
6
7     for(int i = 62; i >= 0; -- i) {
8         for(int j = k; j < n; ++ j) {
9             if(a[j] >> i & 1) {
10                 swap(a[j], a[k]);
11                 break;
12             }
13         }
14         if(!(a[k] >> i & 1)) continue;
15         for(int j = 0; j < n; ++j) {
```

```

16         if(j != k && (a[j] >> i & 1)) {
17             a[j] ^= a[k];
18         }
19     }
20     ++k;
21     if(k == n) break;
22 }
23
24 }
25

```

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

```

1
2 //矩阵类模板
3 struct Matrix{
4     int n,m;
5     int a[maxn][maxm];
6     void clear(){
7         n=m=0;
8         memset(a,0,sizeof(a));
9     }
10    Matrix operator +(const Matrix &b) const {
11        Matrix tmp;
12        tmp.n=n;tmp.m=m;
13        for (int i=0;i<n;++i)
14            for(int j=0;j<m;++j)
15                tmp.a[i][j]=a[i][j]+b.a[i][j];
16        return tmp;
17    }
18    Matrix operator -(const Matrix &b) const{
19        Matrix tmp;
20        tmp.n=n;tmp.m=m;
21        for (int i=0;i<n;++i){
22            for(int j=0;j<m;++j)
23                tmp.a[i][j]=a[i][j]-b.a[i][j];
24        }
25
26        return tmp;
27    }
28    Matrix operator * (const Matrix &b) const{
29        Matrix tmp;
30        tmp.clear();
31        tmp.n=n;tmp.m=b.m;
32        for (int i=0;i<n;++i)
33            for(int j=0;j<b.m;++j)
34                for (int k=0;k<m;++k){
35                    tmp.a[i][j]+=a[i][k]*b.a[k][j];
36                    tmp.a[i][j]%=mod;
37                }
38        return tmp;
39    }

```

```

40     Matrix get(int x){//幂运算
41         Matrix E;
42         E.clear();
43         E.n=E.m=n;
44         for(int i=0;i<n;++i)
45             E.a[i][i]=1;
46         if(x==0) return E;
47         else if(x==1) return *this;
48         Matrix tmp=get(x/2);
49         tmp=tmp*tmp;
50         if(x%2) tmp=tmp*(*this);
51         return tmp;
52     }
53 };
54 //矩阵模板结束

```

稀疏矩阵乘法

```

1  struct Matrix{
2      int n,m;
3      int a[maxn][maxn];////
4      void clear(){
5          n=m=0;
6          memset(a,0,sizeof(a));
7      }
8      Matrix operator * (const Matrix &b) const{
9          Matrix tmp;
10         tmp.clear();
11         tmp.n=n;tmp.m=b.m;
12         for (int k=0;k<m;++k){
13             for (int i=0;i<n;++i){
14                 if(a[i][k]==0) continue;
15                 for(int j=0;j<b.m;++j){
16                     if(b.a[k][j]==0) continue;
17                     tmp.a[i][j]+=a[i][k]*b.a[k][j];
18                     tmp.a[i][j]%=mod;
19                 }
20             }
21         }
22         return tmp;
23     }
24 };
25 //稀疏矩阵乘法

```

杂项

mt19937

```

1  #include <random>
2  #include <iostream>
3
4  int main()
5  {
6      std::random_device rd; //获取随机数种子

```

```

7      std::mt19937 gen(rd()); //Standard mersenne_twister_engine seeded with
rd()
8      std::uniform_int_distribution<> dis(0, 9);
9
10     for (int n = 0; n<20; ++n)
11         std::cout << dis(gen) << ' ';
12     std::cout << '\n';
13     system("pause");
14     return 0;
15 }
16
17 //可能的结果: 7 2 2 1 4 1 4 0 4 7 2 1 0 9 1 9 2 3 5 1

```

double : std::uniform_real_distribution<> dis(0, 9);

```

1  #include <iostream>
2  #include <chrono>
3  #include <random>
4  using namespace std;
5  int main()
6  {
7      // 随机数种子
8      unsigned seed =
std::chrono::system_clock::now().time_since_epoch().count();
9      mt19937 rand_num(seed); // 大随机数
10     uniform_int_distribution<long long> dist(0, 1000000000); // 给定范围
11     cout << dist(rand_num) << endl;
12     return 0;
13 }
14

```

注意：代码中的 rand_num 和 dist 都是自己定义的对象，不是系统的。

洗牌算法

```

1  #include <random>
2  #include <algorithm>
3  #include <iterator>
4  #include <iostream>
5
6  int main()
7  {
8      std::vector<int> v = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
9
10     std::random_device rd;
11     std::mt19937 g(rd());
12
13     std::shuffle(v.begin(), v.end(), g);
14
15     std::copy(v.begin(), v.end(), std::ostream_iterator<int>(std::cout, "
"));
16     std::cout << "\n";
17
18     system("pause");
19     return 0;
20 }

```

快读

```
1 inline int read(){
2     int x=0,w=0;char ch=0;
3     while(!isdigit(ch)){w|=ch=='-';ch=getchar();}
4     while(isdigit(ch))x=(x<<3)+(x<<1)+(ch^48),ch=getchar();
5     return w?-x:x;
6 }
```

fread快读

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 char next_char() {
5     static char buf[1 << 20], *first, *last;
6     if(first == last) {
7         last = buf + fread(buf, 1, 1 << 20, stdin);
8         first = buf;
9     }
10    return first == last ? EOF : *first ++;
11 }
12
13 inline int read(){
14     int x = 0, w = 0; char ch = 0;
15     while(!isdigit(ch)) {w |= ch == '-'; ch = next_char(); }
16     while(isdigit(ch)) {x = (x << 3) + (x << 1) + (ch ^ 48), ch =
next_char(); }
17     return w ? -x : x;
18 }
19
20 int main(){
21     freopen("1.txt", "r", stdin); // 交代码的时候一定要去掉aaa
22     int T;
23     cin >> T;
24     while(T --){
25         int x = read();
26         cout << x << endl;
27     }
28 }
```

朝鲜大哥快读

```
1 #define FI(n) FastIO::read(n)
2 #define FO(n) FastIO::write(n)
3 #define Flush FastIO::Flush()
4 //程序末尾写上 Flush;
5
6 namespace FastIO {
7     const int SIZE = 1 << 16;
8     char buf[SIZE], obuf[SIZE], str[60];
9     int bi = SIZE, bn = SIZE, opt;
```



```

10     double D[] = {0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001,
11     0.00000001, 0.000000001, 0.0000000001};
12
13     int read(char *s) {
14         while (bn) {
15             for (; bi < bn && buf[bi] <= ' '; bi++);
16             if (bi < bn)
17                 break;
18             bn = fread(buf, 1, SIZE, stdin);
19             bi = 0;
20         }
21         int sn = 0;
22         while (bn) {
23             for (; bi < bn && buf[bi] > ' '; bi++)
24                 s[sn++] = buf[bi];
25             if (bi < bn)
26                 break;
27             bn = fread(buf, 1, SIZE, stdin);
28             bi = 0;
29         }
30         s[sn] = 0;
31         return sn;
32     }
33
34     bool read(int &x) {
35         int n = read(str), bf = 0;
36         if (!n)
37             return 0;
38         int i = 0;
39         if (str[i] == '-')
40             bf = 1, i++;
41         else if (str[i] == '+')
42             i++;
43         for (x = 0; i < n; i++)
44             x = x * 10 + str[i] - '0';
45         if (bf)
46             x = -x;
47         return 1;
48     }
49
50     bool read(long long &x) {
51         int n = read(str), bf;
52         if (!n)
53             return 0;
54         int i = 0;
55         if (str[i] == '-')
56             bf = -1, i++;
57         else
58             bf = 1;
59         for (x = 0; i < n; i++)
60             x = x * 10 + str[i] - '0';
61         if (bf < 0)
62             x = -x;
63         return 1;
64     }
65
66     void write(int x) {
67         if (x == 0)

```

```

67         obuf[opt++] = '0';
68     else {
69         if (x < 0)
70             obuf[opt++] = '-', x = -x;
71         int sn = 0;
72         while (x)
73             str[sn++] = x % 10 + '0', x /= 10;
74         for (int i = sn - 1; i >= 0; i--)
75             obuf[opt++] = str[i];
76     }
77     if (opt >= (SIZE >> 1)) {
78         fwrite(obuf, 1, opt, stdout);
79         opt = 0;
80     }
81 }
82
83 void write(long long x) {
84     if (x == 0)
85         obuf[opt++] = '0';
86     else {
87         if (x < 0)
88             obuf[opt++] = '-', x = -x;
89         int sn = 0;
90         while (x)
91             str[sn++] = x % 10 + '0', x /= 10;
92         for (int i = sn - 1; i >= 0; i--)
93             obuf[opt++] = str[i];
94     }
95     if (opt >= (SIZE >> 1)) {
96         fwrite(obuf, 1, opt, stdout);
97         opt = 0;
98     }
99 }
100
101 void write(unsigned long long x) {
102     if (x == 0)
103         obuf[opt++] = '0';
104     else {
105         int sn = 0;
106         while (x)
107             str[sn++] = x % 10 + '0', x /= 10;
108         for (int i = sn - 1; i >= 0; i--)
109             obuf[opt++] = str[i];
110     }
111     if (opt >= (SIZE >> 1)) {
112         fwrite(obuf, 1, opt, stdout);
113         opt = 0;
114     }
115 }
116
117 void write(char x) {
118     obuf[opt++] = x;
119     if (opt >= (SIZE >> 1)) {
120         fwrite(obuf, 1, opt, stdout);
121         opt = 0;
122     }
123 }
124

```

```

125     void Fflush() {
126         if (opt)
127             fwrite(obuf, 1, opt, stdout);
128         opt = 0;
129     }
130 }; // namespace FastIO
131

```

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

```

1  #include <bits/stdc++.h>
2  #include <ctime>
3  using namespace std;
4
5  const int maxn = 110;
6
7  int n;
8
9  #define x first
10 #define y second
11
12 typedef pair<double, double> PDD;
13
14 PDD q[maxn];
15 double ans = 1e8;
16
17 double rand(double l, double r) {
18     return (double) rand() / RAND_MAX * (r - l) + l;
19 }
20
21 double getDist(PDD a, PDD b) {
22     double dx = a.x - b.x;
23     double dy = a.y - b.y;
24     return sqrt(dx * dx + dy * dy) ;
25 }
26
27 double calc(PDD p) {
28     double res = 0;
29     for(int i = 0; i < n; ++ i) {
30         res += getDist(q[i], p);
31     }
32     ans = min(ans, res);
33     return res;
34 }
35
36 double simulate_anneal() {
37     PDD cur(rand(0, 10000), rand(0, 10000)); // 随机一个起点

```

```

38     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
39         PDD np(rand(cur.x - T, cur.x + T), rand(cur.y - T, cur.y + T)); //
        随机新点
40         double delta = calc(np) - calc(cur);
41         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
42     }
43
44 }
45
46 int main() {
47     cin >> n;
48     for(int i = 0; i < n; ++i) {
49         cin >> q[i].x >> q[i].y;
50     }
51
52     while((double) clock() / CLOCKS_PER_SEC < 0.8) { // 卡时 // 或for (100)
53         simulate_anneal();
54     }
55
56     cout << (int)(ans + 0.5) << endl;
57
58     return 0;
59 }

```

// 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

```

1  #include <cmath>
2  #include <cstdio>
3  #include <cstdlib>
4  #include <ctime>
5
6  const int N = 10005;
7  int n, x[N], y[N], w[N];
8  double ansx, ansy, dis;
9
10 double Rand() { return (double)rand() / RAND_MAX; }
11 double calc(double xx, double yy) {
12     double res = 0;
13     for (int i = 1; i <= n; ++i) {
14         double dx = x[i] - xx, dy = y[i] - yy;
15         res += sqrt(dx * dx + dy * dy) * w[i];
16     }
17     if (res < dis) dis = res, ansx = xx, ansy = yy;
18     return res;
19 }
20 void simulateAnneal() {
21     double t = 100000;
22     double nowx = ansx, nowy = ansy;
23     while (t > 0.001) {
24         double nextx = nowx + t * (Rand() * 2 - 1);

```

```

25     double nxy = nowy + t * (Rand() * 2 - 1);
26     double delta = calc(nxtx, nxy) - calc(nowx, nowy);
27     if (exp(-delta / t) > Rand()) nowx = nxtx, nowy = nxy;
28     t *= 0.97;
29 }
30 for (int i = 1; i <= 1000; ++i) {
31     double nxtx = ansx + t * (Rand() * 2 - 1);
32     double nxy = ansy + t * (Rand() * 2 - 1);
33     calc(nxtx, nxy);
34 }
35 }
36 int main() {
37     srand(time(0));
38     scanf("%d", &n);
39     for (int i = 1; i <= n; ++i) {
40         scanf("%d%d%d", &x[i], &y[i], &w[i]);
41         ansx += x[i], ansy += y[i];
42     }
43     ansx /= n, ansy /= n, dis = calc(ansx, ansy);
44     simulateAnneal();
45     printf("%.31f %.31f\n", ansx, ansy);
46     return 0;
47 }

```

整体二分

```

1  ll bit[N];
2
3  void add_bit(ll k, ll a) {
4      while (k < N) {
5          bit[k] = bit[k] + a;
6          k += k & -k;
7      }
8  }
9
10 ll query_bit(ll k) {
11     ll ans = 0;
12     while (k) {
13         ans = ans + bit[k];
14         k -= k & -k;
15     }
16     return ans;
17 }
18
19 struct node {
20     ll x, y, k, id, type;
21 };
22 node q[N], q1[N], q2[N];
23 ll ans[N], now[N], tot, totx;
24
25 void solve(ll l, ll r, ll q1, ll qr) {
26     if (q1 > qr) return;
27     if (l == r) {
28         for (ll i = q1; i <= qr; i++) {
29             if (q[i].type == 2) {
30                 ans[q[i].id] = l;
31             }

```

```

32     }
33     return;
34 }
35 ll mid = (l + r) >> 1;
36 ll cq1 = 0, cq2 = 0;
37 for (ll i = ql; i <= qr; i++) {
38     if (q[i].type == 1) {
39         if (q[i].y <= mid) {
40             add_bit(q[i].x, q[i].k);
41             q1[++cq1] = q[i];
42         } else {
43             q2[++cq2] = q[i];
44         }
45     } else {
46         ll sum = query_bit(q[i].y) - query_bit(q[i].x - 1);
47         if (sum >= q[i].k) {
48             q1[++cq1] = q[i];
49         } else {
50             q2[++cq2] = q[i];
51             q2[cq2].k -= sum;
52         }
53     }
54 }
55 for (ll i = 1; i <= cq1; i++) if (q1[i].type == 1) add_bit(q1[i].x, -
q1[i].k);
56 for (ll i = 1; i <= cq1; i++) q[q1 + i - 1] = q1[i];
57 for (ll i = 1; i <= cq2; i++) q[q1 + cq1 + i - 1] = q2[i];
58 solve(l, mid, q1, q1 + cq1 - 1);
59 solve(mid + 1, r, q1 + cq1, qr);
60
61 }
62
63 void init() {
64     totx = 0;
65     tot = 0;
66     memset(bit, 0, sizeof bit);
67 }

```

字符串

马拉车

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  const int maxn = 100005;
4  char s[maxn];
5  char s_new[maxn * 2];
6  int p[maxn * 2];
7
8  int Manacher(char* a, int l) {
9     s_new[0] = '$';
10    s_new[1] = '#';

```

```

11     int len = 2;
12     for (int i = 0; i < l; i++) {
13         s_new[len++] = a[i];
14         s_new[len++] = '#';
15     }
16     s_new[len] = '\0';
17     int id;
18     int mx = 0;
19     int mmax = 0;
20
21     for (int i = 1; i < len; i++) {
22         p[i] = i < mx ? min(p[2 * id - i], mx - i) : 1;
23         while (s_new[i + p[i]] == s_new[i - p[i]]) p[i]++;
24         if (mx < i + p[i]) {
25             id = i;
26             mx = i + p[i];
27         }
28         mmax = max(mmax, p[i] - 1);
29     }
30     return mmax;
31 }
32
33 int main() {
34     cin >> s;
35     cout << Manacher(s, strlen(s));
36 }
37

```

AC自动机

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  struct AC {
4      static const int maxnode = 200005;
5      static const int sigma_size = 26;
6      char T[maxnode];
7      int ch[maxnode][sigma_size];
8      int val[maxnode], fail[maxnode], last[maxnode];
9      int sz;
10     vector<pair<int, int> > ans;
11
12     void init() {
13         sz = 1;
14         memset(ch[0], 0, sizeof(ch[0]));
15         ans.clear();
16     }
17
18     int idx(const char &c) { return c - 'a'; }
19
20     void insert(string s, int v) {
21         int u = 0, n = s.length();
22         for (int i = 0; i < n; i++) {
23             int c = idx(s[i]);
24             if (!ch[u][c]) {
25                 memset(ch[sz], 0, sizeof(ch[sz]));
26                 val[sz] = 0;
27                 ch[u][c] = sz++;

```

```

28         }
29         u = ch[u][c];
30     }
31     val[u] = v;
32 }
33
34 void get_fail() {
35     queue<int> que;
36     fail[0] = 0;
37     for (int c = 0; c < sigma_size; c++) {
38         int u = ch[0][c];
39         if (u) {
40             fail[u] = 0;
41             que.push(u);
42             last[u] = 0;
43         }
44     }
45     while (!que.empty()) {
46         int r = que.front();
47         que.pop();
48         for (int c = 0; c < sigma_size; c++) {
49             int u = ch[r][c];
50             if (!u) continue;
51             que.push(u);
52             int v = fail[r];
53             while (v && !ch[v][c]) v = fail[v];
54             fail[u] = ch[v][c];
55             last[u] = val[fail[u]] ? fail[u] : last[fail[u]];
56         }
57     }
58 }
59
60 void print(int j) {
61     if (j) {
62         ans.push_back(pair<int, int>(j, val[j]));
63         print(last[j]);
64     }
65 }
66
67 void find() {
68     int n = strlen(T);
69     int j = 0;
70     for (int i = 0; i < n; i++) {
71         int c = idx(T[i]);
72         while (j && !ch[j][c]) j = fail[j];
73         j = ch[j][c];
74         if (val[j])
75             print(j);
76         else if (last[j])
77             print(last[j]);
78     }
79 }
80 } ac; //字符串下标从0开始
81

```


KMP

```
1 //next数组等价于前缀函数
2 #include<bits/stdc++.h>
3 using namespace std;
4 typedef long long ll;
5
6 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);
7     int n=strlen(s1);
8     if(p2==0){
9         p1[0]=0;
10        for(int i=1;s1[i]!='\0';i++){
11            int j=p1[i-1];
12            while(j>0&&s1[i]!=s1[j])j=p1[j-1];
13            if(s1[i]==s1[j])j++;
14            p1[i]=j;
15        }
16    }
17    else{
18        for(int i=0;s2[i]!='\0';i++){
19            int j=i==0?0:p2[i-1];
20            while(j>0&&s2[i]!=s1[j])j=p1[j-1];
21            if(s2[i]==s1[j])j++;
22            p2[i]=j;
23            if(j==n)return i-n+2;//返回位置
24        }
25    }
26    return 0;
27 }
28 int main(){
29     char s1[15],s2[105];
30     int p1[15],p2[105];
31     cin>>s1>>s2;
32     kmp(s1,p1);
33     cout<<kmp(s1,p1,s2,p2)<<endl;
34     return 0;
35 }
36
```

KMP 2

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 struct KMP {
4     static const int MAXN = 1000010;
5     char T[MAXN], P[MAXN];
6     int fail[MAXN];
7     vector<int> ans;
8
9     void init() { ans.clear(); }
10
11     void get_fail() {
12         int m = strlen(P);
13         fail[0] = fail[1] = 0;
```

```

14     for (int i = 1; i < m; i++) {
15         int j = fail[i];
16         while (j && P[i] != P[j]) j = fail[j];
17         fail[i + 1] = (P[i] == P[j] ? j + 1 : 0);
18     }
19 }
20
21 void find() {
22     int n = strlen(T), m = strlen(P);
23     get_fail();
24     int j = 0;
25     for (int i = 0; i < n; i++) {
26         while (j && P[j] != T[i]) j = fail[j];
27         if (P[j] == T[i]) j++;
28         if (j == m) ans.push_back(i - m + 1);
29     }
30 }
31 } kmp; //P为模式串，下标从0开始，输入后直接调用find()
32

```

Tire

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  struct Trie {
4      static const int maxnode = 200005;
5      static const int sigma_size = 26;
6      int ch[maxnode][sigma_size];
7      int val[maxnode];
8      int sz;
9
10     Trie() {
11         sz = 1;
12         memset(ch[0], 0, sizeof(ch[0]));
13     }
14
15     int idx(const char &c) { return c - 'a'; }
16
17     void insert(string s, int v) {
18         int u = 0, n = s.length();
19         for (int i = 0; i < n; i++) {
20             int c = idx(s[i]);
21             if (!ch[u][c]) {
22                 memset(ch[sz], 0, sizeof(ch[sz]));
23                 val[sz] = 0;
24                 ch[u][c] = sz++;
25             }
26             u = ch[u][c];
27         }
28         val[u] = v;
29     }
30
31     int find(string s) {
32         int u = 0, n = s.length();
33         for (int i = 0; i < n; i++) {
34             int c = idx(s[i]);

```

```

35         if (!ch[u][c]) return 0;
36         u = ch[u][c];
37     }
38     return val[u];
39 }
40 } trie;
41

```

后缀数组

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  struct SuffixArray {
4      static const int MAXN = 1100000;
5      char s[MAXN];
6      int sa[MAXN], t[MAXN], t1[MAXN], c[MAXN], ra[MAXN], height[MAXN], m;
7      inline void init() { memset(this, 0, sizeof(SuffixArray)); }
8
9      inline void get_sa(int n) {
10         m = 256;
11         int *x = t, *y = t1;
12         for (int i = 1; i <= m; i++) c[i] = 0;
13         for (int i = 1; i <= n; i++) c[x[i] = s[i]]++;
14         for (int i = 1; i <= m; i++) c[i] += c[i - 1];
15         for (int i = n; i >= 1; i--) sa[c[x[i]]--] = i;
16         for (int k = 1; k <= n; k <= 1) {
17             int p = 0;
18             for (int i = n - k + 1; i <= n; i++) y[++p] = i;
19             for (int i = 1; i <= n; i++)
20                 if (sa[i] > k) y[++p] = sa[i] - k;
21             for (int i = 1; i <= m; i++) c[i] = 0;
22             for (int i = 1; i <= n; i++) c[x[y[i]]]++;
23             for (int i = 1; i <= m; i++) c[i] += c[i - 1];
24             for (int i = n; i >= 1; i--) sa[c[x[y[i]]]--] = y[i];
25             std::swap(x, y);
26             p = x[sa[1]] = 1;
27             for (int i = 2; i <= n; i++) {
28                 x[sa[i]] = (y[sa[i - 1]] == y[sa[i]] &&
29                     y[sa[i - 1] + k] == y[sa[i] + k])
30                     ? p
31                     : ++p;
32             }
33             if (p >= n) break;
34             m = p;
35         }
36     }
37
38     inline void get_height(int n) {
39         int i, j, k = 0;
40         for (int i = 1; i <= n; i++) ra[sa[i]] = i;
41         for (int i = 1; i <= n; i++) {
42             if (k) k--;
43             int j = sa[ra[i] - 1];
44             while (s[i + k] == s[j + k]) k++;
45             height[ra[i]] = k;
46         }

```

```
47     }
48
49     } SA;    //字符串下标从一开始
50
```

可持久化字典树

```
1  struct Trie01 {
2      static const int maxnode = 2000005;
3      static const int sigma_size = 2;
4      int ch[maxnode << 5][sigma_size], val[maxnode << 5];
5      int rt[maxnode];
6      int sz;
7
8      Trie01() {
9          sz = 0;
10         memset(ch[0], 0, sizeof(ch[0]));
11     }
12
13     void insert(int &now, int pre, int v) {
14         now = ++sz;
15         for (int i = 30; i >= 0; i--) {
16             int k = ((v >> i) & 1);
17             ch[now][k] = ++sz;
18             ch[now][k ^ 1] = ch[pre][k ^ 1];
19             val[ch[now][k]] = val[ch[pre][k]] + 1;
20             now = ch[now][k];
21             pre = ch[pre][k];
22         }
23     }
24 } trie;
25
```

对拍

windows环境下bat对拍

```
1 @echo off
2 :loop
3     dataa.exe > data.txt
4     biaocheng.exe < data.txt > ac.txt
5     A.exe < data.txt > test.txt
6     fc ac.txt test.txt
7     if not errorlevel 1 goto loop
8 pause
9 goto loop
```

其中要改的部分 (标红辽) :

```
@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
```

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

```
1  #include <iostream>
2  #include <cstdlib>
3  #include <ctime>
4  using namespace std;
5
6  int main(){
7      freopen("data.txt", "w", stdout);
8
9      srand((int)time(0));
10     int T = rand() % 100000;
11     cout << T << endl;
12
13     for (int i = 0; i < T; i++){
14         cout << rand() % 100;
15     }
16 }
```

rand() 似乎只有三万多，需要更大的数的话要乘一下