EarthCat 2022-10-26

目录

STL

bitset

C++ bitset 用法

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

```
	bitset<4> bitset1;　　//无参构造，长度为４，默认每一位为０

    bitset<8> bitset2(12);　　//长度为８，二进制保存，前面用０补充

    string s = "100101";
    bitset<10> bitset3(s);　　//长度为10，前面用０补充

    char s2[] = "10101";
    bitset<13> bitset4(s2);　　//长度为13，前面用０补充

    cout << bitset1 << endl;　　//0000
    cout << bitset2 << endl;　　//00001100
    cout << bitset3 << endl;　　//0000100101
    cout << bitset4 << endl;　　//0000000010101
```

函数

```
	bitset<8> foo ("10011011");

    cout << foo.count() << endl;　　//5　　（count函数用来求bitset中1的位数，foo中共有５个１
    cout << foo.size() << endl;　　 //8　　（size函数用来求bitset的大小，一共有８位

    cout << foo.test(0) << endl;　　//true　　（test函数用来查下标处的元素是０还是１，并返回false或true，此处foo[0]为１，返回true
    cout << foo.test(2) << endl;　　//false　　（同理，foo[2]为０，返回false

    cout << foo.any() << endl;　　//true　　（any函数检查bitset中是否有１
    cout << foo.none() << endl;　　//false　　（none函数检查bitset中是否没有１
    cout << foo.all() << endl;　　//false　　（all函数检查bitset中是全部为１
```

2019-2020 ICPC Asia Taipei-Hsinchu Regional Contest（H

H

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

windows 环境下的对拍

```
@echo off
:loop
```

```
        dataa.exe > data.txt
        biaocheng.exe < data.txt > ac.txt
        A.exe < data.txt > test.txt
        fc ac.txt test.txt
        if not errorlevel 1 goto loop
pause
goto loop
```

其中要改的部分（标红辽）：

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

文件以`.bat`作为后缀

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

记得加 `freopen`

随机数记得加`srand((int)time(0));`

随机数生成 code

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

int main(){
        freopen("data.txt", "w", stdout);
    srand((int)time(0));
    int T = rand() % 100000;
    cout << T << endl;

    for (int i = 0; i < T; i++){
        cout << rand() % 100;
    }
}
```

`rand()` 似乎只有三万多，需要更大的数的话要乘一下

#

图论

KM

```cpp
#include<bits/stdc++.h>

using namespace std;

const int inf = 0x3f3f3f3f;
const int maxN = 505;

namespace KM {
    int mp[maxN][maxN], link_x[maxN], link_y[maxN], N;
    bool visx[maxN], visy[maxN];
    int que[maxN << 1], top, fail, pre[maxN];
    int hx[maxN], hy[maxN], slk[maxN];

    inline int check(int i) {
        visx[i] = true;
        if (link_x[i]) {
            que[fail++] = link_x[i];
            return visy[link_x[i]] = true;
        }
        while (i) {
            link_x[i] = pre[i];
            swap(i, link_y[pre[i]]);
        }
        return 0;
    }
```

```
    void bfs(int S) {
        for (int i = 1; i <= N; i++) {
            slk[i] = inf;
            visx[i] = visy[i] = false;
        }
        top = 0;
        fail = 1;
        que[0] = S;
        visy[S] = true;
        while (true) {
            int d;
            while (top < fail) {
                for (int i = 1, j = que[top++]; i <= N; i++) {
                    if (!visx[i] && slk[i] >= (d = hx[i] + hy[j] - mp[i]
[j])) {
                        pre[i] = j;
                        if (d) slk[i] = d;
                        else if (!check(i)) return;
                    }
                }
            }
            d = inf;
            for (int i = 1; i <= N; i++) {
                if (!visx[i] && d > slk[i]) d = slk[i];
            }
            for (int i = 1; i <= N; i++) {
                if (visx[i]) hx[i] += d;
                else slk[i] -= d;
                if (visy[i]) hy[i] -= d;
            }
            for (int i = 1; i <= N; i++) {
                if (!visx[i] && !slk[i] && !check(i)) return;
            }
        }
    }

    void prework() {
        for (int i = 1; i <= N; i++) {
            link_x[i] = link_y[i] = 0;
            visy[i] = false;
        }
        for (int i = 1; i <= N; i++) {
            hx[i] = 0;
            for (int j = 1; j <= N; j++) {
                if (hx[i] < mp[i][j]) hx[i] = mp[i][j];
            }
        }
    }

    void init(int n) {
        N = n;
        top = fail = 0;
        for (int i = 1; i <= N; i++) {
            link_x[i] = link_y[i] = visx[i] = visy[i] = pre[i] = hx[i] =
hy[i] = slk[i] = 0;
            for (int j = 1; j <= N; j++) {
                mp[i][j] = 0;
            }
        }
    }
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);

    int n, m;
    cin >> n >> m;
    KM::init(max(n, m));
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            cin >> KM::mp[i][j];
        }
    }
    KM::prework();
    int ans = 0;
    for (int i = 1; i <= KM::N; i++) KM::bfs(i);
    for (int i = 1; i <= KM::N; i++) ans += KM::mp[i][KM::link_x[i]];

}
```

prufer 序列

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
```

```
using namespace std;

const int N = 100010;

int n, m;
int f[N], d[N], p[N];

void tree2prufer()
{
    for (int i = 1; i < n; i ++ )
    {
        scanf("%d", &f[i]);
        d[f[i]] ++ ;
    }

    for (int i = 0, j = 1; i < n - 2; j ++ )
    {
        while (d[j]) j ++ ;
        p[i ++ ] = f[j];
        while (i < n - 2 && -- d[p[i - 1]] == 0 && p[i - 1] < j) p[i ++ ]
 = f[p[i - 1]];
    }

    for (int i = 0; i < n - 2; i ++ ) printf("%d ", p[i]);
}

void prufer2tree()
{
    for (int i = 1; i <= n - 2; i ++ )
    {
        scanf("%d", &p[i]);
        d[p[i]] ++ ;
    }
    p[n - 1] = n;

    for (int i = 1, j = 1; i < n; i ++, j ++ )
    {
        while (d[j]) j ++ ;
        f[j] = p[i];
        while (i < n - 1 && -- d[p[i]] == 0 && p[i] < j) f[p[i]] = p[i +
1], i ++ ;
    }

    for (int i = 1; i <= n - 1; i ++ ) printf("%d ", f[i]);
}

int main()
{
    scanf("%d%d", &n, &m);
    if (m == 1) tree2prufer();
    else prufer2tree();

    return 0;
}
```

spfa 最短路及负环

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 1 << 20;
struct edge {
    ll to, len;
};

vector<edge> g[N];
ll d[N], cnt[N], vis[N];

bool spfa(ll s, ll n) {
    queue<int> que;
    for (int i = 1; i <= n; i++) {  //防止不连通，全加进去
        que.push(i);
        vis[i] = 1;
    }
    while (!que.empty()) {
        ll p = que.front();
        que.pop();
        vis[p] = 0;
        for (auto x:g[p]) {
            if (d[x.to] > d[p] + x.len) {
                d[x.to] = d[p] + x.len;
                cnt[x.to] = cnt[p] + 1;
                if (!vis[x.to]) {
                    if (cnt[x.to] > n) return 0;
                    vis[x.to] = 1;
                    que.push(x.to);
                }
            }
        }
```

```
    }
    return 1;
}
```

一些定理

Hall 定理：若二分图存在完美匹配，且大小为 n，那么取任意 1≤k≤n，均满足 X 集选出 k 个不同的点，它们连向 Y 集的点的个数不小于 k。

二分图匹配（HK 匈牙利匹配）

```
//大量使用了memset，但常数貌似很小？HDU6808 跑了998ms（限制5000ms），然而这个代int main()不是HDU6808 的
#include<bits/stdc++.h>
using namespace std;

const int maxn=505;// 最大点数
const int inf=0x3f3f3f3f;// 距离初始值
struct HK_Hungary{//这个板子从1 开始，0 点不能用,nx 为左边点数，ny 为右边点数
    int nx,ny;//左右顶点数量
    vector<int>bmap[maxn];
    int cx[maxn];//cx[i]表示左集合i 顶点所匹配的右集合的顶点序号
    int cy[maxn]; //cy[i]表示右集合i 顶点所匹配的左集合的顶点序号
    int dx[maxn];
    int dy[maxn];
    int dis;
    bool bmask[maxn];
    void init(int a,int b){
        nx=a,ny=b;
        for(int i=0;i<=nx;i++){
            bmap[i].clear();
        }
    }
    void add_edge(int u,int v){
        bmap[u].push_back(v);
    }
    bool searchpath(){//寻找 增广路径
        queue<int>Q;
        dis=inf;
        memset(dx,-1,sizeof(dx));
        memset(dy,-1,sizeof(dy));
        for(int i=1;i<=nx;i++){//cx[i]表示左集合i 顶点所匹配的右集合的顶点
序号
            if(cx[i]==-1){//将未遍历的节点 入队 并初始化次节点距离为0
                Q.push(i);
                dx[i]=0;
            }
        }//广度搜索增广路径
        while(!Q.empty()){
            int u=Q.front();
            Q.pop();
            if(dx[u]>dis) break;//取右侧节点
            for(int i=0;i<bmap[u].size();i++){
                int v=bmap[u][i];//右侧节点的增广路径的距离
                if(dy[v]==-1){
                    dy[v]=dx[u]+1;//v 对应的距离 为u 对应距离加1
                    if(cy[v]==-1)dis=dy[v];
                    else{
                        dx[cy[v]]=dy[v]+1;
                        Q.push(cy[v]);
                    }
                }
            }
        }
        return dis!=inf;
    }
    int findpath(int u){//寻找路径 深度搜索
        for(int i=0;i<bmap[u].size();i++){
            int v=bmap[u][i];//如果该点没有被遍历过 并且距离为上一节点+1
            if(!bmask[v]&&dy[v]==dx[u]+1){//对该点染色
                bmask[v]=1;
                if(cy[v]!=-1&&dy[v]==dis)continue;
                if(cy[v]==-1||findpath(cy[v])){
                    cy[v]=u;cx[u]=v;
                    return 1;
                }
            }
        }
        return 0;
    }
    int MaxMatch(){//得到最大匹配的数目
        int res=0;
        memset(cx,-1,sizeof(cx));
        memset(cy,-1,sizeof(cy));
        while(searchpath()){
            memset(bmask,0,sizeof(bmask));
            for(int i=1;i<=nx;i++){
                if(cx[i]==-1){
```

```
                res+=findpath(i);
            }
        }
      }
      return res;
    }
}HK;

int main(){
    int nn,n,m;
    cin>>nn;
    while(nn--){
        scanf("%d%d",&n,&m);
        HK.init(n,m);//左端点和右端点数量
        for(int i=1;i<=n;i++){
            int snum;
            cin>>snum;
            int v;
            for(int j=1;j<=snum;j++){
                cin>>v;
                HK.add_edge(i,v);//连边
            }
        }
        cout<<HK.MaxMatch()<<endl;//求最大匹配
    }
    return 0;
}
```

带花树

```
/*
����ʱ�� n ���� m ��� �����ϵ������ϵ�������þ�輸
�����ʽ��
��ʱ��ʱ����������ʾ���þ�������
�ڶ��� n ���������� i ������ʾ���� i þ��Ľ���ţ��
��ý�����þ��������� 0��
*/

#include<bits/stdc++.h>
using namespace std;
#define I inline int
#define V inline void
#define FOR(i,a,b) for(int i=a;i<=b;i++)
#define REP(u) for(int i=h[u],v;v=e[i].t,i;i=e[i].n)
const int N=1e3+1,M=1e5+1;
queue<int>q;
int n,m,tot,qwq,ans;
int h[N],lk[N],tag[N],fa[N],pre[N],dfn[N];
struct edge{int t,n;}e[M];
V link(int x,int y){lk[x]=y,lk[y]=x;}
V add_edge(int x,int y){
        if(!lk[x]&&!lk[y])link(x,y),ans++;
        e[++tot]=(edge){y,h[x]},h[x]=tot;
        e[++tot]=(edge){x,h[y]},h[y]=tot;
}
V rev(int x){if(x)rev(x[pre][lk]),link(x,pre[x]);}
I find(int x){return fa[x]==x?x:fa[x]=find(fa[x]);}
I lca(int x,int y){
        for(qwq++;;x=x[lk][pre],swap(x,y))
                if(dfn[x=find(x)]==qwq)return x;
                else if(x)dfn[x]=qwq;
}
V shrink(int x,int y,int p){
        for(;find(x)!=p;x=pre[y]){
                pre[x]=y,y=lk[x],fa[x]=fa[y]=p;
                if(tag[y]==2)tag[y]=1,q.push(y);
        }
}
I blossom(int u){
        FOR(i,1,n)tag[i]=pre[i]=0,fa[i]=i;
        tag[u]=1,q=queue<int>(),q.push(u);
        for(int p;!q.empty();q.pop())REP(u=q.front())
                if(tag[v]==1)
                        p=lca(u,v),shrink(u,v,p),shrink(v,u,p);
                else if(!tag[v]){
                        pre[v]=u,tag[v]=2;
                        if(!lk[v])return rev(v),1;
                        else tag[lk[v]]=1,q.push(lk[v]);
                }
        return 0;
}
int main(){
        scanf("%d%d",&n,&m);
        for(int x,y;m--;add_edge(x,y))scanf("%d%d",&x,&y);
        FOR(i,1,n)ans+=!lk[i]&&blossom(i);
        cout<<ans<<'\n';
        FOR(i,1,n)cout<<lk[i]<<' ';
        return 0;
}
```

带花树 2

```
// graph
template <typename T>
class graph {
 public:
  struct edge {
    int from;
    int to;
    T cost;
  };
  vector<edge> edges;
  vector<vector<int> > g;
  int n;
  graph(int _n) : n(_n) { g.resize(n); }
  virtual int add(int from, int to, T cost) = 0;
};

// undirectedgraph
template <typename T>
class undirectedgraph : public graph<T> {
 public:
  using graph<T>::edges;
  using graph<T>::g;
  using graph<T>::n;

  undirectedgraph(int _n) : graph<T>(_n) {}
  int add(int from, int to, T cost = 1) {
    assert(0 <= from && from < n && 0 <= to && to < n);
    int id = (int)edges.size();
    g[from].push_back(id);
    g[to].push_back(id);
    edges.push_back({from, to, cost});
    return id;
  }
};

// blossom / find_max_unweighted_matching
template <typename T>
vector<int> find_max_unweighted_matching(const undirectedgraph<T> &g) {
  std::mt19937 rng(chrono::steady_clock::now().time_since_epoch().count
());
  vector<int> match(g.n, -1);   // ƥ��
  vector<int> aux(g.n, -1);     // ʱ�����
  vector<int> label(g.n);       // "o" or "i"
  vector<int> orig(g.n);        // ����
  vector<int> parent(g.n, -1);  // ���ڵ�
  queue<int> q;
  int aux_time = -1;

  auto lca = [&](int v, int u) {
    aux_time++;
    while (true) {
      if (v != -1) {
        if (aux[v] == aux_time) {  // �ҵ��ù��ĵ� ҲΪ����LCA
          return v;
        }
        aux[v] = aux_time;
        if (match[v] == -1) {
          v = -1;
        } else {
          v = orig[parent[match[v]]];  // ��ƥ���Ӧ��ĸ��ڵ��������
        }
      }
      swap(v, u);
    }
  };  // lca

  auto blossom = [&](int v, int u, int a) {
    while (orig[v] != a) {
      parent[v] = u;
      u = match[v];
      if (label[u] == 1) {  // ��ʹ����Ï"o" ������·
        label[u] = 0;
        q.push(u);
      }
      orig[v] = orig[u] = a;  // ����
      v = parent[u];
    }
  };  // blossom

  auto augment = [&](int v) {
    while (v != -1) {
      int pv = parent[v];
      int next_v = match[pv];
      match[v] = pv;
      match[pv] = v;
      v = next_v;
    }
  };  // augment

  auto bfs = [&](int root) {
```

```
  fill(label.begin(), label.end(), -1);
  iota(orig.begin(), orig.end(), 0);
  while (!q.empty()) {
    q.pop();
  }
  q.push(root);
  // ��'����Ï "o", ������"0"����"o", "1"����"i"
  label[root] = 0;
  while (!q.empty()) {
    int v = q.front();
    q.pop();
    for (int id : g.g[v]) {
      auto &e = g.edges[id];
      int u = e.from ^ e.to ^ v;
      if (label[u] == -1) {  // �ũ�δ�õ�
        label[u] = 1;        // ��� "i"
        parent[u] = v;
        if (match[u] == -1) {  // �ũ�δþ���
          augment(u);          // Ψ������·��
          return true;
        }
        // �ũ����þ��� �������þ��ĵ灭��queue ���쿉����
        label[match[u]] = 0;
        q.push(match[u]);
        continue;
      } else if (label[u] == 0 && orig[v] != orig[u]) {
        // �ũ���Ψõ� �ɣ�"Ï"o" �����ũ�"��"
        int a = lca(orig[v], orig[u]);
        // ��LCA Ȼ������
        blossom(u, v, a);
        blossom(v, u, a);
      }
    }
  }
  return false;
}; // bfs

auto greedy = [&]() {
  vector<int> order(g.n);
  // ������� order
  iota(order.begin(), order.end(), 0);
  shuffle(order.begin(), order.end(), rng);

  // ������þ��ĵþ��
  for (int i : order) {
    if (match[i] == -1) {
      for (auto id : g.g[i]) {
        auto &e = g.edges[id];
        int to = e.from ^ e.to ^ i;
        if (match[to] == -1) {
          match[i] = to;
          match[to] = i;
          break;
        }
      }
    }
  }
}; // greedy

// ʜ��'������þ��
greedy();
// ��δþ���������·
for (int i = 0; i < g.n; i++) {
  if (match[i] == -1) {
    bfs(i);
  }
}
return match;
}
```

强连通（**kosaraju**）

```
#include <bits/stdc++.h>
using namespace std;
struct SCC {
    static const int MAXV = 100000;
    int V;
    vector<int> g[MAXV], rg[MAXV], vs;
    bool used[MAXV];
    int cmp[MAXV];

    void add_edge(int from, int to) {
        g[from].push_back(to);
        rg[to].push_back(from);
    }

    void dfs(int v) {
        used[v] = 1;
        for (int i = 0; i < g[v].size(); i++) {
            if (!used[g[v][i]]) dfs(g[v][i]);
```

```
        }
        vs.push_back(v);
    }

    void rdfs(int v, int k) {
        used[v] = 1;
        cmp[v] = k;
        for (int i = 0; i < rg[v].size(); i++) {
            if (!used[rg[v][i]]) rdfs(rg[v][i], k);
        }
    }

    int solve() {
        memset(used, 0, sizeof(used));
        vs.clear();
        for (int v = 1; v <= V; v++) {
            if (!used[v]) dfs(v);
        }
        memset(used, 0, sizeof(used));
        int k = 0;
        for (int i = (int)vs.size() - 1; i >= 0; i--) {
            if (!used[vs[i]]) rdfs(vs[i], ++k);
        }
        return k;
    }

    void init(int n) {
        V = n;
        vs.clear();
        for (int i = 0; i < MAXV; i++) {
            g[i].clear();
            rg[i].clear();
            used[i] = 0;
            cmp[i] = 0;
        }
    }

} scc;

//记得调用init()
```

强连通（tarjan 无 vector）

```
#include <bits/stdc++.h>
using namespace std;
struct SCC {
    static const int MAXN = 5000;
    static const int MAXM = 2000000;
    int dfs_clock, edge_cnt = 1, scc_cnt;
    int head[MAXN];
    int dfn[MAXN], lowlink[MAXN];
    int sccno[MAXN];
    stack<int> s;

    struct edge {
        int v, next;
    } e[MAXM];

    void add_edge(int u, int v) {
        e[edge_cnt].v = v;
        e[edge_cnt].next = head[u];
        head[u] = edge_cnt++;
    }

    void tarjan(int u) {
        int v;
        dfn[u] = lowlink[u] = ++dfs_clock;  //每次dfs，u的次序号增加1
        s.push(u);                          //将u入栈
        for (int i = head[u]; i != -1; i = e[i].next)  //访问从u出发的边
        {
            v = e[i].v;
            if (!dfn[v])  //如果v没被处理过
            {
                tarjan(v);  // dfs(v)
                lowlink[u] = min(lowlink[u], lowlink[v]);
            } else if (!sccno[v])
                lowlink[u] = min(lowlink[u], dfn[v]);
        }
        if (dfn[u] == lowlink[u]) {
            scc_cnt++;
            do {
                v = s.top();
                s.pop();
                sccno[v] = scc_cnt;
            } while (u != v);
        }
    }

    int find_scc(int n) {
```

```
        for (int i = 1; i <= n; i++)
            if (!dfn[i]) tarjan(i);
        return scc_cnt;
    }

    void init() {
        scc_cnt = dfs_clock = 0;
        edge_cnt = 1;  //不用初始化 e 数组，省时间
        while (!s.empty()) s.pop();
        memset(head, -1, sizeof(head));
        memset(sccno, 0, sizeof(sccno));
        memset(dfn, 0, sizeof(dfn));
        memset(lowlink, 0, sizeof(lowlink));
    }
} scc;
```

强连通（tarjan）

```
#include <bits/stdc++.h>
using namespace std;

struct SCC {
    static const int MAXN = 100000;
    vector<int> g[MAXN];
    int dfn[MAXN], lowlink[MAXN], sccno[MAXN], dfs_clock, scc_cnt;
    stack<int> S;

    void dfs(int u) {
        dfn[u] = lowlink[u] = ++dfs_clock;
        S.push(u);
        for (int i = 0; i < g[u].size(); i++) {
            int v = g[u][i];
            if (!dfn[v]) {
                dfs(v);
                lowlink[u] = min(lowlink[u], lowlink[v]);
            } else if (!sccno[v]) {
                lowlink[u] = min(lowlink[u], dfn[v]);
            }
        }
        if (lowlink[u] == dfn[u]) {
            ++scc_cnt;
            for (;;) {
                int x = S.top();
                S.pop();
                sccno[x] = scc_cnt;
                if (x == u) break;
            }
        }
    }

    void solve(int n) {
        dfs_clock = scc_cnt = 0;
        memset(sccno, 0, sizeof(sccno));
        memset(dfn, 0, sizeof(dfn));
        memset(lowlink, 0, sizeof(lowlink));
        for (int i = 1; i <= n; i++) {
            if (!dfn[i]) dfs(i);
        }
    }
} scc;

// scc_cnt 为SCC 计数器，sccno[i]为i 所在SCC 的编号
// vector<int> g[MAXN]中加边
//之后再补充 init()
```

拓扑排序

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 100000;

int c[MAXN];
int topo[MAXN], t, V;
vector<int> g[MAXN];

bool dfs(int u) {
    c[u] = -1;
    for (int i = 0; i < g[u].size(); i++) {
        int v = g[u][i];
        if (c[v] < 0)
            return false;
        else if (!c[v] && !dfs(v))
            return false;
    }
    c[u] = 1;
    topo[t--] = u;
    return true;
}
```

```
bool toposort(int n) {
    V = n;
    t = n;
    memset(c, 0, sizeof(c));
    for (int u = 1; u <= V; u++)
        if (!c[u] && !dfs(u)) return false;
    return true;
}
```

数链剖分

```
ll fa[N], son[N], dep[N], siz[N], dfn[N], rnk[N], top[N];
ll dfscnt;
vector<ll> g[N];
ll tree[N << 1];
ll lazy[N << 1];

void dfs1(ll u, ll f, ll d) {
    son[u] = -1;
    siz[u] = 1;
    fa[u] = f;
    dep[u] = d;
    for (auto v:g[u]) {
        if (v == f) continue;
        dfs1(v, u, d + 1);
        siz[u] += siz[v];
        if (son[u] == -1 || siz[v] > siz[son[u]]) son[u] = v;
    }
}

void dfs2(ll u, ll t) {
    dfn[u] = ++dfscnt;
    rnk[dfscnt] = u;
    top[u] = t;
    if (son[u] == -1) return;
    dfs2(son[u], t);
    for (auto v:g[u]) {
        if (v == son[u] || v == fa[u]) continue;
        dfs2(v, v);
    }
}

ll lca(ll a, ll b) {
    while (top[a] != top[b]) {
        if (dep[top[a]] < dep[top[b]]) swap(a, b);
        a = fa[top[a]];
    }
    return dep[a] < dep[b] ? a : b;
}

void init() {
    for (ll i = 0; i < N; i++) g[i].clear();
    for (ll i = 0; i < (N << 1); i++) {
        tree[i] = 0;
        lazy[i] = 0;
    }
    dfscnt = 0;
}


void pushdown(ll k, ll l, ll r) {
    if (k >= N || lazy[k] == 0) return;
    ll len = (r - l + 1) / 2;
    tree[k << 1] = tree[k << 1] + len * lazy[k];
    tree[k << 1 | 1] = tree[k << 1 | 1] + len * lazy[k];
    lazy[k << 1] = lazy[k << 1] + lazy[k];
    lazy[k << 1 | 1] = lazy[k << 1 | 1] + lazy[k];
    lazy[k] = 0;
}

ll merge_range(ll a, ll b) {
    ll ans = a + b;
    return ans;
}

void change_range(ll k, ll l, ll r, ll ql, ll qr, ll x) {
    if (r < ql || qr < l)return;
    if (ql <= l && r <= qr) {
        tree[k] = tree[k] + x * (r - l + 1);
        lazy[k] = lazy[k] + x;
        return;
    }
    pushdown(k, l, r);
    ll mid = (l + r) >> 1;
    change_range(k << 1, l, mid, ql, qr, x);
    change_range(k << 1 | 1, mid + 1, r, ql, qr, x);
    tree[k] = merge_range(tree[k << 1], tree[k << 1 | 1]);
}
```

```cpp
ll query_range(ll k, ll l, ll r, ll ql, ll qr) {
    if (r < ql || qr < l)return 0;
    if (ql <= l && r <= qr) {
        return tree[k];
    }
    pushdown(k, l, r);
    ll mid = (l + r) >> 1;
    ll lq = query_range(k << 1, l, mid, ql, qr);
    ll rq = query_range(k << 1 | 1, mid + 1, r, ql, qr);
    return merge_range(lq, rq);
}

ll query_path(ll a, ll b) {
    ll sum = 0;
    while (top[a] != top[b]) {
        if (dep[top[a]] < dep[top[b]]) swap(a, b);
        sum = sum + query_range(1, 1, N, dfn[top[a]], dfn[a]);
        //dfn[top[a]]~dfn[a]
        a = fa[top[a]];
    }
    if (dep[a] > dep[b]) swap(a, b);
    //点权
    sum = sum + query_range(1, 1, N, dfn[a], dfn[b]);
    //边权
    //if (a != b) sum = sum + query_range(1, 1, N, dfn[a] + 1, dfn[b]);
    //dfn[a]~dfn[b],x
    return sum;
}

void change_path(ll a, ll b, ll x) {
    while (top[a] != top[b]) {
        if (dep[top[a]] < dep[top[b]]) swap(a, b);
        change_range(1, 1, N, dfn[top[a]], dfn[a], x);
        //dfn[top[a]]~dfn[a]
        a = fa[top[a]];
    }
    if (dep[a] > dep[b]) swap(a, b);
    //点权
    change_range(1, 1, N, dfn[a], dfn[b], x);
    //边权
    //if (a != b) change_range(1, 1, N, dfn[a] + 1, dfn[b], x);
    //dfn[a]~dfn[b],x
}
```

最大流

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

struct Edge {
    ll from, to, cap, flow;
    Edge(ll a, ll b, ll c, ll d) : from(a), to(b), cap(c), flow(d) {}
};

struct Dinic {
    static const ll maxn = 10000;
    static const ll inf = 0x3f3f3f3f3f3f3f3f;
    ll N, M, S, T;
    vector<Edge> edges;
    vector<ll> G[maxn];
    bool vis[maxn];
    ll d[maxn];
    ll cur[maxn];

    void AddEdge(ll from, ll to, ll cap) {
        edges.push_back(Edge(from, to, cap, 0));
        edges.push_back(Edge(to, from, 0, 0));
        M = edges.size();
        G[from].push_back(M - 2);
        G[to].push_back(M - 1);
    }

    bool BFS() {
        memset(vis, 0, sizeof(vis));
        queue<ll> Q;
        Q.push(S);
        d[S] = 0;
        vis[S] = 1;
        while (!Q.empty()) {
            ll x = Q.front();
            Q.pop();
            for (ll i = 0; i < G[x].size(); i++) {
                Edge& e = edges[G[x][i]];
                if (!vis[e.to] && e.cap > e.flow) {
                    vis[e.to] = 1;
                    d[e.to] = d[x] + 1;
                    Q.push(e.to);
                }
            }
```

```
        }
        return vis[T];
    }

    ll DFS(ll x, ll a) {
        if (x == T || a == 0) return a;
        ll flow = 0, f;
        for (ll& i = cur[x]; i < G[x].size(); i++) {
            Edge& e = edges[G[x][i]];
            if (d[x] + 1 == d[e.to] &&
                (f = DFS(e.to, min(a, e.cap - e.flow))) > 0) {
                e.flow += f;
                edges[G[x][i] ^ 1].flow -= f;
                flow += f;
                a -= f;
                if (a == 0) break;
            }
        }
        return flow;
    }

    ll Maxflow(ll S, ll T) {
        this->S = S, this->T = T;
        ll flow = 0;
        while (BFS()) {
            memset(cur, 0, sizeof(cur));
            flow += DFS(S, inf);
        }
        return flow;
    }
} MF;
```

//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

//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

//最大权闭合子图
//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，
//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。
//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。

//最大密度子图
//01 分数规划
//addedge(S,V,m),addedge(E,1),addedge(V,T,2*g-deg(v)+m)
//h(g)=n*m-maxflow(S,T)

最大流（double）

```
#include <iostream>
#include <cstring>
#include <algorithm>

using namespace std;

struct Dinic {
        static constexpr int N = 10010, M = 100010, INF = 1e8;
        static constexpr double eps = 1e-8;
//      int n, m, S, T;
        int S, T;
        int h[N], e[M], ne[M], idx;
        double f[M];
        int q[N], d[N], cur[N]; // d ��'��ʾ�ĸ�'� ��õ��·���� ���б ��������C_

        void AddEdge(int a, int b, double c)
        {
            e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx ++ ;
            e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx ++ ;
        }

        bool bfs()
        {
            int hh = 0, tt = 0;
            memset(d, -1, sizeof d);
            q[0] = S, d[S] = 0, cur[S] = h[S];
            while (hh <= tt)
            {
                int t = q[hh ++ ];
                for (int i = h[t]; ~i; i = ne[i])
                {
                    int ver = e[i];
                    if (d[ver] == -1 && f[i] > 0)
                    {
                        d[ver] = d[t] + 1;
```

```
                    cur[ver] = h[ver];
                    if (ver == T) return true;
                    q[ ++ tt] = ver;
                }
            }
        }
        return false;
    }

    double find(int u, double limit)
    {
        if (u == T) return limit;
        double flow = 0;
        for (int i = cur[u]; ~i && flow < limit; i = ne[i])
        {
            cur[u] = i;
            int ver = e[i];
            if (d[ver] == d[u] + 1 && f[i] > 0)
            {
                double t = find(ver, min(f[i], limit - flow));
                if (t < eps) d[ver] = -1;
                f[i] -= t, f[i ^ 1] += t, flow += t;
            }
        }
        return flow;
    }

    double Maxflow(int S, int T)
    {
            this->S = S, this->T = T;
        double r = 0, flow;
        while (bfs()) while (flow = find(S, INF)) r += flow;
        return r;
    }
    void init() ////////
    {
            memset(h, -1, sizeof h);
            idx = 0;
    }
} MF;

// ?��init
```

最小费用最大流

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

struct Edge {
    ll from, to, cap, flow, cost;
    Edge(ll u, ll v, ll c, ll f, ll w):from(u), to(v), cap(c), flow(f),
cost(w) {}
};

struct MCMF {
    static const ll maxn = 6000;
    static const ll INF = 0x3f3f3f3f3f3f3f3f;
    ll n, m;
    vector<Edge> edges;
    vector<ll> G[maxn];
    ll inq[maxn];
    ll d[maxn];
    ll p[maxn];
    ll a[maxn];

    void init(ll n) {
        this->n = n;
        for (ll i = 1; i <= n; i++) G[i].clear();
        edges.clear();
    }

    void add_edge(ll from, ll to, ll cap, ll cost) {
        from++,to++;//原板子无法使用0点，故修改
        edges.push_back(Edge(from, to, cap, 0, cost));
        edges.push_back(Edge(to, from, 0, 0, -cost));
        m = edges.size();
        G[from].push_back(m - 2);
        G[to].push_back(m - 1);
    }

    bool BellmanFord(ll s, ll t, ll& flow, ll& cost) {
        for (ll i = 1; i <= n; ++i) d[i] = INF;
        memset(inq, 0, sizeof(inq));
        d[s] = 0, inq[s] = 1, p[s] = 0, a[s] = INF;
        queue<ll> Q;
        Q.push(s);
        while (!Q.empty()) {
```

```
            ll u = Q.front();
            Q.pop();
            inq[u] = 0;
            for (ll i = 0; i < G[u].size(); ++i) {
                Edge& e = edges[G[u][i]];
                if (e.cap > e.flow && d[e.to] > d[u] + e.cost) {
                    d[e.to] = d[u] + e.cost;
                    p[e.to] = G[u][i];
                    a[e.to] = min(a[u], e.cap - e.flow);
                    if (!inq[e.to]) {
                        Q.push(e.to);
                        inq[e.to] = 1;
                    }
                }
            }
        }
        if (d[t] == INF) return false;
        flow += a[t];
        cost += (ll)d[t] * (ll)a[t];
        for (ll u = t; u != s; u = edges[p[u]].from) {
            edges[p[u]].flow += a[t];
            edges[p[u] ^ 1].flow -= a[t];
        }
        return true;
    }

    //需要保证初始网络中没有负权圈
    ll MincostMaxflow(ll s, ll t, ll& cost) {
        s++,t++;//原板子无法使用0点，故修改
        ll flow = 0;
        cost = 0;
        while (BellmanFord(s, t, flow, cost));
        return flow;
    }
} mcmf;  //  若固定流量k，增广时在flow+a>=k的时候只增广k-flow单位的流量，然后终止程序
//下标从0开始
```

最小路径覆盖

对于有向无环图（DAG）

定义：在一个有向图中，找出最少的路径，使得这些路径经过了所有的点。

最小路径覆盖分为**最小不相交路径覆盖**和**最小可相交路径覆盖**。

最小不相交路径覆盖：每一条路径经过的顶点各不相同。

最小可相交路径覆盖：每一条路径经过的顶点可以相同。

DAG 的最小不相交路径覆盖：

把原图的每个点 v 拆成v_x和v_y两个点，如果有一条有向边$A \to B$，就加边$A_x \to B_y$，这样就得到一个二分图，最小路径覆盖=原图的节点数-新图的最大匹配数

DAG 的最小可相交路径覆盖：

先用 floyd 求出原图的传递闭包，即若 a 到 b 有路径，则加边$a \to b$，转化为最小不相交路径覆盖问题

最近公共祖先（倍增）

```
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <iostream>
using namespace std;
const int MAX = 600000;

struct edge {
    int t, nex;
} e[MAX << 1];
int head[MAX], tot;

int depth[MAX], fa[MAX][22], lg[MAX];

void add_edge(int x, int y) {
    e[++tot].t = y;
    e[tot].nex = head[x];
    head[x] = tot;

    e[++tot].t = x;
    e[tot].nex = head[y];
    head[y] = tot;
}

void dfs(int now, int fath) {
    fa[now][0] = fath;
    depth[now] = depth[fath] + 1;
```

```
    for (int i = 1; i <= lg[depth[now]]; ++i)
        fa[now][i] = fa[fa[now][i - 1]][i - 1];
    for (int i = head[now]; i; i = e[i].nex)
        if (e[i].t != fath) dfs(e[i].t, now);
}

int lca(int x, int y) {
    if (depth[x] < depth[y]) swap(x, y);
    while (depth[x] > depth[y]) x = fa[x][lg[depth[x] - depth[y]] - 1];
    if (x == y) return x;
    for (int k = lg[depth[x]] - 1; k >= 0; --k)
        if (fa[x][k] != fa[y][k]) x = fa[x][k], y = fa[y][k];
    return fa[x][0];
}

void init(int n, int root) {
    for (int i = 1; i <= n; ++i) lg[i] = lg[i - 1] + (1 << lg[i - 1] ==
i);
    dfs(root, 0);
}
```

最近公共祖先（线段树）

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

有源汇上下界最大小流

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

struct Edge {
    ll from, to, cap, flow, mn;
    Edge(ll a, ll b, ll c, ll d, ll e) : from(a), to(b), cap(c), flow(d),
 mn(e) {}
};

ll n, m;

struct Dinic {
    static const ll maxn = 50010; // 点的大小，记得改
    static const ll inf = 0x3f3f3f3f3f3f3f3f;
    ll N, M, S, T;
    vector<Edge> edges;
    vector<ll> G[maxn];
    bool vis[maxn];
    ll d[maxn];
    ll cur[maxn];

    void AddEdge(ll from, ll to, ll cap, ll c) {
        edges.push_back(Edge(from, to, cap, 0, c));
        edges.push_back(Edge(to, from, 0, 0, c));
        M = edges.size();
        G[from].push_back(M - 2);
        G[to].push_back(M - 1);
    }

    bool BFS() {
        memset(vis, 0, sizeof(vis));
        queue<ll> Q;
        Q.push(S);
        d[S] = 0;
        vis[S] = 1;
        while (!Q.empty()) {
            ll x = Q.front();
            Q.pop();
            for (ll i = 0; i < G[x].size(); i++) {
                Edge& e = edges[G[x][i]];
                if (!vis[e.to] && e.cap > e.flow) {
                    vis[e.to] = 1;
                    d[e.to] = d[x] + 1;
                    Q.push(e.to);
                }
            }
        }
        return vis[T];
    }

    ll DFS(ll x, ll a) {
        if (x == T || a == 0) return a;
        ll flow = 0, f;
        for (ll& i = cur[x]; i < G[x].size(); i++) {
            Edge& e = edges[G[x][i]];
            if (d[x] + 1 == d[e.to] &&
                (f = DFS(e.to, min(a, e.cap - e.flow))) > 0) {
                e.flow += f;
                edges[G[x][i] ^ 1].flow -= f;
                flow += f;
                a -= f;
                if (a == 0) break;
            }
        }
        return flow;
    }

    void deleteEdge(ll u, ll v) {
        ll siz = edges.size();
        for(ll i = 0; i < siz; ++ i) {
            if(edges[i].from == u && edges[i].to == v) {
                edges[i].cap = edges[i].flow = 0;
                edges[i ^ 1].cap = edges[i ^ 1].flow = 0;
                break;
            }

        }

    }

    ll getValue() {
        return edges[2 * m].flow;
    }

    ll Maxflow(ll S, ll T) {
        this->S = S, this->T = T;
        ll flow = 0;
        while (BFS()) {
```

```
            memset(cur, 0, sizeof(cur));
            flow += DFS(S, inf);
        }
        return flow;
    }
} MF;

int main() {
    ll s, t;
    cin >> n >> m >> s >> t;
  // n 个点，m 条边，给的源点汇点

    ll mp[50010] = {0}; // 点的大小，记得改
    for(ll i = 1; i <= m; ++ i) {
        ll a, b, c, d; // 从 a 到 b 有一条下界 c 上界 d 的边
        cin >> a >> b >> c >> d;
        mp[b] += c;
        mp[a] -= c;
        MF.AddEdge(a, b, d - c, c);
    }
    MF.AddEdge(t, s, 1e18, 0); //
    ll tot = 0;
    for(ll i = 1; i <= n; ++ i) {
        if(mp[i] > 0) {
            tot += mp[i];
            MF.AddEdge(0, i , mp[i], 0);
        }
        else {
            MF.AddEdge(i, n + 1, -mp[i], 0);
        }
    }

    if( MF.Maxflow(0, n + 1) != tot) {
        cout << "No Solution" << endl;
    }
    else {
        ll res = MF.getValue(); // 从 t 到 s 边的流量
        MF.deleteEdge(t, s);
      //cout << res + MF.Maxflow(s, t) << endl; // 最大流
        cout << res - MF.Maxflow(t, s) << endl; // 最小流
    }

    return 0;
}
```

朱刘算法

```
#include <iostream>
#include <cstring>
#include <cstdio>
#include <algorithm>
#include <cmath>

#define x first
#define y second

using namespace std;

typedef pair<double, double> PDD;

const int N = 110;
const double INF = 1e8;

int n, m;
PDD q[N];
bool g[N][N];
double d[N][N], bd[N][N];
int pre[N], bpre[N];
int dfn[N], low[N], ts, stk[N], top;
int id[N], cnt;
bool st[N], ins[N];

void dfs(int u) {
    st[u] = true;
    for (int i = 1; i <= n; i++)
        if (g[u][i] && !st[i])
            dfs(i);
}

bool check_con() {
    memset(st, 0, sizeof st);
    dfs(1);
    for (int i = 1; i <= n; i++)
        if (!st[i])
            return false;
    return true;
}

double get_dist(int a, int b) {
    double dx = q[a].x - q[b].x;
```

```cpp
    double dy = q[a].y - q[b].y;
    return sqrt(dx * dx + dy * dy);
}

void tarjan(int u) {
    dfn[u] = low[u] = ++ts;
    stk[++top] = u, ins[u] = true;

    int j = pre[u];
    if (!dfn[j]) {
        tarjan(j);
        low[u] = min(low[u], low[j]);
    } else if (ins[j]) low[u] = min(low[u], dfn[j]);

    if (low[u] == dfn[u]) {
        int y;
        ++cnt;
        do {
            y = stk[top--], ins[y] = false, id[y] = cnt;
        } while (y != u);
    }
}

double work() {
    double res = 0;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            if (g[i][j]) d[i][j] = get_dist(i, j);
            else d[i][j] = INF;

    while (true) {
        for (int i = 1; i <= n; i++) {
            pre[i] = i;
            for (int j = 1; j <= n; j++)
                if (d[pre[i]][i] > d[j][i])
                    pre[i] = j;
        }

        memset(dfn, 0, sizeof dfn);
        ts = cnt = 0;
        for (int i = 1; i <= n; i++)
            if (!dfn[i])
                tarjan(i);

        if (cnt == n) {
            for (int i = 2; i <= n; i++) res += d[pre[i]][i];
            break;
        }

        for (int i = 2; i <= n; i++)
            if (id[pre[i]] == id[i])
                res += d[pre[i]][i];

        for (int i = 1; i <= cnt; i++)
            for (int j = 1; j <= cnt; j++)
                bd[i][j] = INF;

        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++)
                if (d[i][j] < INF && id[i] != id[j]) {
                    int a = id[i], b = id[j];
                    if (id[pre[j]] == id[j]) bd[a][b] = min(bd[a][b], d[i]
[j] - d[pre[j]][j]);
                    else bd[a][b] = min(bd[a][b], d[i][j]);
                }

        n = cnt;
        memcpy(d, bd, sizeof d);
    }

    return res;
}

int main() {
    while (~scanf("%d%d", &n, &m)) {
        for (int i = 1; i <= n; i++) scanf("%lf%lf", &q[i].x, &q[i].y);

        memset(g, 0, sizeof g);
        while (m--) {
            int a, b;
            scanf("%d%d", &a, &b);
            if (a != b && b != 1) g[a][b] = true;
        }

        if (!check_con()) puts("poor snoopy");
        else printf("%.2lf\n", work());
    }

    return 0;
}
```

树上启发式合并

```cpp
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;
const int N = 2e5 + 10;

int vis[N], now;

vector<int> g[N];
int fa[N], son[N], siz[N], ans[N];

void insert(int pos) {
    vis[pos] = 1;
    now = now + 1 - vis[pos - 1] - vis[pos + 1];
}

void remove(int pos) {
    vis[pos] = 0;
    now = now - 1 + vis[pos - 1] + vis[pos + 1];
}

void dfs1(ll u, ll f) {
    siz[u] = 1;
    fa[u] = f;
    son[u] = -1;
    for (auto v:g[u]) {
        if (v == f) continue;
        dfs1(v, u);
        siz[u] += siz[v];
        if (son[u] == -1 || siz[v] > siz[son[u]]) son[u] = v;
    }
}

void add(int u, int exc, int op) {
    if (op) insert(u);
    else remove(u);
    for (auto x:g[u]) {
        if (x == fa[u] || x == exc) continue;
        add(x, exc, op);
    }
}

void dfs(ll u, ll opt) {
    for (auto x:g[u]) {
        if (x == fa[u] || x == son[u]) continue;
        dfs(x, 0);
    }
    if (son[u] != -1) dfs(son[u], 1);
    add(u, son[u], 1);
    ans[u] = now;
    if (!opt) {
        add(u, 0, 0);
    }
}


int main() {
    ios::sync_with_stdio(false),
            cin.tie(nullptr),
            cout.tie(nullptr);
    int t;
    cin >> t;
    int test = 0;
    while (t--) {
        int n;
        cin >> n;
        for (int i = 1; i < n; i++) {
            int a, b;
            cin >> a >> b;
            g[a].push_back(b);
            g[b].push_back(a);
        }
        cout << "Case #" << ++test << ": ";
        dfs1(1, -1);
        dfs(1, 0);
        for (int i = 1; i <= n; i++) {
            if (i != 1) cout << ' ';
            cout << ans[i];
        }
        cout << endl;
        for (int i = 1; i <= n; i++) g[i].clear();
    }
}
```

树分治

```cpp
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 10005;
const int INF = 1000000000;
struct edge {
    int to, length;
    edge() {}
    edge(int a, int b) : to(a), length(b) {}
};


vector<edge> g[MAXN];

bool centroid[MAXN];
int subtree_size[MAXN];

int ans;

//计算子树大小
int compute_subtree_size(int v, int p) {
    int c = 1;
    for (int i = 0; i < g[v].size(); i++) {
        int w = g[v][i].to;
        if (w == p || centroid[w]) continue;
        c += compute_subtree_size(w, v);
    }
    subtree_size[v] = c;
    return c;
}

//查找重心，t 为连通分量大小
// pair（最大子树顶点数，顶点编号）
pair<int, int> search_centroid(int v, int p, int t) {
    pair<int, int> res = pair<int, int>(INF, -1);
    int s = 1, m = 0;
    for (int i = 0; i < g[v].size(); i++) {
        int w = g[v][i].to;
        if (w == p || centroid[w]) continue;
        res = min(res, search_centroid(w, v, t));
        m = max(m, subtree_size[w]);
        s += subtree_size[w];
    }
    m = max(m, t - s);
    res = min(res, pair<int, int>(m, v));
    return res;
}

void init(int n) {
    memset(centroid, 0, sizeof(centroid));
    memset(subtree_size, 0, sizeof(subtree_size));
    for (int i = 0; i <= n; i++) g[i].clear();
    ans = 0;
}

int solve(int u) {
    compute_subtree_size(u, -1);
    int s = search_centroid(u, -1, subtree_size[u]).second;
    centroid[s] = 1;
    for (int i = 0; i < g[s].size(); i++) {
        int v = g[s][i].to;
        if (centroid[v]) continue;
        /*solve()*/
    }
    /*do something*/
    centroid[s] = 0;
    return ans;
}
```

欧拉回路

```cpp
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;
const int N = 1e6 + 10;


int stk[N], top;
struct edge {
    int to, idx;
};

vector<edge> g[N];

namespace Euler1 {  //无向图欧拉回路
    bool vis[N];
    int cur[N];
```

```cpp
    void dfs(int u, const int &w) {
        vis[abs(w)] = true;
        for (int &i = cur[u]; i < g[u].size();) {
            int idx = g[u][i].idx, v = g[u][i].to;
            i++;
            if (!vis[abs(idx)]) dfs(v, idx);
        }
        stk[++top] = w;
    }

    bool solve(int n) {
        // init();
        for (int i = 0; i <= n; i++) cur[i] = 0;
        for (int i = 0; i <= n; i++) vis[i] = 0;
        // calculate degree
        for (int i = 1; i <= n; i++) {
            if (g[i].size() & 1) return false;
        }
        // Hierholzer
        for (int i = 1; i <= n; i++)
            if (!g[i].empty()) {
                dfs(i, 0);
                break;
            }
        return true;
    }
}  // namespace Euler1

namespace Euler2 {  // 有向图欧拉回路
    int deg[N], cur[N];

    void dfs(int u, const int &w) {
        for (int &i = cur[u]; i < g[u].size();) {
            int idx = g[u][i].idx, v = g[u][i].to;
            i++;
            dfs(v, idx);
        }
        stk[++top] = w;
    }

    bool solve(int n) {
        // init
        for (int i = 0; i <= n; i++) deg[i] = 0;
        for (int i = 0; i <= n; i++) cur[i] = 0;
        // calculate degree
        for (int i = 1; i <= n; ++i) {
            for (auto x: g[i]) deg[i]++, deg[x.to]--;
        }
        for (int i = 1; i <= n; ++i)
            if (deg[i]) return false;
        // Hierholzer
        for (int i = 1; i <= n; ++i)
            if (!g[i].empty()) {
                dfs(i, 0);
                break;
            }
        return true;
    }
}  // namespace Euler2

int main() {
    int t, n, m;
    cin >> t >> n >> m;
    for (int u, v, i = 1; i <= m; i++) {
        cin >> u >> v;
        g[u].push_back({v, i});
        if (t == 1) g[v].push_back({u, -i});
    }
    // solve
    bool flag = t == 1 ? Euler1::solve(n) : Euler2::solve(n);
    // output
    if (!flag || (m > 0 && top - 1 < m))
        puts("NO");
    else {
        puts("YES");
        for (int i = top - 1; i > 0; --i) printf("%d%c", stk[i], " \n"[i
== 1]);
    }
    return 0;
}
```

点分树

```cpp
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;
const ll N = 2e5 + 10;
```

```cpp
ll age[N];
struct edge {
    ll to, val;
};

struct father {
    ll u, num;
    ll dist;
};

struct son {
    ll age, dist;

    bool operator<(const son &s) const {
        return age < s.age;
    }
};

vector<father> f[N];
vector<vector<son> > s[N];
vector<edge> g[N];
bool st[N];
ll siz[N];

ll getsiz(ll u, ll fa) {
    if (st[u]) return 0;
    siz[u] = 1;
    for (auto x:g[u]) {
        if (x.to == fa) continue;
        if (st[x.to]) continue;
        siz[u] += getsiz(x.to, u);
    }
    return siz[u];
}

void getwc(ll u, ll fa, ll tot, ll &wc) {
    if (st[u]) return;
    ll mmax = 0, sum = 1;
    for (auto x:g[u]) {
        if (x.to == fa) continue;
        if (st[x.to]) continue;
        getwc(x.to, u, tot, wc);
        mmax = max(mmax, siz[x.to]);
        sum += siz[x.to];
    }
    mmax = max(mmax, tot - sum);
    if (2 * mmax <= tot) wc = u;
}

void getdist(ll u, ll fa, ll now, ll rt, ll kth, vector<son> &v) {
    if (st[u]) return;
    f[u].push_back({rt, kth, now});
    v.push_back({age[u], now});
    for (auto x:g[u]) {
        if (x.to == fa || st[x.to]) continue;
        getdist(x.to, u, now + x.val, rt, kth, v);
    }
}

void calc(ll u) {
    if (st[u]) return;
    getwc(u, -1, getsiz(u, -1), u);

    st[u] = 1;

    for (auto x: g[u]) {
        if (st[x.to]) continue;
        s[u].push_back(vector<son>(0));
        auto &v = s[u].back();
        v.push_back({-0x3f3f3f3f, 0});
        v.push_back({0x3f3f3f3f, 0});
        getdist(x.to, u, x.val, u, (ll) s[u].size() - 1, v);
        sort(v.begin(), v.end(), [](son a, son b) { return a.age < b.age;
 });
        for (ll i = 1; i < v.size(); i++) {
            v[i].dist += v[i - 1].dist;
        }
    }
    for (auto x:g[u]) {
        calc(x.to);
    }
}

ll query(ll u, ll l, ll r) {
    ll ans = 0;
    for (auto x:f[u]) {
        if (l <= age[x.u] && age[x.u] <= r) ans += x.dist;
        for (ll i = 0; i < s[x.u].size(); i++) {
            if (i == x.num) continue;
            auto &v = s[x.u][i];
            ll btn = lower_bound(v.begin(), v.end(), (son) {l, 0}) - v.be
gin() - 1;
```

```
            ll top = upper_bound(v.begin(), v.end(), (son) {r, 0}) - v.be
gin() - 1;
            ans += v[top].dist - v[btn].dist;
            ans += (top - btn) * x.dist;
        }
    }
    for (auto v:s[u]) {
        ll btn = lower_bound(v.begin(), v.end(), (son) {l, 0}) - v.begin
() - 1;
        ll top = upper_bound(v.begin(), v.end(), (son) {r, 0}) - v.begin
() - 1;
        ans += v[top].dist - v[btn].dist;

    }
    return ans;
}

signed main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);

    ll n, q, a;
    cin >> n >> q >> a;
    for (ll i = 1; i <= n; i++) cin >> age[i];
    for (ll i = 1; i < n; i++) {
        ll x, y, z;
        cin >> x >> y >> z;
        g[x].push_back({y, z});
        g[y].push_back({x, z});
    }

    calc(1);

    ll ans = 0;
    while (q--) {
        ll u, l, r;
        cin >> u >> l >> r;
        l = (l + ans) % a;
        r = (r + ans) % a;
        if (l > r) swap(l, r);
        ans = query(u, l, r);
        cout << ans << endl;
    }
}
```

点双连通分量+缩点建图

```
#include <bits/stdc++.h>

#define ll long long
using namespace std;
const int N = 10010;
const int M = 10010 * 4;
int head[N];
int ver[M];
int Next[M];
int tot, n, m;

void add(int x, int y) {
    ver[++tot] = y;
    Next[tot] = head[x];
    head[x] = tot;
}

int root;
vector<int> dcc[N];
int stackk[N];
int dfn[N], low[N];
int num = 0;//ʱ���
int top;//stackk
int cnt = 0;//�ʿ����L
bool cut[N];//����ж�
void tarjan(int x) {
    dfn[x] = low[x] = ++num;
    stackk[++top] = x;
    if (x == root && head[x] == 0) {
        dcc[++cnt].push_back(x);//cnt�ʿ����
        return;
    }
    int flag = 0;
    for (int i = head[x]; i; i = Next[i]) {
        int y = ver[i];
        if (!dfn[y]) {
            tarjan(y);
            low[x] = min(low[x], low[y]);
            if (low[y] >= dfn[x]) {
                flag++;
                if (x != root || flag > 1)cut[x] = true;
                cnt++;
                int z;
                do//������ж����xh� �h���ʿ��(����ˇ���
```

```
������eĽЖ�+���?)
                {
                    z = stackk[top--];
                    dcc[cnt].push_back(z);
                } while (z != y);
                dcc[cnt].push_back(x);
            }
        } else low[x] = min(low[x], dfn[y]);
    }
}

int tot2 = 1;
int new_id[N];

int hc[N];
int vc[M];
int nc[M];

void add_c(int x, int y) {
    vc[++tot2] = y;
    nc[tot2] = hc[x];
    hc[x] = tot2;
}

int main() {
    while (cin >> n >> m) {
        tot = 1;//������^������ʸ�� ��ȷ�
        for (int i = 1; i <= m; ++i) {
            int x, y;
            cin >> x >> y;
            if (x == y)continue;
            add(x, y);
            add(y, x);
        }
        for (int i = 1; i <= n; ++i) {
            if (!dfn[i])root = i, tarjan(i);
        }
        /*for(int i=1;i<=n;++i)
            if(cut[i])printf("%d ",i);*/
        //�������ʿ��V-DCC
        //�������ÿ���ʼ���eĵ�
        for (int i = 1; i <= cnt; ++i) {
            for (int j = 0; j < dcc[i].size(); ++j)cout << i << " " << dc
c[i][j] << endl;
        }
        //����
        tot2 = 1;
        int num2 = cnt;
        for (int i = 1; i <= n; ++i) {
            if (cut[i])new_id[i] = ++num2;//����������±��,� ��
ÿ����滻����Ϊ���ʼ��
        }
        for (int i = 1; i <= cnt; ++i) {
            for (int j = 0; j < dcc[i].size(); ++j) {
                int x = dcc[i][j];
                if (cut[x])//ʱ���ʼ���������ʶ�ʱ����ʼ����
��ĵ���Ш�ʼ���������;
                {
                    add_c(i, new_id[x]);
                    add_c(new_id[x], i);
                } else new_id[x] = i;//������ʶ����ʱ����ʼ��
            }
        }

        //���������ϵu����ǿ����Ǖ�є����ˋ�ʸ��^����ŵ�'ǎ
�i��2��'��ÿ�µ2��<tot2����<=��
        for (int i = 2; i < tot2; i += 2)
            cout << vc[i ^ 1] << "   " << vc[i] << endl;


    }
    return 0;
}

/*
 * tot2Ï���������2��'����
 * num2Ï�����ǖ�ĵ���
 * ��ͱ���得������ʩ��ʩ�ӫ́��N��C
*/
```

虚树

```
ll fa[N], son[N], dep[N], siz[N], dfn[N], rnk[N], top[N];
ll dfscnt;
vector<ll> g[N];
ll mmin[N];

void dfs1(ll u, ll f, ll d) {
    son[u] = -1;
```

```cpp
    siz[u] = 1;
    fa[u] = f;
    dep[u] = d;
    for (auto v:g[u]) {
        if (v == f) continue;
        dfs1(v, u, d + 1);
        siz[u] += siz[v];
        if (son[u] == -1 || siz[v] > siz[son[u]]) son[u] = v;
    }
}

void dfs2(ll u, ll t) {
    dfn[u] = ++dfscnt;
    rnk[dfscnt] = u;
    top[u] = t;
    if (son[u] == -1) return;
    dfs2(son[u], t);
    for (auto v:g[u]) {
        if (v == son[u] || v == fa[u]) continue;
        dfs2(v, v);
    }
}

ll lca(ll a, ll b) {
    while (top[a] != top[b]) {
        if (dep[top[a]] < dep[top[b]]) swap(a, b);
        a = fa[top[a]];
    }
    return dep[a] < dep[b] ? a : b;
}

struct edge {
    ll s, t, v;
};
edge e[N];

vector<int> vg[N];
int sta[N], tot;
int h[N];

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

多项式

fft

```cpp
const double Pi = acos(-1.0);
```

```cpp
struct Complex {
    double x, y;

    Complex(double xx = 0, double yy = 0) { x = xx, y = yy; }
} a[N], b[N];

Complex operator+(Complex _a, Complex _b) { return Complex(_a.x + _b.x,
_a.y + _b.y); }

Complex operator-(Complex _a, Complex _b) { return Complex(_a.x - _b.x,
_a.y - _b.y); }

Complex operator*(Complex _a, Complex _b) {
    return Complex(_a.x * _b.x - _a.y * _b.y, _a.x * _b.y + _a.y * _b.x);
} //不懂的看复数的运算那部分

int L, r[N];
int limit = 1;

void fft(Complex *A, int type) {
    for (int i = 0; i < limit; i++)
        if (i < r[i]) swap(A[i], A[r[i]]); //求出要迭代的序列
    for (int mid = 1; mid < limit; mid <<= 1) { //待合并区间的长度的一半
        Complex Wn(cos(Pi / mid), type * sin(Pi / mid)); //单位根
        for (int R = mid << 1, j = 0; j < limit; j += R) { //R 是区间的长
度，j 表示前已经到哪个位置了
            Complex w(1, 0); //幂
            for (int k = 0; k < mid; k++, w = w * Wn) { //枚举左半部分
                Complex x = A[j + k], y = w * A[j + mid + k]; //蝴蝶效应
                A[j + k] = x + y;
                A[j + mid + k] = x - y;
            }
        }
    }
}

void FFT(int n, int m) {
    limit = 1;
    L = 0;
    while (limit <= n + m) limit <<= 1, L++;
    for (int i = 0; i < limit; i++) r[i] = (r[i >> 1] >> 1) | ((i & 1) <<
 (L - 1));
    // 在原序列中 i 与 i/2 的关系是 ： i 可以看做是 i/2 的二进制上的每一位左移
一位得来
    // 那么在反转后的数组中就需要右移一位，同时特殊处理一下奇数
    fft(a, 1), fft(b, 1);
    for (int i = 0; i <= limit; i++) a[i] = a[i] * b[i];
    fft(a, -1);
    for (int i = 0; i <= n + m; i++) a[i].x /= limit;
}
```

ntt

```cpp
const ll mod = 998244353, G = 3, Gi = 332748118;

int limit = 1, L, r[N];
ll a[N], b[N];

ll qpow(ll _a, ll _b) {
    ll ans = 1;
    while (_b) {
        if (_b & 1) ans = (ans * _a) % mod;
        _b >>= 1;
        _a = (_a * _a) % mod;
    }
    return ans;
}

void ntt(ll *A, int type) {
    auto swap = [](ll &_a, ll &_b) {
        _a ^= _b, _b ^= _a, _a ^= _b;
    };
    for (int i = 0; i < limit; i++)
        if (i < r[i]) swap(A[i], A[r[i]]);
    for (int mid = 1; mid < limit; mid <<= 1) {
        ll Wn = qpow(type == 1 ? G : Gi, (mod - 1) / (mid << 1));
        for (int j = 0; j < limit; j += (mid << 1)) {
            ll w = 1;
            for (int k = 0; k < mid; k++, w = (w * Wn) % mod) {
                int x = A[j + k], y = w * A[j + k + mid] % mod;
                A[j + k] = (x + y) % mod,
                        A[j + k + mid] = (x - y + mod) % mod;
            }
        }
    }
}

void NTT(int n, int m) {
```

```
    limit = 1;
    L = 0;
    while (limit <= n + m) limit <<= 1, L++;
    for (int i = 0; i < limit; i++) r[i] = (r[i >> 1] >> 1) | ((i & 1) <<
 (L - 1));
    ntt(a, 1), ntt(b, 1);
    for (int i = 0; i < limit; i++) a[i] = (a[i] * b[i]) % mod;
    ntt(a, -1);
    ll inv = qpow(limit, mod - 2);
    for (int i = 0; i <= n + m; i++) a[i] = a[i] * inv % mod;
}
```

多项式全家桶

```
//#pragma GCC optimize(2)
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;

const ll N = 3000007;
const ll p = 998244353, gg = 3, ig = 332738118, img = 86583718;
const ll mod = 998244353;

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

namespace Poly {
#define mul(x, y) (1ll * x * y >= mod ? 1ll * x * y % mod : 1ll * x * y)
#define minus(x, y) (1ll * x - y < 0 ? 1ll * x - y + mod : 1ll * x - y)
#define plus(x, y) (1ll * x + y >= mod ? 1ll * x + y - mod : 1ll * x + y)
#define ck(x) (x >= mod ? x - mod : x)//取模运算太慢了

    typedef vector<ll> poly;
    const ll G = 3;//根据具体的模数而定，原根可不一定不一样！！！
    //一般模数的原根为 2 3 5 7 10 6
    const ll inv_G = qpow(G, mod - 2);
    ll RR[N], inv[N];

    void init() {
        inv[0] = inv[1] = 1;
        for (ll i = 2; i < N; ++i)
            inv[i] = 1ll * inv[mod % i] * (mod - mod / i) % mod;
    }

    ll NTT_init(ll n) {//快速数论变换预处理
        ll limit = 1, L = 0;
        while (limit <= n) limit <<= 1, L++;
        for (ll i = 0; i < limit; ++i)
            RR[i] = (RR[i >> 1] >> 1) | ((i & 1) << (L - 1));
        return limit;
    }

//  省空间用
    ll deer[2][N];

    void NTT(poly &A, ll type, ll limit) {//快速数论变换
        A.resize(limit);
        for (ll i = 0; i < limit; ++i)
            if (i < RR[i])
                swap(A[i], A[RR[i]]);
        for (ll mid = 2, j = 1; mid <= limit; mid <<= 1, ++j) {
            ll len = mid >> 1;

//          省空间用
            ll buf1 = qpow(G, (mod - 1) / (1 << j));
            ll buf0 = qpow(inv_G, (mod - 1) / (1 << j));
            deer[0][0] = deer[1][0] = 1;
            for (ll i = 1; i < (1 << j); ++i) {
                deer[0][i] = 1ll * deer[0][i - 1] * buf0 % mod;//逆
                deer[1][i] = 1ll * deer[1][i - 1] * buf1 % mod;
            }

            for (ll pos = 0; pos < limit; pos += mid) {
//                ll *wn = deer[type][j];
//              省空间用
                ll *wn = deer[type];
                for (ll i = pos; i < pos + len; ++i, ++wn) {
                    ll tmp = 1ll * (*wn) * A[i + len] % mod;
                    A[i + len] = ck(A[i] - tmp + mod);
                    A[i] = ck(A[i] + tmp);
                }
```

```cpp
        }
    }
    if (type == 0) {
        for (ll i = 0; i < limit; ++i)
            A[i] = 1ll * A[i] * inv[limit] % mod;
    }
}

poly poly_mul(poly A, poly B) {//多项式乘法
    ll deg = A.size() + B.size() - 1;
    ll limit = NTT_init(deg);
    poly C(limit);
    NTT(A, 1, limit);
    NTT(B, 1, limit);
    for (ll i = 0; i < limit; ++i)
        C[i] = 1ll * A[i] * B[i] % mod;
    NTT(C, 0, limit);
    C.resize(deg);
    return C;
}

poly poly_inv(poly &f, ll deg) {//多项式求逆
    if (deg == 1)
        return poly(1, qpow(f[0], mod - 2));

    poly A(f.begin(), f.begin() + deg);
    poly B = poly_inv(f, (deg + 1) >> 1);
    ll limit = NTT_init(deg << 1);
    NTT(A, 1, limit), NTT(B, 1, limit);
    for (ll i = 0; i < limit; ++i)
        A[i] = B[i] * (2 - 1ll * A[i] * B[i] % mod + mod) % mod;
    NTT(A, 0, limit);
    A.resize(deg);
    return A;
}

poly poly_dev(poly f) {//多项式求导
    ll n = f.size();
    for (ll i = 1; i < n; ++i) f[i - 1] = 1ll * f[i] * i % mod;
    return f.resize(n - 1), f;//f[0] = 0，这里直接扔了,从 1 开始
}

poly poly_idev(poly f) {//多项式求积分
    ll n = f.size();
    for (ll i = n - 1; i; --i) f[i] = 1ll * f[i - 1] * inv[i] % mod;
    return f[0] = 0, f;
}

poly poly_ln(poly f, ll deg) {//多项式求对数
    poly A = poly_idev(poly_mul(poly_dev(f), poly_inv(f, deg)));
    return A.resize(deg), A;
}

poly poly_exp(poly &f, ll deg) {//多项式求指数
    if (deg == 1)
        return poly(1, 1);

    poly B = poly_exp(f, (deg + 1) >> 1);
    B.resize(deg);
    poly lnB = poly_ln(B, deg);
    for (ll i = 0; i < deg; ++i)
        lnB[i] = ck(f[i] - lnB[i] + mod);

    ll limit = NTT_init(deg << 1);//n -> n^2
    NTT(B, 1, limit), NTT(lnB, 1, limit);
    for (ll i = 0; i < limit; ++i)
        B[i] = 1ll * B[i] * (1 + lnB[i]) % mod;
    NTT(B, 0, limit);
    B.resize(deg);
    return B;
}

poly poly_sqrt(poly &f, ll deg) {//多项式开方
    if (deg == 1) return poly(1, 1);
    poly A(f.begin(), f.begin() + deg);
    poly B = poly_sqrt(f, (deg + 1) >> 1);
    poly IB = poly_inv(B, deg);
    ll limit = NTT_init(deg << 1);
    NTT(A, 1, limit), NTT(IB, 1, limit);
    for (ll i = 0; i < limit; ++i)
        A[i] = 1ll * A[i] * IB[i] % mod;
    NTT(A, 0, limit);
    for (ll i = 0; i < deg; ++i)
        A[i] = 1ll * (A[i] + B[i]) * inv[2] % mod;
    A.resize(deg);
    return A;
}

poly poly_pow(poly f, ll k) {//多项式快速幂
    if (f.size() == 1) {
```

```
        f[0] = qpow(f[0], k);
        return f;
    }
    f = poly_ln(f, f.size());
    for (auto &x: f) x = 1ll * x * k % mod;
    return poly_exp(f, f.size());
}

poly poly_cos(poly f, ll deg) {//多项式三角函数（cos）
    poly A(f.begin(), f.begin() + deg);
    poly B(deg), C(deg);
    for (ll i = 0; i < deg; ++i)
        A[i] = 1ll * A[i] * img % mod;

    B = poly_exp(A, deg);
    C = poly_inv(B, deg);
    ll inv2 = qpow(2, mod - 2);
    for (ll i = 0; i < deg; ++i)
        A[i] = 1ll * (1ll * B[i] + C[i]) % mod * inv2 % mod;
    return A;
}

poly poly_sin(poly f, ll deg) {//多项式三角函数（sin）
    poly A(f.begin(), f.begin() + deg);
    poly B(deg), C(deg);
    for (ll i = 0; i < deg; ++i)
        A[i] = 1ll * A[i] * img % mod;

    B = poly_exp(A, deg);
    C = poly_inv(B, deg);
    ll inv2i = qpow(img << 1, mod - 2);
    for (ll i = 0; i < deg; ++i)
        A[i] = 1ll * (1ll * B[i] - C[i] + mod) % mod * inv2i % mod;
    return A;
}

poly poly_arcsin(poly f, ll deg) {
    poly A(f.size()), B(f.size()), C(f.size());
    A = poly_dev(f);
    B = poly_mul(f, f);
    for (ll i = 0; i < deg; ++i)
        B[i] = minus(mod, B[i]);
    B[0] = plus(B[0], 1);
    C = poly_sqrt(B, deg);
    C = poly_inv(C, deg);
        C = poly_mul(A, C);
        C = poly_idev(C);
        return C;
    }

    poly poly_arctan(poly f, ll deg) {
        poly A(f.size()), B(f.size()), C(f.size());
        A = poly_dev(f);
        B = poly_mul(f, f);
        B[0] = plus(B[0], 1);
        C = poly_inv(B, deg);
        C = poly_mul(A, C);
        C = poly_idev(C);
        return C;
    }
}

using namespace Poly;

signed main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);

    ll n, k;
    cin >> n >> k;
    Poly::init();
    vector<ll> v(n);
    for (ll i = 0; i < n; i++) cin >> v[i];
    auto res = Poly::poly_pow(v, k);
    for (auto x: res) cout << x << ' ';
    cout << endl;
}
```

字符串

AC 自动机

```
#include <bits/stdc++.h>
using namespace std;
struct AC {
    static const int maxnode = 200005;
    static const int sigma_size = 26;
```

```
char T[maxnode];
int ch[maxnode][sigma_size];
int val[maxnode], fail[maxnode], last[maxnode];
int sz;
vector<pair<int, int> > ans;

void init() {
    sz = 1;
    memset(ch[0], 0, sizeof(ch[0]));
    ans.clear();
}

int idx(const char &c) { return c - 'a'; }

void insert(string s, int v) {
    int u = 0, n = s.length();
    for (int i = 0; i < n; i++) {
        int c = idx(s[i]);
        if (!ch[u][c]) {
            memset(ch[sz], 0, sizeof(ch[sz]));
            val[sz] = 0;
            ch[u][c] = sz++;
        }
        u = ch[u][c];
    }
    val[u] = v;
}

void get_fail() {
    queue<int> que;
    fail[0] = 0;
    for (int c = 0; c < sigma_size; c++) {
        int u = ch[0][c];
        if (u) {
            fail[u] = 0;
            que.push(u);
            last[u] = 0;
        }
    }
    while (!que.empty()) {
        int r = que.front();
        que.pop();
        for (int c = 0; c < sigma_size; c++) {
            int u = ch[r][c];
            if (!u) continue;
            que.push(u);
            int v = fail[r];
            while (v && !ch[v][c]) v = fail[v];
            fail[u] = ch[v][c];
            last[u] = val[fail[u]] ? fail[u] : last[fail[u]];
        }
    }
}

void print(int j) {
    if (j) {
        ans.push_back(pair<int, int>(j, val[j]));
        print(last[j]);
    }
}

void find() {
    int n = strlen(T);
    int j = 0;
    for (int i = 0; i < n; i++) {
        int c = idx(T[i]);
        while (j && !ch[j][c]) j = fail[j];
        j = ch[j][c];
        if (val[j])
            print(j);
        else if (last[j])
            print(last[j]);
    }
}
} ac;   //字符串下标从0开始
```

KMP 2

```
#include <bits/stdc++.h>
using namespace std;
struct KMP {
    static const int MAXN = 1000010;
    char T[MAXN], P[MAXN];
    int fail[MAXN];
    vector<int> ans;

    void init() { ans.clear(); }

    void get_fail() {
        int m = strlen(P);
```

```
        fail[0] = fail[1] = 0;
        for (int i = 1; i < m; i++) {
            int j = fail[i];
            while (j && P[i] != P[j]) j = fail[j];
            fail[i + 1] = (P[i] == P[j] ? j + 1 : 0);
        }
    }

    void find() {
        int n = strlen(T), m = strlen(P);
        get_fail();
        int j = 0;
        for (int i = 0; i < n; i++) {
            while (j && P[j] != T[i]) j = fail[j];
            if (P[j] == T[i]) j++;
            if (j == m) ans.push_back(i - m + 1);
        }
    }
} kmp;  //P 为模式串，下标从 0 开始，输入后直接调用 find()
```

kmp

```
//next 数组等价于前缀函数
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;

int kmp(char *s1,int *p1,char *s2=0,int *p2=0){//必须先求 s1 的 next 数组，即 kmp(s1,p1);再 kmp(s1,p1,s2,p2);
    int n=strlen(s1);
    if(p2==0){
        p1[0]=0;
        for(int i=1;s1[i]!='\0';i++){
            int j=p1[i-1];
            while(j>0&&s1[i]!=s1[j])j=p1[j-1];
            if(s1[i]==s1[j])j++;
            p1[i]=j;
        }
    }
    else{
        for(int i=0;s2[i]!='\0';i++){
            int j=i==0?0:p2[i-1];
            while(j>0&&s2[i]!=s1[j])j=p1[j-1];
            if(s2[i]==s1[j])j++;
            p2[i]=j;
            if(j==n)return i-n+2;//返回位置
        }
    }
    return 0;
}
int main(){
    char s1[15],s2[105];
    int p1[15],p2[105];
    cin>>s1>>s2;
    kmp(s1,p1);
    cout<<kmp(s1,p1,s2,p2)<<endl;
    return 0;
}
```

regex

元字符	描述
\	将下一个字符标记符、或一个向后引用、或一个八进制转义符。例如，“\n”匹配\n。“\n”匹配换行符。序列“\”匹配“\”而“(”则匹配“(”。即相当于多种编程语言中都有的“转义字符”的概念。
^	匹配输入字行首。如果设置了 RegExp 对象的 Multiline 属性，^也匹配“\n”或“\r”之后的位置。
$	匹配输入行尾。如果设置了 RegExp 对象的 Multiline 属性，$也匹配“\n”或“\r”之前的位置。
*	匹配前面的子表达式任意次。例如，zo *能匹配“z”，也能匹配“zo”以及“zoo”*。等价于{0,}。
+	匹配前面的子表达式一次或多次(大于等于 1 次）。例如，“zo+”能匹配“zo”以及“zoo”，但不能匹配“z”。+等价于{1,}。
?	匹配前面的子表达式零次或一次。例如，“do(es)?”可以匹配“do”或“does”。?等价于{0,1}。
{*n*}	*n* 是一个非负整数。匹配确定的 *n* 次。例如，“o{2}”不能匹配“Bob”中的“o”，但是能匹配“food”中的两个 o。
{*n*,}	*n* 是一个非负整数。至少匹配 *n* 次。例如，“o{2,}”不能匹配“Bob”中的“o”，但能匹配“foooood”中的所有 o。“o{1,}”等价于“o+”。“o{0,}”则等价于“o*”。
{*n*,*m*}	*m* 和 *n* 均为非负整数，其中 *n*<=*m*。最少匹配 *n* 次且最多匹配 *m* 次。例如，“o{1,3}”将匹配“fooooood”中的前三个 o 为一组，后三个 o 为一组。“o{0,1}”等价于“o?”。请注意在逗号和两个数之间不能有空格。
?	当该字符紧跟在任何一个其他限制符（,+,?，{n}，{n,}，{n,m*}）后面时，匹

<table>
<tr><th>元字符</th><th>描述</th></tr>
<tr><td></td><td>配模式是非贪婪的。非贪婪模式尽可能少地匹配所搜索的字符串，而默认的贪婪模式则尽可能多地匹配所搜索的字符串。例如，对于字符串“oooo”，“o+”将尽可能多地匹配“o”，得到结果[“oooo”]，而“o+?”将尽可能少地匹配“o”，得到结果 ['o', 'o', 'o', 'o']</td></tr>
<tr><td>.点</td><td>匹配除“\n”和"\r"之外的任何单个字符。要匹配包括“\n”和"\r"在内的任何字符，请使用像“[\s\S]”的模式。</td></tr>
<tr><td>(pattern)</td><td>匹配 pattern 并获取这一匹配。所获取的匹配可以从产生的 Matches 集合得到，在 VBScript 中使用 SubMatches 集合，在 JScript 中则使用 0…9 属性。要匹配圆括号字符，请使用“(”或“)”。</td></tr>
<tr><td>(?:pattern)</td><td>非获取匹配，匹配 pattern 但不获取匹配结果，不进行存储供以后使用。这在使用或字符“(|)”来组合一个模式的各个部分时很有用。例如“industr(?:y|ies)”就是一个比“industry|industries”更简略的表达式。</td></tr>
<tr><td>(?=pattern)</td><td>非获取匹配，正向肯定预查，在任何匹配 pattern 的字符串开始处匹配查找字符串，该匹配不需要获取供以后使用。例如，“Windows(?=95|98|NT|2000)”能匹配“Windows2000”中的“Windows”，但不能匹配“Windows3.1”中的“Windows”。预查不消耗字符，也就是说，在一个匹配发生后，在最后一次匹配之后立即开始下一次匹配的搜索，而不是从包含预查的字符之后开始。</td></tr>
<tr><td>(?!pattern)</td><td>非获取匹配，正向否定预查，在任何不匹配 pattern 的字符串开始处匹配查找字符串，该匹配不需要获取供以后使用。例如“Windows(?!95|98|NT|2000)”能匹配“Windows3.1”中的“Windows”，但不能匹配“Windows2000”中的“Windows”。</td></tr>
<tr><td>(?<=pattern)</td><td>非获取匹配，反向肯定预查，与正向肯定预查类似，只是方向相反。例如，“(?<=95|98|NT|2000)Windows”能匹配“2000Windows”中的“Windows”，但不能匹配“3.1Windows”中的“Windows”。*python 的正则表达式没有完全按照正则表达式规范实现，所以一些高级特性建议使用其他语言如 java、scala 等</td></tr>
<tr><td>(?<!pattern)</td><td>非获取匹配，反向否定预查，与正向否定预查类似，只是方向相反。例如“(?<!95|98|NT|2000)Windows”能匹配“3.1Windows”中的“Windows”，但不能匹配“2000Windows”中的“Windows”。*python 的正则表达式没有完全按照正则表达式规范实现，所以一些高级特性建议使用其他语言如 java、scala 等</td></tr>
<tr><td>x|y</td><td>匹配 x 或 y。例如，“z|food”能匹配“z”或“food”(此处请谨慎)。“[z|f]ood”则匹配“zood”或“food”。</td></tr>
<tr><td>[xyz]</td><td>字符集合。匹配所包含的任意一个字符。例如，“[abc]”可以匹配“plain”中的“a”。</td></tr>
</table>

<table>
<tr><th>元字符</th><th>描述</th></tr>
<tr><td>[^xyz]</td><td>负值字符集合。匹配未包含的任意字符。例如，“abc”可以匹配“plain”中的“plin”任一字符。</td></tr>
<tr><td>[a-z]</td><td>字符范围。匹配指定范围内的任意字符。例如，“[a-z]”可以匹配“a”到“z”范围内的任意小写字母字符。注意:只有连字符在字符组内部时,并且出现在两个字符之间时,才能表示字符的范围; 如果出字符组的开头,则只能表示连字符本身.</td></tr>
<tr><td>[^a-z]</td><td>负值字符范围。匹配任何不在指定范围内的任意字符。例如，“a-z”可以匹配任何不在“a”到“z”范围内的任意字符。</td></tr>
<tr><td>\b</td><td>匹配一个单词的边界，也就是指单词和空格间的位置（即正则表达式的“匹配”有两种概念，一种是匹配字符，一种是匹配位置，这里的\b 就是匹配位置的）。例如，“er\b”可以匹配“never”中的“er”，但不能匹配“verb”中的“er”；“\b1”可以匹配“123”中的“1”，但不能匹配“213”中的“1_”。</td></tr>
<tr><td>\B</td><td>匹配非单词边界。“er\B”能匹配“verb”中的“er”，但不能匹配“never”中的“er”。</td></tr>
<tr><td>\cx</td><td>匹配由 x 指明的控制字符。例如，\cM 匹配一个 Control-M 或回车符。x 的值必须为 A-Z 或 a-z 之一。否则，将 c 视为一个原义的“c”字符。</td></tr>
<tr><td>\d</td><td>匹配一个数字字符。等价于[0-9]。grep 要加上-P，perl 正则支持</td></tr>
<tr><td>\D</td><td>匹配一个非数字字符。等价于 0-9。grep 要加上-P，perl 正则支持</td></tr>
<tr><td>\f</td><td>匹配一个换页符。等价于\x0c 和\cL。</td></tr>
<tr><td>\n</td><td>匹配一个换行符。等价于\x0a 和\cJ。</td></tr>
<tr><td>\r</td><td>匹配一个回车符。等价于\x0d 和\cM。</td></tr>
<tr><td>\s</td><td>匹配任何不可见字符，包括空格、制表符、换页符等等。等价于[\f\n\r\t\v]。</td></tr>
<tr><td>\S</td><td>匹配任何可见字符。等价于 \f\n\r\t\v。</td></tr>
<tr><td>\t</td><td>匹配一个制表符。等价于\x09 和\cI。</td></tr>
<tr><td>\v</td><td>匹配一个垂直制表符。等价于\x0b 和\cK。</td></tr>
<tr><td>\w</td><td>匹配包括下划线的任何单词字符。类似但不等价于“[A-Za-z0-9_]”，这里的"单词"字符使用 Unicode 字符集。</td></tr>
<tr><td>\W</td><td>匹配任何非单词字符。等价于“A-Za-z0-9_”。</td></tr>
<tr><td>\xn</td><td>匹配 n，其中 n 为十六进制转义值。十六进制转义值必须为确定的两个数字长。例如，“\x41”匹配“A”。“\x041”则等价于“\x04&1”。正则表达式中可以使用 ASCII 编码。</td></tr>
<tr><td>*num*</td><td>匹配 num，其中 num 是一个正整数。对所获取的匹配的引用。例如，</td></tr>
</table>

元字符	描述
	“(.)\1”匹配两个连续的相同字符。
n	标识一个八进制转义值或一个向后引用。如果*n *之前至少* n *个获取的子表达式，则* n *为向后引用。否则，如果* n *为八进制数字（0-7），则* n*为一个八进制转义值。
nm	标识一个八进制转义值或一个向后引用。如果*nm *之前至少有* nm *个获得子表达式，则* nm *为向后引用。如果\nm 之前至少有* n *个获取，则* n *为一个后跟文字* m *的向后引用。如果前面的条件都不满足，若* n *和* m *均为八进制数字（0-7），则\nm 将匹配八进制转义值* nm*。
nml	如果 *n* 为八进制数字（0-7），且 *m* 和 *l* 均为八进制数字（0-7），则匹配八进制转义值 *nml*。
\u*n*	匹配 *n*，其中 *n* 是一个用四个十六进制数字表示的 Unicode 字符。例如，\u00A9 匹配版权符号（©）。
\p{P}	小写 p 是 property 的意思，表示 Unicode 属性，用于 Unicode 正表达式的前缀。中括号内的“P”表示 Unicode 字符集七个字符属性之一：标点字符。其他六个属性：L：字母；M：标记符号（一般不会单独出现）；Z：分隔符（比如空格、换行等）；S：符号（比如数学符号、货币符号等）；N：数字（比如阿拉伯数字、罗马数字等）；C：其他字符。**注：此语法部分语言不支持，例：javascript。*
<>	匹配词（word）的开始（<）和结束（>）。例如正则表达式<the>能够匹配字符串"for the wise"中的"the"，但是不能匹配字符串"otherwise"中的"the"。注意：这个元字符不是所有的软件都支持的。
( )	将(和) 之间的表达式定义为“组”（group），并且将匹配这个表达式的字符保存到一个临时区域（一个正则表达式中最多可以保存 9 个），它们可以用 \1 到\9 的符号来引用。
|	将两个匹配条件进行逻辑“或”（or）运算。例如正则表达式(him|her) 匹配"it belongs to him"和"it belongs to her"，但是不能匹配"it belongs to them."。注意：这个元字符不是所有的软件都支持的。

Trie

```cpp
#include <bits/stdc++.h>
using namespace std;
struct Trie {
    static const int maxnode = 200005;
    static const int sigma_size = 26;
    int ch[maxnode][sigma_size];
    int val[maxnode];
    int sz;

    Trie() {
        sz = 1;
        memset(ch[0], 0, sizeof(ch[0]));
    }

    int idx(const char &c) { return c - 'a'; }

    void insert(string s, int v) {
        int u = 0, n = s.length();
        for (int i = 0; i < n; i++) {
            int c = idx(s[i]);
            if (!ch[u][c]) {
                memset(ch[sz], 0, sizeof(ch[sz]));
                val[sz] = 0;
                ch[u][c] = sz++;
            }
            u = ch[u][c];
        }
        val[u] = v;
    }

    int find(string s) {
        int u = 0, n = s.length();
        for (int i = 0; i < n; i++) {
            int c = idx(s[i]);
            if (!ch[u][c]) return 0;
            u = ch[u][c];
        }
        return val[u];
    }
} trie;
```

可持久化字典树

```cpp
struct Trie01 {
    static const int maxnode = 2000005;
    static const int sigma_size = 2;
    int ch[maxnode << 5][sigma_size], val[maxnode << 5];
    int rt[maxnode];
    int sz;

    Trie01() {
        sz = 0;
```

```
        memset(ch[0], 0, sizeof(ch[0]));
    }

    void insert(int &now, int pre, int v) {
        now = ++sz;
        for (int i = 30; i >= 0; i--) {
            int k = ((v >> i) & 1);
            ch[now][k] = ++sz;
            ch[now][k ^ 1] = ch[pre][k ^ 1];
            val[ch[now][k]] = val[ch[pre][k]] + 1;
            now = ch[now][k];
            pre = ch[pre][k];
        }
    }
} trie;
```

后缀数组

```
#include <bits/stdc++.h>
using namespace std;
struct SuffixArray {
    static const int MAXN = 1100000;
    char s[MAXN];
    int sa[MAXN], t[MAXN], t1[MAXN], c[MAXN], ra[MAXN], height[MAXN], m;
    inline void init() { memset(this, 0, sizeof(SuffixArray)); }

    inline void get_sa(int n) {
        m = 256;
        int *x = t, *y = t1;
        for (int i = 1; i <= m; i++) c[i] = 0;
        for (int i = 1; i <= n; i++) c[x[i] = s[i]]++;
        for (int i = 1; i <= m; i++) c[i] += c[i - 1];
        for (int i = n; i >= 1; i--) sa[c[x[i]]--] = i;
        for (int k = 1; k <= n; k <<= 1) {
            int p = 0;
            for (int i = n - k + 1; i <= n; i++) y[++p] = i;
            for (int i = 1; i <= n; i++)
                if (sa[i] > k) y[++p] = sa[i] - k;
            for (int i = 1; i <= m; i++) c[i] = 0;
            for (int i = 1; i <= n; i++) c[x[y[i]]]++;
            for (int i = 1; i <= m; i++) c[i] += c[i - 1];
            for (int i = n; i >= 1; i--) sa[c[x[y[i]]]--] = y[i];
            std::swap(x, y);
            p = x[sa[1]] = 1;
            for (int i = 2; i <= n; i++) {
                x[sa[i]] = (y[sa[i - 1]] == y[sa[i]] &&
                            y[sa[i - 1] + k] == y[sa[i] + k])
                               ? p
                               : ++p;
            }
            if (p >= n) break;
            m = p;
        }
    }

    inline void get_height(int n) {
        int i, j, k = 0;
        for (int i = 1; i <= n; i++) ra[sa[i]] = i;
        for (int i = 1; i <= n; i++) {
            if (k) k--;
            int j = sa[ra[i] - 1];
            while (s[i + k] == s[j + k]) k++;
            height[ra[i]] = k;
        }
    }

} SA;   //字符串下标从一开始
```

后缀自动机

```
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;
const int N = 2e6 + 10;

int tot = 1, last = 1;
struct Node {
    int len, fa;
    int ch[26];
} node[N];
char str[N];
ll f[N], ans;
int h[N], e[N], ne[N], idx;

void extend(int c) {
    int p = last, np = last = ++tot;
    f[tot] = 1;
```

```
    node[np].len = node[p].len + 1;
    for (; p && !node[p].ch[c]; p = node[p].fa) node[p].ch[c] = np;
    if (!p) node[np].fa = 1;
    else {
        int q = node[p].ch[c];
        if (node[q].len == node[p].len + 1) node[np].fa = q;
        else {
            int nq = ++tot;
            node[nq] = node[q], node[nq].len = node[p].len + 1;
            node[q].fa = node[np].fa = nq;
            for (; p && node[p].ch[c] == q; p = node[p].fa) node[p].ch[c]
 = nq;
        }
    }
}

void add(int a, int b) {
    e[idx] = b, ne[idx] = h[a], h[a] = idx++;
}

void dfs(int u) {
    for (int i = h[u]; ~i; i = ne[i]) {
        dfs(e[i]);
        f[u] += f[e[i]];
    }
    if (f[u] > 1) ans = max(ans, f[u] * node[u].len);
}

int main() {
    scanf("%s", str);
    for (int i = 0; str[i]; i++) extend(str[i] - 'a');
    memset(h, -1, sizeof h);
    for (int i = 2; i <= tot; i++) add(node[i].fa, i);
    dfs(1);
    printf("%lld\n", ans);

    return 0;
}
```

回文自动机

```
#include <bits/stdc++.h>

using namespace std;

const int maxn = 300000 + 5;

namespace pam {
    int sz, tot, last;
    int cnt[maxn], ch[maxn][26], len[maxn], fail[maxn];
    char s[maxn];

    int node(int l) {  // 建立一个新节点，长度为 l
        sz++;
        memset(ch[sz], 0, sizeof(ch[sz]));
        len[sz] = l;
        fail[sz] = cnt[sz] = 0;
        return sz;
    }

    void clear() {  // 初始化
        sz = -1;
        last = 0;
        s[tot = 0] = '$';
        node(0);
        node(-1);
        fail[0] = 1;
    }

    int getfail(int x) {  // 找后缀回文
        while (s[tot - len[x] - 1] != s[tot]) x = fail[x];
        return x;
    }

    void insert(char c) {  // 建树
        s[++tot] = c;
        int now = getfail(last);
        if (!ch[now][c - 'a']) {
            int x = node(len[now] + 2);
            fail[x] = ch[getfail(fail[now])][c - 'a'];
            ch[now][c - 'a'] = x;
        }
        last = ch[now][c - 'a'];
        cnt[last]++;
    }

    long long solve() {
        long long ans = 0;
        for (int i = sz; i >= 0; i--) {
            cnt[fail[i]] += cnt[i];
```

```
        }
        for (int i = 1; i <= sz; i++) {  // 更新答案
            ans = max(ans, 1ll * len[i] * cnt[i]);
        }
        return ans;
    }
}  // namespace pam

char s[maxn];

int main() {
    pam::clear();
    scanf("%s", s + 1);
    for (int i = 1; s[i]; i++) {
        pam::insert(s[i]);
    }
    printf("%lld\n", pam::solve());
    return 0;
}
```

马拉车

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 100005;
char s[maxn];
char s_new[maxn * 2];
int p[maxn * 2];

int Manacher(char* a, int l) {
    s_new[0] = '$';
    s_new[1] = '#';
    int len = 2;
    for (int i = 0; i < l; i++) {
        s_new[len++] = a[i];
        s_new[len++] = '#';
    }
    s_new[len] = '\0';
    int id;
    int mx = 0;
    int mmax = 0;

    for (int i = 1; i < len; i++) {
        p[i] = i < mx ? min(p[2 * id - i], mx - i) : 1;
        while (s_new[i + p[i]] == s_new[i - p[i]]) p[i]++;
        if (mx < i + p[i]) {
            id = i;
            mx = i + p[i];
        }
        mmax = max(mmax, p[i] - 1);
    }
    return mmax;
}

int main() {
    cin >> s;
    cout << Manacher(s, strlen(s));
}
```

数据结构

CDQ 分治

```
/*
处理三维偏序问题,
每个node的三维不能完全相等，完全相等的话加权做
*/

#include <iostream>
#include <cstring>
#include <algorithm>

using namespace std;

const int N = 100010, M = 200010;

int n, m;
struct Data
{
    int a, b, c, s, res;

    bool operator< (const Data& t) const
    {
        if (a != t.a) return a < t.a;
```

```
        if (b != t.b) return b < t.b;
        return c < t.c;
    }
    bool operator== (const Data& t) const
    {
        return a == t.a && b == t.b && c == t.c;
    }
}q[N], w[N];
int tr[M], ans[N];

int lowbit(int x)
{
    return x & -x;
}

void add(int x, int v)
{
    for (int i = x; i < M; i += lowbit(i)) tr[i] += v;
}

int query(int x)
{
    int res = 0;
    for (int i = x; i; i -= lowbit(i)) res += tr[i];
    return res;
}

void merge_sort(int l, int r)
{
    if (l >= r) return;
    int mid = l + r >> 1;
    merge_sort(l, mid), merge_sort(mid + 1, r);
    int i = l, j = mid + 1, k = 0;
    while (i <= mid && j <= r)
        if (q[i].b <= q[j].b) add(q[i].c, q[i].s), w[k ++ ] = q[i ++ ];
        else q[j].res += query(q[j].c), w[k ++ ] = q[j ++ ];
    while (i <= mid) add(q[i].c, q[i].s), w[k ++ ] = q[i ++ ];
    while (j <= r) q[j].res += query(q[j].c), w[k ++ ] = q[j ++ ];
    for (i = l; i <= mid; i ++ ) add(q[i].c, -q[i].s);
    for (i = l, j = 0; j < k; i ++, j ++ ) q[i] = w[j];
}

int main()
{
    scanf("%d%d", &n, &m);
    for (int i = 0; i < n; i ++ )
    {
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        q[i] = {a, b, c, 1};
    }
    sort(q, q + n);

    int k = 1;
    for (int i = 1; i < n; i ++ )
        if (q[i] == q[k - 1]) q[k - 1].s ++ ;
        else q[k ++ ] = q[i];

    merge_sort(0, k - 1);
    for (int i = 0; i < k; i ++ )
        ans[q[i].res + q[i].s - 1] += q[i].s;

    for (int i = 0; i < n; i ++ ) printf("%d\n", ans[i]);

    return 0;
}
```

kruskal 重构树

```
int pa[N];

void init(int n) {
    for (int i = 0; i <= n; i++) {
        pa[i] = i;
    }
}

int find(int a) {
    return pa[a] == a ? a : pa[a] = find(pa[a]);
}

struct edge {
    int from, to, l;
};

int w[N];
edge e[N];
vector<int> g[N];

int kruskal(int n, int m) {
```

```cpp
    int kcnt = n;
    init(n);
    sort(e + 1, e + 1 + m, [](edge a, edge b) { return a.l < b.l; });
    for (int i = 1; i <= m; i++) {
        int u = find(e[i].from);
        int v = find(e[i].to);
        if (u == v) continue;
        w[++kcnt] = e[i].l;
        pa[kcnt] = pa[u] = pa[v] = kcnt;
        g[u].push_back(kcnt);
        g[v].push_back(kcnt);
        g[kcnt].push_back(u);
        g[kcnt].push_back(v);
    }
    return kcnt;
}
```

LCT

```cpp
ll ch[N][2], f[N], sum[N], val[N], tag[N], siz[N], siz2[N];

inline void pushup(ll p) {
    sum[p] = sum[ch[p][0]] ^ sum[ch[p][1]] ^ val[p];
    siz[p] = siz[ch[p][0]] + siz[ch[p][1]] + 1 + siz2[p];
}

inline void pushdown(ll p) {
    if (tag[p]) {
        if (ch[p][0]) swap(ch[ch[p][0]][0], ch[ch[p][0]][1]), tag[ch[p]
[0]] ^= 1;
        if (ch[p][1]) swap(ch[ch[p][1]][0], ch[ch[p][1]][1]), tag[ch[p]
[1]] ^= 1;
        tag[p] = 0;
    }
}

ll getch(ll x) { return ch[f[x]][1] == x; }

bool isroot(ll x) { return ch[f[x]][0] != x && ch[f[x]][1] != x; }

inline void rotate(ll x) {
    ll y = f[x], z = f[y], k = getch(x);
    if (!isroot(y)) ch[z][ch[z][1] == y] = x;
    // 上面这句一定要写在前面，普通的Splay 是不用的，因为 isRoot  (后面会讲)
    ch[y][k] = ch[x][!k], f[ch[x][!k]] = y;
    ch[x][!k] = y, f[y] = x, f[x] = z;
    pushup(y), pushup(x);
}

// 从上到下一层一层 pushDown 即可
void update(ll p) {
    if (!isroot(p)) update(f[p]);
    pushdown(p);
}

inline void splay(ll x) {
    update(x);  // 马上就能看到啦。 在
    // Splay 之前要把旋转会经过的路径上的点都PushDown
    for (ll fa; fa = f[x], !isroot(x); rotate(x)) {
        if (!isroot(fa)) rotate(getch(fa) == getch(x) ? fa : x);
    }
}

// 回顾一下代码
inline void access(ll x) {
    for (ll p = 0; x; p = x, x = f[x]) {
        splay(x), siz2[x] += siz[ch[x][1]] - siz[p], ch[x][1] = p, pushu
p(x);
    }
}

inline void makeroot(ll p) {
    access(p);
    splay(p);
    swap(ch[p][0], ch[p][1]);
    tag[p] ^= 1;
}

inline void split(ll a, ll b) {
    makeroot(a);
    access(b);
    splay(b);
}


inline ll find(ll p) {
    access(p), splay(p);
    while (ch[p][0]) pushdown(p), p = ch[p][0];
    splay(p);
```

```
    return p;
}

inline void link(ll x, ll y) {
    makeroot(y);
    makeroot(x);
    if (find(y) != x) {
        f[x] = y;
        siz2[y] += siz[x];
    }
}

inline void cut(ll x, ll y) {
    makeroot(x);
    if (find(y) == x && f[y] == x) {
        ch[x][1] = f[y] = 0;
        pushup(x);
    }
}

void init(int n) {
    for (int i = 1; i <= n; i++) siz[i] = 1;
}
```

Splay

```
ll ch[N][2], f[N], sum[N], val[N], tag[N], siz[N];

inline void pushup(ll p) {
    sum[p] = sum[ch[p][0]] ^ sum[ch[p][1]] ^ val[p];
    siz[p] = siz[ch[p][0]] + siz[ch[p][1]] + 1;
}

inline void pushdown(ll p) {
    if (tag[p]) {
        if (ch[p][0]) swap(ch[ch[p][0]][0], ch[ch[p][0]][1]), tag[ch[p]
[0]] ^= 1;
        if (ch[p][1]) swap(ch[ch[p][1]][0], ch[ch[p][1]][1]), tag[ch[p]
[1]] ^= 1;
        tag[p] = 0;
    }
}

ll getch(ll x) { return ch[f[x]][1] == x; }

bool isroot(ll x) { return ch[f[x]][0] != x && ch[f[x]][1] != x; }

inline void rotate(ll x) {
    ll y = f[x], z = f[y], k = getch(x);
    if (!isroot(y)) ch[z][ch[z][1] == y] = x;
    // 上面这句一定要写在前面，普通的Splay 是不用的，因为 isRoot  (后面会讲)
    ch[y][k] = ch[x][!k], f[ch[x][!k]] = y;
    ch[x][!k] = y, f[y] = x, f[x] = z;
    pushup(y), pushup(x);
}

// 从上到下一层一层 pushDown  即可
void update(ll p) {
    if (!isroot(p)) update(f[p]);
    pushdown(p);
}

inline void splay(ll x) {
    update(x);  // 马上就能看到啦。 在
    // Splay 之前要把旋转会经过的路径上的点都PushDown
    for (ll fa; fa = f[x], !isroot(x); rotate(x)) {
        if (!isroot(fa)) rotate(getch(fa) == getch(x) ? fa : x);
    }
}

// 回顾一下代码
inline void access(ll x) {
    for (ll p = 0; x; p = x, x = f[x]) {
        splay(x), ch[x][1] = p, pushup(x);
    }
}

inline void makeroot(ll p) {
    access(p);
    splay(p);
    swap(ch[p][0], ch[p][1]);
    tag[p] ^= 1;
}

inline void split(ll a, ll b) {
    makeroot(a);
    access(b);
    splay(b);
}
```

```cpp
inline ll find(ll p) {
    access(p), splay(p);
    while (ch[p][0]) pushdown(p), p = ch[p][0];
    splay(p);
    return p;
}

inline void link(ll x, ll y) {
    makeroot(x);
    if (find(y) != x) f[x] = y;
}

inline void cut(ll x, ll y) {
    makeroot(x);
    if (find(y) == x && f[y] == x) {
        ch[x][1] = f[y] = 0;
        pushup(x);
    }
}
```

ST 表

```cpp
#include <bits/stdc++.h>

using namespace std;
const int logn = 21;
const int N = 2000001;
int f[N][logn + 1], lg[N + 1];

void pre() {
    lg[1] = 0;
    for (int i = 2; i < N; i++) {
        lg[i] = lg[i / 2] + 1;
    }
}

int main() {
    ios::sync_with_stdio(false);
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= n; i++) cin >> f[i][0];
    pre();
    for (int j = 1; j <= logn; j++)
        for (int i = 1; i + (1 << j) - 1 <= n; i++)
            f[i][j] = max(f[i][j - 1], f[i + (1 << (j - 1))][j - 1]);
    for (int i = 1; i <= m; i++) {
        int x, y;
        cin >> x >> y;
        int s = lg[y - x + 1];
        printf("%d\n", max(f[x][s], f[y - (1 << s) + 1][s]));
    }
    return 0;
}
```

Treap

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

```
    if (d == -1) {
        if (a->ch[0] == NULL)
            a = a->ch[1];
        else if (a->ch[1] == NULL)
            a = a->ch[0];
        else {
            int d2 = a->ch[1]->r > a->ch[0]->r ? 0 : 1;
            rotate(a, d2);
            remove(a->ch[d2], x);
        }
    } else {
        remove(a->ch[d], x);
    }
}
int find(node*& a, int x) {
    if (a == NULL)
        return 0;
    else if (a->v == x)
        return 1;
    else {
        int d = a->cmp(x);
        return find(a->ch[d], x);
    }
}
int main() {
    node* a = NULL;
    int k, l;
    while (cin >> k >> l) {
        if (k == 1)
            insert(a, l);
        else if (k == 2)
            remove(a, l);
        else {
            cout << find(a, l) << endl;
        }
    }
}
```

y 总 Splay Plus

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>

using namespace std;

const int N = 500010, INF = 1e9;

int n, m;
struct Node
{
    int s[2], p, v;
    int rev, same;
    int size, sum, ms, ls, rs;

    void init(int _v, int _p)
    {
        s[0] = s[1] = 0, p = _p, v = _v;
        rev = same = 0;
        size = 1, sum = ms = v;
        ls = rs = max(v, 0);
    }
}tr[N];
int root, nodes[N], tt;
int w[N];

void pushup(int x)
{
    auto &u = tr[x], &l = tr[u.s[0]], &r = tr[u.s[1]];
    u.size = l.size + r.size + 1;
    u.sum = l.sum + r.sum + u.v;
    u.ls = max(l.ls, l.sum + u.v + r.ls);
    u.rs = max(r.rs, r.sum + u.v + l.rs);
    u.ms = max(max(l.ms, r.ms), l.rs + u.v + r.ls);
}

void pushdown(int x)
{
    auto &u = tr[x], &l = tr[u.s[0]], &r = tr[u.s[1]];
    if (u.same)
    {
        u.same = u.rev = 0;
        if (u.s[0]) l.same = 1, l.v = u.v, l.sum = l.v * l.size;
        if (u.s[1]) r.same = 1, r.v = u.v, r.sum = r.v * r.size;
        if (u.v > 0)
        {
            if (u.s[0]) l.ms = l.ls = l.rs = l.sum;
            if (u.s[1]) r.ms = r.ls = r.rs = r.sum;
        }
```

```
        else
        {
            if (u.s[0]) l.ms = l.v, l.ls = l.rs = 0;
            if (u.s[1]) r.ms = r.v, r.ls = r.rs = 0;
        }
    }
    else if (u.rev)
    {
        u.rev = 0, l.rev ^= 1, r.rev ^= 1;
        swap(l.ls, l.rs), swap(r.ls, r.rs);
        swap(l.s[0], l.s[1]), swap(r.s[0], r.s[1]);
    }
}

void rotate(int x)
{
    int y = tr[x].p, z = tr[y].p;
    int k = tr[y].s[1] == x;
    tr[z].s[tr[z].s[1] == y] = x, tr[x].p = z;
    tr[y].s[k] = tr[x].s[k ^ 1], tr[tr[x].s[k ^ 1]].p = y;
    tr[x].s[k ^ 1] = y, tr[y].p = x;
    pushup(y), pushup(x);
}

void splay(int x, int k)
{
    while (tr[x].p != k)
    {
        int y = tr[x].p, z = tr[y].p;
        if (z != k)
            if ((tr[y].s[1] == x) ^ (tr[z].s[1] == y)) rotate(x);
            else rotate(y);
        rotate(x);
    }
    if (!k) root = x;
}

int get_k(int k)
{
    int u = root;
    while (u)
    {
        pushdown(u);
        if (tr[tr[u].s[0]].size >= k) u = tr[u].s[0];
        else if (tr[tr[u].s[0]].size + 1 == k) return u;
        else k -= tr[tr[u].s[0]].size + 1, u = tr[u].s[1];
    }
}

int build(int l, int r, int p)
{
    int mid = l + r >> 1;
    int u = nodes[tt -- ];
    tr[u].init(w[mid], p);
    if (l < mid) tr[u].s[0] = build(l, mid - 1, u);
    if (mid < r) tr[u].s[1] = build(mid + 1, r, u);
    pushup(u);
    return u;
}

void dfs(int u)
{
    if (tr[u].s[0]) dfs(tr[u].s[0]);
    if (tr[u].s[1]) dfs(tr[u].s[1]);
    nodes[ ++ tt] = u;
}

int main()
{
    for (int i = 1; i < N; i ++ ) nodes[ ++ tt] = i;
    scanf("%d%d", &n, &m);
    tr[0].ms = w[0] = w[n + 1] = -INF;
    for (int i = 1; i <= n; i ++ ) scanf("%d", &w[i]);
    root = build(0, n + 1, 0);

    char op[20];
    while (m -- )
    {
        scanf("%s", op);
        if (!strcmp(op, "INSERT"))
        {
            int posi, tot;
            scanf("%d%d", &posi, &tot);
            for (int i = 0; i < tot; i ++ ) scanf("%d", &w[i]);
            int l = get_k(posi + 1), r = get_k(posi + 2);
            splay(l, 0), splay(r, l);
            int u = build(0, tot - 1, r);
            tr[r].s[0] = u;
            pushup(r), pushup(l);
        }
        else if (!strcmp(op, "DELETE"))
        {
```

```
        int posi, tot;
        scanf("%d%d", &posi, &tot);
        int l = get_k(posi), r = get_k(posi + tot + 1);
        splay(l, 0), splay(r, l);
        dfs(tr[r].s[0]);
        tr[r].s[0] = 0;
        pushup(r), pushup(l);
    }
    else if (!strcmp(op, "MAKE-SAME"))
    {
        int posi, tot, c;
        scanf("%d%d%d", &posi, &tot, &c);
        int l = get_k(posi), r = get_k(posi + tot + 1);
        splay(l, 0), splay(r, l);
        auto& son = tr[tr[r].s[0]];
        son.same = 1, son.v = c, son.sum = c * son.size;
        if (c > 0) son.ms = son.ls = son.rs = son.sum;
        else son.ms = c, son.ls = son.rs = 0;
        pushup(r), pushup(l);
    }
    else if (!strcmp(op, "REVERSE"))
    {
        int posi, tot;
        scanf("%d%d", &posi, &tot);
        int l = get_k(posi), r = get_k(posi + tot + 1);
        splay(l, 0), splay(r, l);
        auto& son = tr[tr[r].s[0]];
        son.rev ^= 1;
        swap(son.ls, son.rs);
        swap(son.s[0], son.s[1]);
        pushup(r), pushup(l);
    }
    else if (!strcmp(op, "GET-SUM"))
    {
        int posi, tot;
        scanf("%d%d", &posi, &tot);
        int l = get_k(posi), r = get_k(posi + tot + 1);
        splay(l, 0), splay(r, l);
        printf("%d\n", tr[tr[r].s[0]].sum);
    }
    else printf("%d\n", tr[root].ms);
  }

  return 0;
}
```

y 总 Splay

```
#include <bits/stdc++.h>

using namespace std;
const int N = 1e6 + 10;
struct node {
    int p, v, s[2];
    int siz, tag;
    void init(int _v, int _p) {
        v = _v, p = _p;
        siz = 1;
    }
};
node tr[N];
int root, idx;

void pushup(int x) { tr[x].siz = tr[tr[x].s[0]].siz + tr[tr[x].s[1]].si
z + 1; }

void pushdown(int x) {
    if (tr[x].tag) {
        swap(tr[x].s[0], tr[x].s[1]);
        tr[tr[x].s[0]].tag ^= 1;
        tr[tr[x].s[1]].tag ^= 1;
        tr[x].tag = 0;
    }
}
void rotate(int x) {
         pushdown(x);
    int y = tr[x].p, z = tr[y].p;
    int k = tr[y].s[1] == x;
    tr[y].s[k] = tr[x].s[k ^ 1], tr[tr[y].s[k]].p = y;
    tr[x].s[k ^ 1] = y, tr[y].p = x;
    tr[z].s[tr[z].s[1] == y] = x, tr[x].p = z;
    pushup(y), pushup(x);
}

void splay(int x, int k) {
    while (tr[x].p != k) {
        int y = tr[x].p, z = tr[y].p;
        if (z != k) {
            if ((tr[z].s[1] == y) ^ (tr[y].s[1] == x)) {
                rotate(x);
            } else {
                rotate(y);
```

```
            }
        }
        rotate(x);
    }
    if (!k) root = x;
}

void insert(int v) {
    int u = root, p = 0;
    while (u) p = u, u = tr[u].s[v > tr[u].v];
    u = ++idx;
    if (p) tr[p].s[v > tr[p].v] = u;
    tr[u].init(v, p);
    splay(u, 0);
}

int getk(int k) {
    int u = root;
    while (1) {
        pushdown(u);
        if (k <= tr[tr[u].s[0]].siz) {
            u = tr[u].s[0];
        } else if (k == tr[tr[u].s[0]].siz + 1) {
            splay(u, 0);
            return u;
        } else {
            k -= tr[tr[u].s[0]].siz + 1, u = tr[u].s[1];
        }
    }
}

int n, m;
void output(int u) {
    if (u == 0) return;
    pushdown(u);
    output(tr[u].s[0]);
    if (1 <= tr[u].v && tr[u].v <= n) cout << tr[u].v << ' ';
    output(tr[u].s[1]);
}

int main() {
    ios::sync_with_stdio(0), cin.tie(0), cout.tie(0);
    cin >> n >> m;
    for (int i = 0; i <= n + 1; i++) insert(i);
    while (m--) {
        int a, b;
        cin >> a >> b;
        int id1 = getk(a), id2 = getk(b + 2);
        splay(id1, 0), splay(id2, id1);
        tr[tr[id2].s[0]].tag ^= 1;
    }
    output(root);
}
```

主席树

```
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;
const ll N = 1 << 20;

ll ch[N << 5][2], rt[N], tot;
ll val[N << 5];

ll update(ll a, ll b) {
    return a + b;
}

ll build(ll l, ll r) {  // 建树
    ll p = ++tot;
    if (l == r) {
        //初始化
        val[p] = 0;
        return p;
    }
    ll mid = (l + r) >> 1;
    ch[p][0] = build(l, mid);
    ch[p][1] = build(mid + 1, r);
    val[p] = update(val[ch[p][0]], val[ch[p][1]]);
    return p;  // 返回该子树的根节点
}

ll modify(ll pre, ll l, ll r, ll pos, ll v) {  // 插入操作
    ll now = ++tot;
    ch[now][0] = ch[pre][0], ch[now][1] = ch[pre][1];
    if (l == r) {
        val[now] = val[pre] + v;
        return now;
    }
```

```
    ll mid = (l + r) >> 1;
    if (pos <= mid)
        ch[now][0] = modify(ch[now][0], l, mid, pos, v);
    else
        ch[now][1] = modify(ch[now][1], mid + 1, r, pos, v);
    val[now] = update(val[ch[now][0]], val[ch[now][1]]);
    return now;
}

ll kth(ll pre, ll now, ll l, ll r, ll k) {  // 查询操作
    ll mid = (l + r) >> 1;
    ll x = val[ch[now][0]] - val[ch[pre][0]];  // 通过区间减法得到左儿子的信息
    if (l == r) return l;
    if (k <= x)  // 说明在左儿子中
        return kth(ch[pre][0], ch[now][0], l, mid, k);
    else  // 说明在右儿子中
        return kth(ch[pre][1], ch[now][1], mid + 1, r, k - x);
}

ll query(ll pre, ll now, ll l, ll r, ll ql, ll qr) {  // 查询操作
    if (ql <= l && r <= qr) {
        return val[now] - val[pre];
    }
    if (qr < l || r < ql) {
        return 0;
    }
    ll mid = (l + r) >> 1;
    ll lv = query(ch[pre][0], ch[now][0], l, mid, ql, qr);
    ll rv = query(ch[pre][1], ch[now][1], mid + 1, r, ql, qr);
    return update(lv, rv);
}
//修改查询记得用rt[]!!!
```

仙人掌

```
/*
 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。
 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
 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
 */

#include <iostream>
#include <cstring>
#include <algorithm>

using namespace std;

const int N = 12010, M = N * 3;

int n, m, Q, new_n;
int h1[N], h2[N], e[M], w[M], ne[M], idx;
int dfn[N], low[N], cnt;
int s[N], stot[N], fu[N], fw[N];
int fa[N][14], depth[N], d[N];
int A, B;

void add(int h[], int a, int b, int c)
{
    e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx ++ ;
}

void build_circle(int x, int y, int z)
{
    int sum = z;
    for (int k = y; k != x; k = fu[k])
    {
        s[k] = sum;
        sum += fw[k];
    }
    s[x] = stot[x] = sum;
    add(h2, x, ++ new_n, 0);
    for (int k = y; k != x; k = fu[k])
    {
        stot[k] = sum;
        add(h2, new_n, k, min(s[k], sum - s[k]));
    }
}

void tarjan(int u, int from)
{
    dfn[u] = low[u] = ++ cnt;
    for (int i = h1[u]; ~i; i = ne[i])
    {
        int j = e[i];
        if (!dfn[j])
        {
            fu[j] = u, fw[j] = w[i];
            tarjan(j, i);
```

```
            low[u] = min(low[u], low[j]);
            if (dfn[u] < low[j]) add(h2, u, j, w[i]);
        }
        else if (i != (from ^ 1)) low[u] = min(low[u], dfn[j]);
    }
    for (int i = h1[u]; ~i; i = ne[i])
    {
        int j = e[i];
        if (dfn[u] < dfn[j] && fu[j] != u)
            build_circle(u, j, w[i]);
    }
}

void dfs_lca(int u, int father)
{
    depth[u] = depth[father] + 1;
    fa[u][0] = father;
    for (int k = 1; k <= 13; k ++ )
        fa[u][k] = fa[fa[u][k - 1]][k - 1];
    for (int i = h2[u]; ~i; i = ne[i])
    {
        int j = e[i];
        d[j] = d[u] + w[i];
        dfs_lca(j, u);
    }
}

int lca(int a, int b)
{
    if (depth[a] < depth[b]) swap(a, b);
    for (int k = 13; k >= 0; k -- )
        if (depth[fa[a][k]] >= depth[b])
            a = fa[a][k];
    if (a == b) return a;
    for (int k = 13; k >= 0; k -- )
        if (fa[a][k] != fa[b][k])
        {
            a = fa[a][k];
            b = fa[b][k];
        }
    A = a, B = b;
    return fa[a][0];
}

int main()
{
    scanf("%d%d%d", &n, &m, &Q);
    new_n = n;
    memset(h1, -1, sizeof h1);
    memset(h2, -1, sizeof h2);
    while (m -- )
    {
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        add(h1, a, b, c), add(h1, b, a, c);
    }
    tarjan(1, -1);
    dfs_lca(1, 0);

    while (Q -- )
    {
        int a, b;
        scanf("%d%d", &a, &b);
        int p = lca(a, b);
        if (p <= n) printf("%d\n", d[a] + d[b] - d[p] * 2);
        else
        {
            int da = d[a] - d[A], db = d[b] - d[B];
            int l = abs(s[A] - s[B]);
            int dm = min(l, stot[A] - l);
            printf("%d\n", da + dm + db);
        }
    }

    return 0;
}
```

区间 max

```
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;
const int N = 1 << 20;

struct node {
    int mmax, semax, cnt;
    ll sum;
};
```

```cpp
node tree[N << 1];
int init[N << 1];

node merge_range(node a, node b) {
    node ans;
    ans.sum = a.sum + b.sum;
    if (a.mmax == b.mmax) {
        ans.mmax = a.mmax;
        ans.cnt = a.cnt + b.cnt;
        ans.semax = max(a.semax, b.semax);
    } else {
        if (a.mmax < b.mmax) swap(a, b);
        ans.mmax = a.mmax;
        ans.cnt = a.cnt;
        ans.semax = max(a.semax, b.mmax);
    }
    return ans;
}

void build(int k, int l, int r) {
    if (l == r) {
        tree[k] = {init[l], -1, 1, init[l]};
        return;
    }
    int mid = (l + r) >> 1;
    build(k << 1, l, mid);
    build(k << 1 | 1, mid + 1, r);
    tree[k] = merge_range(tree[k << 1], tree[k << 1 | 1]);
}


void pushdown(int k, int l, int r) {
    if (l == r) return;
    if (tree[k].mmax < tree[k << 1].mmax) {
        tree[k << 1].sum -= 1LL * (tree[k << 1].mmax - tree[k].mmax) * tr
ee[k << 1].cnt;
        tree[k << 1].mmax = tree[k].mmax;
    }
    if (tree[k].mmax < tree[k << 1 | 1].mmax) {
        tree[k << 1 | 1].sum -= 1LL * (tree[k << 1 | 1].mmax - tree[k].mm
ax) * tree[k << 1 | 1].cnt;
        tree[k << 1 | 1].mmax = tree[k].mmax;
    }
}
```

```cpp
node query(int k, int l, int r, int ql, int qr) {
    if (qr < l || r < ql) return {0, -1, 1, 0};
    if (ql <= l && r <= qr) {
        return tree[k];
    }
    pushdown(k, l, r);
    int mid = (l + r) >> 1;
    node lq = query(k << 1, l, mid, ql, qr);
    node rq = query(k << 1 | 1, mid + 1, r, ql, qr);
    return merge_range(lq, rq);
}

void modify(int k, int l, int r, int ql, int qr, int x) {
    if (qr < l || r < ql) return;
    if (ql <= l && r <= qr && tree[k].semax < x) {
        if (x < tree[k].mmax) {
            tree[k].sum -= 1LL * (tree[k].mmax - x) * tree[k].cnt;
            tree[k].mmax = x;
        }
        return;
    }
    pushdown(k, l, r);
    int mid = (l + r) >> 1;
    modify(k << 1, l, mid, ql, qr, x);
    modify(k << 1 | 1, mid + 1, r, ql, qr, x);
    tree[k] = merge_range(tree[k << 1], tree[k << 1 | 1]);
}


signed main() {
//    freopen("data.txt", "r", stdin);
//    freopen("test1.txt", "w", stdout);
    int t;
    scanf("%d", &t);
    while (t--) {
        int n, q;
        scanf("%d%d", &n, &q);
        for (int i = 1; i <= n; i++) scanf("%d", &init[i]);
        build(1, 1, n);
        while (q--) {
            int x, y, op, val;
            scanf("%d%d%d", &op, &x, &y);
            if (op == 0) {
                scanf("%d", &val);
                modify(1, 1, n, x, y, val);
            } else if (op == 1) {
```

```
                node ans = query(1, 1, n, x, y);
                printf("%d\n", ans.mmax);
            } else {
                node ans = query(1, 1, n, x, y);
                printf("%lld\n", ans.sum);
            }
        }
    }
}
```

回滚莫队

```
/*
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
对于都在块内的点直接暴力，否则跨块：
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
左端点每次暴力，右端点单调
*/

#include <iostream>
#include <cstring>
#include <cstdio>
#include <algorithm>
#include <cmath>
#include <vector>

using namespace std;

typedef long long LL;
const int N = 100010;

int n, m, len;
int w[N], cnt[N];
LL ans[N];
struct Query
{
    int id, l, r;
}q[N];
vector<int> nums;

int get(int x)
{
    return x / len;
}

bool cmp(const Query& a, const Query& b)
{
    int i = get(a.l), j = get(b.l);
    if (i != j) return i < j;
    return a.r < b.r;
}

void add(int x, LL& res)
{
    cnt[x] ++ ;
    res = max(res, (LL)cnt[x] * nums[x]);
}

int main()
{
    scanf("%d%d", &n, &m);
    len = sqrt(n);
    for (int i = 1; i <= n; i ++ ) scanf("%d", &w[i]), nums.push_back(w[i]);
    sort(nums.begin(), nums.end());
    nums.erase(unique(nums.begin(), nums.end()), nums.end());
    for (int i = 1; i <= n; i ++ )
        w[i] = lower_bound(nums.begin(), nums.end(), w[i]) - nums.begin();

    for (int i = 0; i < m; i ++ )
    {
        int l, r;
        scanf("%d%d", &l, &r);
        q[i] = {i, l, r};
    }
    sort(q, q + m, cmp);

    for (int x = 0; x < m;)
    {
        int y = x;
        while (y < m && get(q[y].l) == get(q[x].l)) y ++ ;
        int right = get(q[x].l) * len + len - 1;

        // 暴力求块内的询问
        while (x < y && q[x].r <= right)
        {
            LL res = 0;
            int id = q[x].id, l = q[x].l, r = q[x].r;
```

```
            for (int k = l; k <= r; k ++ ) add(w[k], res);
            ans[id] = res;
            for (int k = l; k <= r; k ++ ) cnt[w[k]] -- ;
            x ++ ;
        }

        // 求块外的询问
        LL res = 0;
        int i = right, j = right + 1;
        while (x < y)
        {
            int id = q[x].id, l = q[x].l, r = q[x].r;
            while (i < r) add(w[ ++ i], res);
            LL backup = res;
            while (j > l) add(w[ -- j], res);
            ans[id] = res;
            while (j < right + 1) cnt[w[j ++ ]] -- ;
            res = backup;
            x ++ ;
        }
        memset(cnt, 0, sizeof cnt);
    }

    for (int i = 0; i < m; i ++ ) printf("%lld\n", ans[i]);
    return 0;
}
```

带修莫队

```
#include <bits/stdc++.h>
using namespace std;

const int N = 10010;

int a[N], cnt[1000010], ans[N];

int len, mq, mc;

struct Query {
        int id, l, r, t;
} q[N];

struct Modify {
        int p, c;
} c[N];

int getNum(int x) {
        return x / len;
}

// l 所在块的编号，r 所在块的编号，t 升序

bool cmp(const Query& a, const Query& b) {
        if(getNum(a.l) == getNum(b.l) && getNum(a.r) == getNum(b.r)) {
                return a.t < b.t;
        }
        if(getNum(a.l) == getNum(b.l)) return a.r < b.r;
        return a.l < b.l;
}

void add(int x, int& res) {
    if (!cnt[x]) res ++ ;
    cnt[x] ++ ;
}

void del(int x, int& res) {
    cnt[x] -- ;
    if (!cnt[x]) res -- ;
}


int main() {
        ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);

        int n, m;
        cin >> n >> m;
        char op;
        int x, y;
        for(int i = 1; i <= n; ++ i) {
                cin >> a[i];
        }
        for(int i = 1; i <= m; ++ i) {
                cin >> op >> x >> y;
        if (op == 'Q') q[++ mq] = {mq, x, y, mc};
        else c[ ++ mc] = {x, y};
        }

  ///
        len = cbrt((double)n * mc) + 1;
```

```
  sort(q + 1, q + mq + 1, cmp);

        int i = 1, j = 0, t = 0, res = 0;
        for(int k = 1; k <= mq; ++ k) {
                int id = q[k].id, l = q[k].l, r = q[k].r, tm = q[k].t;
                while(j < r) add(a[++ j], res);
                while(j > r) del(a[j --], res);
                while(i < l) del(a[i ++], res);
                while(i > l) add(a[-- i], res);
                while(t < tm) {
                        ++ t;
                        if(c[t].p >= i && c[t].p <= j) {
                                del(a[c[t].p], res);
                                add(c[t].c, res);
                        }
                        swap(a[c[t].p], c[t].c);
                }
                while(t > tm) {
                        if(c[t].p >= i && c[t].p <= j) {
                                del(a[c[t].p], res);
                                add(c[t].c, res);
                        }
                        swap(a[c[t].p], c[t].c);
                        -- t;
                }
                ans[id] = res;
        }

        for(int i = 1; i <= mq; ++ i) {
                cout << ans[i] << endl;
        }
}
```

普通莫队

```
#include <bits/stdc++.h>
using namespace std;

const int N = 1e6 + 10, M = 1e6 + 10;
int a[N];

struct node {
        int id, l, r;
} mp[M];

int len;
int ans[M], cnt[1000010];

int getNum(int l) {
        return l / len;
}

//左指针的分块，右指针的大小
bool cmp (const node &a, const node & b) {
        if(getNum(a.l) == getNum(b.l)) return a.r < b.r;
        return a.l < b.l;
}
/* 奇偶优化
struct node {
  int l, r, id;
  bool operator<(const node &x) const {
    if (l / unit != x.l / unit) return l < x.l;
    if ((l / unit) & 1)
      return r <  x.r;  // 注意这里和下面一行不能写小于（大于）等于
    return r > x.r;
  }
};
*/

void add(int x, int& res) {
        if(cnt[x] == 0) res++;
        cnt[x] ++;
}

void del(int x, int& res) {
        cnt[x] --;
        if(cnt[x] == 0) res --;
}

int main() {
        ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);

        int n;
        cin >> n;
        for(int i = 1; i <= n; ++ i) {
                cin >> a[i];
        }
        int m;
        cin >> m;
```

```
        len = sqrt((double)n * n / m);
        for(int i = 1; i <= m; ++ i) {
                mp[i].id = i;
                cin >> mp[i].l >> mp[i].r;
        }
        sort(mp + 1, mp + m + 1, cmp);

        //离线处理询问
        int res = 0, i = 0, j = 0;
        for(int k = 1; k <= m; ++ k) {
                int id = mp[k].id, l = mp[k].l, r = mp[k].r;
                while(j < r) add(a[++j], res);
                while(j > r) del(a[j--], res);
                while(i < l) del(a[i++], res);
                while(i > l) add(a[--i], res);
                ans[id] = res;
        }

        for(int i = 1; i <= m; ++ i) {
                cout << ans[i] << endl;
        }
        return 0;
}
```

树状数组（fenwick）

```
template <typename T>
struct fenwick {
    vector<T> fenw;
    int n;

    fenwick(int _n) : n(_n) {
        fenw.resize(n);
    }

    void clear(){
        fenw.clear();
        fenw.resize(n);
    }

    void modify(int x, T v) {
        while (x < n) {
            fenw[x] += v;
            //if(fenw[x]>=mod)fenw[x]-=mod;
            x |= (x + 1);
        }
    }

    T get(int x) {
        T v{};
        while (x >= 0) {
            v += fenw[x];
            //if(v>=mod)v-=mod;
            x = (x & (x + 1)) - 1;
        }
        return v;
    }

    T gets(int l,int r){
        T res=get(r)-get(l-1);
        //if(res<0)res+=mod;
        return res;
    }
};
```

线段树合并分裂

```
ll nodetot, recycnt, bac[N << 5], ch[N << 5][2], rt[N];
ll val[N << 5];

ll newnod() { return (recycnt ? bac[recycnt--] : ++nodetot); }

void recyc(ll p) {
    bac[++recycnt] = p, ch[p][0] = ch[p][1] = val[p] = 0;
    return;
}

void pushdown(ll p) {

}

void pushup(ll p) {
    val[p] = 0;
    if (ch[p][0]) val[p] += val[ch[p][0]];
    if (ch[p][1]) val[p] += val[ch[p][1]];
}

void modify(ll &p, ll l, ll r, ll pos, ll v) {
```

```
    if (!p) { p = newnod(); }
    if (l == r) {
        val[p] += v;
        return;
    }
    ll mid = (l + r) >> 1;
//    pushdown(p);
    if (pos <= mid) { modify(ch[p][0], l, mid, pos, v); }
    else { modify(ch[p][1], mid + 1, r, pos, v); }
    pushup(p);
    return;
}

ll query(ll p, ll l, ll r, ll xl, ll xr) {
    if (xr < l || r < xl) { return 0; }
    if (xl <= l && r <= xr) { return val[p]; }
    ll mid = (l + r) >> 1;
//    pushdown(p);
    return query(ch[p][0], l, mid, xl, xr) + query(ch[p][1], mid + 1, r,
 xl, xr);
}

ll kth(ll p, ll l, ll r, ll k) {
    if (l == r) { return l; }
    ll mid = (l + r) >> 1;
//    pushdown(p);
    if (val[ch[p][0]] >= k) { return kth(ch[p][0], l, mid, k); }
    else { return kth(ch[p][1], mid + 1, r, k - val[ch[p][0]]); }
}

ll merge(ll x, ll y, ll l, ll r) {
    if (!x || !y) {
        return x + y;
    }   // 只有一边有点，不用合并
    ll p = newnod(); // 创建一个新结点 p
    if (l == r) {                  // 边界（某些时候可以省略，见下面一个代
码）
        val[p] = val[x] + val[y];
        return p;
    }
//    pushdown(x), pushdown(y);
    ll mid = (l + r) >> 1;
    ch[p][0] = merge(ch[x][0], ch[y][0], l, mid);
    ch[p][1] = merge(ch[x][1], ch[y][1], mid + 1, r);
    recyc(x), recyc(y);            // 垃圾回收
    pushup(p);                     // pushup
    return p;
}

void split(ll x, ll &y, ll k) {
    if (x == 0) return;
    y = newnod();
    ll v = val[ch[x][0]];
//    pushdown(x);
    if (k > v) { split(ch[x][1], ch[y][1], k - v); }
    else { swap(ch[x][1], ch[y][1]); }
    if (k < v) { split(ch[x][0], ch[y][0], k); }
    val[y] = val[x] - k;
    val[x] = k;
    return;
}
```

舞蹈链（多重覆盖）

```
#include <bits/stdc++.h>
using namespace std;
struct DLX {
    static const int maxn = 1000;     //列的上限
    static const int maxr = 1000;     //解的上限
    static const int maxnode = 5000;  //总结点数上限
    static const int INF = 1000000000;
    int n, sz;
    int S[maxn];

    int row[maxnode], col[maxnode];
    int L[maxnode], R[maxnode], U[maxnode], D[maxnode];

    int ansd, ans[maxr];

    int vis[maxnode];

    void init(int n) {
        this->n = n;

        //虚拟节点
        for (int i = 0; i <= n; i++) {
            U[i] = i;
            D[i] = i;
            L[i] = i - 1;
```

```
        R[i] = i + 1;
    }
    R[n] = 0;
    L[0] = n;

    sz = n + 1;
    memset(S, 0, sizeof(S));
}

void addRow(int r, vector<int> columns) {
    int first = sz;
    for (int i = 0; i < columns.size(); i++) {
        int c = columns[i];
        L[sz] = sz - 1;
        R[sz] = sz + 1;
        D[sz] = c;
        U[sz] = U[c];
        D[U[c]] = sz;
        U[c] = sz;
        row[sz] = r;
        col[sz] = c;
        S[c]++;
        sz++;
    }
    R[sz - 1] = first;
    L[first] = sz - 1;
}
#define FOR(i, A, s) for (int i = A[s]; i != s; i = A[i])
void remove(int c) {
    FOR(i, D, c) { L[R[i]] = L[i], R[L[i]] = R[i]; }
}

void restore(int c) {
    FOR(i, U, c) { L[R[i]] = i, R[L[i]] = i; }
}
int f_check()  //精确覆盖区估算剪枝
{
    /*
    强剪枝。这个
    剪枝利用的思想是A*搜索中的估价函数。即，对于当前的递归深度K 下的矩
阵，估计其最好情况下（即最少还需要多少步）才能出解。也就是，如果将能够覆盖当
    前列的所有行全部选中，去掉这些行能够覆盖到的列，将这个操作作为一层深
度。重复此操作直到所有列全部出解的深度是多少。如果当前深度加上这个估价函数返
    回值，其和已然不能更优（也就是已经超过当前最优解），则直接返回，不必再
搜。
    */

    int ret = 0;
    FOR(c, R, 0) vis[c] = true;
    FOR(c, R, 0)
    if (vis[c]) {
        ret++;
        vis[c] = false;
        FOR(i, D, c)
        FOR(j, R, i) vis[col[j]] = false;
    }
    return ret;
}
// d 为递归深度
void dfs(int d, vector<int>& v) {
    if (d + f_check() >= ansd) return;
    if (R[0] == 0) {
        if (d < ansd) {
            ansd = d;
            v.clear();
            for (int i = 0; i < ansd; i++) {
                v.push_back(ans[i]);
            }
        }       //找到解
        return;  //记录解的长度
    }

    //找到S 最小的列c
    int c = R[0];
    FOR(i, R, 0)
    if (S[i] < S[c])
        c = i;      //第一个未删除的列
                    //删除第c 列
    FOR(i, D, c) {  //用结点i 所在的行能覆盖的所有其他列
        ans[d] = row[i];
        remove(i);
        FOR(j, R, i) remove(j);  //删除结点i 所在的能覆的所有其他列
        dfs(d + 1, v);
        FOR(j, L, i) restore(j);
        restore(i);  //恢复结点i 所在的行能覆盖的所有其他列
    }               //恢复第c 列
}

bool solve(vector<int>& v) {
    v.clear();
```

```
        ansd = INF;
        dfs(0, v);
        return !v.empty();
    }
};
//使用时 init 初始化，vector 中存入 r 行结点列表用 addRow 加行，solve(ans)后答
案按行的选择在 ans 中
DLX dlx;
int main() {
    int n, m;
    cin >> n >> m;
    dlx.init(m);
    for (int i = 1; i <= n; i++) {
        vector<int> v;
        for (int j = 1; j <= m; j++) {
            int a;
            cin >> a;
            if (a == 1) v.push_back(j);
        }
        dlx.addRow(i, v);
    }
    vector<int> ans;
    dlx.solve(ans);
    for (int i = 0; i < ans.size(); i++) cout << ans[i];
}
```

舞蹈链（精确覆盖）

```
#include <bits/stdc++.h>
using namespace std;
struct DLX {
    static const int maxn = 1000;     //列的上限
    static const int maxr = 1000;     //解的上限
    static const int maxnode = 5000;  //总结点数上限
    int n, sz;
    int S[maxn];

    int row[maxnode], col[maxnode];
    int L[maxnode], R[maxnode], U[maxnode], D[maxnode];

    int ansd, ans[maxr];

    void init(int n) {
        this->n = n;
        //虚拟节点
        for (int i = 0; i <= n; i++) {
            U[i] = i;
            D[i] = i;
            L[i] = i - 1;
            R[i] = i + 1;
        }
        R[n] = 0;
        L[0] = n;

        sz = n + 1;
        memset(S, 0, sizeof(S));
    }

    void addRow(int r, vector<int> columns) {
        int first = sz;
        for (int i = 0; i < columns.size(); i++) {
            int c = columns[i];
            L[sz] = sz - 1;
            R[sz] = sz + 1;
            D[sz] = c;
            U[sz] = U[c];
            D[U[c]] = sz;
            U[c] = sz;
            row[sz] = r;
            col[sz] = c;
            S[c]++;
            sz++;
        }
        R[sz - 1] = first;
        L[first] = sz - 1;
    }
#define FOR(i, A, s) for (int i = A[s]; i != s; i = A[i])
    void remove(int c) {
        L[R[c]] = L[c];
        R[L[c]] = R[c];
        FOR(i, D, c)
        FOR(j, R, i) {
            U[D[j]] = U[j];
            D[U[j]] = D[j];
            --S[col[j]];
        }
    }

    void restore(int c) {
```

```
        FOR(i, U, c)
        FOR(j, L, i) {
            ++S[col[j]];
            U[D[j]] = j;
            D[U[j]] = j;
        }
        L[R[c]] = c;
        R[L[c]] = c;
    }

    // d 为递归深度
    bool dfs(int d) {
        if (R[0] == 0) {
            ansd = d;     //找到解
            return true;  //记录解的长度
        }

        //找到S 最小的列c
        int c = R[0];
        FOR(i, R, 0) if (S[i] < S[c]) c = i;  //第一个未删除的列

        remove(c);      //删除第 c 列
        FOR(i, D, c) {  //用结点i 所在的行能覆盖的所有其他列
            ans[d] = row[i];
            FOR(j, R, i) remove(col[j]);  //删除结点i 所在的能覆的所有其他
列

            if (dfs(d + 1)) return true;
            FOR(j, L, i) restore(col[j]);  //恢复结点i 所在的行能覆盖的所有
其他列
        }
        restore(c);  //恢复第 c 列

        return false;
    }

    bool solve(vector<int>& v) {
        v.clear();
        if (!dfs(0)) return false;
        for (int i = 0; i < ansd; i++) v.push_back(ans[i]);
        return true;
    }
};
//使用时 init 初始化，vector 中存入 r 行结点列表用 addRow 加行，solve(ans)后答
案按行的选择在 ans 中
```

数论

BSGS 扩展 BSGS

BSGS

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

$$t = x \times k - y, x \in [1,k], y \in [0,k-1]$$
$$t \in [1,k^2]$$

$$a^k x \equiv b \times a^y (\bmod p)$$

对 $b \times a^y$ 建立 hash 表，枚举 x 看是否有解

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

unordered_map<int , int> mp;

int bsgs(int a, int p, int b) {

        if (1 % p == b % p) return 0; // 特判0 是不是解
        mp.clear();

        int k = sqrt(p) + 1;

        for(int i = 0, j = b % p; i < k; ++ i, j = (ll)j * a % p) {
                mp[j] = i;
        }

        int ak = 1;
        for(int i = 0; i < k; ++i) {
                ak = (ll)ak * a % p;
        }

        for(int i = 1, j = ak % p; i <= k; ++ i, j = (ll)j * ak % p) {
                if(mp.count(j)) return (ll)i * k - mp[j];
        }
```

```
	return -1;
}

int main() {
	ios::sync_with_stdio(0);
	cin.tie(0); cout.tie(0);

	int a, p, b;
	while(cin >> a >> p >> b, a | p | b) {
		int res;
		res = bsgs(a, p, b);
		if(res == -1) {
			cout << "No Solution\n";
		}
		else {
			cout << res << endl;
		}
	}

	return 0;
}
```

扩展 BSGS

求$a^t \equiv b(\mod p)$ 的最小的 t

当$(a,p)!=1$

$(a,p)=d \quad d\nmid b$ 无解

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

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

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

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

unordered_map<ll, ll> mp;

ll bsgs(ll a, ll p, ll b) {

	if(1 % p == b % p) return 0; // 特判0 是不是解
	mp.clear();

	ll k = sqrt(p) + 1;

	for(ll i = 0, j = b % p; i < k; ++i, j = (ll)j * a % p) {
		mp[j] = i;
	}

	ll ak = 1;
	for(ll i = 0; i < k; ++i) {
		ak = (ll) ak * a % p;
	}

	for(ll i = 1, j = ak % p;i <= k; ++i, j = (ll)j * ak % p) {
		if(mp.count(j)) return (ll) i * k - mp[j];
	}

	return -1;
}

ll gcd(ll x, ll y) {
	return x % y == 0 ? y : gcd(y, x % y);
}

void extgcd(ll a,ll b,ll& d,ll& x,ll& y){
    if(!b){
        d = a; x = 1; y = 0;
    }
    else{
        extgcd(b, a%b, d, y, x);
        y -= x * (a / b);
    }
}

ll inverse(ll a,ll n){
    ll d,x,y;
    extgcd(a,n,d,x,y);
    return d == 1 ? (x + n) % n : -1;

}
```

```cpp
int main() {
	ll a, p, b;

	while(cin >> a >> p >> b, a | p | b) {
		ll d = gcd(a, p);
		if(d == 1) {
			ll res = bsgs(a, p, b);
			if(res == -1) {
				cout << "No Solution\n";
			}
			else {
				cout << res << endl;
			}
		}
		else {
			if(b % d != 0) {
				cout << "No Solution\n";
				continue;
			}
			else {
				p = p / d;
				b = (b / d) * inverse(a / d, p);
				ll res = bsgs(a, p, b);
				if(res == -1) {
					cout << "No Solution\n";
				}
				else {
					cout << res + 1 << endl;
				}
			}
		}
	}

	return 0;
}
```

burnside&polya

burnside 引理

burnside ：用$D(a_j)$ 表示在置换a_j下不变的元素的个数，L 表示本质不同的方案数（等价类）：

$$L = \frac{1}{|G|}\sum_{j=1}^{s} D(a_j)$$

定理：$|E_k| \times |Z_k| = |G|, k = 1,2,\ldots,n,$ 该定理的一个重要研究对象是群的元素个数，其中Z_k是 K 不动置换类，设 G 是 1,2，.. n 的置换群，若 k 是 1 到 n 中某个元素，则 G 中使 K 保持不变的置换的全体，为Z_k. E_k是等价类，设 G 是 1,2，.. n 的置换群，若 k 是 1 到 n 中某个元素，k 在 G 的作用下的轨迹，为E_k, 即 k 在 G 的作用下能变化成的所有元素的集合

$$\sum_{k=1}^{n}|Z_k| = \sum_{i=1}^{L}\sum_{k\in E_i}|Z_k| = \sum_{i=1}^{L}|E_i| \times |Z_i| = L \times |G|$$

每个置换的不动点的平均值就是不同方案数

任何一个置换都可以拆解成若干个循环置换

polya 定理

polya: 设 G 是 p 个对象的一个置换群，用 m 种颜色涂染 p 个对象，则不同染色的方案为：

$$L = \frac{1}{|G|}\left(m^{c(g_1)} + m^{c(g_2)} + \ldots + m^{c(g_s)}\right)$$

其中$G = \{g_1, g_2 \ldots g_s\}, c(g_i)$ 为置换g_i为置换的循环节数

浅证：$D(a_j) = m^{c(g_i)}$

每个置换的不动点有公式可以求 $\textit{每个循环的方案数}^{\textit{循环数}}$

(不同循环直接完全独立)

Cipolla

```cpp
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;

ll mod;
```

```cpp
ll I_mul_I; // 虚数单位的平方

struct Complex {
    ll real, imag;

    Complex(ll real = 0, ll imag = 0) : real(real), imag(imag) {}
};

inline bool operator==(Complex x, Complex y) {
    return x.real == y.real and x.imag == y.imag;
}

inline Complex operator*(Complex x, Complex y) {
    return Complex((x.real * y.real + I_mul_I * x.imag % mod * y.imag) %
 mod,
                   (x.imag * y.real + x.real * y.imag) % mod);
}

Complex power(Complex x, ll k) {
    Complex res = 1;
    while (k) {
        if (k & 1) res = res * x;
        x = x * x;
        k >>= 1;
    }
    return res;
}

bool check_if_residue(ll x) {
    return power(x, (mod - 1) >> 1) == 1;
}

void solve(ll n, ll &x0, ll &x1) {

    ll a = rand() % mod;
    while (!a or check_if_residue((a * a + mod - n) % mod))
        a = rand() % mod;
    I_mul_I = (a * a + mod - n) % mod;
    x0 = ll(power(Complex(a, 1), (mod + 1) >> 1).real);
    x1 = mod - x0;
}

signed main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);

    ll t;
    cin >> t;
    while (t--) {
        ll n;
        cin >> n >> mod;
        if (n == 0) {
            cout << 0 << endl;
            continue;
        }
        if (!check_if_residue(n)) {
            cout << "Hola!" << endl;
            continue;
        }
        ll x0, x1;
        solve(n, x0, x1);
        if (x0 > x1) swap(x0, x1);
        cout << x0 << ' ' << x1 << endl;
    }
}
```

exgcd

```cpp
ll ex_gcd(ll a, ll b, ll &x, ll &y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    ll d = ex_gcd(b, a % b, x, y);
    ll temp = x;
    x = y;
    y = temp - a / b * y;
    return d;
}
```

FFT

```cpp
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;
```

```cpp
const int N = 1e7 + 10;


const double Pi = acos(-1.0);

struct Complex {
    double x, y;

    Complex(double xx = 0, double yy = 0) { x = xx, y = yy; }
} a[N], b[N];

Complex operator+(Complex _a, Complex _b) { return Complex(_a.x + _b.x,
_a.y + _b.y); }

Complex operator-(Complex _a, Complex _b) { return Complex(_a.x - _b.x,
_a.y - _b.y); }

Complex operator*(Complex _a, Complex _b) {
    return Complex(_a.x * _b.x - _a.y * _b.y, _a.x * _b.y + _a.y * _b.x);
} //不懂的看复数的运算那部分

int L, r[N];
int limit = 1;

void fft(Complex *A, int type) {
    for (int i = 0; i < limit; i++)
        if (i < r[i]) swap(A[i], A[r[i]]); //求出要迭代的序列
    for (int mid = 1; mid < limit; mid <<= 1) { //待合并区间的长度的一半
        Complex Wn(cos(Pi / mid), type * sin(Pi / mid)); //单位根
        for (int R = mid << 1, j = 0; j < limit; j += R) { //R 是区间的长
度，j 表示前已经到哪个位置了
            Complex w(1, 0); //幂
            for (int k = 0; k < mid; k++, w = w * Wn) { //枚举左半部分
                Complex x = A[j + k], y = w * A[j + mid + k]; //蝴蝶效应
                A[j + k] = x + y;
                A[j + mid + k] = x - y;

            }
        }
    }
}

void FFT(int n, int m) {
    limit = 1;
    L = 0;
    while (limit <= n + m) limit <<= 1, L++;
    for (int i = 0; i < limit; i++) r[i] = (r[i >> 1] >> 1) | ((i & 1) <<
 (L - 1));
    // 在原序列中 i 与 i/2 的关系是 ： i 可以看做是 i/2 的二进制上的每一位左移
一位得来
    // 那么在反转后的数组中就需要右移一位，同时特殊处理一下奇数
    fft(a, 1), fft(b, 1);
    for (int i = 0; i <= limit; i++) a[i] = a[i] * b[i];
    fft(a, -1);
    for (int i = 0; i <= n + m; i++) a[i].x /= limit;
}

int main() {
    int n, m;
    cin >> n >> m;
    for (int i = 0; i <= n; i++) cin >> a[i].x;
    for (int i = 0; i <= m; i++) cin >> b[i].x;
    FFT(n, m);
    for (int i = 0; i <= n + m; i++) cout << (int) (a[i].x + 0.5) << ' ';
    return 0;
}
```

FWT

```cpp
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;
const int mod = 998244353;

void add(int &x, int y) {
    (x += y) >= mod && (x -= mod);
}

void sub(int &x, int y) {
    (x -= y) < 0 && (x += mod);
}

namespace FWT {
    int extend(int n) {
        int N = 1;
        for (; N < n; N <<= 1);
        return N;
    }
```

```cpp
void FWTor(std::vector<int> &a, bool rev) {
    int n = a.size();
    for (int l = 2, m = 1; l <= n; l <<= 1, m <<= 1) {
        for (int j = 0; j < n; j += l)
            for (int i = 0; i < m; i++) {
                if (!rev) add(a[i + j + m], a[i + j]);
                else sub(a[i + j + m], a[i + j]);
            }
    }
}

void FWTand(std::vector<int> &a, bool rev) {
    int n = a.size();
    for (int l = 2, m = 1; l <= n; l <<= 1, m <<= 1) {
        for (int j = 0; j < n; j += l)
            for (int i = 0; i < m; i++) {
                if (!rev) add(a[i + j], a[i + j + m]);
                else sub(a[i + j], a[i + j + m]);
            }
    }
}

void FWTxor(std::vector<int> &a, bool rev) {
    int n = a.size(), inv2 = (mod + 1) >> 1;
    for (int l = 2, m = 1; l <= n; l <<= 1, m <<= 1) {
        for (int j = 0; j < n; j += l)
            for (int i = 0; i < m; i++) {
                int x = a[i + j], y = a[i + j + m];
                if (!rev) {
                    a[i + j] = (x + y) % mod;
                    a[i + j + m] = (x - y + mod) % mod;
                } else {
                    a[i + j] = 1LL * (x + y) * inv2 % mod;
                    a[i + j + m] = 1LL * (x - y + mod) * inv2 % mod;
                }
            }
    }
}

std::vector<int> Or(std::vector<int> a1, std::vector<int> a2) {
    int n = std::max(a1.size(), a2.size()), N = extend(n);
    a1.resize(N), FWTor(a1, false);
    a2.resize(N), FWTor(a2, false);
    std::vector<int> A(N);
    for (int i = 0; i < N; i++) A[i] = 1LL * a1[i] * a2[i] % mod;
    FWTor(A, true);
    return A;
}

std::vector<int> And(std::vector<int> a1, std::vector<int> a2) {
    int n = std::max(a1.size(), a2.size()), N = extend(n);
    a1.resize(N), FWTand(a1, false);
    a2.resize(N), FWTand(a2, false);
    std::vector<int> A(N);
    for (int i = 0; i < N; i++) A[i] = 1LL * a1[i] * a2[i] % mod;
    FWTand(A, true);
    return A;
}

std::vector<int> Xor(std::vector<int> a1, std::vector<int> a2) {
    int n = std::max(a1.size(), a2.size()), N = extend(n);
    a1.resize(N), FWTxor(a1, false);
    a2.resize(N), FWTxor(a2, false);
    std::vector<int> A(N);
    for (int i = 0; i < N; i++) A[i] = 1LL * a1[i] * a2[i] % mod;
    FWTxor(A, true);
    return A;
}
};

int main() {
    int n;
    scanf("%d", &n);
    n = (1 << n);
    std::vector<int> a1(n), a2(n);
    for (int i = 0; i < n; i++) scanf("%d", &a1[i]);
    for (int i = 0; i < n; i++) scanf("%d", &a2[i]);
    std::vector<int> A;
    A = FWT::Or(a1, a2);
    for (int i = 0; i < n; i++) {
        printf("%d%c", A[i], " \n"[i == n - 1]);
    }
    A = FWT::And(a1, a2);
    for (int i = 0; i < n; i++) {
        printf("%d%c", A[i], " \n"[i == n - 1]);
    }
    A = FWT::Xor(a1, a2);
    for (int i = 0; i < n; i++) {
        printf("%d%c", A[i], " \n"[i == n - 1]);
    }
    return 0;
```

```
}
```

lucas 求组合数

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

ll p;

const int maxn = 1e5 + 10;

ll qpow(ll x, ll n){
	ll res = 1;
	while(n){
		if(n & 1) res = (res * x) % p;
		x = (x * x) % p;
		n >>= 1;
	}

	return res;
}

ll C(ll up, ll down){
	if(up > down) return 0;
	ll res = 1;

//	for(int i = up + 1; i <= down; ++ i){
//		res = (res * i) % p;
//	}
//	for(int i = 1; i <= down - up; ++ i){
//		res = (res * qpow(i, p - 2)) % p;
//	}

	for(int i = 1, j = down; i <= up; ++ i, -- j){
		res = (res * j) % p;
		res = (res * qpow(i, p - 2)) % p;
	}

	return res;
}

ll lucas(ll up, ll down){
	if(up < p && down < p) return C(up, down);
	return C(up % p, down % p) * lucas(up / p, down / p) % p;
}

int main(){
	ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);

	int T;
	cin >> T;
	while (T --){
		ll down, up;
		cin >> down >> up >> p;

		cout << lucas(up, down) % p << endl;
	}

	return 0;
}
```

min_25 筛

```
/*
https://loj.ac/p/6053
筛积性函数 f 的前缀和
f(1)=1
f(p^e)=f xor e
n<=1e10，LOJ 347ms 本地 1100ms
*/
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll mod=1e9+7,inv3=333333336;
const int N=1e5+5;//开到 sqrt(n)即可

ll prime[N],sp0[N],sp1[N],sp2[N],g0[N<<1],g1[N<<1],g2[N<<1];
ll pnum,min25n,sqrn,w[N<<1],ind1[N],ind2[N];
bool notp[N];

void pre() { //预处理，线性筛
    notp[1]=1;
    for(int i=1; i<N; i++) {
        if(!notp[i]) {
            prime[++pnum]=i;
```

```
            sp0[pnum]=(sp0[pnum-1]+1)%mod;//p^0 前缀和（p 指质数），可以按需增删，下标意义为第 pnum 个质数的前缀和，而 g 的实际下标意义为 w 之前的前缀和，两者有所区别
            sp1[pnum]=(sp1[pnum-1]+i)%mod;//p^1 前缀和
            sp2[pnum]=(sp2[pnum-1]+1ll*i*i)%mod;//p^2 前缀和
        }
        for(int j=1; j<=pnum&&prime[j]*i<N; j++) {
            notp[i*prime[j]]=1;
            if(i%prime[j]==0)break;
        }
    }
}

void min25(ll n) {
    ll tot=0;
    min25n=n;
    sqrn=sqrt(n);
    for(ll i=1; i<=n; i=n/(n/i)+1) {
        w[++tot]=n/i;//实际下标
        ll x=w[tot]%mod;
        g0[tot]=x-1;//x^0 前缀和
        g1[tot]=x*(x+1)/2%mod-1;//x^1 前缀和
        g2[tot]=x*(x+1)/2%mod*(2*x+1)%mod*inv3%mod-1;//x^2 前缀和
        if(n/i<=sqrn)ind1[n/i]=tot;//离散下标
        else ind2[n/(n/i)]=tot;//离散下标
    }
    for(int i=1; i<=pnum; i++) {//扩展埃氏筛，筛质数部分前缀和
        for(int j=1; j<=tot&&prime[i]*prime[i]<=w[j]; j++) {
            int id=w[j]/prime[i]<=sqrn?ind1[w[j]/prime[i]]:ind2[n/(w[j]/prime[i])];
            g0[j]-=(g0[id]-sp0[i-1]+mod)%mod;
            g1[j]-=prime[i]*(g1[id]-sp1[i-1]+mod)%mod;
            g2[j]-=prime[i]*prime[i]%mod*(g2[id]-sp2[i-1]+mod)%mod;
            g0[j]%=mod,g1[j]%=mod,g2[j]%=mod;
            if(g0[j]<0)g0[j]+=mod;
            if(g1[j]<0)g1[j]+=mod;
            if(g2[j]<0)g2[j]+=mod;
        }
    }
}

//该前缀和不计算 f(1)，需要自行加上
ll S(ll x,int y) {//x 以内最小质因子大于第 y 个因子的前缀和
    if(prime[y]>=x)return 0;
    int id=x<=sqrn?ind1[x]:ind2[min25n/x];
    ll ans=(((g1[id]-g0[id])-(sp1[y]-sp0[y]))%mod+mod)%mod;//x 以内大于第 y 个因子的质数部分前缀和
    if(x>=2&&y<1)ans=(ans+2)%mod;//特判包含 f(2)的情况
    for(int i=y+1; i<=pnum&&prime[i]*prime[i]<=x; i++) {//筛合数部分前缀和
        ll pe=prime[i];
        for(int e=1; pe<=x; e++,pe=pe*prime[i]) {
            ll fpe=prime[i]^e;//f(p^e)
            ans=(ans+fpe%mod*(S(x/pe,i)+(e!=1)))%mod;
        }
    }
    return ans%mod;
}

int main() {
    pre();//预处理一次即可
    ll n;
    scanf("%lld",&n);
    min25(n);//每个不同的 n 都要调用一次该函数，再调用 S(n,0)
    printf("%lld\n",S(n,0)+1);//加上 f(1)
    return 0;
}
```

NTT

```
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;

const int N = 4e6 + 10;
const ll mod = 998244353, G = 3, Gi = 332748118;

int limit = 1, L, r[N];
ll a[N], b[N];

ll qpow(ll _a, ll _b) {
    ll ans = 1;
    while (_b) {
        if (_b & 1) ans = (ans * _a) % mod;
        _b >>= 1;
        _a = (_a * _a) % mod;
    }
```

```
    return ans;
}

void ntt(ll *A, int type) {
    auto swap = [](ll &_a, ll &_b) {
        _a ^= _b, _b ^= _a, _a ^= _b;
    };
    for (int i = 0; i < limit; i++)
        if (i < r[i]) swap(A[i], A[r[i]]);
    for (int mid = 1; mid < limit; mid <<= 1) {
        ll Wn = qpow(type == 1 ? G : Gi, (mod - 1) / (mid << 1));
        for (int j = 0; j < limit; j += (mid << 1)) {
            ll w = 1;
            for (int k = 0; k < mid; k++, w = (w * Wn) % mod) {
                int x = A[j + k], y = w * A[j + k + mid] % mod;
                A[j + k] = (x + y) % mod,
                        A[j + k + mid] = (x - y + mod) % mod;
            }
        }
    }
}

void NTT(int n, int m) {
    limit = 1;
    L = 0;
    while (limit <= n + m) limit <<= 1, L++;
    for (int i = 0; i < limit; i++) r[i] = (r[i >> 1] >> 1) | ((i & 1) <<
 (L - 1));
    ntt(a, 1), ntt(b, 1);
    for (int i = 0; i < limit; i++) a[i] = (a[i] * b[i]) % mod;
    ntt(a, -1);
    ll inv = qpow(limit, mod - 2);
    for (int i = 0; i <= n + m; i++) a[i] = a[i] * inv % mod;
}

int main() {
    int n, m;
    cin >> n >> m;
    for (int i = 0; i <= n; i++) {
        cin >> a[i];
        a[i] = (a[i] + mod) % mod;
    }
    for (int i = 0; i <= m; i++) {
        cin >> b[i];
        b[i] = (b[i] + mod) % mod;
    }
    NTT(n, m);
    for (int i = 0; i <= n + m; i++) cout << a[i] << ' ';
}

/*
#include<cstdio>
#include<cctype>
#include<cstring>
#include<cmath>
namespace fast_IO
{
        const int IN_LEN=10000000,OUT_LEN=10000000;
        char ibuf[IN_LEN],obuf[OUT_LEN],*ih=ibuf+IN_LEN,*oh=obuf,*lasti
n=ibuf+IN_LEN,*lastout=obuf+OUT_LEN-1;
        inline char getchar_(){return (ih==lastin)&&(lastin=(ih=ibuf)+f
read(ibuf,1,IN_LEN,stdin),ih==lastin)?EOF:*ih++;}
        inline void putchar_(const char x){if(oh==lastout)fwrite(obuf,1,
oh-obuf,stdout),oh=obuf;*oh++=x;}
        inline void flush(){fwrite(obuf,1,oh-obuf,stdout);}
}
using namespace fast_IO;
#define getchar() getchar_()
#define putchar(x) putchar_((x))
typedef long long LL;
#define rg register
template <typename T> inline T max(const T a,const T b){return a>b?a:b;}
template <typename T> inline T min(const T a,const T b){return a<b?a:b;}
template <typename T> inline T mind(T&a,const T b){a=a<b?a:b;}
template <typename T> inline T maxd(T&a,const T b){a=a>b?a:b;}
template <typename T> inline T abs(const T a){return a>0?a:-a;}
template <typename T> inline void swap(T&a,T&b){T c=a;a=b;b=c;}
template <typename T> inline void swap(T*a,T*b){T c=a;a=b;b=c;}
template <typename T> inline T gcd(const T a,const T b){if(!b)return a;r
eturn gcd(b,a%b);}
template <typename T> inline T square(const T x){return x*x;};
template <typename T> inline void read(T&x)
{
    char cu=getchar();x=0;bool fla=0;
    while(!isdigit(cu)){if(cu=='-')fla=1;cu=getchar();}
    while(isdigit(cu))x=x*10+cu-'0',cu=getchar();
    if(fla)x=-x;
}
template <typename T> void printe(const T x)
{
    if(x>=10)printe(x/10);
    putchar(x%10+'0');
```

```
}
template <typename T> inline void print(const T x)
{
    if(x<0)putchar('-'),printe(-x);
    else printe(x);
}
const int maxn=262145;
int n,m;
struct Ntt
{
        LL mod,a[maxn],b[maxn];;
        inline LL pow(LL x,LL y)
        {
                rg LL res=1;
                for(;y;y>>=1,x=x*x%mod)if(y&1)res=res*x%mod;
                return res;
        }
        int lenth,Reverse[maxn];
        inline void init(const int x)
        {
                rg int tim=0;lenth=1;
                while(lenth<=x)lenth<<=1,tim++;
                for(rg int i=0;i<lenth;i++)Reverse[i]=(Reverse[i>>1]>>1)
|((i&1)<<(tim-1));
        }
        inline void NTT(LL*A,const int fla)
        {
                for(rg int i=0;i<lenth;i++)if(i<Reverse[i])swap(A[i],A
[Reverse[i]]);
                for(rg int i=1;i<lenth;i<<=1)
                {
                        LL w=pow(3,(mod-1)/i/2);
                        if(fla==-1)w=pow(w,mod-2);
                        for(rg int j=0;j<lenth;j+=(i<<1))
                        {
                                LL K=1;
                                for(rg int k=0;k<i;k++,K=K*w%mod)
                                {
                                        const LL x=A[j+k],y=A[j+k+i]*
K%mod;
                                        A[j+k]=(x+y)%mod;
                                        A[j+k+i]=(mod+x-y)%mod;
                                }
                        }
                }
                if(fla==-1)
                {
                        const int inv=pow(lenth,mod-2);
                        for(rg int i=0;i<lenth;i++)A[i]=A[i]*inv%mod;
                }
        }
}Q[3];
LL EXgcd(const LL a,const LL b,LL &x,LL &y)
{
    if(!b)
    {
                x=1,y=0;
        return a;
    }
    const LL res=EXgcd(b,a%b,y,x);
    y-=a/b*x;
    return res;
}
inline LL msc(LL a,LL b,LL mod)
{
    LL v=(a*b-(LL)((long double)a/mod*b+1e-8)*mod);
    return v<0?v+mod:v;
}
int N,a[3],p[3];
LL CRT()
{
    LL P=1,sum=0;
    for(rg int i=1;i<=N;i++)P*=p[i];
    for(rg int i=1;i<=N;i++)
        {
        const LL m=P/p[i];
        LL x,y;
        EXgcd(p[i],m,x,y);
        sum=(sum+msc(msc(y,m,P),a[i],P))%P;
        }
        return sum;
}
int P;
int main()
{
        read(n),read(m),read(P);
        Q[0].mod=469762049,Q[0].init(n+m);
        Q[1].mod=998244353,Q[1].init(n+m);
        Q[2].mod=1004535809,Q[2].init(n+m);
        for(rg int i=0;i<=n;i++)read(Q[0].a[i]),Q[2].a[i]=Q[1].a[i]=Q
[0].a[i];
        for(rg int i=0;i<=m;i++)read(Q[0].b[i]),Q[2].b[i]=Q[1].b[i]=Q
```

```
[0].b[i];
        Q[0].NTT(Q[0].a,1),Q[0].NTT(Q[0].b,1);
        Q[1].NTT(Q[1].a,1),Q[1].NTT(Q[1].b,1);
        Q[2].NTT(Q[2].a,1),Q[2].NTT(Q[2].b,1);
        for(rg int i=0;i<Q[0].lenth;i++)
                Q[0].a[i]=(LL)Q[0].a[i]*Q[0].b[i]%Q[0].mod,
                Q[1].a[i]=(LL)Q[1].a[i]*Q[1].b[i]%Q[1].mod,
                Q[2].a[i]=(LL)Q[2].a[i]*Q[2].b[i]%Q[2].mod;
        Q[0].NTT(Q[0].a,-1);
        Q[1].NTT(Q[1].a,-1);
        Q[2].NTT(Q[2].a,-1);
        N=2,p[1]=Q[0].mod,p[2]=Q[1].mod;
        const int INV=Q[2].pow(Q[0].mod,Q[2].mod-2)*Q[2].pow(Q[1].mod,Q
[2].mod-2)%Q[2].mod;
        for(rg int i=0;i<=n+m;i++)
        {
                a[1]=Q[0].a[i],a[2]=Q[1].a[i];
                const LL ans1=CRT();
                const LL ans2=((Q[2].a[i]-ans1)%Q[2].mod+Q[2].mod)%Q[2].
mod*INV%Q[2].mod;
                print((ans2*Q[0].mod%P*Q[1].mod%P+ans1)%P),putchar('
');
        }
        return flush(),0;
}

*/
```

PN 筛+2022HDU 多校 5-1002

相关证明可以看：https://oi-wiki.org/math/number-theory/powerful-number/

PN 筛

积性函数：对于所有互质的a和b，总有$f(ab)=f(a)f(b)$，则称f为积性函数。

PN(power number)：对于正整数n，记n的质因数分解为$\prod p_i^{c_i}$。 是 PN 当且仅当$\forall 1<=i<=m, c_i>1$。（1 也是 PN）

性质 1：所有 PN 都可以表示成a^2b^3的形式，因为大于等于 2 的数总能分解成$2x+3y$的形式。

性质 2： n以内的 PN 至多有$O(\sqrt{n})$个，可以通过 dfs 枚举下一个质数的次数在$O(\sqrt{n})$时间内找到这些 PN。

已知f为积性函数，求$F(n)=\sum_{i=1}^{n} f(i)$

通过 PN 筛求解的一般过程：

1.构造积性函数g满足$g(p)=f(p)$，且$G(n)=\sum_{i=1}^{n} g(i)$能够快速求出（或者快速预处理$2\sqrt{n}$个有效值）

2.构造（其实不用真的构造）h满足$g*h=f$，且$h(p^c)$能够快速求出，由$g*h=f$可得h也是积性函数

3.其实h满足$h(1)=1, h(p)=0, f(p^c)=\sum_{i=0}^{c} g(p^i)h(p^{c-i})$即可，也可以移项递推$h(p^c)=f(p^c)-\sum_{i=1}^{c} g(p^i)h(p^{c-i})$

3.5.如果你在程序里递推$h(p^c)$复杂度会是$O\left(\sqrt{n}log(n)\right)$的，但是实测跑不满，耗时较少。

4.dfs 求出小于等于 n 的 PN 的同时计算$h(PN)$和答案，$F(n)=\sum_{i=1}^{n}[i是PN]\,h(i)G\left(\lfloor\frac{n}{i}\rfloor\right)$

复杂度与计算h和G的复杂度有关（有时需要预处理），两者均为$O(1)$时，总复杂度为$O(\sqrt{n})$

2022HDU 多校 5-1002

$f(p^c)=\frac{p^c}{c}$，求$\frac{1}{n}\sum_{i=1}^{n} f(i), n<=1e12$

构造积性函数$g(p^c)=p^c$,即$g(x)=x, G(x)=\frac{x(x+1)}{2}$，并求出$h$

这里先递推找规律：

c	$f(p^c)=\frac{p^c}{c}$	$g(p^c)=p^c$	$h(p^c)$
0	1	1	1
1	p	p	0

c	$f(p^c) = \frac{p^c}{c}$	$g(p^c) = p^c$	$h(p^c)$
2	$\frac{p^2}{2}$	p^2	$-\frac{p^2}{2}$
3	$\frac{p^3}{3}$	p^3	$-\frac{p^3}{6}$
4	$\frac{p^4}{4}$	p^4	$-\frac{p^4}{12}$
5	$\frac{p^5}{5}$	p^5	$-\frac{p^5}{20}$

容易发现$c >= 2$ 时$h(p^c) =-\frac{p^c}{c(c-1)}$

也可以直接推式子：

$$f(p^c) = \sum_{i=0}^{c} g(p^i)h(p^{c-i})$$

$$\frac{p^c}{c} = \sum_{i=0}^{c} p^i h(p^{c-i})$$

$$\frac{1}{c} = \sum_{i=0}^{c} \frac{h(p^{c-i})}{p^{c-i}} = \sum_{i=0}^{c} \frac{h(p^i)}{p^i}$$

$c >= 2$ 时：

$$\frac{1}{c} - \frac{1}{c-1} = \sum_{i=0}^{c} \frac{h(p^i)}{p^i} - \sum_{i=0}^{c-1} \frac{h(p^i)}{p^i}$$

$$\frac{h(p^c)}{p^c} = \frac{1}{c} - \frac{1}{c-1} = -\frac{1}{c(c-1)}$$

$$h(p^c) = -\frac{p^c}{c(c-1)}$$

预处理逆元（或者记忆化 h 函数后）可以$O(1)$求出。

dfs 出n以内的所有 PN，在 dfs 过程中，对于每个 PN——x,将$h(x)G\left(\lfloor\frac{n}{x}\rfloor\right)$累加到答案上即可。

实际代码时要注意的点：

1.质数需要处理到$\sqrt{n}$，可以多处理一点例如处理到 $100 + \sqrt{maxn}$

2.h 函数只会用到 PN 数处的值，预处理/记忆化时，只需要存$c >= 2$ 的$h(p^c)$，$h\left(p_1^{c_1}p_2^{c_2}p_3^{c_3}\right)$之类的可以直接在 dfs 中通过积性函数性质$O(1)$运算。

3.满足$c >= 2$ 且$p^c <= n$的p^c都是 PN，所以其数量是不超过$O\left(\sqrt{n}\right)$的。

4.$h(p^c)$可以按质数的下标（即第几个质数） 和c存来省空间。

5.对于同一个 n，G 的有效取值会有 $2\sqrt{n}$个，即$G(1)$、$G(2)...G\left(\sqrt{n}\right)$和$G\left(\frac{n}{1}\right)$、$G\left(\frac{n}{2}\right)...G\left(\frac{n}{\sqrt{n}}\right)$，有时需要预处理。

6.这题部分乘法要先转__int128 再乘再取模。

PN 筛

```
///2022 杭电多校 5-1002
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int,int> pii;

const int inf=0x3f3f3f3f,N=1e7+9;
const ll mod=4179340454199820289;

const int PMAX=N,PN=N;//PN 开到n 以内 P 的最大数量可以省空间
int prime[PN],pcnt;//prime[0]=1,prime[1]=2
bool notp[PMAX];//motp[1]=1,notp[2]=0,notp[4]=1
void Prime(){
    pcnt=0;
    for(int i=2;i<PMAX;i++){
                if(!notp[i])prime[++pcnt]=i;
                for(int j=1;j<=pcnt&&i*prime[j]<PMAX;j++){
                        notp[i*prime[j]]=1;
                        if(i%prime[j]==0)break;
```

```
            }
        }
        notp[1]=1;
}

ll qpow(ll a,ll b){
    ll ans=1;
    while(b){
        if(b&1)ans=(__int128)ans*a%mod;
        a=(__int128)a*a%mod;
        b>>=1;
    }
    return ans;
}

struct PNS{//修改G，f，g 即可使用，不同的n 只需要初始化ans 和n。
    ll ans,n;
    ll G(ll x){
        return (__int128)x*(x+1)/2%mod;
    }
    //f 和g 均只在h 函数中使用，如果可以直接公式求h 函数则不用定义这两个函数
    ll f(int pid,ll c){
        return (__int128)qpow(prime[pid],c)*qpow(c,mod-2)%mod;
    }
    //g 函数不用记忆化，实际调用次数很少
    ll g(int pid,ll c){
        return qpow(prime[pid],c);
    }

    vector<ll>vh[PN];//这里要确保c>2 时调用h(pid,c)前调用过h(pid,c-1)
    ll h(int pid,ll c){
        if(c==0)return 1;
        if(c==1)return 0;
        if(c-2>=(ll)vh[pid].size()){//n=1e12、1e13、1e14 时会进80070、230567、670121 次，跑不满根号n，所以递推的耗时也是不高的。
            //vh[pid].push_back((LL)((-(__int128)qpow(prime[pid],c)*qpow(c*(c-1),mod-2)%mod+mod)%mod));
            //递推h 函数，需要配合f 函数和g 函数一起使用
            ll ans=f(pid,c);
            for(ll i=1;i<=c;i++){
                ans=((ans-(__int128)g(pid,i)*h(pid,c-i))%mod+mod)%mod;
            }
            vh[pid].push_back(ans);
        }
        return vh[pid][c-2];
    }
    void dfs(ll prod,ll hprod,int pid){
        ans=(ans+(__int128)hprod*G(n/prod))%mod;
        for(int i=pid;i<=pcnt;i++){
            if(prod>n/prime[i]/prime[i])break;
            for(ll c=2,x=prod*prime[i]*prime[i];x<=n;c++,x*=prime[i]){
                dfs(x,(__int128)hprod*h(i,c)%mod,i+1);
                if(x>n/prime[i])break;
            }
        }
    }
}pns;

int main(){
    #ifdef ONLINE_JUDGE
        //std::ios::sync_with_stdio(false);
    #else
        freopen("1002.in","r",stdin);
        //freopen("out.txt","w",stdout);
    #endif
    Prime();
    int t;
    scanf("%d",&t);
    while(t--){
        pns.ans=0;
        scanf("%lld",&pns.n);
        pns.dfs(1,1,1);
        //printf("%lld\n",pns.ans);
        printf("%lld\n",(ll)((__int128)pns.ans*qpow(pns.n,mod-2)%mod));
    }
    return 0;
}
```

pn 筛_zyx

```
#include <bits/stdc++.h>

using namespace std;
#define de(x) cout << #x << " = " << x << endl
#define dd(x) cout << #x << " = " << x << " "
typedef long long ll;
const __int128 N = 1e6 + 10;
const __int128 M = 41;
const __int128 mod = 4179340454199820289ll;
```

```cpp
const __int128 inv2 = 208967022709991014511;

__int128 isp[N], pri[N], pcnt;
vector<__int128> h[N];

void getPrime() {
    fill(isp + 2, isp + N, 1);
    for (__int128 i = 2; i < N; i++) {
        if (isp[i]) {
            pri[pcnt++] = i;
        }
        for (__int128 j = 0; j < pcnt && i * pri[j] < N; j++) {
            isp[i * pri[j]] = 0;
            if (i % pri[j] == 0) break;
        }
    }
}

__int128 qpow(__int128 x, __int128 y) {
    __int128 ans = 1;
    while (y) {
        if (y & 1) ans = ans * x % mod;
        x = x * x % mod;
        y >>= 1;
    }
    return ans;
}

__int128 g(__int128 x) {
    return x;
}

__int128 G(__int128 x) {
    return x * (x + 1) % mod * inv2 % mod;
}

__int128 f(__int128 x, __int128 c) {
    return x * qpow(c, mod - 2) % mod;
}

__int128 ans;
ll n;

void dfs(__int128 deep, __int128 hpn, __int128 pn, bool flag) {
    if (flag) {
        ans = (ans + hpn * G(n / pn)) % mod;
    }
    if (deep >= pcnt) return;
    if (pri[deep] * pri[deep] * pn > n) return;
    dfs(deep + 1, hpn, pn, false);
    for (__int128 i = 2, pi = pri[deep] * pri[deep] % mod; pn * pi <= n;
i++, pi = pi * pri[deep] % mod) {
        dfs(deep + 1, hpn * h[deep][i] % mod, pn * pi, true);
    }
}

signed main() {
    ios::sync_with_stdio(0);
    getPrime();
    for (__int128 pid = 0; pid < pcnt; pid++) {
        h[pid].push_back(1);
        h[pid].push_back(0);
        __int128 invp = qpow(pri[pid], mod - 2);
        for (__int128 c = 2, pc = pri[pid] * pri[pid]; c < M && pc <= 1e1
2; c++, pc = pc * pri[pid]) {
            h[pid].push_back(f(pri[pid], c));//唯一 f 使用，传入参数类型自
定义
            __int128 pci = qpow(pri[pid], c);
            for (__int128 i = 1, pi = pri[pid]; i <= c; i++, pi = pi * pr
i[pid] % mod) {
                pci = pci * invp % mod;
                h[pid][c] = (h[pid][c] - g(pi) * h[pid][c - i] % mod + mo
d) % mod;
            }
        }
    }

    ll t;
    cin >> t;
    while (t--) {
        cin >> n;
        ans = G(n);
        dfs(0, 1, 1, false);
        cout << (ll) ans << endl;
    }
}
```

Pollard_Rho+Miller-Robin

```cpp
typedef long long ll;
```

```cpp
namespace Miller_Rabin {
    const ll Pcnt = 12;
    const ll p[Pcnt] = {2, 3, 5, 7, 11, 13, 17, 19, 61, 2333, 4567, 2425
1};

    ll pow(ll a, ll b, ll p) {
        ll ans = 1;
        for (; b; a = (__int128) a * a % p, b >>= 1)if (b & 1)ans = (__in
t128) ans * a % p;
        return ans;
    }

    bool check(ll x, ll p) {
        if (x % p == 0 || pow(p % x, x - 1, x) ^ 1)return true;
        ll t, k = x - 1;
        while ((k ^ 1) & 1) {
            t = pow(p % x, k >>= 1, x);
            if (t ^ 1 && t ^ x - 1)return true;
            if (!(t ^ x - 1))return false;
        }
        return false;
    }

    inline bool MR(ll x) {  //用这个
        if (x < 2)return false;
        for (int i = 0; i ^ Pcnt; ++i) {
            if (!(x ^ p[i]))return true;
            if (check(x, p[i]))return false;
        }
        return true;
    }
}
namespace Pollard_Rho {
#define Rand(x) (1ll*rand()*rand()%(x)+1)

    ll gcd(const ll a, const ll b) { return b ? gcd(b, a % b) : a; }

    ll mul(const ll x, const ll y, const ll X) {
        ll k = (1.0L * x * y) / (1.0L * X) - 1, t = (__int128) x * y - (_
_int128) k * X;
        while (t < 0)t += X;
        return t;
    }

    ll PR(const ll x, const ll y) {
        int t = 0, k = 1;
        ll v0 = Rand(x - 1), v = v0, d, s = 1;
        while (true) {
            v = (mul(v, v, x) + y) % x, s = mul(s, abs(v - v0), x);
            if (!(v ^ v0) || !s)return x;
            if (++t == k) {
                if ((d = gcd(s, x)) ^ 1)return d;
                v0 = v, k <<= 1;
            }
        }
    }

    void Resolve(ll x, ll &ans) {
        if (!(x ^ 1) || x <= ans)return;
        if (Miller_Rabin::MR(x)) {
            if (ans < x)ans = x;
            return;
        }
        ll y = x;
        while ((y = PR(x, Rand(x))) == x);
        while (!(x % y))x /= y;
        Resolve(x, ans);
        Resolve(y, ans);
    }

    long long check(ll x) { //用这个，素数返回本身
        ll ans = 0;
        Resolve(x, ans);
        return ans;
    }
}
```

prufer

Prufer 序列 (Prufer code)，这是一种将带标号的树用一个唯一的整数序列表示的方法。

Prufer 序列可以将一个带标号 n 个结点的树用$[1,n]$中的$n-2$个整数表示。你也可以把它理解为完全图的生成树与数列之间的双射。

显然你不会想不开拿这玩意儿去维护树结构。这玩意儿常用组合计数问题上。

线性建立 prufer

Prufer 是这样建立的：每次选择一个编号最小的叶结点并删掉它，然后在序列中记录下它连接到的那个结点。重复 n - 2 次后就只剩下两个结点，算法结束。

线性构造的本质就是维护一个指针指向我们将要删除的结点。首先发现，叶结点数是非严格单调递减的。要么删一个，要么删一个得一个。

于是我们考虑这样一个过程：维护一个指针 p 。初始时 p 指向编号最小的叶结点。同时我们维护每个结点的度数，方便我们知道在删除结点的时侯是否产生新的叶结点。操作如下：

1. 删除 指向的结点，并检查是否产生新的叶结点。
2. 如果产生新的叶结点，假设编号为 x ，我们比较 p, x 的大小关系。如果 x>p，那么不做其他操作；否则就立刻删除 x，然后检查删除 x 后是否产生新的叶结点，重复 2 步骤，直到未产生新节点或者新节点的编号>p 。
3. 让指针 p 自增直到遇到一个未被删除叶结点为止；

循环上述操作 n - 2 次，就完成了序列的构造。

```
// 从原文摘的代码，同样以 0 为起点
vector<vector<int>> adj;
vector<int> parent;

void dfs(int v) {
  for (int u : adj[v]) {
    if (u != parent[v]) parent[u] = v, dfs(u);
  }
}

vector<int> pruefer_code() {
  int n = adj.size();
  parent.resize(n), parent[n - 1] = -1;
  dfs(n - 1);

  int ptr = -1;
  vector<int> degree(n);
  for (int i = 0; i < n; i++) {
    degree[i] = adj[i].size();
    if (degree[i] == 1 && ptr == -1) ptr = i;
  }

  vector<int> code(n - 2);
  int leaf = ptr;
  for (int i = 0; i < n - 2; i++) {
    int next = parent[leaf];
    code[i] = next;
    if (--degree[next] == 1 && next < ptr) {
      leaf = next;
    } else {
      ptr++;
      while (degree[ptr] != 1) ptr++;
      leaf = ptr;
    }
  }
  return code;
}
```

性质

1. 在构造完 Prufer 序列后原树中会剩下两个结点，其中一个一定是编号最大的点 。
2. 每个结点在序列中出现的次数是其度数减 1 。（没有出现的就是叶结点）

线性 prufer 转化成树

同线性构造 Prufer 序列的方法。在删度数的时侯会产生新的叶结点，于是判断这个叶结点与指针 p 的大小关系，如果更小就优先考虑它

```
// 原文摘代码
vector<pair<int, int>> pruefer_decode(vector<int> const& code) {
  int n = code.size() + 2;
  vector<int> degree(n, 1);
  for (int i : code) degree[i]++;

  int ptr = 0;
  while (degree[ptr] != 1) ptr++;
  int leaf = ptr;

  vector<pair<int, int>> edges;
  for (int v : code) {
    edges.emplace_back(leaf, v);
    if (--degree[v] == 1 && v < ptr) {
      leaf = v;
    } else {
      ptr++;
      while (degree[ptr] != 1) ptr++;
      leaf = ptr;
    }
  }
```

```
  edges.emplace_back(leaf, n - 1);
  return edges;
}
```

cayley 公式

完全图K_n 有n^{n-2} 棵生成树。

用 Prufer 序列证:任意一个长度为 n - 2 的值域 [1, n] 的整数序列都可以通过 Prufer 序列双射对应一个生成树，于是方案数就是n^{n-2} 。

图连通方案数

一个 n 个点 m 条边的带标号无向图有 k 个连通块。我们希望添加 k - 1 条边使得整个图连通。求方案数。

设s_i表示每个连通块的数量。我们对 k 个连通块构造 Prufer 序列，然后你发现这并不是普通的 Prufer 序列。因为每个连通块的连接方法很多。不能直接淦就设啊。于是设d_i为第 i 个连通块的度数。由于度数之和是边数的两倍，于是$\sum_{i=1}^{k} d_i = 2k - 2$ 。则对于给定的 d 序列构造 Prufer 序列的方案数是

$$\tbinom{k - 2}{d_1 - 1, d_2 - 1, \dots, d_k - 1} = \frac{(k - 2)!}{(d_1 - 1)!(d_2 - 1)! \dots (d_k - 1)!}$$

对于第 i 个连通块，它的连接方式有$s_i^{d_i}$种，因此对于给定 d 序列使图连通的方案数是

$$\tbinom{k - 2}{d_1 - 1, d_2 - 1, \dots, d_k - 1} \prod_{i = 1}^{k}s_i^{d_i}$$

现在我们要枚举 d 序列，式子变成

$$\sum_{d_i\geq 1. \sum_{i = 1}^{k} d_i = 2k - 2} \tbinom{k - 2}{d_1 - 1, d2 - 1, \dots ,d_k - 1} \prod_{i = 1}^{k}s_i^{d_i}$$

根据多元二项式定理

$$(x_1+\dots+x_m)^{p}= \sum_{c_i \geq 0, \sum_{i = 1}^{m} c_i = p} \tbinom{p}{C_1, C_2, \dots , C_m} \prod_{i= 1}^{m}x_i^{C_i}$$

对原式换元，设$e_i = d_i - 1$ ，显然有$\sum_{i=1}^{k} e_i = k - 2$

$$\Rightarrow \sum_{e_i\ge 0, \sum_{i= 1}^{k} e_i = k - 2} \tbinom{k -2}{e_1, e_2, \dots, e_k} \prod_{i = 1}^{k}s_i ^{e _i+ 1} \\ 化简 \Rightarrow (s_1 + s_2 + \dots + s_k)^{k - 2} \prod_{i = 1}^{k}s_i \\ \Rightarrow n^{k - 2} \prod_{i = 1}^{k} s_i$$

中国剩余定理

```
#include<cstdio>

using namespace std;
typedef long long ll;

ll n;
ll a[100010], b[100010];

ll mul(ll A, ll B, ll mod) //快速乘取余 模板
{
    ll ans = 0;
    while (B > 0) {
        if (B & 1) ans = (ans + A % mod) % mod;
        A = (A + A) % mod;
        B >>= 1;
    }
    return ans;
}

ll exgcd(ll A, ll B, ll &x, ll &y) //扩展欧几里得 模板
{
    if (!B) {
        x = 1, y = 0;
        return A;
    }
    ll d = exgcd(B, A % B, x, y);
    ll tmp = x;
    x = y, y = tmp - A / B * y;
    return d;
}

ll lcm(ll A, ll B) //求最小公倍数
{
    ll xxx, yyy;
    ll g = exgcd(A, B, xxx, yyy);
    return (A / g * B);
}

ll excrt() //重点:求解同余方程组
{
    ll x, y;
    ll M = b[1], ans = a[1]; //赋初值
    //M 为前 k-1 个数的最小公倍数，ans 为前 k-1 个方程的通解
```

```
    for (int i = 2; i <= n; i++) {
        ll A = M, B = b[i];
        ll C = (a[i] - ans % B + B) % B; //代表同余方程 ax≡c(mod b) 中 a,b,
c

        ll g = exgcd(A, B, x, y);
        //求得 A,B 的最大公约数，与同余方程 ax≡gcd(a,b)(mod b)的解，

        if (C % g) return -1; //无解的情况

        x = mul(x, C / g, B); //求得 x 的值,x 即 t
        ans += x * M;  //获得前 k 个方程的通解
        M = lcm(M, B); //更改 M 的值
        ans = (ans % M + M) % M;
    }
    return ans;
}

int main() {
    scanf("%lld", &n);
    for (int i = 1; i <= n; i++)
        scanf("%lld%lld", &b[i], &a[i]);
    ll ans = excrt();
    printf("%lld", ans);
}
```

二次剩余

解的数量

对于$x^2 \equiv n(\bmod p)$ 能满足 n 是 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

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

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

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

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 和 都是模意义下的数，类似复数中的实部和虚部。

在有了 i 和 a 后可以直接得到答案，$x^2 \equiv n(\bmod p)$ 的解为$(a+i)^{\frac{p+1}{2}}$。

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
int t;
ll n, p;
ll w;

struct num {     //建立一个复数域

        ll x, y;
};

num mul(num a, num b, ll p) {     //复数乘法
        num ans = {0, 0};
        ans.x = ((a.x * b.x % p + a.y * b.y % p * w % p) % p + p) % p;
        ans.y = ((a.x * b.y % p + a.y * b.x % p) % p + p) % p;
        return ans;
}

ll binpow_real(ll a, ll b, ll p) {        //实部快速幂
        ll ans = 1;
        while (b) {
                if (b & 1) ans = ans * a % p;
                a = a * a % p;
```

```
            b >>= 1;
        }
        return ans % p;
}

ll binpow_imag(num a, ll b, ll p) {        //虚部快速幂
        num ans = {1, 0};
        while (b) {
                if (b & 1) ans = mul(ans, a, p);
                a = mul(a, a, p);
                b >>= 1;
        }
        return ans.x % p;
}

ll cipolla(ll n, ll p) {
        n %= p;
        if (p == 2) return n;
        if (binpow_real(n, (p - 1) / 2, p) == p - 1) return -1;
        ll a;
        while (1) {       //生成随机数再检验找到满足非二次剩余的a
                a = rand() % p;
                w = ((a * a % p - n) % p + p) % p;
                if (binpow_real(w, (p - 1) / 2, p) == p - 1) break;
        }
        num x = {a, 1};
        return binpow_imag(x, (p + 1) / 2, p);
}
```

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 ①

其中 m ≥3

1. 当 m 确定为任意一个 ≥3 的奇数时，k={1，m^2 的所有小于 m 的因子}
2. 当 m 确定为任意一个 ≥4 的偶数时，k={m^2 / 2 的所有小于 m 的偶数因子}

高斯整数/高斯素数

3B1B 的视频

洛谷某题

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

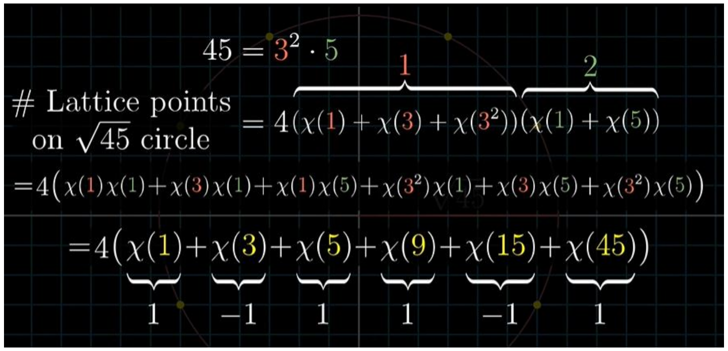
（乘以 1， -1， i，-i 四个

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，

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

最后转化为 45 的所有约数

$$f(x) = \begin{cases} 1 ,x 为素数 x = 4n+1 \\ -1, x 为素数 x = 4n+3 \\ 0, x 为偶数\\ \end{cases}\\ 半径为\sqrt { n}的圆上的格点数（二维坐标轴 xy 都为整数的点）是 4 \times \sum_{d|n}f(d)$$

博弈拾遗

SG 定理：

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

Nimk：

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

anti_nim

描述

和最普通的 Nim 游戏相同，不过是取走最后一个石子的人输。

先手必胜条件

以下两个条件满足其一即可：

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

卡特兰

‘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)~~

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

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）

卡特兰三角

卡特兰三角

卡特兰数：由 n 个 X 和 n 个 Y 组成的一个序列中，满足**所有前缀中 Y 出现的次数不超过 X 出现的次数**的序列的个数

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

卡特兰三角：由 n 个 X 和 k 个 Y 组成的一个序列，满足**所有前缀中 Y 出现的次数-X 出现的次数小于 m** 的序列的个数

$$C_m(n,k) = \begin{cases} \binom{n+k}{k}, 0 \le k < m \\ \binom{n+k}{k} - \binom{n+k}{k-m}, m \le k \le n+m-1 \\ 0, k > n+m-1 \end{cases}$$

卡特兰三角（OEIS）：$T(n,k) = T(n-1,k) + T(n,k-1)$

$$T(n+1,n+1) = \sum_{k=0}^{n} T(n,k)$$

n	Catalan triangle														Row sums $\sum_{k=0}^{n} T(n,k)$ = $C(n+1)$ Catalan numbers (A000108$(n+1)$)
0	1														1
1	1	1													2
2	1	2	2												5
3	1	3	5	5											14
4	1	4	9	14	14										42
5	1	5	14	28	42	42									132
6	1	6	20	48	90	132	132								429
7	1	7	27	75	165	297	429	429							1430
8	1	8	35	110	275	572	1001	1430	1430						4862
9	1	9	44	154	429	1001	2002	3432	4862	4862					16796
10	1	10	54	208	637	1638	3640	7072	11934	16796	16796				58786
11	1	11	65	273	910	2548	6188	13260	25194	41990	58786	58786			208012
12	1	12	77	350	A005587: $\frac{n^4+18n^3+107n^2+210n}{24}$										742900
13	1	13	90	A005586: Number of walks on square lattice $\frac{n^3+9n^2+20n}{6}$											2674440

原根

```
#include<bits/stdc++.h>

using namespace std;

ll qpow(ll n, ll m, ll p) {
    ll ans = 1;
    n %= p;
    for (; m; m >>= 1, n = n * n % p)
        if (m & 1)ans = ans * n % p;
    return ans;
}

ll pri[N], tot;

ll getRoot(ll p) //求质数p 的最小原根
{
    tot = 0;
    ll n = p - 1, sq = sqrt(p + 0.5);
    for (ll i = 2; i <= sq; i++)
```

```
        if (n % i == 0) {
            pri[tot++] = i;
            while (n % i == 0)n /= i;
        }
    if (n > 1)pri[tot++] = n;
    for (ll g = 2; g <= p - 1; g++) //试探每一个 g 是否原根
    {
        ll flag = 1;
        for (ll i = 0; i < tot; i++)
            if (qpow(g, (p - 1) / pri[i], p) == 1) {
                flag = 0;
                break;
            }
        if (flag)return g;
    }
    return -1; //没有原根
}
```

快速幂

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

扩展欧拉定理

用于在底数与模数不互质的情况下将质数降将至与模数同阶大小，从而使用快速幂

$$a^c = \begin{cases} a^{c \mod \phi(m)} , \gcd(a, m) = 1 \\ a^c, gcd(a, m)\not = 1 \and c < \phi(m) \\ a^{c \mod \phi(m) + \phi(m)}, gcd(a, m)\not = 1 \and c \ge \phi(m) \end{cases}$$

证明以及引理：

欧拉定理： $a^{\phi(m)} \equiv 1 (\mod m)$

证明欧拉：记 x_i 为第 i 个与 m 互质的数，则小于 m 的范围内共有$\phi(m)$ 个这样的数

$p_i = a \times x_i$

△ : {pi}*两两不同余且与m互质*，{xi}$ 两两不同余

所有p_i 的模 m 的集合与$\{x_i\}$ 相等 $\Rightarrow$ 他们的积模 m 相等

$$\Rightarrow \prod_{i=1}^{\phi m} p_i = a^{\phi(m)} \prod_{i=1}^{\phi(m)} x_i = \prod_{i=1}^{\phi(m)} x_i (\mod m)$$

扩展欧拉：

$$a^c = \begin{cases} a^{c \mod \phi(m)} , \gcd(a, m) = 1 \\ a^c, gcd(a, m)\not = 1 \and c < \phi(m) \\ a^{c \mod \phi(m) + \phi(m)}, gcd(a, m)\not = 1 \and c \ge \phi(m) \end{cases}$$

证明扩展欧拉(3)：

1. $\phi(p^r) = (p-1) \times p^r$, P 为质数
2. $\exist a, b, x, y , s.t. x^a \times y^b = k, 都有 a， b\le \phi(k) $
3. $\exist r \le c ,s.t. a^{\phi(m)+r} \equiv a ^r (\mod m)$

证明其中 3：$m = t \times a^r$，其中 $gcd(a, t) = 1$

又 ϕ 是一个积性函数，故 $\phi(t) \mid \phi(m)$

$$a^{\phi(t)} \equiv 1 (\mod t) \Rightarrow a^{\phi(m)} \equiv 1 (\mod t)$$

两边同乘以$a^r \Rightarrow a^{\phi(m)+r} \equiv a^r (\mod m)$

根据 2，$r \le \phi(m)$ 又 $c \ge \phi(m)$ ，得证

$$a^c \equiv a^{c-r+r} \equiv a^{c-r+\phi(m)+r} \equiv a^{c+\phi(m)} (\mod m)$$

扩欧求逆元

```cpp
#include <bits/stdc++.h>
using namespace std;

typedef  long long ll;

void extgcd(ll a,ll b,ll& d,ll& x,ll& y){
    if(!b){ d=a; x=1; y=0;}
    else{ extgcd(b,a%b,d,y,x); y-=x*(a/b); }
}

ll inverse(ll a,ll n){
    ll d,x,y;
    extgcd(a,n,d,x,y);
    return d==1?(x+n)%n:-1;
}

int main(){
        int x, y;
        //cin >> x >> y;
        while(1){
                cin >> x >> y;
                cout << inverse(x, y) << endl;
        }
        //cout << inverse(x, y) << endl;
}
```

数学知识

数学知识的一些范围（？

1 ~ n 的质数个数

$\frac{n}{l_n n}$

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

判断一个数是否为 11 的倍数

奇偶位置上的数位和的差是否为 11 的倍数

平方前缀和

$$\frac{n \times (n+1) \times (2 \times n + 1)}{6}$$

立方前缀和

$$\left(\frac{n \times (n+1)}{2}\right)^2$$

库默尔定理

设 m,n 为正整数，p 为素数，则C_{m+n}^{m} 含 p 的幂次等于 m+n 在 p 进制下的进位次数

原根存在定理

一个数 m 存在原根当且仅当$m = 2,4,p^{\alpha},2p^{\alpha}$，其中 p 为奇素数，$\alpha \in N^*$

整除分块（向上向下取整）

```cpp
int x;
scanf("%d",&x);
int ans1=0,ans2=0;
//向下取整
for(int l=1,r;l<=x;l=r+1){
    int m=x/l;
    r=x/m;
    ans1+=(r-l+1)*m;
}
//向上取整
int R=1e5;
for(int l=1,r;l<=R;l=r+1){
    int m=(x+l-1)/l;
    r=m!=1?(x-1)/(m-1):R;
    ans2+=(r-l+1)*m;
}
```

格雷码

```cpp
int gray_encode(int num) {
    return num ^ (num >> 1);
```

```
}

int gray_decode(int num) {
    int head;
    if (!num) return 0;
    head = 1 << int(log(num) / log(2));
    return head + gray_decode((num ^ head) ^ (head >> 1));
}
```

欧拉筛（素数）

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 1000005;
int phi[N], prime[N], cnt;
bool st[N];

void get_eulers() {
    phi[1] = 1;
    for (int i = 2; i < N; i++) {
        if (!st[i]) {
            prime[cnt++] = i;
            phi[i] = i - 1;
        }
        for (int j = 0; prime[j] * i < N; j++) {
            st[prime[j] * i] = 1;
            if (i % prime[j] == 0) {
                phi[prime[j] * i] = phi[i] * prime[j];
                break;
            }
            phi[prime[j] * i] = phi[i] * (prime[j] - 1);
        }
    }
}

int main() {
    get_eulers();
    ll n;
    cin >> n;
    ll ans = 0;
    for (int i = 1; i <= n; i++) ans += phi[i];
    cout << ans;
}
```

欧拉筛（莫比乌斯）

```
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;
const int N = 1e5 + 10;

bool vis[N];
ll prime[N], mu[N];

void init_mu() {
    ll cnt = 0;
    mu[1] = 1;
    for (ll i = 2; i < N; i++) {
        if (!vis[i]) {
            prime[cnt++] = i;
            mu[i] = -1;
        }
        for (ll j = 0; j < cnt && i * prime[j] < N; j++) {
            vis[i * prime[j]] = 1;
            if (i % prime[j] == 0) {
                mu[i * prime[j]] = 0;
                break;
            } else { mu[i * prime[j]] = -mu[i]; }
        }
    }
}

int main() {
    init_mu();
}
```

欧拉降幂

~~不知道它有什么用毕竟已经有快速幂了~~

这里有一张图可以很好的说明欧拉降幂是什么

$$a^b \equiv \begin{cases} a^{b\%\varphi(n)} (\bmod n) & n,a\text{互质} \\ a^b (\bmod n) & b < \varphi(n) \\ a^{b\%\varphi(n)+\varphi(n)} (\bmod n) & b \geq \varphi(n) \end{cases}$$

```
//其实只是想试一下 markdown 怎么用
//假装这里有代码
```

然后下面这个是用 $\LaTeX$公式写的

$$a^b \equiv \begin{cases} a^{b\%\varphi(n)} (\mathrm{mod} n) & n,a\text{互质} \\ a^b (\mathrm{mod} n) & b < \varphi(n) \\ a^{b\%\varphi(n)+\varphi(n)} (\mathrm{mod} n) & b \geq \varphi(n) \end{cases}$$

正多面体

Polyhedron, $a = 2$	Radius			Surface area, A	Volume	
	In-, r	Mid-, ρ	Circum-, R		V	Ur
tetrahedron	$\frac{1}{\sqrt{6}}$	$\frac{1}{\sqrt{2}}$	$\sqrt{\frac{3}{2}}$	$4\sqrt{3}$	$\frac{\sqrt{8}}{3} \approx 0.942809$	≈ (
cube	1	$\sqrt{2}$	$\sqrt{3}$	24	8	
octahedron	$\sqrt{\frac{2}{3}}$	1	$\sqrt{2}$	$8\sqrt{3}$	$\frac{\sqrt{128}}{3} \approx 3.771236$	≈ (
dodecahedron	$\frac{\varphi^2}{\xi}$	φ^2	$\sqrt{3}\,\varphi$	$12\sqrt{25+10\sqrt{5}}$	$\frac{20\varphi^3}{\xi^2} \approx 61.304952$	≈ [illegible]
icosahedron	$\frac{\varphi^2}{\sqrt{3}}$	φ	$\xi\varphi$	$20\sqrt{3}$	$\frac{20\varphi^2}{3} \approx 17.453560$	≈ [illegible]

The constants φ and ξ in the above are given by

$$\varphi = 2\cos\frac{\pi}{5} = \frac{1+\sqrt{5}}{2}, \qquad \xi = 2\sin\frac{\pi}{5} = \sqrt{\frac{5-\sqrt{5}}{2}} = \sqrt{3-\varphi}.$$

4 6 8 12 20

组合数

```
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;
const ll mod = 1e9 + 7;
const ll maxn = 3e4 + 5;
ll inv[maxn], fac[maxn];

ll qpow(ll a, ll b) {
    ll ans = 1;
    while (b) {
        if (b & 1) ans = (ans * a) % mod;
        a = (a * a) % mod;
```

```
        b >>= 1;
    }
    return ans;
}

ll c(ll n, ll m) {
    if (n < 0 || m < 0 || n < m) return 0;
    return fac[n] * inv[n - m] % mod * inv[m] % mod;
}

void init() {
    fac[0] = 1;
    for (int i = 1; i < maxn; i++) {
        fac[i] = fac[i - 1] * i % mod;
    }
    inv[maxn - 1] = qpow(fac[maxn - 1], mod - 2);
    for (ll i = maxn - 2; i >= 0; i--) {
        inv[i] = (inv[i + 1] * (i + 1)) % mod;
    }
}
```

莫比乌斯反演

莫比乌斯函数

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

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

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

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

构造[MathProcessingError]$F(n)和f(n)$ 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

$$F(n) = gcd(x,y) = n的倍数的xy的对数$$

$$f(n) = gcd(x,y) = n的xy的对数$$

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

const int N = 50010;

ll primes[N], mu[N], sum[N], cnt;
bool st[N];

void init() {
        mu[1] = 1;

        for(int i = 2; i < N; ++ i) {
                if(!st[i]) {
                        primes[cnt ++] = i;
                        mu[i] = -1;
                }

                for(int j = 0; primes[j] * i < N; ++ j) {
                        st[primes[j] * i] = 1;
                        if(i % primes[j] == 0) break;
                        mu[primes[j] * i] = -mu[i];
                }
        }

        for(int i = 1; i < N; ++ i) {
                sum[i] = sum[i - 1] + mu[i];
```

```
        }
}

ll g(ll n, ll x) {
        return n / (n / x);
}

ll f (int a, int b, int k) {
        a = a / k, b = b / k;

        ll res = 0;

        ll n = min(a, b);

        for(ll l = 1, r; l <= n; l = r + 1) {
                r = min(n, min(g(a, l), g(b, l)));
                res += (sum[r] - sum[l - 1]) * (a / l) * (b / l);
        }

        return res;
}

int main() {
        ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);

        init();

        int T;
        cin >> T;
        while(T --) {
                int a, b, c, d, k;
                cin >> a >> b >> c >> d >> k;
                cout << f(b, d, k) - f(a - 1, d, k) - f(b, c - 1, k)
                                + f(a - 1, c - 1, k) << endl;
        }

        return 0;
}
```

2

求$\sum_{i=1}^{N}\sum_{j=1}^{M} d(ij)$

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

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

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

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

两次整数分块

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
const int N = 50010;

int primes[N], cnt, mu[N], sum[N], h[N];
bool st[N];

inline int g(int n, int x) {
        return n / (n / x);
}

void init() {
        mu[1] = 1;
        for(int i = 2; i < N; ++i) {
                if(!st[i]){
                        primes[cnt++] = i;
                        mu[i] = -1;
                }
                for(int j = 0; primes[j] * i < N; ++j) {
                        st[primes[j] * i] = 1;
                        if(i % primes[j] == 0) break;
                        mu[primes[j] * i] = -mu[i];
                }


        }

        for(int i = 1; i < N; ++ i) {
```

```
            sum[i] = sum[i - 1] + mu[i];
        }

        for(int i = 1; i < N; ++i) {
                for(int l = 1, r; l <= i; l = r + 1) {
                        r = min(i, g(i, l));
                        h[i] += (r - l + 1) * (i / l);
                }
        }
}

int main() {
        //ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
        init();

        int T;
        scanf("%d", &T);
        while(T--) {
                int n, m;
                scanf("%d %d", &n, &m);
                ll res = 0;
                int k = min(n, m);
                for(int l = 1, r; l <= k; l = r + 1) {
                        r = min(k, min(g(n, l), g(m, l)));
                        res += (ll)(sum[r] - sum[l - 1]) * h[n / l] * h
[m / l];
                }
            printf("%lld\n", res);
        }

        return 0;
}
```

逆元线性递推 **inv** 阶乘逆元组合数

```
ll fac[N];// n!
ll invfac[N]; // n!的inv
ll invn[N]; //n 的inv

inline void init() {
    fac[0] = fac[1] = invfac[0] = invfac[1] = invn[0] = invn[1] = 1;
    for (int i = 2; i < N; ++i) {
        fac[i] = fac[i - 1] * i % mod;
        invn[i] = (mod - mod / i) * invn[mod % i] % mod;
        invfac[i] = invfac[i - 1] * invn[i] % mod;
    }
}

ll C(ll up, ll down) {
    if (up > down) return 0;
    if (up < 0 || down < 0) return 0;
    ll res = fac[down];
    res = res * invfac[down - up] % mod;
    res = res * invfac[up] % mod;
    return res;
}

//先init
```

杂项

fread 快读

```
#include <bits/stdc++.h>
using namespace std;

char next_char() {
        static char buf[1 << 20], *first, *last;
        if(first == last) {
                last = buf + fread(buf, 1, 1 << 20, stdin);
                first = buf;
        }
        return first == last ? EOF : *first ++;
}

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

int main(){
        freopen("1.txt", "r", stdin); // �������ʱ��h��Çz��aa
a
        int T;
        cin >> T;
```

```
	while(T --){
		int x = read();
		cout << x << endl;
	}
}
```

int128 输出

```
inline void print(__int128 x) {
    if (x < 0) {
        putchar('-');
        x = -x;
    }
    if (x > 9)
        print(x / 10);
    putchar(x % 10 + '0');
}
```

mt19937

mt19937

```
#include <random>
#include <iostream>

int main()
{
    std::random_device rd;  //获取随机数种子
    std::mt19937 gen(rd()); //Standard mersenne_twister_engine seeded wi
th rd()
    std::uniform_int_distribution<> dis(0, 9);

    for (int n = 0; n<20; ++n)
        std::cout << dis(gen) << ' ';
    std::cout << '\n';
    system("pause");
    return 0;
}

//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
```

doule ： std::uniform*real*distribution<> dis(0, 9);

```
#include <iostream>
#include <chrono>
#include <random>
using namespace std;
int main()
{
	// 随机数种子
	unsigned seed = std::chrono::system_clock::now().time_since_epo
ch().count();
	mt19937 rand_num(seed);  // 大随机数
	uniform_int_distribution<long long> dist(0, 1000000000);  // 给
定范围
	cout << dist(rand_num) << endl;
	return 0;
}
```

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

```
#include <random>
#include <algorithm>
#include <iterator>
#include <iostream>

int main()
{
    std::vector<int> v = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

    std::random_device rd;
    std::mt19937 g(rd());

    std::shuffle(v.begin(), v.end(), g);

    std::copy(v.begin(), v.end(), std::ostream_iterator<int>(std::cout,
 " "));
    std::cout << "\n";

    system("pause");
    return 0;
}
```

大质数表

https://www.cnblogs.com/ljxtt/p/13514346.html

1e17	1e18
100000000000000003	1000000000000000003
100000000000000013	1000000000000000009
100000000000000019	1000000000000000031
100000000000000021	1000000000000000079
100000000000000049	1000000000000000177
100000000000000081	1000000000000000183
100000000000000099	1000000000000000201
100000000000000141	1000000000000000283
100000000000000181	1000000000000000381
100000000000000337	1000000000000000387
100000000000000339	1000000000000000507
100000000000000369	1000000000000000523
100000000000000379	1000000000000000583
100000000000000423	1000000000000000603
100000000000000519	1000000000000000619
100000000000000543	1000000000000000621
100000000000000589	1000000000000000799
100000000000000591	1000000000000000841
100000000000000609	1000000000000000861
100000000000000669	1000000000000000877
100000000000000691	1000000000000000913
100000000000000781	1000000000000000931
100000000000000787	1000000000000000997

快读 read

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

整体二分

```cpp
ll bit[N];

void add_bit(ll k, ll a) {
    while (k < N) {
        bit[k] = bit[k] + a;
        k += k & -k;
    }
}

ll query_bit(ll k) {
    ll ans = 0;
    while (k) {
        ans = ans + bit[k];
        k -= k & -k;
    }
    return ans;
}

struct node {
    ll x, y, k, id, type;
};
node q[N], q1[N], q2[N];
ll ans[N], now[N], tot, totx;

void solve(ll l, ll r, ll ql, ll qr) {
    if (ql > qr) return;
    if (l == r) {
        for (ll i = ql; i <= qr; i++) {
            if (q[i].type == 2) {
                ans[q[i].id] = l;
            }
        }
        return;
    }
    ll mid = (l + r) >> 1;
    ll cq1 = 0, cq2 = 0;
    for (ll i = ql; i <= qr; i++) {
        if (q[i].type == 1) {
            if (q[i].y <= mid) {
                add_bit(q[i].x, q[i].k);
                q1[++cq1] = q[i];
            } else {
                q2[++cq2] = q[i];
            }
```

```
        } else {
            ll sum = query_bit(q[i].y) - query_bit(q[i].x - 1);
            if (sum >= q[i].k) {
                q1[++cq1] = q[i];
            } else {
                q2[++cq2] = q[i];
                q2[cq2].k -= sum;
            }
        }
    }
    for (ll i = 1; i <= cq1; i++) if (q1[i].type == 1) add_bit(q1[i].x,
-q1[i].k);
    for (ll i = 1; i <= cq1; i++) q[ql + i - 1] = q1[i];
    for (ll i = 1; i <= cq2; i++) q[ql + cq1 + i - 1] = q2[i];
    solve(l, mid, ql, ql + cq1 - 1);
    solve(mid + 1, r, ql + cq1, qr);

}

void init() {
    totx = 0;
    tot = 0;
    memset(bit, 0, sizeof bit);
}
```

朝鲜大哥快读

```
#define FI(n) FastIO::read(n)
#define FO(n) FastIO::write(n)
#define Flush FastIO::Fflush()
//程序末尾写上  Flush;

namespace FastIO {
    const int SIZE = 1 << 16;
    char buf[SIZE], obuf[SIZE], str[60];
    int bi = SIZE, bn = SIZE, opt;
    double D[] = {0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001,
 0.00000001, 0.000000001, 0.0000000001};

    int read(char *s) {
        while (bn) {
            for (; bi < bn && buf[bi] <= ' '; bi++);
            if (bi < bn)
                break;
            bn = fread(buf, 1, SIZE, stdin);
            bi = 0;
        }
        int sn = 0;
        while (bn) {
            for (; bi < bn && buf[bi] > ' '; bi++)
                s[sn++] = buf[bi];
            if (bi < bn)
                break;
            bn = fread(buf, 1, SIZE, stdin);
            bi = 0;
        }
        s[sn] = 0;
        return sn;
    }

    bool read(int &x) {
        int n = read(str), bf = 0;
        if (!n)
            return 0;
        int i = 0;
        if (str[i] == '-')
            bf = 1, i++;
        else if (str[i] == '+')
            i++;
        for (x = 0; i < n; i++)
            x = x * 10 + str[i] - '0';
        if (bf)
            x = -x;
        return 1;
    }

    bool read(long long &x) {
        int n = read(str), bf;
        if (!n)
            return 0;
        int i = 0;
        if (str[i] == '-')
            bf = -1, i++;
        else
            bf = 1;
        for (x = 0; i < n; i++)
            x = x * 10 + str[i] - '0';
        if (bf < 0)
            x = -x;
        return 1;
```

```
}

void write(int x) {
    if (x == 0)
        obuf[opt++] = '0';
    else {
        if (x < 0)
            obuf[opt++] = '-', x = -x;
        int sn = 0;
        while (x)
            str[sn++] = x % 10 + '0', x /= 10;
        for (int i = sn - 1; i >= 0; i--)
            obuf[opt++] = str[i];
    }
    if (opt >= (SIZE >> 1)) {
        fwrite(obuf, 1, opt, stdout);
        opt = 0;
    }
}

void write(long long x) {
    if (x == 0)
        obuf[opt++] = '0';
    else {
        if (x < 0)
            obuf[opt++] = '-', x = -x;
        int sn = 0;
        while (x)
            str[sn++] = x % 10 + '0', x /= 10;
        for (int i = sn - 1; i >= 0; i--)
            obuf[opt++] = str[i];
    }
    if (opt >= (SIZE >> 1)) {
        fwrite(obuf, 1, opt, stdout);
        opt = 0;
    }
}

void write(unsigned long long x) {
    if (x == 0)
        obuf[opt++] = '0';
    else {
        int sn = 0;
        while (x)
            str[sn++] = x % 10 + '0', x /= 10;
        for (int i = sn - 1; i >= 0; i--)
            obuf[opt++] = str[i];
    }
    if (opt >= (SIZE >> 1)) {
        fwrite(obuf, 1, opt, stdout);
        opt = 0;
    }
}

void write(char x) {
    obuf[opt++] = x;
    if (opt >= (SIZE >> 1)) {
        fwrite(obuf, 1, opt, stdout);
        opt = 0;
    }
}

void Fflush() {
    if (opt)
        fwrite(obuf, 1, opt, stdout);
    opt = 0;
}
}; // namespace FastIO
```

枚举子集

```
cin >> n;
for (int s = n; s; s = (s - 1) & n) {
    cout << bitset<8>(s) << endl;
}
```

模拟退火

“优化的随机算法”

连续函数找区间最优

// 找一个点，与平面中的 n 个点的距离和最近

//进行多次模拟退火避免局部最大值

```
#include <bits/stdc++.h>
#include <ctime>
using namespace std;
```

```
const int maxn = 110;

int n;

#define x first
#define y second

typedef pair<double, double> PDD;

PDD q[maxn];
double ans = 1e8;

double rand(double l, double r) {
    return (double) rand() / RAND_MAX * (r - l) + l;
}

double getDist(PDD a, PDD b) {
    double dx = a.x - b.x;
    double dy = a.y - b.y;
    return sqrt(dx * dx + dy * dy) ;
}

double calc(PDD p) {
    double res = 0;
    for(int i = 0; i < n; ++ i) {
        res += getDist(q[i], p);
    }
    ans = min(ans, res);
    return res;
}

double simulate_anneal() {
    PDD cur(rand(0, 10000), rand(0, 10000)); // 随机一个起点
    for(double T = 1e4; T > 1e-4; T = T * 0.99) { // 初始温度，末态温度，
衰减系数，一般调整衰减系数0.999 0.95
        PDD np(rand(cur.x - T, cur.x + T), rand(cur.y - T, cur.y + T)); /
/ 随机新点
        double delta = calc(np) - calc(cur);
        if(exp(-delta / T) > rand(0, 1)) cur = np; //如果新点比现在的点更
优，必过去，不然有一定概率过去
    }

}
```

```
int main() {
    cin >> n;
    for(int i = 0; i < n; ++ i) {
        cin >> q[i].x >> q[i].y;
    }

    while((double) clock() / CLOCKS_PER_SEC < 0.8) { // 卡时 // 或for（1
00）
        simulate_anneal();
    }

    cout << (int)(ans + 0.5) << endl;

    return 0;
}
```

//n 个点带权费马点 // 平衡点||吊打 XXX

//n 个二维坐标点，带重物重量，找平衡点

//进行一次模拟退火，但是在局部最大值周围多次跳动（以提高精度

```
#include <cmath>
#include <cstdio>
#include <cstdlib>
#include <ctime>

const int N = 10005;
int n, x[N], y[N], w[N];
double ansx, ansy, dis;

double Rand() { return (double)rand() / RAND_MAX; }
double calc(double xx, double yy) {
  double res = 0;
  for (int i = 1; i <= n; ++i) {
    double dx = x[i] - xx, dy = y[i] - yy;
    res += sqrt(dx * dx + dy * dy) * w[i];
  }
  if (res < dis) dis = res, ansx = xx, ansy = yy;
  return res;
}
void simulateAnneal() {
  double t = 100000;
```

```
  double nowx = ansx, nowy = ansy;
  while (t > 0.001) {
    double nxtx = nowx + t * (Rand() * 2 - 1);
    double nxty = nowy + t * (Rand() * 2 - 1);
    double delta = calc(nxtx, nxty) - calc(nowx, nowy);
    if (exp(-delta / t) > Rand()) nowx = nxtx, nowy = nxty;
    t *= 0.97;
  }
  for (int i = 1; i <= 1000; ++i) {
    double nxtx = ansx + t * (Rand() * 2 - 1);
    double nxty = ansy + t * (Rand() * 2 - 1);
    calc(nxtx, nxty);
  }
}
int main() {
  srand(time(0));
  scanf("%d", &n);
  for (int i = 1; i <= n; ++i) {
    scanf("%d%d%d", &x[i], &y[i], &w[i]);
    ansx += x[i], ansy += y[i];
  }
  ansx /= n, ansy /= n, dis = calc(ansx, ansy);
  simulateAnneal();
  printf("%.3lf %.3lf\n", ansx, ansy);
  return 0;
}
```

测试时常用的代码

```
//�Ļ�����������ʱ�ύ��������������

#ifdef ONLINE_JUDGE
#else
    freopen("in.txt","r",stdin);
    //freopen("out.txt","w",stdout);
#endif
//����Ŀ���stdin/stdout��in.txt/out.txt��"r"��"w"Ï����ð


#include<ctime>
    clock_t ST,ED;
    ST=clock();
    //��������ʱ���
    ED=clock();
    cout<<ED-ST<<"ms"<<endl;


#include<ctime>
#include<cstdlib>
    srand(time(0));//��’��
    rand();//����[0,RAND_MAX]��������������(int)��RAND_MAX��
cstdlib�eĺ�Ɖh��Ï0x7fff(32767)
```

线性代数

矩阵类模板_加减乘快速幂

```
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;
const ll N = 305;
const ll mod = 998244353;

//矩阵类模板
struct Matrix {
    ll n, m;
    ll a[N][N];

    void set(ll _a, ll _b) {
        n = _a, m = _b;
    }

    Matrix() {
        clear();
    }

    void clear() {
        n = m = 0;
        memset(a, 0, sizeof(a));
    }

    Matrix operator+(const Matrix &b) const {
        Matrix tmp;
        tmp.n = n;
        tmp.m = m;
        for (ll i = 0; i < n; ++i)
```

```cpp
        for (ll j = 0; j < m; ++j)
            tmp.a[i][j] = (a[i][j] + b.a[i][j]) % mod;
    return tmp;
}

Matrix operator-(const Matrix &b) const {
    Matrix tmp;
    tmp.n = n;
    tmp.m = m;
    for (ll i = 0; i < n; ++i) {
        for (ll j = 0; j < m; ++j)
            tmp.a[i][j] = (a[i][j] - b.a[i][j] + mod) % mod;
    }

    return tmp;
}

Matrix operator*(const Matrix &b) const {
    Matrix tmp;
    tmp.clear();
    tmp.n = n;
    tmp.m = b.m;
    for (ll i = 0; i < n; ++i)
        for (ll j = 0; j < b.m; ++j)
            for (ll k = 0; k < m; ++k) {
                tmp.a[i][j] += a[i][k] * b.a[k][j];
                tmp.a[i][j] %= mod;
            }
    return tmp;
}

Matrix get(ll x) {//幂运算
    Matrix E;
    E.clear();
    E.set(n, m);
    for (ll i = 0; i < n; ++i)
        E.a[i][i] = 1;
    if (x == 0) return E;
    else if (x == 1) return *this;
    Matrix tmp = get(x / 2);
    tmp = tmp * tmp;
    if (x % 2) tmp = tmp * (*this);
    return tmp;
}

void exgcd(ll _a, ll _b, ll &x, ll &y) {
    if (!_b)return x = 1, y = 0, void();
    exgcd(_b, _a % _b, y, x);
    y -= x * (_a / _b);
}

ll inv(ll p) {
    ll x, y;
    exgcd(p, mod, x, y);
    return (x + mod) % mod;
}

Matrix inv() {
    Matrix E = *this;
    ll is[N], js[N];
    for (ll k = 0; k < E.n; k++) {
        is[k] = js[k] = -1;
        for (ll i = k; i < E.n; i++) // 1
            for (ll j = k; j < E.n; j++)
                if (E.a[i][j]) {
                    is[k] = i, js[k] = j;
                    break;
                }
        if (is[k] == -1) {
            E.clear();
            return E;
        }
        for (ll i = 0; i < E.n; i++) // 2
            swap(E.a[k][i], E.a[is[k]][i]);
        for (ll i = 0; i < E.n; i++)
            swap(E.a[i][k], E.a[i][js[k]]);
        if (!E.a[k][k]) {
            E.clear();
            return E;
        }
        E.a[k][k] = inv(E.a[k][k]); // 3
        for (ll j = 0; j < E.n; j++)
            if (j != k) // 4
                (E.a[k][j] *= E.a[k][k]) %= mod;
        for (ll i = 0; i < E.n; i++)
            if (i != k) // 5
                for (ll j = 0; j < E.n; j++)
                    if (j != k)
                        (E.a[i][j] += mod - E.a[i][k] * E.a[k][j] % mo
d) %= mod;
        for (ll i = 0; i < E.n; i++)
            if (i != k) // 就是这里不同
```

```
                E.a[i][k] = (mod - E.a[i][k] * E.a[k][k] % mod) % mod;
        }
        for (ll k = E.n - 1; k >= 0; k--) { // 6
            for (ll i = 0; i < E.n; i++)
                swap(E.a[js[k]][i], E.a[k][i]);
            for (ll i = 0; i < E.n; i++)
                swap(E.a[i][is[k]], E.a[i][k]);
        }
        return E;
    }
};
//矩阵模板结束
```

矩阵类模板_稀疏矩阵乘法

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

矩阵行列式

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll mod = 1e9 + 7;
struct Matrix {
    static const ll MAXN = 300;
    ll a[MAXN][MAXN];

    void init() { memset(a, 0, sizeof(a)); }

    ll det(ll n) {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++) a[i][j] = (a[i][j] + mod) % mod;
        ll res = 1;
        for (int i = 0; i < n; i++) {
            if (!a[i][i]) {
                bool flag = false;
                for (int j = i + 1; j < n; j++) {
                    if (a[j][i]) {
                        flag = true;
                        for (int k = i; k < n; k++) {
                            swap(a[i][k], a[j][k]);
                        }
                        res = -res;
                        break;
                    }
                }
                if (!flag) return 0;
            }

            for (int j = i + 1; j < n; j++) {
                while (a[j][i]) {
                    ll t = a[i][i] / a[j][i];
                    for (int k = i; k < n; k++) {
                        a[i][k] = (a[i][k] - t * a[j][k]) % mod;
                        swap(a[i][k], a[j][k]);
                    }
                    res = -res;
                }
            }
            res *= a[i][i];
            res %= mod;
        }
        return (res + mod) % mod;
```

```
    }
} mat;
```

线性基 2

线性基 能表示的线性空间与原向量 能表示的线性空间等价

用高斯消元得到线性基

先输入数组 a[] 中

```
int n, k;
ll a[N];

void getVec() {
    k = 0;

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

}
```

这里注意最后的线性基是 a[]中从 0 到 k-1 个，在前的是**高位**

线性基模板

```
//
const int maxbit = 62;          //maxbit���_��

struct L_B{
        ll lba[maxbit];
        L_B(){
       memset(lba, 0, sizeof(lba));
    }

        void Insert(ll val){            //����
       for(int i = maxbit - 1; i >= 0; -- i) // �ӸߵͰλ����λ
           if(val & (1ll << i)){ //
               if(!lba[i]){
                   lba[i] = val;
                   break;
               }
               val ^= lba[i];
           }
    }
};
//��ȡ���Є�ÿ����valȡÏ2���ϵ��ߵ�Ы�λ���λŧ������Ж�ǰλÏ1�ǵ
���lba[i]�����ǁ���lba[i]=x���������val=val`xor`lba[i]
//'�ã� „��insert
// --------------���ի�ġ��
```

高斯消元

```
#include <iostream>
#include <vector>
using namespace std;
const double eps = 1e-8;
void sway(vector<double>& a, vector<double>& b) {
    vector<double> s;
    for (int i = 0; i < a.size(); i++) {
        s.push_back(a[i]);
    }
    a.clear();
    for (int i = 0; i < b.size(); i++) {
        a.push_back(b[i]);
    }
    b.clear();
    for (int i = 0; i < s.size(); i++) {
        b.push_back(s[i]);
    }
}
```

```cpp
vector<double> gauss_jordan(const vector<vector<double> >& A,
                            const vector<double>& b) {
    int n = A.size();
    vector<vector<double> > B(n, vector<double>(n + 1));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++) B[i][j] = A[i][j];
    for (int i = 0; i < n; i++) B[i][n] = b[i];

    for (int i = 0; i < n; i++) {
        int pivot = i;
        for (int j = i; j < n; j++) {
            if (abs(B[j][i]) > abs(B[pivot][i])) pivot = j;
        }
        swap(B[i], B[pivot]);
        if (abs(B[i][i]) < eps) return vector<double>();
        for (int j = i + 1; j <= n; j++) B[i][j] /= B[i][i];
        for (int j = 0; j < n; j++) {
            if (i != j) {
                for (int k = i + 1; k <= n; k++) B[j][k] -= B[j][i] * B[i]
[k];
            }
        }
    }
    vector<double> x(n);
    for (int i = 0; i < n; i++) x[i] = B[i][n];
    return x;
}
int main() {
    int n, m;
    cin >> n >> m;
    vector<vector<double> > mat(n, vector<double>(m));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cin >> mat[i][j];
        }
    }
    vector<double> val(n);
    for (int i = 0; i < n; i++) cin >> val[i];
    vector<double> ans = gauss_jordan(mat, val);
    for (int i = 0; i < ans.size(); i++) cout << ans[i] << ' ';
}
```

组合数学

斯特林数

百度百科讲的超好

第一类斯特林数（无符号第一类）

定义：$\left[{n \atop k}\right]$ 表示将 n 个两两不同的元素，划分为 k 个非空圆排列的方案数。

递推式 $\left[{k \atop n}\right] = \left[{n-1 \atop k-1}\right] + (n-1)\left[{n-1 \atop k}\right]$

升阶函数

$$x^{n\uparrow} = x(x+1)(x+2)\cdots(x+n-1) = \sum_{k=0}^{n} s_u(n,k)\,x^k$$

（每一项系数则为无符号第一类斯特林数，求前 n 项和则为取 x=1）

性质

无符号Stirling数有如下性质：

① $s_u(0,0)=1$

② $s_u(n,0)=0$

③ $s_u(n,n)=1$

④ $s_u(n,1)=(n-1)!$

⑤ $s_u(n,n-1)=C(n,2)$

⑥ $s_u(n,2)=(n-1)!\cdot\sum_{i=1}^{n-1}\frac{1}{i}$

⑦ $s_u(n,n-2)=2\cdot C(n,3)+3\cdot C(n,4)$

⑧ $\sum_{k=0}^{n}s_u(n,k)=n!$ 证明可令升阶函数中的x=1，比较两边系数。

第二类斯特林数

定义：$\left\{ n \atop k \right\}$ 表示将 n 个两两不同的元素，划分为 k 个非空子集的方案数。

递推式 $\left\{ n \atop k \right\}=\left\{ n-1 \atop k-1 \right\}+k\left\{ n-1 \atop k \right\}$

性质

① $S(n,0)=0^n$

② $S(n,1)=1$

③ $S(n,n)=1$

④ $S(n,2)=2^{n-1}-1$

⑤ $S(n,n-1)=C(n,2)$

⑥ $S(n,n-2)=C(n,3)+3\cdot C(n,4)$

⑦ $S(n,3)=\frac{1}{2}(3^{n-1}+1)-2^{n-1}$

⑧ $S(n,n-3)=C(n,4)+10\cdot C(n,5)+15\cdot C(n,6)$

⑨ $\sum_{k=0}^{n}S(n,k)=B_n$ ， B_n 是贝尔数。

通项公式：

$$S(n,m)=\frac{1}{m!}\sum_{k=0}^{m}(-1)^kC(m,k)(m-k)^n$$

两类Stirling数之间的关系

两类Stirling数之间的递推式和实际含义很类似，事实上他们之间存在一个互为转置的转化关系：

$$\sum_{k=0}^{n} S(n,k)s(k,m) = \sum_{k=0}^{n} s(n,k)S(k,m)$$

计算几何

jls

```cpp
#define mp make_pair
#define fi first
#define se second
#define pb push_back
typedef double db;
const db eps=1e-6;
const db pi=acos(-1);
int sign(db k){
    if (k>eps) return 1; else if (k<-eps) return -1; return 0;
}
int cmp(db k1,db k2){return sign(k1-k2);}
int inmid(db k1,db k2,db k3){return sign(k1-k3)*sign(k2-k3)<=0;}// k3 在 [k1,k2] 内
struct point{
    db x,y;
    point operator + (const point &k1) const{return (point){k1.x+x,k1.y+y};}
    point operator - (const point &k1) const{return (point){x-k1.x,y-k1.y};}
    point operator * (db k1) const{return (point){x*k1,y*k1};}
    point operator / (db k1) const{return (point){x/k1,y/k1};}
    int operator == (const point &k1) const{return cmp(x,k1.x)==0&&cmp(y,k1.y)==0;}
    // 逆时针旋转
    point turn(db k1){return (point){x*cos(k1)-y*sin(k1),x*sin(k1)+y*cos(k1)};}
    point turn90(){return (point){-y,x};}
    bool operator < (const point k1) const{
        int a=cmp(x,k1.x);
        if (a==-1) return 1; else if (a==1) return 0; else return cmp(y,k1.y)==-1;
    }
    db abs(){return sqrt(x*x+y*y);}
    db abs2(){return x*x+y*y;}
    db dis(point k1){return ((*this)-k1).abs();}
    point unit(){db w=abs(); return (point){x/w,y/w};}
    void scan(){double k1,k2; scanf("%lf%lf",&k1,&k2); x=k1; y=k2;}
    void print(){printf("%.11lf %.11lf\n",x,y);}
    db getw(){return atan2(y,x);}
    point getdel(){if (sign(x)==-1||(sign(x)==0&&sign(y)==-1)) return (*this)*(-1); else return (*this);}
        int getP() const{return sign(y)==1||(sign(y)==0&&sign(x)==-1);}
};
int inmid(point k1,point k2,point k3){return inmid(k1.x,k2.x,k3.x)&&inmid(k1.y,k2.y,k3.y);}
db cross(point k1,point k2){return k1.x*k2.y-k1.y*k2.x;}
db dot(point k1,point k2){return k1.x*k2.x+k1.y*k2.y;}
db rad(point k1,point k2){return atan2(cross(k1,k2),dot(k1,k2));}
// -pi -> pi
int compareangle (point k1,point k2){
    return k1.getP()<k2.getP()||(k1.getP()==k2.getP()&&sign(cross(k1,k2))>0);
}
point proj(point k1,point k2,point q){ // q 到直线 k1,k2 的投影
    point k=k2-k1; return k1+k*(dot(q-k1,k)/k.abs2());
}
point reflect(point k1,point k2,point q){return proj(k1,k2,q)*2-q;}
int clockwise(point k1,point k2,point k3){// k1 k2 k3 逆时针 1 顺时针 -1 否则 0
    return sign(cross(k2-k1,k3-k1));
}
int checkLL(point k1,point k2,point k3,point k4){// 求直线 (L) 线段 (S)k1,k2 和 k3,k4 的交点
    return cmp(cross(k3-k1,k4-k1),cross(k3-k2,k4-k2))!=0;
}
point getLL(point k1,point k2,point k3,point k4){
    db w1=cross(k1-k3,k4-k3),w2=cross(k4-k3,k2-k3); return (k1*w2+k2*w1)/(w1+w2);
}
int intersect(db l1,db r1,db l2,db r2){
    if (l1>r1) swap(l1,r1); if (l2>r2) swap(l2,r2); return cmp(r1,l2)!=-1&&cmp(r2,l1)!=-1;
}
int checkSS(point k1,point k2,point k3,point k4){
```

```
    return intersect(k1.x,k2.x,k3.x,k4.x)&&intersect(k1.y,k2.y,k3.y,k4.
y)&&
    sign(cross(k3-k1,k4-k1))*sign(cross(k3-k2,k4-k2))<=0&&
    sign(cross(k1-k3,k2-k3))*sign(cross(k1-k4,k2-k4))<=0;
}
db disSP(point k1,point k2,point q){
    point k3=proj(k1,k2,q);
    if (inmid(k1,k2,k3)) return q.dis(k3); else return min(q.dis(k1),q.d
is(k2));
}
db disSS(point k1,point k2,point k3,point k4){
    if (checkSS(k1,k2,k3,k4)) return 0;
    else return min(min(disSP(k1,k2,k3),disSP(k1,k2,k4)),min(disSP(k3,k
4,k1),disSP(k3,k4,k2)));
}
int onS(point k1,point k2,point q){return inmid(k1,k2,q)&&sign(cross(k1
-q,k2-k1))==0;}
struct circle{
    point o; db r;
    void scan(){o.scan(); scanf("%lf",&r);}
    int inside(point k){return cmp(r,o.dis(k));}
};
struct line{
    // p[0]->p[1]
    point p[2];
    line(point k1,point k2){p[0]=k1; p[1]=k2;}
    point& operator [] (int k){return p[k];}
    int include(point k){return sign(cross(p[1]-p[0],k-p[0]))>0;}
    point dir(){return p[1]-p[0];}
    line push(){ // 向外 ( 左手边 ) 平移 eps
        const db eps = 1e-6;
        point delta=(p[1]-p[0]).turn90().unit()*eps;
        return {p[0]-delta,p[1]-delta};
    }
};
point getLL(line k1,line k2){return getLL(k1[0],k1[1],k2[0],k2[1]);}
int parallel(line k1,line k2){return sign(cross(k1.dir(),k2.dir()))==0;}
int sameDir(line k1,line k2){return parallel(k1,k2)&&sign(dot(k1.dir(),
k2.dir()))==1;}
int operator < (line k1,line k2){
    if (sameDir(k1,k2)) return k2.include(k1[0]);
    return compareangle(k1.dir(),k2.dir());
}
int checkpos(line k1,line k2,line k3){return k3.include(getLL(k1,k2));}
vector<line> getHL(vector<line> &L){ // 求半平面交 , 半平面是逆时针方向 ,
输出按照逆时针
    sort(L.begin(),L.end()); deque<line> q;
    for (int i=0;i<(int)L.size();i++){
        if (i&&sameDir(L[i],L[i-1])) continue;
        while (q.size()>1&&!checkpos(q[q.size()-2],q[q.size()-1],L[i]))
q.pop_back();
        while (q.size()>1&&!checkpos(q[1],q[0],L[i])) q.pop_front();
        q.push_back(L[i]);
    }
    while (q.size()>2&&!checkpos(q[q.size()-2],q[q.size()-1],q[0])) q.p
op_back();
    while (q.size()>2&&!checkpos(q[1],q[0],q[q.size()-1])) q.pop_front
();
    vector<line>ans; for (int i=0;i<q.size();i++) ans.push_back(q[i]);
    return ans;
}
db closepoint(vector<point>&A,int l,int r){ // 最近点对 , 先要按照 x 坐标
排序
    if (r-l<=5){
        db ans=1e20;
        for (int i=l;i<=r;i++) for (int j=i+1;j<=r;j++) ans=min(ans,A[i].
dis(A[j]));
        return ans;
    }
    int mid=l+r>>1; db ans=min(closepoint(A,l,mid),closepoint(A,mid+1,
r));
    vector<point>B; for (int i=l;i<=r;i++) if (abs(A[i].x-A[mid].x)<=ans)
 B.push_back(A[i]);
    sort(B.begin(),B.end(),[](point k1,point k2){return k1.y<k2.y;});
    for (int i=0;i<B.size();i++) for (int j=i+1;j<B.size()&&B[j].y-B[i].
y<ans;j++) ans=min(ans,B[i].dis(B[j]));
    return ans;
}
int checkposCC(circle k1,circle k2){// 返回两个圆的公切线数量
    if (cmp(k1.r,k2.r)==-1) swap(k1,k2);
    db dis=k1.o.dis(k2.o);  int w1=cmp(dis,k1.r+k2.r),w2=cmp(dis,k1.r-k
2.r);
    if (w1>0) return 4; else if (w1==0) return 3; else if (w2>0) return
2;
    else if (w2==0) return 1; else return 0;
}
vector<point> getCL(circle k1,point k2,point k3){ // 沿着 k2->k3 方向给出
 , 相切给出两个
    point k=proj(k2,k3,k1.o); db d=k1.r*k1.r-(k-k1.o).abs2();
    if (sign(d)==-1) return {};
```

```
    point del=(k3-k2).unit()*sqrt(max((db)0.0,d)); return {k-del,k+del};
}
vector<point> getCC(circle k1,circle k2){// 沿圆 k1 逆时针给出 , 相切给出两个
    int pd=checkposCC(k1,k2); if (pd==0||pd==4) return {};
    db a=(k2.o-k1.o).abs2(),cosA=(k1.r*k1.r+a-k2.r*k2.r)/(2*k1.r*sqrt(max(a,(db)0.0)));
    db b=k1.r*cosA,c=sqrt(max((db)0.0,k1.r*k1.r-b*b));
    point k=(k2.o-k1.o).unit(),m=k1.o+k*b,del=k.turn90()*c;
    return {m-del,m+del};
}
vector<point> TangentCP(circle k1,point k2){// 沿圆 k1 逆时针给出
    db a=(k2-k1.o).abs(),b=k1.r*k1.r/a,c=sqrt(max((db)0.0,k1.r*k1.r-b*b));
    point k=(k2-k1.o).unit(),m=k1.o+k*b,del=k.turn90()*c;
    return {m-del,m+del};
}
vector<line> TangentoutCC(circle k1,circle k2){
    int pd=checkposCC(k1,k2); if (pd==0) return {};
    if (pd==1){point k=getCC(k1,k2)[0]; return {(line){k,k}};}
    if (cmp(k1.r,k2.r)==0){
        point del=(k2.o-k1.o).unit().turn90().getdel();
        return {(line){k1.o-del*k1.r,k2.o-del*k2.r},(line){k1.o+del*k1.r,k2.o+del*k2.r}};
    } else {
        point p=(k2.o*k1.r-k1.o*k2.r)/(k1.r-k2.r);
        vector<point>A=TangentCP(k1,p),B=TangentCP(k2,p);
        vector<line>ans; for (int i=0;i<A.size();i++) ans.push_back((line){A[i],B[i]});
        return ans;
    }
}
vector<line> TangentinCC(circle k1,circle k2){
    int pd=checkposCC(k1,k2); if (pd<=2) return {};
    if (pd==3){point k=getCC(k1,k2)[0]; return {(line){k,k}};}
    point p=(k2.o*k1.r+k1.o*k2.r)/(k1.r+k2.r);
    vector<point>A=TangentCP(k1,p),B=TangentCP(k2,p);
    vector<line>ans; for (int i=0;i<A.size();i++) ans.push_back((line){A[i],B[i]});
    return ans;
}
vector<line> TangentCC(circle k1,circle k2){
    int flag=0; if (k1.r<k2.r) swap(k1,k2),flag=1;
    vector<line>A=TangentoutCC(k1,k2),B=TangentinCC(k1,k2);
    for (line k:B) A.push_back(k);
    if (flag) for (line &k:A) swap(k[0],k[1]);
    return A;
}
db getarea(circle k1,point k2,point k3){
    // 圆 k1 与三角形 k2 k3 k1.o 的有向面积交
    point k=k1.o; k1.o=k1.o-k; k2=k2-k; k3=k3-k;
    int pd1=k1.inside(k2),pd2=k1.inside(k3);
    vector<point>A=getCL(k1,k2,k3);
    if (pd1>=0){
        if (pd2>=0) return cross(k2,k3)/2;
        return k1.r*k1.r*rad(A[1],k3)/2+cross(k2,A[1])/2;
    } else if (pd2>=0){
        return k1.r*k1.r*rad(k2,A[0])/2+cross(A[0],k3)/2;
    }else {
        int pd=cmp(k1.r,disSP(k2,k3,k1.o));
        if (pd<=0) return k1.r*k1.r*rad(k2,k3)/2;
        return cross(A[0],A[1])/2+k1.r*k1.r*(rad(k2,A[0])+rad(A[1],k3))/2;
    }
}
circle getcircle(point k1,point k2,point k3){
    db a1=k2.x-k1.x,b1=k2.y-k1.y,c1=(a1*a1+b1*b1)/2;
    db a2=k3.x-k1.x,b2=k3.y-k1.y,c2=(a2*a2+b2*b2)/2;
    db d=a1*b2-a2*b1;
    point o=(point){k1.x+(c1*b2-c2*b1)/d,k1.y+(a1*c2-a2*c1)/d};
    return (circle){o,k1.dis(o)};
}
circle getScircle(vector<point> A){
    random_shuffle(A.begin(),A.end());
    circle ans=(circle){A[0],0};
    for (int i=1;i<A.size();i++)
        if (ans.inside(A[i])==-1){
            ans=(circle){A[i],0};
            for (int j=0;j<i;j++)
                if (ans.inside(A[j])==-1){
                    ans.o=(A[i]+A[j])/2; ans.r=ans.o.dis(A[i]);
                    for (int k=0;k<j;k++)
                        if (ans.inside(A[k])==-1)
                            ans=getcircle(A[i],A[j],A[k]);
                }
        }
    return ans;
}
db area(vector<point> A){ // 多边形用 vector<point> 表示 , 逆时针
    db ans=0;
```

```
    for (int i=0;i<A.size();i++) ans+=cross(A[i],A[(i+1)%A.size()]);
    return ans/2;
}
int checkconvex(vector<point>A){
    int n=A.size(); A.push_back(A[0]); A.push_back(A[1]);
    for (int i=0;i<n;i++) if (sign(cross(A[i+1]-A[i],A[i+2]-A[i]))==-1)
return 0;
    return 1;
}
int contain(vector<point>A,point q){ // 2 内部 1 边界 0 外部
    int pd=0; A.push_back(A[0]);
    for (int i=1;i<A.size();i++){
        point u=A[i-1],v=A[i];
        if (onS(u,v,q)) return 1; if (cmp(u.y,v.y)>0) swap(u,v);
        if (cmp(u.y,q.y)>=0||cmp(v.y,q.y)<0) continue;
        if (sign(cross(u-v,q-v))<0) pd^=1;
    }
    return pd<<1;
}
vector<point> ConvexHull(vector<point>A,int flag=1){ // flag=0 不严格 fl
ag=1 严格
    int n=A.size(); vector<point>ans(n*2);
    sort(A.begin(),A.end()); int now=-1;
    for (int i=0;i<A.size();i++){
        while (now>0&&sign(cross(ans[now]-ans[now-1],A[i]-ans[now-1]))<f
lag) now--;
        ans[++now]=A[i];
    } int pre=now;
    for (int i=n-2;i>=0;i--){
        while (now>pre&&sign(cross(ans[now]-ans[now-1],A[i]-ans[now-1]))
<flag) now--;
        ans[++now]=A[i];
    } ans.resize(now); return ans;
}
db convexDiameter(vector<point>A){
    int now=0,n=A.size(); db ans=0;
    for (int i=0;i<A.size();i++){
        now=max(now,i);
        while (1){
            db k1=A[i].dis(A[now%n]),k2=A[i].dis(A[(now+1)%n]);
            ans=max(ans,max(k1,k2)); if (k2>k1) now++; else break;
        }
    }
    return ans;
}
```

```
int rotating_calipers()  //卡壳
{
    int i , q=1;
    int ans = 0;
    stack[top]=0;
    for(i = 0 ; i < top ; i++)
    {
        while( xmult( p[stack[i+1]] , p[stack[q+1]] , p[stack[i]] ) >
           xmult( p[stack[i+1]] , p[stack[q]] , p[stack[i]] ) )
            q = (q+1)%(top);
        ans = max(ans , max( dis(p[stack[i]] , p[stack[q]]) ,
           dis(p[stack[i+1]] , p[stack[q+1]])));
    }
    return ans;
}
vector<point> convexcut(vector<point>A,point k1,point k2){
    // 保留 k1,k2,p 逆时针的所有点
    int n=A.size(); A.push_back(A[0]); vector<point>ans;
    for (int i=0;i<n;i++){
        int w1=clockwise(k1,k2,A[i]),w2=clockwise(k1,k2,A[i+1]);
        if (w1>=0) ans.push_back(A[i]);
        if (w1*w2<0) ans.push_back(getLL(k1,k2,A[i],A[i+1]));
    }
    return ans;
}
int checkPoS(vector<point>A,point k1,point k2){
    // 多边形 A 和直线 ( 线段 )k1->k2 严格相交 , 注释部分为线段
    struct ins{
        point m,u,v;
        int operator < (const ins& k) const {return m<k.m;}
    }; vector<ins>B;
    //if (contain(A,k1)==2||contain(A,k2)==2) return 1;
    vector<point>poly=A; A.push_back(A[0]);
    for (int i=1;i<A.size();i++) if (checkLL(A[i-1],A[i],k1,k2)){
        point m=getLL(A[i-1],A[i],k1,k2);
        if (inmid(A[i-1],A[i],m)/*&&inmid(k1,k2,m)*/) B.push_back((ins)
{m,A[i-1],A[i]});
    }
    if (B.size()==0) return 0; sort(B.begin(),B.end());
    int now=1; while (now<B.size()&&B[now].m==B[0].m) now++;
    if (now==B.size()) return 0;
    int flag=contain(poly,(B[0].m+B[now].m)/2);
    if (flag==2) return 1;
    point d=B[now].m-B[0].m;
    for (int i=now;i<B.size();i++){
```

```
        if (!(B[i].m==B[i-1].m)&&flag==2) return 1;
        int tag=sign(cross(B[i].v-B[i].u,B[i].m+d-B[i].u));
        if (B[i].m==B[i].u||B[i].m==B[i].v) flag+=tag; else flag+=tag*2;
    }
    //return 0;
    return flag==2;
}
int checkinp(point r,point l,point m){
        if (compareangle(l,r)){return compareangle(l,m)&&compareangle(m,
r);}
        return compareangle(l,m)||compareangle(m,r);
}
int checkPosFast(vector<point>A,point k1,point k2){ // 快速检查线段是否和
多边形严格相交
        if (contain(A,k1)==2||contain(A,k2)==2) return 1; if (k1==k2) r
eturn 0;
        A.push_back(A[0]); A.push_back(A[1]);
        for (int i=1;i+1<A.size();i++)
                if (checkLL(A[i-1],A[i],k1,k2)){
                        point now=getLL(A[i-1],A[i],k1,k2);
                        if (inmid(A[i-1],A[i],now)==0||inmid(k1,k2,now)
==0) continue;
                        if (now==A[i]){
                                if (A[i]==k2) continue;
                                point pre=A[i-1],ne=A[i+1];
                                if (checkinp(pre-now,ne-now,k2-now)) r
eturn 1;
                        } else if (now==k1){
                                if (k1==A[i-1]||k1==A[i]) continue;
                                if (checkinp(A[i-1]-k1,A[i]-k1,k2-k1))
 return 1;
                        } else if (now==k2||now==A[i-1]) continue;
                        else return 1;
                }
        return 0;
}
// 拆分凸包成上下凸壳 凸包尽量都随机旋转一个角度来避免出现相同横坐标
// 尽量特判只有一个点的情况 凸包逆时针
void getUDP(vector<point>A,vector<point>&U,vector<point>&D){
    db l=1e100,r=-1e100;
    for (int i=0;i<A.size();i++) l=min(l,A[i].x),r=max(r,A[i].x);
    int wherel,wherer;
    for (int i=0;i<A.size();i++) if (cmp(A[i].x,l)==0) wherel=i;
    for (int i=A.size();i;i--) if (cmp(A[i-1].x,r)==0) wherer=i-1;
    U.clear(); D.clear(); int now=wherel;
    while (1){D.push_back(A[now]); if (now==wherer) break; now++; if (no
w>=A.size()) now=0;}
    now=wherel;
    while (1){U.push_back(A[now]); if (now==wherer) break; now--; if (no
w<0) now=A.size()-1;}
}
// 需要保证凸包点数大于等于 3,2 内部 ,1 边界 ,0 外部
int containCoP(const vector<point>&U,const vector<point>&D,point k){
    db lx=U[0].x,rx=U[U.size()-1].x;
    if (k==U[0]||k==U[U.size()-1]) return 1;
    if (cmp(k.x,lx)==-1||cmp(k.x,rx)==1) return 0;
    int where1=lower_bound(U.begin(),U.end(),(point){k.x,-1e100})-U.beg
in();
    int where2=lower_bound(D.begin(),D.end(),(point){k.x,-1e100})-D.beg
in();
    int w1=clockwise(U[where1-1],U[where1],k),w2=clockwise(D[where2-1],
D[where2],k);
    if (w1==1||w2==-1) return 0; else if (w1==0||w2==0) return 1; return
 2;
}
// d 是方向 , 输出上方切点和下方切点
pair<point,point> getTangentCow(const vector<point> &U,const vector<poi
nt> &D,point d){
    if (sign(d.x)<0||(sign(d.x)==0&&sign(d.y)<0)) d=d*(-1);
    point whereU,whereD;
    if (sign(d.x)==0) return mp(U[0],U[U.size()-1]);
    int l=0,r=U.size()-1,ans=0;
    while (l<r){int mid=l+r>>1; if (sign(cross(U[mid+1]-U[mid],d))<=0) l
=mid+1,ans=mid+1; else r=mid;}
    whereU=U[ans]; l=0,r=D.size()-1,ans=0;
    while (l<r){int mid=l+r>>1; if (sign(cross(D[mid+1]-D[mid],d))>=0) l
=mid+1,ans=mid+1; else r=mid;}
    whereD=D[ans]; return mp(whereU,whereD);
}
// 先检查 contain, 逆时针给出
pair<point,point> getTangentCoP(const vector<point>&U,const vector<poin
t>&D,point k){
    db lx=U[0].x,rx=U[U.size()-1].x;
    if (k.x<lx){
        int l=0,r=U.size()-1,ans=U.size()-1;
        while (l<r){int mid=l+r>>1; if (clockwise(k,U[mid],U[mid+1])==1)
 l=mid+1; else ans=mid,r=mid;}
        point w1=U[ans]; l=0,r=D.size()-1,ans=D.size()-1;
        while (l<r){int mid=l+r>>1; if (clockwise(k,D[mid],D[mid+1])==-1)
 l=mid+1; else ans=mid,r=mid;}
```

```
        point w2=D[ans]; return mp(w1,w2);
    } else if (k.x>rx){
        int l=1,r=U.size(),ans=0;
        while (l<r){int mid=l+r>>1; if (clockwise(k,U[mid],U[mid-1])==-1)
 r=mid; else ans=mid,l=mid+1;}
        point w1=U[ans]; l=1,r=D.size(),ans=0;
        while (l<r){int mid=l+r>>1; if (clockwise(k,D[mid],D[mid-1])==1)
 r=mid; else ans=mid,l=mid+1;}
        point w2=D[ans]; return mp(w2,w1);
    } else {
        int where1=lower_bound(U.begin(),U.end(),(point){k.x,-1e100})-U.
begin();
        int where2=lower_bound(D.begin(),D.end(),(point){k.x,-1e100})-D.
begin();
        if ((k.x==lx&&k.y>U[0].y)||(where1&&clockwise(U[where1-1],U[wher
e1],k)==1)){
            int l=1,r=where1+1,ans=0;
            while (l<r){int mid=l+r>>1; if (clockwise(k,U[mid],U[mid-1])
==1) ans=mid,l=mid+1; else r=mid;}
            point w1=U[ans]; l=where1,r=U.size()-1,ans=U.size()-1;
            while (l<r){int mid=l+r>>1; if (clockwise(k,U[mid],U[mid+1])
==1) l=mid+1; else ans=mid,r=mid;}
            point w2=U[ans]; return mp(w2,w1);
        } else {
            int l=1,r=where2+1,ans=0;
            while (l<r){int mid=l+r>>1; if (clockwise(k,D[mid],D[mid-1])
==-1) ans=mid,l=mid+1; else r=mid;}
            point w1=D[ans]; l=where2,r=D.size()-1,ans=D.size()-1;
            while (l<r){int mid=l+r>>1; if (clockwise(k,D[mid],D[mid+1])
==-1) l=mid+1; else ans=mid,r=mid;}
            point w2=D[ans]; return mp(w1,w2);
        }
    }
}
struct P3{
    db x,y,z;
    P3 operator + (P3 k1){return (P3){x+k1.x,y+k1.y,z+k1.z};}
    P3 operator - (P3 k1){return (P3){x-k1.x,y-k1.y,z-k1.z};}
    P3 operator * (db k1){return (P3){x*k1,y*k1,z*k1};}
    P3 operator / (db k1){return (P3){x/k1,y/k1,z/k1};}
    db abs2(){return x*x+y*y+z*z;}
    db abs(){return sqrt(x*x+y*y+z*z);}
    P3 unit(){return (*this)/abs();}
    int operator < (const P3 k1) const{
        if (cmp(x,k1.x)!=0) return x<k1.x;
        if (cmp(y,k1.y)!=0) return y<k1.y;
        return cmp(z,k1.z)==-1;
    }
    int operator == (const P3 k1){
        return cmp(x,k1.x)==0&&cmp(y,k1.y)==0&&cmp(z,k1.z)==0;
    }
    void scan(){
        double k1,k2,k3; scanf("%lf%lf%lf",&k1,&k2,&k3);
        x=k1; y=k2; z=k3;
    }
};
P3 cross(P3 k1,P3 k2){return (P3){k1.y*k2.z-k1.z*k2.y,k1.z*k2.x-k1.x*k2.
z,k1.x*k2.y-k1.y*k2.x};}
db dot(P3 k1,P3 k2){return k1.x*k2.x+k1.y*k2.y+k1.z*k2.z;}
//p=(3,4,5),l=(13,19,21),theta=85 ans=(2.83,4.62,1.77)
P3 turn3D(db k1,P3 l,P3 p){
    l=l.unit(); P3 ans; db c=cos(k1),s=sin(k1);
    ans.x=p.x*(l.x*l.x*(1-c)+c)+p.y*(l.x*l.y*(1-c)-l.z*s)+p.z*(l.x*l.z*
(1-c)+l.y*s);
    ans.y=p.x*(l.x*l.y*(1-c)+l.z*s)+p.y*(l.y*l.y*(1-c)+c)+p.z*(l.y*l.z*
(1-c)-l.x*s);
    ans.z=p.x*(l.x*l.z*(1-c)-l.y*s)+p.y*(l.y*l.z*(1-c)+l.x*s)+p.z*(l.z*
l.z*(1-c)+c);
    return ans;
}
typedef vector<P3> VP;
typedef vector<VP> VVP;
db Acos(db x){return acos(max(-(db)1,min(x,(db)1)));}
// 球面距离 , 圆心原点 , 半径 1
db Odist(P3 a,P3 b){db r=Acos(dot(a,b)); return r;}
db r; P3 rnd;
vector<db> solve(db a,db b,db c){
    db r=sqrt(a*a+b*b),th=atan2(b,a);
    if (cmp(c,-r)==-1) return {0};
    else if (cmp(r,c)<=0) return {1};
    else {
        db tr=pi-Acos(c/r); return {th+pi-tr,th+pi+tr};
    }
}
vector<db> jiao(P3 a,P3 b){
    // dot(rd+x*cos(t)+y*sin(t),b) >= cos(r)
    if (cmp(Odist(a,b),2*r)>0) return {0};
    P3 rd=a*cos(r),z=a.unit(),y=cross(z,rnd).unit(),x=cross(y,z).unit();
    vector<db> ret = solve(-(dot(x,b)*sin(r)),-(dot(y,b)*sin(r)),-(cos
(r)-dot(rd,b)));
    return ret;
}
```

```
db norm(db x,db l=0,db r=2*pi){ // change x into [l,r)
    while (cmp(x,l)==-1) x+=(r-l); while (cmp(x,r)>=0) x-=(r-l);
    return x;
}
db disLP(P3 k1,P3 k2,P3 q){
    return (cross(k2-k1,q-k1)).abs()/(k2-k1).abs();
}
db disLL(P3 k1,P3 k2,P3 k3,P3 k4){
    P3 dir=cross(k2-k1,k4-k3); if (sign(dir.abs())==0) return disLP(k1,k
2,k3);
    return fabs(dot(dir.unit(),k1-k2));
}
VP getFL(P3 p,P3 dir,P3 k1,P3 k2){
    db a=dot(k2-p,dir),b=dot(k1-p,dir),d=a-b;
    if (sign(fabs(d))==0) return {};
    return {(k1*a-k2*b)/d};
}
VP getFF(P3 p1,P3 dir1,P3 p2,P3 dir2){// 返回一条线
    P3 e=cross(dir1,dir2),v=cross(dir1,e);
    db d=dot(dir2,v); if (sign(abs(d))==0) return {};
    P3 q=p1+v*dot(dir2,p2-p1)/d; return {q,q+e};
}
// 3D Covex Hull Template
db getV(P3 k1,P3 k2,P3 k3,P3 k4){ // get the Volume
    return dot(cross(k2-k1,k3-k1),k4-k1);
}
db rand_db(){return 1.0*rand()/RAND_MAX;}
VP convexHull2D(VP A,P3 dir){
    P3 x={(db)rand(),(db)rand(),(db)rand()}; x=x.unit();
    x=cross(x,dir).unit(); P3 y=cross(x,dir).unit();
    P3 vec=dir.unit()*dot(A[0],dir);
    vector<point>B;
    for (int i=0;i<A.size();i++) B.push_back((point){dot(A[i],x),dot(A
[i],y)});
    B=ConvexHull(B); A.clear();
    for (int i=0;i<B.size();i++) A.push_back(x*B[i].x+y*B[i].y+vec);
    return A;
}
namespace CH3{
    VVP ret; set<pair<int,int> >e;
    int n; VP p,q;
    void wrap(int a,int b){
        if (e.find({a,b})==e.end()){
            int c=-1;
            for (int i=0;i<n;i++) if (i!=a&&i!=b){
                if (c==-1||sign(getV(q[c],q[a],q[b],q[i]))>0) c=i;
```

```
            }
            if (c!=-1){
                ret.push_back({p[a],p[b],p[c]});
                e.insert({a,b}); e.insert({b,c}); e.insert({c,a});
                wrap(c,b); wrap(a,c);
            }
        }
    }
    VVP ConvexHull3D(VP _p){
        p=q=_p; n=p.size();
        ret.clear(); e.clear();
        for (auto &i:q) i=i+(P3){rand_db()*1e-4,rand_db()*1e-4,rand_db()
*1e-4};
        for (int i=1;i<n;i++) if (q[i].x<q[0].x) swap(p[0],p[i]),swap(q
[0],q[i]);
        for (int i=2;i<n;i++) if ((q[i].x-q[0].x)*(q[1].y-q[0].y)>(q[i].
y-q[0].y)*(q[1].x-q[0].x)) swap(q[1],q[i]),swap(p[1],p[i]);
        wrap(0,1);
        return ret;
    }
}
VVP reduceCH(VVP A){
    VVP ret; map<P3,VP> M;
    for (VP nowF:A){
        P3 dir=cross(nowF[1]-nowF[0],nowF[2]-nowF[0]).unit();
        for (P3 k1:nowF) M[dir].pb(k1);
    }
    for (pair<P3,VP> nowF:M) ret.pb(convexHull2D(nowF.se,nowF.fi));
    return ret;
}
//  把一个面变成 ( 点 , 法向量 ) 的形式
pair<P3,P3> getF(VP F){
    return mp(F[0],cross(F[1]-F[0],F[2]-F[0]).unit());
}
// 3D Cut 保留 dot(dir,x-p)>=0 的部分
VVP ConvexCut3D(VVP A,P3 p,P3 dir){
    VVP ret; VP sec;
    for (VP nowF: A){
        int n=nowF.size(); VP ans; int dif=0;
        for (int i=0;i<n;i++){
            int d1=sign(dot(dir,nowF[i]-p));
            int d2=sign(dot(dir,nowF[(i+1)%n]-p));
            if (d1>=0) ans.pb(nowF[i]);
            if (d1*d2<0){
                P3 q=getFL(p,dir,nowF[i],nowF[(i+1)%n])[0];
```

```
                ans.push_back(q); sec.push_back(q);
            }
            if (d1==0) sec.push_back(nowF[i]); else dif=1;
            dif|=(sign(dot(dir,cross(nowF[(i+1)%n]-nowF[i],nowF[(i+1)%n]
-nowF[i])))==-1);
        }
        if (ans.size()>0&&dif) ret.push_back(ans);
    }
    if (sec.size()>0) ret.push_back(convexHull2D(sec,dir));
    return ret;
}
db vol(VVP A){
    if (A.size()==0) return 0; P3 p=A[0][0]; db ans=0;
    for (VP nowF:A)
        for (int i=2;i<nowF.size();i++)
            ans+=abs(getV(p,nowF[0],nowF[i-1],nowF[i]));
    return ans/6;
}
VVP init(db INF) {
    VVP pss(6,VP(4));
    pss[0][0] = pss[1][0] = pss[2][0] = {-INF, -INF, -INF};
    pss[0][3] = pss[1][1] = pss[5][2] = {-INF, -INF, INF};
    pss[0][1] = pss[2][3] = pss[4][2] = {-INF, INF, -INF};
    pss[0][2] = pss[5][3] = pss[4][1] = {-INF, INF, INF};
    pss[1][3] = pss[2][1] = pss[3][2] = {INF, -INF, -INF};
    pss[1][2] = pss[5][1] = pss[3][3] = {INF, -INF, INF};
    pss[2][2] = pss[4][3] = pss[3][1] = {INF, INF, -INF};
    pss[5][0] = pss[4][0] = pss[3][0] = {INF, INF, INF};
    return pss;
}
```

zyx 的计算几何

```
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;
const int N = 1e6 + 10;

const double eps = 1e-9;
const double PI = acos(-1.0);
const double dinf = 1e99;
const ll inf = 0x3f3f3f3f3f3f3f3f;
struct Line;

struct Point {
    double x, y;

    Point() { x = y = 0; }

    Point(const Line &a);

    Point(const double &a, const double &b) : x(a), y(b) {}

    Point operator+(const Point &a) const {
        return {x + a.x, y + a.y};
    }

    Point operator-(const Point &a) const {
        return {x - a.x, y - a.y};
    }

    Point operator*(const double &a) const {
        return {x * a, y * a};
    }

    Point operator/(const double &d) const {
        return {x / d, y / d};
    }

    bool operator==(const Point &a) const {
        return abs(x - a.x) + abs(y - a.y) < eps;
    }

    // 标准化，转化为膜长为1
    void standardize() {
        *this = *this / sqrt(x * x + y * y);
    }
};


double norm(const Point &p) { return p.x * p.x + p.y * p.y; }

//逆时针转90度
Point orth(const Point &a) { return Point(-a.y, a.x); }

//两点间距离
double dist(const Point &a, const Point &b) {
    return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
```

```cpp
}

//两点间距离的平方
double dist2(const Point &a, const Point &b) {
    return (a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y);
}

struct Line {
    Point s, t;

    Line() {}

    Line(const Point &a, const Point &b) : s(a), t(b) {}

};


struct Circle {
    Point o;
    double r;

    Circle() {}

    Circle(Point P, double R = 0) { o = P, r = R; }
};

//向量的膜长
double length(const Point &p) {
    return sqrt(p.x * p.x + p.y * p.y);
}

//线段的长度
double length(const Line &l) {
    Point p(l);
    return length(p);
}

Point::Point(const Line &a) { *this = a.t - a.s; }

istream &operator>>(istream &in, Point &a) {
    in >> a.x >> a.y;
    return in;
}

ostream &operator<<(ostream &out, Point &a) {
    out << fixed << setprecision(10) << a.x << ' ' << a.y;
    return out;
}

//点积
double dot(const Point &a, const Point &b) { return a.x * b.x + a.y * b.
y; }

//叉积
double det(const Point &a, const Point &b) { return a.x * b.y - a.y * b.
x; }

//正负判断
int sgn(const double &x) { return fabs(x) < eps ? 0 : (x > 0 ? 1 : -1); }

//平方
double sqr(const double &x) { return x * x; }

//将向量 a 逆时针旋转 ang (弧度制)
Point rotate(const Point &a, const double &ang) {
    double x = cos(ang) * a.x - sin(ang) * a.y;
    double y = sin(ang) * a.x + cos(ang) * a.y;
    return {x, y};
}

//点 p 在线段 seg 上，<=0 则包含端点
bool sp_on(const Line &seg, const Point &p) {
    Point a = seg.s, b = seg.t;
    return !sgn(det(p - a, b - a)) && sgn(dot(p - a, p - b)) <= 0;
}

//点 p 在直线 line 上
bool lp_on(const Line &line, const Point &p) {
    Point a = line.s, b = line.t;
    return !sgn(det(p - a, b - a));
}

//凸包，下标从 0 开始，<=0 则凸包中不包含共线点
int andrew(Point *point, Point *convex, int n) {
    sort(point, point + n, [](Point a, Point b) {
        if (a.x != b.x) return a.x < b.x;
        return a.y < b.y;
    });
    int top = 0;
    for (int i = 0; i < n; i++) {
```

```
        while ((top > 1) && det(convex[top - 1] - convex[top - 2], point
[i] - convex[top - 1]) <= 0)
            top--;
        convex[top++] = point[i];
    }
    int tmp = top;
    for (int i = n - 2; i >= 0; i--) {
        while ((top > tmp) && det(convex[top - 1] - convex[top - 2], poin
t[i] - convex[top - 1]) <= 0)
            top--;
        convex[top++] = point[i];
    }
    if (n > 1) top--;
    return top;
}

//斜率
double slope(const Point &a, const Point &b) { return (a.y - b.y) / (a.x
 - b.x); }

//斜率
double slope(const Line &a) { return slope(a.s, a.t); }

//两直线的焦点
Point ll_intersection(const Line &a, const Line &b) {
    double s1 = det(Point(a), b.s - a.s), s2 = det(Point(a), b.t - a.s);
    if (sgn(s1) == 0 && sgn(s2) == 0) return a.s;
    return (b.s * s2 - b.t * s1) / (s2 - s1);
}

//两线段交点p，返回0为无交点，2为交点为端点，1为相交
int ss_cross(const Line &a, const Line &b, Point &p) {
    int d1 = sgn(det(a.t - a.s, b.s - a.s));
    int d2 = sgn(det(a.t - a.s, b.t - a.s));
    int d3 = sgn(det(b.t - b.s, a.s - b.s));
    int d4 = sgn(det(b.t - b.s, a.t - b.s));
    if ((d1 ^ d2) == -2 && (d3 ^ d4) == -2) {
        p = ll_intersection(a, b);
        return 1;
    }
    if (!d1 && sp_on(a, b.s)) {
        p = b.s;
        return 2;
    }
    if (!d2 && sp_on(a, b.t)) {
        p = b.t;
        return 2;
    }
    if (!d3 && sp_on(b, a.s)) {
        p = a.s;
        return 2;
    }
    if (!d4 && sp_on(b, a.t)) {
        p = a.t;
        return 2;
    }
    return 0;
}

//两向量直接的相对位置关系，含义见英文注释
int ccw(const Point &a, Point b, Point c) {
    b = b - a, c = c - a;
    if (sgn(det(b, c)) > 0) return +1;  // "COUNTER_CLOCKWISE"
    if (sgn(det(b, c)) < 0) return -1; // "CLOCKWISE"
    if (sgn(dot(b, c)) < 0) return +2;     // "ONLINE_BACK"
    if (sgn(norm(b) - norm(c)) < 0) return -2;  // "ONLINE_FRONT"
    return 0;                          // "ON_SEGMENT"
}


//点p在线l上的投影位置
Point project(const Line &l, const Point &p) {
    Point base(l);
    double r = dot(base, p - l.s) / sqr(length(base));
    return l.s + (base * r);
}

//线段l和点p的距离
double sp_dist(const Line &l, const Point &p) {
    if (l.s == l.t) return dist(l.s, p);
    Point x = p - l.s, y = p - l.t, z = l.t - l.s;
    if (sgn(dot(x, z)) < 0)return length(x);//P距离A更近
    if (sgn(dot(y, z)) > 0)return length(y);//P距离B更近
    return abs(det(x, z) / length(z));//面积除以底边长
}

//直线l和点p的距离
double lp_dist(const Line &l, const Point &p) {
    Point x = p - l.s, y = p - l.t, z = l.t - l.s;
    return abs(det(x, z) / length(z));//面积除以底边长
```

```
}

//圆 c 和直线 l 的交点，返回值为交点的数量，ans 为交点位置
int cl_cross(const Circle &c, const Line &l, pair<Point, Point> &ans) {
    Point a = c.o;
    double r = c.r;
    Point pr = project(l, a);
    double dis = dist(pr, a);
    double tmp = r * r - dis * dis;
    if (sgn(tmp) == 1) {
        double base = sqrt(max(0.0, r * r - dis * dis));
        Point e(l);
        e.standardize();
        e = e * base;
        ans = make_pair(pr + e, pr - e);
        return 2;
    } else if (sgn(tmp) == 0) {
        ans = make_pair(pr, pr);
        return 1;
    } else return 0;
}

//圆 c 和线段 l 交点个数，下面 cs_cross 用到
int intersectCS(Circle c, Line l) {
    if (sgn(norm(project(l, c.o) - c.o) - c.r * c.r) > 0) return 0;
    double d1 = length(c.o - l.s), d2 = length(c.o - l.t);
    if (sgn(d1 - c.r) <= 0 && sgn(d2 - c.r) <= 0) return 0;
    if ((sgn(d1 - c.r) < 0 && sgn(d2 - c.r) > 0) || (sgn(d1 - c.r) > 0 &&
 sgn(d2 - c.r) < 0)) return 1;
    Point h = project(l, c.o);
    if (dot(l.s - h, l.t - h) < 0) return 2;
    return 0;
}

//圆和线段交点，返回交点数量
int cs_cross(Circle c, Line s, pair<Point, Point> &ans) {
    Line l(s);
    int num = cl_cross(c, l, ans);
    int res = intersectCS(c, s);
    if (res == 2) return 2;
    if (num > 1) {
        if (dot(l.s - ans.first, l.t - ans.first) > 0) swap(ans.first, a
ns.second);
        ans.second = ans.first;
    }
    return res;
}

//两圆交点，位置关系见注释
int cc_cross(const Circle &cir1, const Circle &cir2, pair<Point, Point>
&ans) {
    const Point &c1 = cir1.o, &c2 = cir2.o;
    const double &r1 = cir1.r, &r2 = cir2.r;
    double x1 = c1.x, x2 = c2.x, y1 = c1.y, y2 = c2.y;
    double d = length(c1 - c2);
    if (sgn(fabs(r1 - r2) - d) > 0) return 0;  //内含
    if (sgn(r1 + r2 - d) < 0) return 4; //相离
    double a = r1 * (x1 - x2) * 2, b = r1 * (y1 - y2) * 2, c = r2 * r2 -
r1 * r1 - d * d;
    double p = a * a + b * b, q = -a * c * 2, r = c * c - b * b;

    double cosa, sina, cosb, sinb;
    //One Intersection
    if (sgn(d - (r1 + r2)) == 0 || sgn(d - fabs(r1 - r2)) == 0) {
        cosa = -q / p / 2;
        sina = sqrt(1 - sqr(cosa));
        Point p0(x1 + r1 * cosa, y1 + r1 * sina);
        if (sgn(dist(p0, c2) - r2)) p0.y = y1 - r1 * sina;
        ans = pair<Point, Point>(p0, p0);
        if (sgn(r1 + r2 - d) == 0) return 3;    //外切
        else return 1;  //内切
    }
    //Two Intersections
    double delta = sqrt(q * q - p * r * 4);
    cosa = (delta - q) / p / 2;
    cosb = (-delta - q) / p / 2;
    sina = sqrt(1 - sqr(cosa));
    sinb = sqrt(1 - sqr(cosb));
    Point p1(x1 + r1 * cosa, y1 + r1 * sina);
    Point p2(x1 + r1 * cosb, y1 + r1 * sinb);
    if (sgn(dist(p1, c2) - r2)) p1.y = y1 - r1 * sina;
    if (sgn(dist(p2, c2) - r2)) p2.y = y1 - r1 * sinb;
    if (p1 == p2) p1.y = y1 - r1 * sina;
    ans = pair<Point, Point>(p1, p2);
    return 2;   //  相交
}

//点 p 关于直线 l 的对称点
Point lp_sym(const Line &l, const Point &p) {
    return p + (project(l, p) - p) * 2;
```

```
}

//返回两向量的夹角
double alpha(const Point &t1, const Point &t2) {
    double theta;
    theta = atan2((double) t2.y, (double) t2.x) - atan2((double) t1.y,
(double) t1.x);
    if (sgn(theta) < 0)
        theta += 2.0 * PI;
    return theta;
}

//【射线法】判断点 A 是否在任意多边形 Poly 以内，下标从 1 开始（为保险起见，可以
在判断前将所有点随机旋转一个角度防止被卡）
int pip(const Point *P, const int &n, const Point &a) {
    int cnt = 0;
    double tmp;
    for (int i = 1; i <= n; ++i) {
        int j = i < n ? i + 1 : 1;
        if (sp_on(Line(P[i], P[j]), a))return 2;//点在多边形上
        if (a.y >= min(P[i].y, P[j].y) && a.y < max(P[i].y, P[j].y))//纵
坐标在该线段两端点之间
            tmp = P[i].x + (a.y - P[i].y) / (P[j].y - P[i].y) * (P[j].x -
 P[i].x), cnt += sgn(tmp - a.x) > 0;//交点在 A 右方
    }
    return cnt & 1;//穿过奇数次则在多边形以内
}

//判断 AL 是否在 AR 右边
bool pip_convex_jud(const Point &a, const Point &L, const Point &R) {
    return sgn(det(L - a, R - a)) > 0;//必须严格以内
}

//【二分法】判断点 A 是否在凸多边形 Poly 以内，下标从 0 开始
bool pip_convex(const Point *P, const int &n, const Point &a) {
    //点按逆时针给出
    if (pip_convex_jud(P[0], a, P[1]) || pip_convex_jud(P[0], P[n - 1],
a)) return 0;//在 P[0_1]或 P[0_n-1]外
    if (sp_on(Line(P[0], P[1]), a) || sp_on(Line(P[0], P[n - 1]), a)) re
turn 2;//在 P[0_1]或 P[0_n-1]上
    int l = 1, r = n - 2;
    while (l < r) {//二分找到一个位置 pos 使得 P[0]_A 在 P[0_pos],P[0_(pos+
1)]之间
        int mid = (l + r + 1) >> 1;
        if (pip_convex_jud(P[0], P[mid], a))l = mid;
        else r = mid - 1;
    }
    if (pip_convex_jud(P[l], a, P[l + 1]))return 0;//在 P[pos_(pos+1)]外
    if (sp_on(Line(P[l], P[l + 1]), a))return 2;//在 P[pos_(pos+1)]上
    return 1;
}
// 多边形是否包含线段
// 因此我们可以先求出所有和线段相交的多边形的顶点，然后按照 X-Y 坐标排序(X 坐标
小的排在前面，对于 X 坐标相同的点，Y 坐标小的排在前面，
// 这种排序准则也是为了保证水平和垂直情况的判断正确)，这样相邻的两个点就是在线
段上相邻的两交点，如果任意相邻两点的中点也在多边形内，
// 则该线段一定在多边形内。

//【判断多边形 A 与多边形 B 是否相离】
int pp_judge(Point *A, int n, Point *B, int m) {
    for (int i1 = 1; i1 <= n; ++i1) {
        int j1 = i1 < n ? i1 + 1 : 1;
        for (int i2 = 1; i2 <= m; ++i2) {
            int j2 = i2 < m ? i2 + 1 : 1;
            Point tmp;
            if (ss_cross(Line(A[i1], A[j1]), Line(B[i2], B[j2]), tmp)) r
eturn 0;//两线段相交
            if (pip(B, m, A[i1]) || pip(A, n, B[i2]))return 0;//点包含在
内
        }
    }
    return 1;
}

//【任意多边形 P 的面积】，下标从 0 开始
double area(Point *P, int n) {
    double S = 0;
    for (int i = 0; i < n; i++) S += det(P[i], P[(i + 1) % n]);
    return S * 0.5;
}

//多边形和圆的面积交 ， 下表从 0 开始
double pc_area(Point *p, int n, const Circle &c) {
    if (n < 3) return 0;
    function<double(Circle, Point, Point)> dfs = [&](Circle c, Point a,
Point b) {
        Point va = c.o - a, vb = c.o - b;
        double f = det(va, vb), res = 0;
```

```cpp
        if (sgn(f) == 0) return res;
        if (sgn(max(length(va), length(vb)) - c.r) <= 0) return f;
        Point d(dot(va, vb), det(va, vb));
        if (sgn(sp_dist(Line(a, b), c.o) - c.r) >= 0) return c.r * c.r *
atan2(d.y, d.x);
        pair<Point, Point> u;
        int cnt = cs_cross(c, Line(a, b), u);
        if (cnt == 0) return res;
        if (cnt > 1 && sgn(dot(u.second - u.first, a - u.first)) > 0) swa
p(u.first, u.second);
        res += dfs(c, a, u.first);
        if (cnt == 2) res += dfs(c, u.first, u.second) + dfs(c, u.second,
 b);
        else if (cnt == 1) res += dfs(c, u.first, b);
        return res;
    };
    double res = 0;
    for (int i = 0; i < n; i++) {
        res += dfs(c, p[i], p[(i + 1) % n]);
    }
    return res * 0.5;
}

Line Q[N];

//【半平面交】
int judge(Line L, Point a) { return sgn(det(a - L.s, L.t - L.s)) > 0; }/
/判断点 a 是否在直线 L 的右边
int halfcut(Line *L, int n, Point *P) {
    sort(L, L + n, [](const Line &a, const Line &b) {
        double d = atan2((a.t - a.s).y, (a.t - a.s).x) - atan2((b.t - b.
s).y, (b.t - b.s).x);
        return sgn(d) ? sgn(d) < 0 : judge(a, b.s);
    });

    int m = n;
    n = 0;
    for (int i = 0; i < m; ++i)
        if (i == 0 || sgn(atan2(Point(L[i]).y, Point(L[i]).x) - atan2(Po
int(L[i - 1]).y, Point(L[i - 1]).x)))
            L[n++] = L[i];
    int h = 1, t = 0;
    for (int i = 0; i < n; ++i) {
        while (h < t && judge(L[i], ll_intersection(Q[t], Q[t - 1]))) --
t;//当队尾两个直线交点不是在直线 L[i]上或者左边时就出队
        while (h < t && judge(L[i], ll_intersection(Q[h], Q[h + 1]))) ++
h;//当队头两个直线交点不是在直线 L[i]上或者左边时就出队
        Q[++t] = L[i];

    }
    while (h < t && judge(Q[h], ll_intersection(Q[t], Q[t - 1]))) --t;
    while (h < t && judge(Q[t], ll_intersection(Q[h], Q[h + 1]))) ++h;
    n = 0;
    for (int i = h; i <= t; ++i) {
        P[n++] = ll_intersection(Q[i], Q[i < t ? i + 1 : h]);
    }
    return n;
}

Point V1[N], V2[N];

//【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
int mincowski(Point *P1, int n, Point *P2, int m, Point *V) {
    for (int i = 0; i < n; ++i) V1[i] = P1[(i + 1) % n] - P1[i];
    for (int i = 0; i < m; ++i) V2[i] = P2[(i + 1) % m] - P2[i];
    int t = 0, i = 0, j = 0;
    V[t++] = P1[0] + P2[0];
    while (i < n && j < m) V[t] = V[t - 1] + (sgn(det(V1[i], V2[j])) > 0 ?
 V1[i++] : V2[j++]), t++;
    while (i < n) V[t] = V[t - 1] + V1[i++], t++;
    while (j < m) V[t] = V[t - 1] + V2[j++], t++;
    return t;
}

//【三点确定一圆】向量垂心法
Circle external_circle(const Point &A, const Point &B, const Point &C) {
    Point P1 = (A + B) * 0.5, P2 = (A + C) * 0.5;
    Line R1 = Line(P1, P1 + orth(B - A));
    Line R2 = Line(P2, P2 + orth(C - A));
    Circle O;
    O.o = ll_intersection(R1, R2);
    O.r = length(A - O.o);
    return O;
}

//三角形内接圆
Circle internal_circle(const Point &A, const Point &B, const Point &C) {
    double a = dist(B, C), b = dist(A, C), c = dist(A, B);
    double s = (a + b + c) / 2;
    double S = sqrt(max(0.0, s * (s - a) * (s - b) * (s - c)));
```

```cpp
    double r = S / s;

    return Circle((A * a + B * b + C * c) / (a + b + c), r);
}

//动态凸包
struct ConvexHull {

    int op;

    struct cmp {
        bool operator()(const Point &a, const Point &b) const {
            return sgn(a.x - b.x) < 0 || sgn(a.x - b.x) == 0 && sgn(a.y -
 b.y) < 0;
        }
    };

    set<Point, cmp> s;

    ConvexHull(int o) {
        op = o;
        s.clear();
    }

    inline int PIP(Point P) {
        set<Point>::iterator it = s.lower_bound(Point(P.x, -dinf));//找
到第一个横坐标大于 P 的点
        if (it == s.end())return 0;
        if (sgn(it->x - P.x) == 0) return sgn((P.y - it->y) * op) <= 0;//
比较纵坐标大小
        if (it == s.begin())return 0;
        set<Point>::iterator j = it, k = it;
        --j;
        return sgn(det(P - *j, *k - *j) * op) >= 0;//看叉姬 1
    }

    inline int judge(set<Point>::iterator it) {
        set<Point>::iterator j = it, k = it;
        if (j == s.begin())return 0;
        --j;
        if (++k == s.end())return 0;
        return sgn(det(*it - *j, *k - *j) * op) >= 0;//看叉姬
    }

    inline void insert(Point P) {
        if (PIP(P))return;//如果点 P 已经在凸壳上或凸包里就不插入了
        set<Point>::iterator tmp = s.lower_bound(Point(P.x, -dinf));
        if (tmp != s.end() && sgn(tmp->x - P.x) == 0)s.erase(tmp);//特判
横坐标相等的点要去掉
        s.insert(P);
        set<Point>::iterator it = s.find(P), p = it;
        if (p != s.begin()) {
            --p;
            while (judge(p)) {
                set<Point>::iterator temp = p--;
                s.erase(temp);
            }
        }
        if ((p = ++it) != s.end()) {
            while (judge(p)) {
                set<Point>::iterator temp = p++;
                s.erase(temp);
            }
        }
    }
} up(1), down(-1);

int PIC(Circle C, Point a) { return sgn(length(a - C.o) - C.r) <= 0; }//
判断点 A 是否在圆 C 内
void Random(Point *P, int n) { for (int i = 0; i < n; ++i)swap(P[i], P
[(rand() + 1) % n]); }//随机一个排列
//【求点集 P 的最小覆盖圆】 O(n)
Circle min_circle(Point *P, int n) {
//  random_shuffle(P,P+n);
    Random(P, n);
    Circle C = Circle(P[0], 0);
    for (int i = 1; i < n; ++i)
        if (!PIC(C, P[i])) {
            C = Circle(P[i], 0);
            for (int j = 0; j < i; ++j)
                if (!PIC(C, P[j])) {
                    C.o = (P[i] + P[j]) * 0.5, C.r = length(P[j] - C.o);
                    for (int k = 0; k < j; ++k) if (!PIC(C, P[k])) C = ext
ernal_circle(P[i], P[j], P[k]);
                }
        }
    return C;
}
```

```cpp
int temp[N];

//最近点对
double closest_point(Point *p, int n) {
    function<double(int, int)> merge = [&](int l, int r) {
        double d = dinf;
        if (l == r) return d;
        if (l + 1 == r) return dist(p[l], p[r]);
        int mid = (l + r) >> 1;
        double d1 = merge(l, mid);
        double d2 = merge(mid + 1, r);
        d = min(d1, d2);
        int i, j, k = 0;
        for (i = l; i <= r; i++) {
            if (sgn(abs(p[mid].x - p[i].x) - d) <= 0)
                temp[k++] = i;

        }
        sort(temp, temp + k, [&](const int &a, const int &b) {
            return sgn(p[a].y - p[b].y) < 0;
        });
        for (i = 0; i < k; i++) {
            for (j = i + 1; j < k && sgn(p[temp[j]].y - p[temp[i]].y - d)
 <= 0; j++) {
                double d3 = dist(p[temp[i]], p[temp[j]]);
                d = min(d, d3);
            }
        }
        return d;
    };
    sort(p, p + n, [&](const Point &a, const Point &b) {
        if (sgn(a.x - b.x) == 0) return sgn(a.y - b.y) < 0;
        else return sgn(a.x - b.x) < 0;
    });
    return merge(0, n - 1);
}

//圆和点的切线
int tangent(const Circle &c1, const Point &p2, pair<Point, Point> &ans)
{
    Point tmp = c1.o - p2;
    int sta;
    if (sgn(norm(tmp) - c1.r * c1.r) < 0) return 0;
    else if (sgn(norm(tmp) - c1.r * c1.r) == 0) sta = 1;
    else sta = 2;
    Circle c2 = Circle(p2, sqrt(max(0.0, norm(tmp) - c1.r * c1.r)));
    cc_cross(c1, c2, ans);
    return sta;
}

//圆和圆的切线
int tangent(Circle c1, Circle c2, vector<Line> &ans) {
    ans.clear();
    if (sgn(c1.r - c2.r) < 0) swap(c1, c2);
    double g = norm(c1.o - c2.o);
    if (sgn(g) == 0) return 0;
    Point u = (c2.o - c1.o) / sqrt(g);
    Point v = orth(u);
    for (int s = 1; s >= -1; s -= 2) {
        double h = (c1.r + s * c2.r) / sqrt(g);
        if (sgn(1 - h * h) == 0) {
            ans.push_back(Line(c1.o + u * c1.r, c1.o + (u + v) * c1.r));
        } else if (sgn(1 - h * h) >= 0) {
            Point uu = u * h, vv = v * sqrt(1 - h * h);
            ans.push_back(Line(c1.o + (uu + vv) * c1.r, c2.o - (uu + vv)
* c2.r * s));
            ans.push_back(Line(c1.o + (uu - vv) * c1.r, c2.o - (uu - vv)
* c2.r * s));
        }
    }

    return ans.size();
}

//两圆面积交
double areaofCC(Circle c1, Circle c2) {
    if (c1.r > c2.r) swap(c1, c2);
    double nor = norm(c1.o - c2.o);
    double dist = sqrt(max(0.0, nor));

    if (sgn(c1.r + c2.r - dist) <= 0) return 0;

    if (sgn(dist + c1.r - c2.r) <= 0) return c1.r * c1.r * PI;

    double val;
    val = (nor + c1.r * c1.r - c2.r * c2.r) / (2 * c1.r * dist);
    val = max(val, -1.0), val = min(val, 1.0);
    double theta1 = acos(val);
    val = (nor + c2.r * c2.r - c1.r * c1.r) / (2 * c2.r * dist);
    val = max(val, -1.0), val = min(val, 1.0);
```

```
    double theta2 = acos(val);
    return (theta1 - sin(theta1 + theta1) * 0.5) * c1.r * c1.r + (theta2
 - sin(theta2 + theta2) * 0.5) * c2.r * c2.r;
}

//https://onlinejudge.u-aizu.ac.jp/courses/library/4/CGL/all/CGL_4_C
//把凸包切一刀
int convexCut(Point *p, Point *ans, int n, Line l) {
    int top = 0;
    for (int i = 0; i < n; i++) {
        Point a = p[i], b = p[(i + 1) % n];
        if (ccw(l.s, l.t, a) != -1) ans[top++] = a;
        if (ccw(l.s, l.t, a) * ccw(l.s, l.t, b) < 0)
            ans[top++] = ll_intersection(Line(a, b), l);
    }
    return top;
}

//两球体积交
double SphereCross(double d, double r1, double r2) {
    if (r1 < r2) swap(r1, r2);
    if (sgn(d - r1 - r2) >= 0) return 0;
    if (sgn(d + r2 - r1) <= 0) return 4.0 / 3 * PI * r2 * r2 * r2;
    double co = (r1 * r1 + d * d - r2 * r2) / (2.0 * d * r1);
    double h = r1 * (1 - co);
    double ans = (1.0 / 3) * PI * (3.0 * r1 - h) * h * h;
    co = (r2 * r2 + d * d - r1 * r1) / (2.0 * d * r2);
    h = r2 * (1 - co);
    ans += (1.0 / 3) * PI * (3.0 * r2 - h) * h * h;
    return ans;
}
```

几何一些定理（或知识点？

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。

单纯形体积

R^n空间下标准单纯形与原点围成的体积为$\frac{1}{n!}$

球缺体积公式

球缺体积 V=(π/3)(3R-h)*h^2 或写成 V=πh^2(R-h/3)，（R 是球的半径,h 是球缺的高).如果已知球缺高 h，底面半径 r,则 V=[πh(3r^2+h^2)]/6

海伦公式

$$S = \sqrt{p(p-a)(p-b)(p-c)}$$

球体积交和并

```
#include<bits/stdc++.h>
#define fi first
#define sf scanf
#define se second
#define pf printf
#define pb push_back
#define mp make_pair
#define sz(x) ((int)(x).size())
#define all(x) (x).begin(),(x).end()
#define mem(x,y) memset((x),(y),sizeof(x))
#define fup(i,x,y) for(int i=(x);i<=(y);++i)
#define fdn(i,x,y) for(int i=(x);i>=(y);--i)
typedef long long ll;
typedef long double ld;
typedef unsigned long long ull;
typedef std::pair<int,int> pii;
using namespace std;

const ld pi=acos(-1);

ld pow2(ld x){return x*x;}

ld pow3(ld x){return x*x*x;}

ld dis(ld x1,ld y1,ld z1,ld x2,ld y2,ld z2)
{
    return pow2(x1-x2)+pow2(y1-y2)+pow2(z1-z2);
}
```

```
ld cos(ld a,ld b,ld c){return (b*b+c*c-a*a)/(2*b*c);}

ld cap(ld r,ld h){return pi*(r*3-h)*h*h/3;} // 球缺体积公式，h 为球缺的高

//2 球体积交
ld sphere_intersect(ld x1,ld y1,ld z1,ld r1,ld x2,ld y2,ld z2,ld r2)
{
    ld d=dis(x1,y1,z1,x2,y2,z2);
    //相离
    if(d>=pow2(r1+r2))return 0;
    //包含
    if(d<=pow2(r1-r2))return pow3(min(r1,r2))*4*pi/3;
    //相交
    ld h1=r1-r1*cos(r2,r1,sqrt(d)),h2=r2-r2*cos(r1,r2,sqrt(d));
    return cap(r1,h1)+cap(r2,h2);
}

//2 球体积并
ld sphere_union(ld x1,ld y1,ld z1,ld r1,ld x2,ld y2,ld z2,ld r2)
{
    ld d=dis(x1,y1,z1,x2,y2,z2);
    //相离
    if(d>=pow2(r1+r2))return (pow3(r1)+pow3(r2))*4*pi/3;
    //包含
    if(d<=pow2(r1-r2))return pow3(max(r1,r2))*4*pi/3;
    //相交
    ld h1=r1+r1*cos(r2,r1,sqrt(d)),h2=r2+r2*cos(r1,r2,sqrt(d));
    return cap(r1,h1)+cap(r2,h2);
}

int main()
{
    double x1,y1,z1,r1,x2,y2,z2,r2;
    sf("%lf%lf%lf%lf%lf%lf%lf%lf",&x1,&y1,&z1,&r1,&x2,&y2,&z2,&r2);
    pf("%.12Lf\n",sphere_union(x1,y1,z1,r1,x2,y2,z2,r2));
    return 0;
}
```

自适应辛普森

```
double f(double x) {
}

double simpson(double l, double r) {
    double mid = (l + r) / 2;
    return (r - l) * (f(l) + 4 * f(mid) + f(r)) / 6;  // 辛普森公式
}

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

计算几何全家桶

```
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;
const ll N = 1 << 20;
const ll mod = 1e9 + 7;
const double dinf = 1e99;
const int inf = 0x3f3f3f3f;
const ll linf = 0x3f3f3f3f3f3f3f3f;

const double eps = 1e-9;
const double PI = acos(-1.0);

struct Line;

struct Point {
    double x, y;

    Point() { x = y = 0; }

    Point(const Line &a);

    Point(const double &a, const double &b) : x(a), y(b) {}

    Point operator+(const Point &a) const {
        return {x + a.x, y + a.y};
    }
```

```cpp
    Point operator-(const Point &a) const {
        return {x - a.x, y - a.y};
    }

    Point operator*(const double &a) const {
        return {x * a, y * a};
    }

    Point operator/(const double &d) const {
        return {x / d, y / d};
    }

    bool operator==(const Point &a) const {
        return abs(x - a.x) + abs(y - a.y) < eps;
    }

    void standardize() {
        *this = *this / sqrt(x * x + y * y);
    }
};

Point normal(const Point &a) { return Point(-a.y, a.x); }

double dist(const Point &a, const Point &b) {
    return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
}

double dist2(const Point &a, const Point &b) {
    return (a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y);
}

struct Line {
    Point s, t;

    Line() {}

    Line(const Point &a, const Point &b) : s(a), t(b) {}

};

struct circle {
    Point o;
    double r;

    circle() {}

    circle(Point P, double R = 0) { o = P, r = R; }
};

double length(const Point &p) {
    return sqrt(p.x * p.x + p.y * p.y);
}

double length(const Line &l) {
    Point p(l);
    return length(p);
}

Point::Point(const Line &a) { *this = a.t - a.s; }

istream &operator>>(istream &in, Point &a) {
    in >> a.x >> a.y;
    return in;
}

double dot(const Point &a, const Point &b) {
    return a.x * b.x + a.y * b.y;
}

double det(const Point &a, const Point &b) {
    return a.x * b.y - a.y * b.x;
}

int sgn(const double &x) { return fabs(x) < eps ? 0 : (x > 0 ? 1 : -1); }

double sqr(const double &x) { return x * x; }

Point rotate(const Point &a, const double &ang) {
    double x = cos(ang) * a.x - sin(ang) * a.y;
    double y = sin(ang) * a.x + cos(ang) * a.y;
    return {x, y};
}

//点在线段上 <=0 包含端点
bool sp_on(const Line &seg, const Point &p) {
    Point a = seg.s, b = seg.t;
    return !sgn(det(p - a, b - a)) && sgn(dot(p - a, p - b)) <= 0;
}

bool lp_on(const Line &line, const Point &p) {
    Point a = line.s, b = line.t;
```

```
    return !sgn(det(p - a, b - a));
}

//等于不包含共线
int andrew(Point *point, Point *convex, int n) {
    sort(point, point + n, [](Point a, Point b) {
        if (a.x != b.x) return a.x < b.x;
        return a.y < b.y;
    });
    int top = 0;
    for (int i = 0; i < n; i++) {
        while ((top > 1) && det(convex[top - 1] - convex[top - 2], point
[i] - convex[top - 1]) <= 0)
            top--;
        convex[top++] = point[i];
    }
    int tmp = top;
    for (int i = n - 2; i >= 0; i--) {
        while ((top > tmp) && det(convex[top - 1] - convex[top - 2], poin
t[i] - convex[top - 1]) <= 0)
            top--;
        convex[top++] = point[i];
    }
    if (n > 1) top--;
    return top;
}

double slope(const Point &a, const Point &b) {
    return (a.y - b.y) / (a.x - b.x);
}

double slope(const Line &a) {
    return slope(a.s, a.t);
}

Point ll_intersection(const Line &a, const Line &b) {
    double s1 = det(Point(a), b.s - a.s), s2 = det(Point(a), b.t - a.s);
    return (b.s * s2 - b.t * s1) / (s2 - s1);
}

int ss_cross(const Line &a, const Line &b, Point &p) {
    int d1 = sgn(det(a.t - a.s, b.s - a.s));
    int d2 = sgn(det(a.t - a.s, b.t - a.s));
    int d3 = sgn(det(b.t - b.s, a.s - b.s));
    int d4 = sgn(det(b.t - b.s, a.t - b.s));
    if ((d1 ^ d2) == -2 && (d3 ^ d4) == -2) {
        p = ll_intersection(a, b);
        return 1;
    }
    if (!d1 && sp_on(a, b.s)) {
        p = b.s;
        return 2;
    }
    if (!d2 && sp_on(a, b.t)) {
        p = b.t;
        return 2;
    }
    if (!d3 && sp_on(b, a.s)) {
        p = a.s;
        return 2;
    }
    if (!d4 && sp_on(b, a.t)) {
        p = a.t;
        return 2;
    }
    return 0;
}

Point project(const Line &l, const Point &p) {
    Point base(l);
    double r = dot(base, p - l.s) / sqr(length(base));
    return l.s + (base * r);
}

double sp_dist(const Line &l, const Point &p) {
    if (l.s == l.t) return dist(l.s, p);
    Point x = p - l.s, y = p - l.t, z = l.t - l.s;
    if (sgn(dot(x, z)) < 0)return length(x);//P 距离 A 更近
    if (sgn(dot(y, z)) > 0)return length(y);//P 距离 B 更近
    return abs(det(x, z) / length(z));//面积除以底边长
}

double lp_dist(const Line &l, const Point &p) {
    Point x = p - l.s, y = p - l.t, z = l.t - l.s;
    return abs(det(x, z) / length(z));//面积除以底边长
}

int lc_cross(const Line &l, const Point &a, const double &r, pair<Point,
 Point> &ans) {
    int num = 0;
    Point pr = project(l, a);
```

```
    double dis = dist(pr, a);
    double tmp = r * r - dis * dis;
    if (sgn(tmp) == 1) num = 2;
    else if (sgn(tmp) == 0) num = 1;
    else return 0;
    double base = sqrt(r * r - dis * dis);
    Point e(l);
    e.standardize();
    e = e * base;
    ans = make_pair(pr + e, pr - e);
    return num;
}

int cc_cross(const Point &c1, const double &r1, const Point &c2, const d
ouble &r2, pair<Point, Point> &ans) {
    double x1 = c1.x, x2 = c2.x, y1 = c1.y, y2 = c2.y;
    double d = length(c1 - c2);
    if (sgn(fabs(r1 - r2) - d) > 0) return -1;  //内含
    if (sgn(r1 + r2 - d) < 0) return 0; //相离
    double a = r1 * (x1 - x2) * 2, b = r1 * (y1 - y2) * 2, c = r2 * r2 -
r1 * r1 - d * d;
    double p = a * a + b * b, q = -a * c * 2, r = c * c - b * b;

    double cosa, sina, cosb, sinb;
    //One Intersection
    if (sgn(d - (r1 + r2)) == 0 || sgn(d - fabs(r1 - r2)) == 0) {
        cosa = -q / p / 2;
        sina = sqrt(1 - sqr(cosa));
        Point p0(x1 + r1 * cosa, y1 + r1 * sina);
        if (sgn(dist(p0, c2) - r2)) p0.y = y1 - r1 * sina;
        ans = pair<Point, Point>(p0, p0);
        return 1;
    }
    //Two Intersections
    double delta = sqrt(q * q - p * r * 4);
    cosa = (delta - q) / p / 2;
    cosb = (-delta - q) / p / 2;
    sina = sqrt(1 - sqr(cosa));
    sinb = sqrt(1 - sqr(cosb));
    Point p1(x1 + r1 * cosa, y1 + r1 * sina);
    Point p2(x1 + r1 * cosb, y1 + r1 * sinb);
    if (sgn(dist(p1, c2) - r2)) p1.y = y1 - r1 * sina;
    if (sgn(dist(p2, c2) - r2)) p2.y = y1 - r1 * sinb;
    if (p1 == p2) p1.y = y1 - r1 * sina;
    ans = pair<Point, Point>(p1, p2);
    return 2;
}

Point lp_sym(const Line &l, const Point &p) {
    return p + (project(l, p) - p) * 2;
}

double alpha(const Point &t1, const Point &t2) {
    double theta;
    theta = atan2((double) t2.y, (double) t2.x) - atan2((double) t1.y,
(double) t1.x);
    if (sgn(theta) < 0)
        theta += 2.0 * PI;
    return theta;
}

int pip(const Point *P, const int &n, const Point &a) {//【射线法】判断点
A 是否在任意多边形 Poly 以内
    int cnt = 0;
    int tmp;
    for (int i = 1; i <= n; ++i) {
        int j = i < n ? i + 1 : 1;
        if (sp_on(Line(P[i], P[j]), a))return 2;//点在多边形上
        if (a.y >= min(P[i].y, P[j].y) && a.y < max(P[i].y, P[j].y))//纵
坐标在该线段两端点之间
            tmp = P[i].x + (a.y - P[i].y) / (P[j].y - P[i].y) * (P[j].x -
 P[i].x), cnt += sgn(tmp - a.x) > 0;//交点在 A 右方
    }
    return cnt & 1;//穿过奇数次则在多边形以内
}

bool pip_convex_jud(const Point &a, const Point &L, const Point &R) {//
判断 AL 是否在 AR 右边
    return sgn(det(L - a, R - a)) > 0;//必须严格以内
}

bool pip_convex(const Point *P, const int &n, const Point &a) {//【二分
法】判断点 A 是否在凸多边形 Poly 以内
    //点按逆时针给出
    if (pip_convex_jud(P[0], a, P[1]) || pip_convex_jud(P[0], P[n - 1],
a)) return 0;//在 P[0_1]或 P[0_n-1]外
    if (sp_on(Line(P[0], P[1]), a) || sp_on(Line(P[0], P[n - 1]), a)) re
turn 2;//在 P[0_1]或 P[0_n-1]上
    int l = 1, r = n - 2;
```

```
    while (l < r) {//二分找到一个位置pos 使得P[0]_A 在P[0_pos],P[0_(pos+1)]之间
        int mid = (l + r + 1) >> 1;
        if (pip_convex_jud(P[0], P[mid], a))l = mid;
        else r = mid - 1;
    }
    if (pip_convex_jud(P[l], a, P[l + 1]))return 0;//在P[pos_(pos+1)]外
    if (sp_on(Line(P[l], P[l + 1]), a))return 2;//在P[pos_(pos+1)]上
    return 1;
}
// 多边形是否包含线段
// 因此我们可以先求出所有和线段相交的多边形的顶点，然后按照X-Y 坐标排序(X 坐标小的排在前面，对于X 坐标相同的点，Y 坐标小的排在前面，
// 这种排序准则也是为了保证水平和垂直情况的判断正确)，这样相邻的两个点就是在线段上相邻的两交点，如果任意相邻两点的中点也在多边形内，
// 则该线段一定在多边形内。

int pp_judge(Point *A, int n, Point *B, int m) {//【判断多边形A 与多边形B 是否相离】
    for (int i1 = 1; i1 <= n; ++i1) {
        int j1 = i1 < n ? i1 + 1 : 1;
        for (int i2 = 1; i2 <= m; ++i2) {
            int j2 = i2 < m ? i2 + 1 : 1;
            Point tmp;
            if (ss_cross(Line(A[i1], A[j1]), Line(B[i2], B[j2]), tmp)) return 0;//两线段相交
            if (pip(B, m, A[i1]) || pip(A, n, B[i2]))return 0;//点包含在内
        }
    }
    return 1;
}

double area(Point *P, int n) {//【任意多边形P 的面积】
    double S = 0;
    for (int i = 1; i <= n; i++) S += det(P[i], P[i < n ? i + 1 : 1]);
    return S / 2.0;
}

Line Q[N];

int judge(Line L, Point a) { return sgn(det(a - L.s, L.t - L.s)) > 0; }//判断点a 是否在直线L 的右边
int halfcut(Line *L, int n, Point *P) {//【半平面交】
    sort(L, L + n, [](const Line &a, const Line &b) {
        double d = atan2((a.t - a.s).y, (a.t - a.s).x) - atan2((b.t - b.s).y, (b.t - b.s).x);
        return sgn(d) ? sgn(d) < 0 : judge(a, b.s);
    });

    int m = n;
    n = 0;
    for (int i = 0; i < m; ++i)
        if (i == 0 || sgn(atan2(Point(L[i]).y, Point(L[i]).x) - atan2(Point(L[i - 1]).y, Point(L[i - 1]).x)))
            L[n++] = L[i];
    int h = 1, t = 0;
    for (int i = 0; i < n; ++i) {
        while (h < t && judge(L[i], ll_intersection(Q[t], Q[t - 1]))) --t;//当队尾两个直线交点不是在直线L[i]上或者左边时就出队
        while (h < t && judge(L[i], ll_intersection(Q[h], Q[h + 1]))) ++h;//当队头两个直线交点不是在直线L[i]上或者左边时就出队
        Q[++t] = L[i];

    }
    while (h < t && judge(Q[h], ll_intersection(Q[t], Q[t - 1]))) --t;
    while (h < t && judge(Q[t], ll_intersection(Q[h], Q[h + 1]))) ++h;
    n = 0;
    for (int i = h; i <= t; ++i) {
        P[n++] = ll_intersection(Q[i], Q[i < t ? i + 1 : h]);
    }
    return n;
}

Point V1[N], V2[N];

int mincowski(Point *P1, int n, Point *P2, int m, Point *V) {//【闵可夫斯基和】求两个凸包{P1},{P2}的向量集合{V}={P1+P2}构成的凸包
    for (int i = 0; i < n; ++i) V1[i] = P1[(i + 1) % n] - P1[i];
    for (int i = 0; i < m; ++i) V2[i] = P2[(i + 1) % m] - P2[i];
    int t = 0, i = 0, j = 0;
    V[t++] = P1[0] + P2[0];
    while (i < n && j < m) V[t] = V[t - 1] + (sgn(det(V1[i], V2[j])) > 0 ? V1[i++] : V2[j++]), t++;
    while (i < n) V[t] = V[t - 1] + V1[i++], t++;
    while (j < m) V[t] = V[t - 1] + V2[j++], t++;
    return t;
}
```

```
circle getcircle(const Point &A, const Point &B, const Point &C) {//【三点确定一圆】向量垂心法
    Point P1 = (A + B) * 0.5, P2 = (A + C) * 0.5;
    Line R1 = Line(P1, P1 + normal(B - A));
    Line R2 = Line(P2, P2 + normal(C - A));
    circle O;
    O.o = ll_intersection(R1, R2);
    O.r = length(A - O.o);
    return O;
}

struct ConvexHull {

    int op;

    struct cmp {
        bool operator()(const Point &a, const Point &b) const {
            return sgn(a.x - b.x) < 0 || sgn(a.x - b.x) == 0 && sgn(a.y - b.y) < 0;
        }
    };

    set<Point, cmp> s;

    ConvexHull(int o) {
        op = o;
        s.clear();
    }

    inline int PIP(Point P) {
        set<Point>::iterator it = s.lower_bound(Point(P.x, -dinf));//找到第一个横坐标大于P的点
        if (it == s.end())return 0;
        if (sgn(it->x - P.x) == 0) return sgn((P.y - it->y) * op) <= 0;//比较纵坐标大小
        if (it == s.begin())return 0;
        set<Point>::iterator j = it, k = it;
        --j;
        return sgn(det(P - *j, *k - *j) * op) >= 0;//看叉姬1
    }

    inline int judge(set<Point>::iterator it) {
        set<Point>::iterator j = it, k = it;
        if (j == s.begin())return 0;
        --j;
        if (++k == s.end())return 0;
        return sgn(det(*it - *j, *k - *j) * op) >= 0;//看叉姬
    }

    inline void insert(Point P) {
        if (PIP(P))return;//如果点P已经在凸壳上或凸包里就不插入了
        set<Point>::iterator tmp = s.lower_bound(Point(P.x, -inf));
        if (tmp != s.end() && sgn(tmp->x - P.x) == 0)s.erase(tmp);//特判横坐标相等的点要去掉
        s.insert(P);
        set<Point>::iterator it = s.find(P), p = it;
        if (p != s.begin()) {
            --p;
            while (judge(p)) {
                set<Point>::iterator temp = p--;
                s.erase(temp);
            }
        }
        if ((p = ++it) != s.end()) {
            while (judge(p)) {
                set<Point>::iterator temp = p++;
                s.erase(temp);
            }
        }
    }
} up(1), down(-1);

int PIC(circle C, Point a) { return sgn(length(a - C.o) - C.r) <= 0; }//判断点A是否在圆C内
void Random(Point *P, int n) { for (int i = 0; i < n; ++i)swap(P[i], P[(rand() + 1) % n]); }//随机一个排列
circle min_circle(Point *P, int n) {//【求点集P的最小覆盖圆】 O(n)
//  random_shuffle(P,P+n);
    Random(P, n);
    circle C = circle(P[0], 0);
    for (int i = 1; i < n; ++i)
        if (!PIC(C, P[i])) {
            C = circle(P[i], 0);
            for (int j = 0; j < i; ++j)
                if (!PIC(C, P[j])) {
                    C.o = (P[i] + P[j]) * 0.5, C.r = length(P[j] - C.o);
                    for (int k = 0; k < j; ++k) if (!PIC(C, P[k])) C = getcircle(P[i], P[j], P[k]);
                }
        }
```

```
    return C;
}
```

高精度

高精度 GCD

```
#include <bits/stdc++.h>
using namespace std;
string add(string a, string b) {
    const int L = 1e5;
    string ans;
    int na[L] = {0}, nb[L] = {0};
    int la = a.size(), lb = b.size();
    for (int i = 0; i < la; i++) na[la - 1 - i] = a[i] - '0';
    for (int i = 0; i < lb; i++) nb[lb - 1 - i] = b[i] - '0';
    int lmax = la > lb ? la : lb;
    for (int i = 0; i < lmax; i++)
        na[i] += nb[i], na[i + 1] += na[i] / 10, na[i] %= 10;
    if (na[lmax]) lmax++;
    for (int i = lmax - 1; i >= 0; i--) ans += na[i] + '0';
    return ans;
}
string mul(string a, string b) {
    const int L = 1e5;
    string s;
    int na[L], nb[L], nc[L],
        La = a.size(), Lb = b.size();  // na 存储被乘数，nb 存储乘数，nc 存
储积
    fill(na, na + L, 0);
    fill(nb, nb + L, 0);
    fill(nc, nc + L, 0);  //将 na,nb,nc 都置为 0
    for (int i = La - 1; i >= 0; i--)
        na[La - i] =
            a[i] - '0';  //将字符串表示的大整形数转成 i 整形数组表示的大整形数
    for (int i = Lb - 1; i >= 0; i--) nb[Lb - i] = b[i] - '0';
    for (int i = 1; i <= La; i++)
        for (int j = 1; j <= Lb; j++)
            nc[i + j - 1] +=
                na[i] *
                nb[j];  // a 的第 i 位乘以 b 的第 j 位为积的第 i+j-1 位（先不考虑
进位）
    for (int i = 1; i <= La + Lb; i++)
        nc[i + 1] += nc[i] / 10, nc[i] %= 10;  //统一处理进位
    if (nc[La + Lb]) s += nc[La + Lb] + '0';  //判断第 i+j 位上的数字是不是
0
    for (int i = La + Lb - 1; i >= 1; i--)
        s += nc[i] + '0';  //将整形数组转成字符串
    return s;
}
int sub(int *a, int *b, int La, int Lb) {
    if (La < Lb) return -1;  //如果 a 小于 b，则返回-1
    if (La == Lb) {
        for (int i = La - 1; i >= 0; i--)
            if (a[i] > b[i])
                break;
            else if (a[i] < b[i])
                return -1;  //如果 a 小于 b，则返回-1
    }
    for (int i = 0; i < La; i++)  //高精度减法
    {
        a[i] -= b[i];
        if (a[i] < 0) a[i] += 10, a[i + 1]--;
    }
    for (int i = La - 1; i >= 0; i--)
        if (a[i]) return i + 1;  //返回差的位数
    return 0;                    //返回差的位数
}
string div(string n1, string n2,
           int nn)  // n1,n2 是字符串表示的被除数，除数,nn 是选择返回商还是余
数
{
    const int L = 1e5;
    string s, v;  // s 存商,v 存余数
    int a[L], b[L], r[L],
        La = n1.size(), Lb = n2.size(), i,
        tp = La;  // a，b 是整形数组表示被除数，除数，tp 保存被除数的长度
    fill(a, a + L, 0);
    fill(b, b + L, 0);
    fill(r, r + L, 0);  //数组元素都置为 0
    for (i = La - 1; i >= 0; i--) a[La - 1 - i] = n1[i] - '0';
    for (i = Lb - 1; i >= 0; i--) b[Lb - 1 - i] = n2[i] - '0';
    if (La < Lb || (La == Lb && n1 < n2)) {
        // cout<<0<<endl;
        return n1;
    }                  //如果 a<b,则商为 0，余数为被除数
```

```
    int t = La - Lb;  //除被数和除数的位数之差
    for (int i = La - 1; i >= 0; i--)  //将除数扩大10^t 倍
        if (i >= t)
            b[i] = b[i - t];
        else
            b[i] = 0;
    Lb = La;
    for (int j = 0; j <= t; j++) {
        int temp;
        while ((temp = sub(a, b + j, La, Lb - j)) >=
               0)  //如果被除数比除数大继续减
        {
            La = temp;
            r[t - j]++;
        }
    }
    for (i = 0; i < L - 10; i++)
        r[i + 1] += r[i] / 10, r[i] %= 10;  //统一处理进位
    while (!r[i]) i--;  //将整形数组表示的商转化成字符串表示的
    while (i >= 0) s += r[i--] + '0';
    // cout<<s<<endl;
    i = tp;
    while (!a[i]) i--;  //将整形数组表示的余数转化成字符串表示的</span>
    while (i >= 0) v += a[i--] + '0';
    if (v.empty()) v = "0";
    // cout<<v<<endl;
    if (nn == 1) return s;
    if (nn == 2) return v;
}
bool judge(string s)  //判断s 是否为全0 串
{
    for (int i = 0; i < s.size(); i++)
        if (s[i] != '0') return false;
    return true;
}
string gcd(string a, string b)  //求最大公约数
{
    string t;
    while (!judge(b))  //如果余数不为0，继续除
    {
        t = a;             //保存被除数的值
        a = b;             //用除数替换被除数
        b = div(t, b, 2);  //用余数替换除数
    }
    return a;
}

//o(无法估计)
```

高精度乘法（FFT）

```
#include <bits/stdc++.h>
using namespace std;
#define L(x) (1 << (x))
const double PI = acos(-1.0);
const int Maxn = 133015;
double ax[Maxn], ay[Maxn], bx[Maxn], by[Maxn];
char sa[Maxn / 2], sb[Maxn / 2];
int sum[Maxn];
int x1[Maxn], x2[Maxn];
int revv(int x, int bits) {
    int ret = 0;
    for (int i = 0; i < bits; i++) {
        ret <<= 1;
        ret |= x & 1;
        x >>= 1;
    }
    return ret;
}
void fft(double* a, double* b, int n, bool rev) {
    int bits = 0;
    while (1 << bits < n) ++bits;
    for (int i = 0; i < n; i++) {
        int j = revv(i, bits);
        if (i < j) swap(a[i], a[j]), swap(b[i], b[j]);
    }
    for (int len = 2; len <= n; len <<= 1) {
        int half = len >> 1;
        double wmx = cos(2 * PI / len), wmy = sin(2 * PI / len);
        if (rev) wmy = -wmy;
        for (int i = 0; i < n; i += len) {
            double wx = 1, wy = 0;
            for (int j = 0; j < half; j++) {
                double cx = a[i + j], cy = b[i + j];
                double dx = a[i + j + half], dy = b[i + j + half];
                double ex = dx * wx - dy * wy, ey = dx * wy + dy * wx;
                a[i + j] = cx + ex, b[i + j] = cy + ey;
                a[i + j + half] = cx - ex, b[i + j + half] = cy - ey;
```

```
                double wnx = wx * wmx - wy * wmy, wny = wx * wmy + wy * wm
x;
                wx = wnx, wy = wny;
            }
        }
    }
    if (rev) {
        for (int i = 0; i < n; i++) a[i] /= n, b[i] /= n;
    }
}
int solve(int a[], int na, int b[], int nb, int ans[]) {
    int len = max(na, nb), ln;
    for (ln = 0; L(ln) < len; ++ln)
        ;
    len = L(++ln);
    for (int i = 0; i < len; ++i) {
        if (i >= na)
            ax[i] = 0, ay[i] = 0;
        else
            ax[i] = a[i], ay[i] = 0;
    }
    fft(ax, ay, len, 0);
    for (int i = 0; i < len; ++i) {
        if (i >= nb)
            bx[i] = 0, by[i] = 0;
        else
            bx[i] = b[i], by[i] = 0;
    }
    fft(bx, by, len, 0);
    for (int i = 0; i < len; ++i) {
        double cx = ax[i] * bx[i] - ay[i] * by[i];
        double cy = ax[i] * by[i] + ay[i] * bx[i];
        ax[i] = cx, ay[i] = cy;
    }
    fft(ax, ay, len, 1);
    for (int i = 0; i < len; ++i) ans[i] = (int)(ax[i] + 0.5);
    return len;
}
string mul(string sa, string sb) {
    int l1, l2, l;
    int i;
    string ans;
    memset(sum, 0, sizeof(sum));
    l1 = sa.size();
    l2 = sb.size();
    for (i = 0; i < l1; i++) x1[i] = sa[l1 - i - 1] - '0';
    for (i = 0; i < l2; i++) x2[i] = sb[l2 - i - 1] - '0';
    l = solve(x1, l1, x2, l2, sum);
    for (i = 0; i < l || sum[i] >= 10; i++)  // 进位
    {
        sum[i + 1] += sum[i] / 10;
        sum[i] %= 10;
    }
    l = i;
    while (sum[l] <= 0 && l > 0) l--;              // 检索最高位
    for (i = l; i >= 0; i--) ans += sum[i] + '0';  // 倒序输出
    return ans;
}
int main() {
    cin.sync_with_stdio(false);
    string a, b;
    while (cin >> a >> b) cout << mul(a, b) << endl;
    return 0;
}

//o(nlogn)
```

高精度乘法（乘单精）

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

//o(n)
```

高精度乘法（朴素）

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

//o(n^2)
```

高精度减法

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

//o(n)
```

高精度加法

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

//o(n)
```

高精度取模（对单精）

```
#include <bits/stdc++.h>
```

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

//o(n)
```

高精度幂

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

```
    string ans;
    memset(sum, 0, sizeof(sum));
    l1 = sa.size();
    l2 = sb.size();
    for (i = 0; i < l1; i++) x1[i] = sa[l1 - i - 1] - '0';
    for (i = 0; i < l2; i++) x2[i] = sb[l2 - i - 1] - '0';
    l = solve(x1, l1, x2, l2, sum);
    for (i = 0; i < l || sum[i] >= 10; i++)  // 进位
    {
        sum[i + 1] += sum[i] / 10;
        sum[i] %= 10;
    }
    l = i;
    while (sum[l] <= 0 && l > 0) l--;               // 检索最高位
    for (i = l; i >= 0; i--) ans += sum[i] + '0';  // 倒序输出
    return ans;
}
string Pow(string a, int n) {
    if (n == 1) return a;
    if (n & 1) return mul(Pow(a, n - 1), a);
    string ans = Pow(a, n / 2);
    return mul(ans, ans);
}

//o(nlognlogm)
```

高精度平方根

```
#include <bits/stdc++.h>
using namespace std;
const int L = 2015;
string add(string a, string b)  //只限两个非负整数相加
{
    string ans;
    int na[L] = {0}, nb[L] = {0};
    int la = a.size(), lb = b.size();
    for (int i = 0; i < la; i++) na[la - 1 - i] = a[i] - '0';
    for (int i = 0; i < lb; i++) nb[lb - 1 - i] = b[i] - '0';
    int lmax = la > lb ? la : lb;
    for (int i = 0; i < lmax; i++)
        na[i] += nb[i], na[i + 1] += na[i] / 10, na[i] %= 10;
    if (na[lmax]) lmax++;
    for (int i = lmax - 1; i >= 0; i--) ans += na[i] + '0';
    return ans;
}
string sub(string a, string b)  //只限大的非负整数减小的非负整数
{
    string ans;
    int na[L] = {0}, nb[L] = {0};
    int la = a.size(), lb = b.size();
    for (int i = 0; i < la; i++) na[la - 1 - i] = a[i] - '0';
    for (int i = 0; i < lb; i++) nb[lb - 1 - i] = b[i] - '0';
    int lmax = la > lb ? la : lb;
    for (int i = 0; i < lmax; i++) {
        na[i] -= nb[i];
        if (na[i] < 0) na[i] += 10, na[i + 1]--;
    }
    while (!na[--lmax] && lmax > 0)
        ;
    lmax++;
    for (int i = lmax - 1; i >= 0; i--) ans += na[i] + '0';
    return ans;
}
string mul(string a, string b)  //高精度乘法a,b,均为非负整数
{
    string s;
    int na[L], nb[L], nc[L],
        La = a.size(), Lb = b.size();  // na 存储被乘数, nb 存储乘数, nc 存
储积
    fill(na, na + L, 0);
    fill(nb, nb + L, 0);
    fill(nc, nc + L, 0);  //将na,nb,nc 都置为0
    for (int i = La - 1; i >= 0; i--)
        na[La - i] =
            a[i] - '0';  //将字符串表示的大整形数转成i 整形数组表示的大整形数
    for (int i = Lb - 1; i >= 0; i--) nb[Lb - i] = b[i] - '0';
    for (int i = 1; i <= La; i++)
        for (int j = 1; j <= Lb; j++)
            nc[i + j - 1] +=
                na[i] *
                nb[j];  // a 的第i 位乘以b 的第j 位为积的第i+j-1 位（先不考虑
进位）
    for (int i = 1; i <= La + Lb; i++)
        nc[i + 1] += nc[i] / 10, nc[i] %= 10;  //统一处理进位
    if (nc[La + Lb]) s += nc[La + Lb] + '0';  //判断第i+j 位上的数字是不是
0
    for (int i = La + Lb - 1; i >= 1; i--)
        s += nc[i] + '0';  //将整形数组转成字符串
```

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

string BigInterSqrt(string n) {
    n = DeletePreZero(n);
    string l = "1", r = n, mid, ans;
    while (cmp(l, r)) {
        mid = div(add(l, r), "2", 1);
        if (cmp(mul(mid, mid), n))
            ans = mid, l = add(mid, "1");
        else
            r = sub(mid, "1");
    }
    return ans;
```

```
}

// o(n^3)
```

高精度进制转换

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

//o(n^2)
```

高精度阶乘

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

//o(n^2)
```

高精度除法（除单精）

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

```
    return r.substr(p);
}

//o(n)
```

高精度除法（除高精）

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

//o(n^2)
```

龟速乘快速幂（快速幂爆 longlong

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

ll qmul(ll a, ll b, ll p) {
        ll res = 0;
        while(b) {
                if(b & 1) res = (res + a) % p;
                a = (a + a) % p;
```

```
			b >>= 1;
		}
		return res;
}

ll qpow(ll x, ll n, ll p) {
		ll res = 1;
		while(n) {
			if(n & 1) res = qmul(res, x, p);
			x = qmul(x, x, p);
			n >>= 1;
		}
		return res % p; // 1 0 1
}

int main() {
		ll b, p, k;
		cin >> b >> p >> k;
		ll ans = qpow(b, p, k);
		printf("%lld^%lld mod %lld=%lld", b, p, k, ans);

		return 0;
}
```