Standard Code Library

AC Mountains

WUST

December 9102

Contents

1 一切的开始

1.1 宏定义

by 杜教

```cpp
#include <bits/stdc++.h>
using namespace std;
#define rep(i,a,n) for (int i=a;i<n;i++)
#define per(i,a,n) for (int i=n-1;i>=a;i--)
#define pb push_back
#define mp make_pair
#define all(x) (x).begin(),(x).end()
#define fi first
#define se second
#define SZ(x) ((int)(x).size())
#define de(c) cout << #c << " = " << c << endl
#define dd(c) cout << #c << " = " << c << " "
typedef vector<int> VI;
typedef long long ll;
typedef pair<int,int> PII;
mt19937 mrand(random_device{}());
const ll mod=998244353;
int rnd(int x) { return mrand() % x;}
ll powmod(ll a,ll b) {ll res=1;a%=mod; assert(b>=0); for(;b;b>>=1){if(b&1)res=res
    *a%mod;a=a*a%mod;}return res;}
ll gcd(ll a,ll b) { return b?gcd(b,a%b):a;}

int main()
{
#ifdef indiewar
   freopen("in.txt","r",stdin);
   freopen("out.txt","w",stdout);
#endif
}
```

- CMakeLists.txt (for CLion)

```
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -O2 -Dindiewar -Wall")
```

```
project(icpc)

set(CMAKE_CXX_STANDARD 11)

add_executable(icpc main.cpp)
```

- HDU Assert Patch

```cpp
#ifdef ONLINE_JUDGE
#define assert(condition) if (!(condition)) { int x = 1, y = 0; cout << x / y <<
    endl; }
#endif
```

1.2 快速读

```cpp
inline char nc() {
    static char buf[100000], *p1 = buf, *p2 = buf;
    return p1 == p2 && (p2 = (p1 = buf) + fread(buf, 1, 100000, stdin), p1 == p2)
         ? EOF : *p1++;
}
template <typename T>
bool rn(T& v) {
    static char ch;
    while (ch != EOF && !isdigit(ch)) ch = nc();
    if (ch == EOF) return false;
    for (v = 0; isdigit(ch); ch = nc())
        v = v * 10 + ch - '0';
    return true;
}

template <typename T>
void o(T p) {
    static int stk[70], tp;
    if (p == 0) { putchar('0'); return; }
    if (p < 0) { p = -p; putchar('-'); }
    while (p) stk[++tp] = p % 10, p /= 10;
    while (tp) putchar(stk[tp--] + '0');
}
```

- 需要初始化
- 需要一次读入
- 不支持负数

```cpp
const int MAXS = 100 * 1024 * 1024;
char buf[MAXS];
template<typename T>
inline bool read(T& x) {
    static char* p = buf;
    x = 0;
    while (*p && !isdigit(*p)) ++p;
    if (!*p) return false;
    while (isdigit(*p)) x = x * 10 + *p++ - 48;
    return true;
}

fread(buf, 1, MAXS, stdin);
```

1.3 对拍

```bash
#!/usr/bin/env bash
g++ -o r main.cpp -O2 -std=c++11
g++ -o std std.cpp -O2 -std=c++11
while true; do
    python gen.py > in
    ./std < in > stdout
    ./r < in > out
```

```
    if test $? -ne 0; then
        exit 0
    fi
    if diff stdout out; then
        printf "AC\n"
    else
        printf "GG\n"
        exit 0
    fi
done
```

- 快速编译运行 train.sh

```
#!/bin/bash
g++ $1.cpp -o $1 -O2 -std=c++11 -Dindiewar
if [ $? -eq 0 ]; then
    ./$1
fi
```

2 数据结构

2.1 BIT

```
struct Bit
{
    vector<int> a;
    int sz;
    void init(int n)
    {
        sz=n+5;
        for(int i=1;i<=n+5;i++)
            a.push_back(0);
    }
    int lowbit(int x)
    {
        return x&(-x);
    }
    int query(int x)
    {
        int ans = 0;
        for(;x;x-=lowbit(x))ans+=a[x];
        return ans;
    }
    void update(int x,int v)
    {
        for(;x<sz;x+=lowbit(x))
            a[x]+=v;
    }
}bit;
```

2.2 主席树

2.2.1 中位数

有一棵树，n 个节点，每个节点有个能力值 A[i],mid[i]，mid 的值为第 i 节点的子树的中位数（包括本身），现在让你将其中的一个节点的 A 值改为 1e5, 问所有的 mid 的和最大问多少。用主席树来支持查询中位数，然后预处理出每个节点的 mid 的值和 mid+1 的值。

最后用树状数组 +dfs 维护一个最大的差值。

```
const int maxn = 1e5+10;

int n;
VI e[maxn];
int needval[maxn],addval[maxn];
ll allval;
ll ans;


int a[maxn],rt[maxn];
int cnt;
int lson[maxn << 5],rson[maxn << 5],sum[maxn << 5];
#define Lson l,m,lson[x],lson[y]
#define Rson m+1,r,rson[x],rson[y]
void upd(int p,int l,int r,int &x,int y)
{
    lson[++cnt] = lson[y],rson[cnt] = rson[y],sum[cnt] = sum[y]+1,x=cnt;
    // de(x);
    if(l == r) return;
    int m = (l+r) >> 1;
    if(p <= m) upd(p,Lson);
    else upd(p,Rson);
}

int query(int l,int r,int x,int y,int k)
{
    if(l == r) return l;
    int m = (l+r) >> 1;
    int s = sum[lson[y]] - sum[lson[x]];
    if(s >= k) return query(Lson,k);
    else return query(Rson,k-s);
}


struct Bit
{
    ll a[maxn];
    int sz;
    void init()
    {
        sz=maxn;
        for(int i=1;i<maxn;i++)
            a[i] = 0;
    }
    int lowbit(int x)
```

```
    {
        return x&(-x);
    }
    ll query(int x)
    {
        ll ans = 0;
        for(;x;x-=lowbit(x))ans+=a[x];
        return ans;
    }
    void update(int x,int v)
    {
        for(;x<sz;x+=lowbit(x))
            a[x]+=v;
    }
}bit;

int in[maxn],ot[maxn],sz[maxn],idx;

void init()
{
    ans = allval = idx = cnt = 0;
    bit.init();
    rep(i,1,n+1)
    {
        e[i].clear();
    }
}


void dfs(int u)
{
    in[u] = ++idx;
    upd(a[u],1,100000,rt[idx],rt[idx-1]);
    sz[u] = 1;
    for(auto &v:e[u])
    {
        dfs(v);
        sz[u] += sz[v];
    }
    ot[u] = idx;
}

void gao(int u)
{
    bit.update(100001-needval[u],addval[u]);
    ans = max(ans,bit.query(100001-a[u]));
    for(auto &v:e[u])
    {
        gao(v);
    }
    bit.update(100001-needval[u],-addval[u]);
}


int main(int argc, char const *argv[])
{
    while(scanf("%d",&n)!=EOF)
```

```
    {
        init();
        rep(i,1,n+1)
        {
            scanf("%d",&a[i]);
        }
        rep(i,2,n+1)
        {
            int p;
            scanf("%d",&p);
            e[p].pb(i);
        }
        dfs(1);
        rep(i,1,n+1)
        {
            if(sz[i] == 1)
            {
                needval[i] = a[i];
                addval[i] = 100000-a[i];
                allval += a[i];
            }
            else
            {
                int mid = (sz[i] + 1) >> 1;
                needval[i] = query(1,100000,rt[in[i]-1],rt[ot[i]],mid);
                addval[i] = query(1,100000,rt[in[i]-1],rt[ot[i]],mid+1)-needval[i];
                allval += needval[i];
                // de(mid);
            }


        }
        gao(1);
        //de(ans);
        printf("%lld\n",ans+allval);
    }
    return 0;
}
```

2.2.2 二分

给出 n 条直线，每次查询给出一个 x，求这个 x 位置上方的前 k 个线段的高度之和。

首先对高度进行离散化建立主席树，将每条直线对应为两个点，左端点 +1, 右端点-1，然后每个点建一个线段树，然后每次查询对于一个 x，只需要二分查找找到时序点，然后找到线段数量为 k 的最左的前缀即可。

```
const int maxn = 1e5+100;

VI v;
int n,M,X,P;
int a[maxn],rt[maxn];
int cnt;
```

```
int lson[maxn << 5],rson[maxn<< 5],scnt[maxn<< 5];
ll sum[maxn<< 5];
#define Lson l,m,lson[x],lson[y]
#define Rson m+1,r,rson[x],rson[y]

void upd(int p,int l,int r,int& x,int y,int add)
{
    lson[++cnt] = lson[y],rson[cnt] = rson[y],scnt[cnt] = scnt[y] + add,sum[cnt]
         = sum[y] + add * v[p-1],x = cnt;
    if(l == r) return;
    int m = (l+r)>>1;
    if(p <= m) upd(p,Lson,add);
    else upd(p,Rson,add);
}

ll query(int l,int r,int x,int y,int k)
{
    if(l == r)
    {
        return (ll)min(k,scnt[y])*v[l-1];
    }
    int m = (l+r) >> 1;
    int s = scnt[lson[y]] - scnt[lson[x]];
    // de(s);
    if(s >= k) return query(Lson,k);
    else
        return query(Rson,k-s) + sum[lson[y]]-sum[lson[x]];
}

int getid(int x){return lower_bound(all(v),x)-v.begin()+1;}

vector<PII> e[maxn];
void init()
{
    rt[0] = lson[0] = rson[0] = sum[0] = scnt[0] = 0;
    cnt = 0;
    rep(i,0,X+1)
    {
        e[i].clear();
    }
    v.clear();
}

int main(int argc, char const *argv[])
{
    while(scanf("%d%d%d%d",&n,&M,&X,&P)!=EOF)
    {
        init();
        int l,r,d;
        rep(i,0,n)
        {
            scanf("%d%d%d",&l,&r,&d);
            v.pb(d);
            e[l].pb({d,1});
            e[r+1].pb({d,-1});
        }
        sort(all(v));
```

```

        v.erase(unique(all(v)),v.end());
        int tot = v.size();
        rep(i,1,X+1)
        {
            rt[i] = rt[i-1];
            int last = rt[i-1];
            for(auto &x:e[i])
            {
                int pos = getid(x.fi);
                // de(pos);
                upd(pos,1,tot,rt[i],last,x.se);
                last = rt[i];
                // de(rt[i]);
            }
        }
        ll ans = 1;
        ll preans;
        int x;
        ll a,b,c;
        ll k;
        rep(i,0,M)
        {
            scanf("%d%lld%lld%lld",&x,&a,&b,&c);
            preans = ans;
            k = (a%c*preans%c + b) % c;
            ans = query(1,tot,0,rt[x],k);
            if(i > 0 && preans > P) ans *= 2;
            printf("%lld\n",ans);
        }
    }
    return 0;
}
```

2.3 权值线段树

2.3.1 求第 K 大

给你一组序列的逆序数前缀和，让你去还原原来的序列逆序数统计的是前面大于自己的数字个数。我们可以通过做差的方式 a[i]-a[i-1] 求出每一个数的逆序数个数。对于第 i 个数来说，假如逆序数是 x，那么他就是前 i 中第 x 大（更新之后的第 x 大）。那么我们从后往前找，每次找区间内的第 x 大，求出来之后我们必须将那个节点的值更新为 0。

```
const int maxn = 5e4+100;
int sum[maxn << 2];

void push_up(int p)
{
    sum[p] = sum[p<<1] + sum[p<<1|1];
}

#define lson l,m,p<<1
```

```
#define rson m+1,r,p<<1|1
void build(int l,int r,int p)
{
    sum[p] = r-l+1;
    if(l == r) return ;
    int m = (l+r) >> 1;
    build(lson),build(rson);
}

int query(int l,int r,int p,int pos)
{
    if(l == r)
    {
        sum[p] = 0;
        return l;
    }
    int m = (l+r) >> 1,res;
    if(sum[p<<1|1] >= pos) res = query(rson,pos);
    else res = query(lson,pos-sum[p<<1|1]);
    push_up(p);
    return res;
}

int T,n;
int a[maxn],ans[maxn];
int main(int argc, char const *argv[])
{
    scanf("%d",&T);
    while(T--)
    {
        scanf("%d",&n);
        build(1,n,1);
        a[0] = 0;
        rep(i,1,n+1) scanf("%d",&a[i]);
        per(i,1,n+1) ans[i] = query(1,n,1,a[i] - a[i-1] + 1);
        rep(i,1,n+1)
        {
            if(i == n)
            printf("%d\n",ans[i] );
            else
            printf("%d ",ans[i] );
        }
    }
    return 0;
}
```

2.3.2 区间第一个大于 r 的位置

题目大意：给出一个 n 个元素的数组 A，A 中所有元素都是不重复的 [1,n]。有两种操作：1. 将 pos 位置的元素 +1e7 2. 查询不属于 [1,r] 中的最小的 >=k 的值。

考虑建权值线段树，那么线段树存的值就是它的坐标，考虑到答案一定存在且最大值为 $n+1$, 可以多加一个点 $n+1$

对于操作 1 $(1,x)$，直接单点修改位置 $s[x]$ 为 $n+1$(s[x] 为初始位置 x 的值)

对于操作 2 $(2,r,x)$，要查询区间 $[x,n+1]$ 内第一个大于 r 的位置，考虑建一棵维护最大值的线段树，查询时先访问左子树，假如左子树不行才访问右子树，但是这个时候的复杂度是 $log(n)*log(n)$，有用的剪枝是当查询的区间包含现在的区间时，可以直接判断符不符合情况，这样可以把多的 $log(n)$ 变成常数

```
int query(int p,int l,int r,int x,int y,int tmp)//query(1,1,n+1,x,n+1,r)
   {
      if(l == r)//查询到叶子节点
      {
         if(tr[p] > tmp)
         {
            return l;
         }
         else
         {
            return -1;
         }
      }
      if(x <= l && r <= y)//区间包含于查询区间
      {
         if(tr[p] <= tmp)
         {
            return -1;
         }
      }
      int mid = (l+r)/2;
      if(y <= mid)
      {
         return query(p*2,l,mid,x,y,tmp);
      }
      if(x >= mid + 1)
      {
         return query(p*2+1,mid+1,r,x,y,tmp);
      }

      int judge = query(p*2,l,mid,x,y,tmp);//先判断左子树

      if(judge != -1)
      {
         return judge;
      }

      return query(p*2+1,mid+1,r,x,y,tmp);

   }
```

2.4 莫队

2.4.1 bitset+ 莫队

给你一个序列 a，长度为 n，有 m 次操作，每次询问一个区间是否可以选出两个数它们的差为 x，或者询问一个区间是否可以选出两个数它们的和为 x，或者询问一个区间是否可以选出两个数它们的乘积为 x ，这三个操作分别为操作 1,2,3

对于操作 3，我们可以很容易通过枚举约数得到。离线操作，考虑莫队对于操作 1，询问实际上问的就是区间是否存在 a 和 x-a，因为数据范围较小，用 bitset 维护每个值对于操作 2，考虑一个较大值 $maxn$, 根据 $(maxn-i)-(maxn-k)=(k-i)$，维护 $maxn-a[i]$ 即可

```
const int maxn = 1e5+50;

bitset<maxn+50> s1,s2;
struct query { int L, R, id,op,val; } q[maxn];
int ans[maxn+50];
int a[maxn+50];
int cnt[maxn+50];
int n,m,unit;

void add(int x)
{
   cnt[a[x]]++;
   if(cnt[a[x]] == 0)
   s1[a[x]] = s2[maxn-a[x]] = cnt[a[x]];
   else
   s1[a[x]] = s2[maxn-a[x]] = 1;
}

void del(int x)
{
   cnt[a[x]]--;
   if(cnt[a[x]] == 0)
   s1[a[x]] = s2[maxn-a[x]] = cnt[a[x]];
   else
   s1[a[x]] = s2[maxn-a[x]] = 1;
}


void solve(query node[], int m)
{
   memset(ans, 0, sizeof(ans));
   s1.reset();
   s2.reset();
   memset(cnt, 0, sizeof(cnt));
   sort(node+1, node + m+1, [](query a, query b) {
      if(a.L / unit == b.L / unit)
      {
         return a.R < b.R;
      }
      return a.L < b.L;
   });
```

```
    // rep(i,1,m+1)
    // {
    //    dd(node[i].L),dd(node[i].R),dd(node[i].id),dd(node[i].op),de(node[i].val);
    // }
    int L = 1, R = 0;
    for (int i = 1; i < m+1; i++)
    {
        while (node[i].L > L) del(L++);
        while (node[i].L < L) add(--L);
        while (node[i].R > R) add(++R);
        while (node[i].R < R) del(R--);
        // dd(L),de(R);
        if(node[i].op == 1)
        {
            ans[node[i].id] = (s1&(s1<<node[i].val)).any();
        }
        else if(node[i].op == 2)
        {
            ans[node[i].id] = (s1&(s2>>(maxn-node[i].val))).any();
        }
        else
        {
            for(int j=1;j*j<=node[i].val;j++)
            if(node[i].val % j ==0)
            {
                // de(s1[4]);
                if(s1[j] && s1[node[i].val/j])
                    {
                        ans[node[i].id]=1;
                    }
            }
        }
        // ans[node[i].id] = tmp;
    }
}

int main(int argc, char const *argv[])
{
    ios_base::sync_with_stdio(false), cin.tie(0), cout.tie(0);
    cin >> n >> m;
    unit = int(sqrt(n)+0.5);
    rep(i,1,n+1)
    {
        cin >> a[i];
    }
    int op,l,r,x;
    rep(i,1,m+1)
    {
        cin >> op >> l >> r >> x;
        q[i].op = op;
        q[i].L = l;
        q[i].R = r;
        q[i].val = x;
        q[i].id = i;
    }
    solve(q,m);
```

```
    rep(i,1,m+1)
    {
        if(ans[i])
        {
            cout << "hana"<<endl;
        }
        else
        {
            cout << "bi" << endl;
        }
    }
    return 0;
}
```

3 图论

3.1 最短路

3.1.1 floyd

```
for (int k = 1; k <= n; k++) {
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
      f[i][j] = min(f[i][j], f[i][k] + f[k][j]);
    }
  }
}
```

- 找最小环

```
int val[maxn + 1][maxn + 1]; // 原图的邻接矩阵
int floyd(const int &n) {
  static int dis[maxn + 1][maxn + 1]; // 最短路矩阵
  for (int i = 1; i <= n; ++i)
    for (int j = 1; j <= n; ++j) dis[i][j] = val[i][j]; // 初始化最短路矩阵
  int ans = inf;
  for (int k = 1; k <= n; ++k) {
    for (int i = 1; i < k; ++i)
      for (int j = 1; j < i; ++j)
        ans = std::min(ans, dis[i][j] + val[i][k] + val[k][j]); // 更新答案
    for (int i = 1; i <= n; ++i)
      for (int j = 1; j <= n; ++j)
        dis[i][j] = std::min(
            dis[i][j], dis[i][k] + dis[k][j]); // 正常的 floyd 更新最短路矩阵
  }
  return ans;
}

template<typename T>
struct Floyd {
  static const int N = 1000 + 10;
```

```cpp
  std::vector<int> path;
  T dis[N][N], res;
  int src[N][N];
  // 传入结点个数n及权值矩阵graph[][]，返回最小环的长度res，方案记在path中
  // 对于矩阵graph[][]中不存在的边，权值设为1e9+7或0x7F7F7F7F之类的极大值
  // 若最后的返回值大于等于1e9，则不存在最小环
  T run(int n, const T graph[N][N]) {
    res = 1e9 + 7;
    for (int i = 0; i < n; ++i) {
      for (int j = 0; j < n; ++j) {
        src[i][j] = -1;
        dis[i][j] = graph[i][j];
      }
    }
    for (int k = 0; k < n; ++k) {
      for (int i = 0; i < k; ++i) {
        for (int j = i + 1; j < k; ++j) {
          T tmp = graph[k][i] + graph[j][k];
          if (dis[i][j] > res - tmp) continue;
          path.clear();
          get_path(i, j);
          path.push_back(k);
          path.push_back(i);
          res = tmp + dis[i][j];
        }
      }
      for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
          T tmp = dis[i][k] + dis[k][j];
          if (tmp < dis[i][j]) {
            dis[i][j] = tmp;
            src[i][j] = k;
          }
        }
      }
    }
    return res;
  }
  void get_path(int i, int j) {
    int k = src[i][j];
    if (k != -1) {
      get_path(i, k);
      get_path(k, j);
    } else {
      path.push_back(j);
    }
  }
};
```

- 利用 floyd 的 dp 思路求解

```cpp
int dp[maxn][maxn][maxn];
int w[maxn];
int s[maxn];
bool cmp(int a,int b)
{
   return w[a] < w[b];
```

```
}
rep(i,1,n+1)
{
    rep(j,1,n+1)
    {
       scanf("%d",&dp[i][j][0]);
        rep(k,1,n+1)
        {
            dp[i][j][k] = 1e9;
        }
     }
    s[i] = i;
}
sort(s+1,s+n+1,cmp);
rep(k,1,n+1)
{
    rep(i,1,n+1)
    {
        rep(j,1,n+1)
        {
            dp[i][j][k] = min(dp[i][j][k-1],dp[i][s[k]][k-1]+dp[s[k]][j][k-1]);
        }
    }
}
```

- 传递闭包已知一个有向图中任意两点之间是否有连边，要求判断任意两点是否连通。

```
for (int k = 1; k <= n; k++)
  for (int i = 1; i <= n; i++)
    if (f[i][k]) f[i] = f[i] & f[k];
```

3.1.2 Bellman-ford

每次拿每条边去更新最短路（负权边）

```
for(int k = 1 ; k <= n - 1 ; k ++)
{
    for(int i = 1 ; i < m ; i ++)
    {
        if(dis[v[i]] > dis[u[i]] + w[i])
            dis[v[i]] = dis[u[i]] + w[i] ;
    }
}
```

3.1.3 SPFA

一条边 (u,v)，在 dis[u] 没变的情况下不应该重新更新用队列记录要进行扩展的点，每次 dis[x] 被更新后就把 x 扔进队列里去更新其他点的最短路

```
void spfa(int s){
   for(int i=0; i<=n; i++)
      dis[i]=99999999; //初始化每点i到s的距离
   dis[s]=0; vis[s]=1; q[1]=s; 队列初始化,s为起点
   int i, v, head=0, tail=1;
   while (head<tail){ 队列非空
      head++;
      v=q[head]; 取队首元素
      vis[v]=0; 释放队首结点，因为这节点可能下次用来松弛其它节点，重新入队
      for(i=0; i<=n; i++) 对所有顶点
        if (a[v][i]>0 && dis[i]>dis[v]+a[v][i]){
            dis[i] = dis[v]+a[v][i]; 修改最短路
            if (vis[i]==0){ 如果扩展结点i不在队列中，入队
               tail++;
               q[tail]=i;
               vis[i]=1;
            }
        }

   }
}
```

-SPFA 判断负环一个有向图有负环等价于某个点的最短路长度 >n 方法 1：某个点入队次数 >n 时就有负环方法 2：记录 len[x]，表示 1 到 x 的最短路的点的个数，len[x]>=n 时存在负环时间复杂度：O(nm)

3.1.4 Dijkstra

Dijkstra 算法适用于边权为正的无向和有向图，不适用于有负边权的图根据初始点，挨个的把离初始点最近的点一个一个找到并加入集合，集合中所有的点的 d[i] 都是该点到初始点最短路径长度，由于后加入的点是根据集合 S 中的点为基础拓展的，所以也能找到最短路径

```
void dijkstra(int s) {
  priority_queue<DistNode> Q;
  Q.push_back(DistNode{0, s});
  for(int i = 1; i <= n; i++) d[i] = INF;//把d[i]都设置为最大值
  d[s] = 0;
  memset(done, 0, sizeof(done));
  while(!Q.empty()) {
    DistNode x = Q.top(); Q.pop();
    int u = x.u;
    if(done[u]) continue;//如果这个点已经被提出过了，直接抛弃（适用于那种松弛之后重复
         放入队列的点）
    done[u] = true;
    for(int i = 0; i < Q[u].size(); i++)
      Edge& e = edges(Q[u][i]);
      if(e.dist + d[u] < d[e.to] && d[u] < INF) {
        d[e.to] = e.dist + d[u];
        p[e.to] = G[u][i];
        Q.push_back(Edge(d[e.to], e.to));//把松弛过后点的d值重新放入队列
      }
```

```
    }
  }
}
```

3.1.5 差分约束

假设对于一张图来说, 1 号点到 x 的最短路长度为 d[x] 则有: d[y]<=d[x]+w[x][y] 最短路问题可以给出这一类不等式的最大解，取个反可以得到最小解 -例题给定 n,m 和 m 个三元组 (l,r,k)，求一个 01 串 s，使得 s[l...r] 中 1 的个数至少有 k 个，求 1 最少的满足条件的 01 串 n,m<=10000 设 f[i] 为 s[1...i] 中 1 的个数 f[i]<=f[i-1]+1；f[i-1]<=f[i]；f[l-1]<=f[r]-k 等价于求该不等式组中 f[n] 的最小解

3.2 次短路

给定一张有向带正权拓扑图，求有几条 1 到 N 的路径的长度 <=1 到 N 的最短路 +K N,M<=10^{5}，K<=100，边权 <=10^{9} 定义一条路径 (X...Y) 的冗余度为它的长度减去 X...Y 的最短路长度则题目要求的是 1...N 冗余度不超过 K 的路径长度考虑路径 (1,a,b,N) 冗余度为 w(1,a)+w(a,b)+w(b,N)-d(1,N) =[w(1,a)+d(a,N)-d(1,N)]+w(a,b)+w(b,N)-d(a,N) 设 p(x,y)=w(x,y)+d(y,N)-d(x,N)，则冗余度变成了 a...N 的冗余度加上 p(1,a) 设 F(X,L) 表示 1 到 X 有几条路径满足 p 的和为 L F(X,L)=sum(F(Y,L-P(Y,X))) 时间复杂度：O(mlogn+mk+n)

```cpp
#include <cstdio>
#include <queue>
#include <vector>
#include <cstring>
using namespace std;
int t,d[100001],f[100001][51],n,m,k,p;
bool working[100001][51];
struct ac{
   int to,len;
};
vector<ac>head[100001];
vector<ac>h[100001];
int dp(int root, int l){
   int ans=0;
   if (l<0||l> k) return 0;
   if (working[root][l]) {
      working[root][l]=false;
      return -1;
   }
   if(f[root][l]!=-1)
      return f[root][l];
   working[root][l]=true;
   for (int i=0;i<h[root].size();i++) {
      ac e= h[root][i];
      int val=dp(e.to,d[root]+l-d[e.to]-e.len);
      if (val==-1) {
```

```
            working[root][l]=false;
            return -1;
        }
        ans=(ans+val)%p;
    }
    working[root][l] = false;
    if (root==1&&l==0) ans++;
    f[root][l]=ans;
    return ans;
}
int main(){
    scanf("%d",&t);
    while(t--){
        scanf("%d%d%d%d",&n,&m,&k,&p);
        for(int i=1;i<=n;i++){
            head[i].clear();
            h[i].clear();
        }
        int a,b,c;
        for(int i=1;i<=m;i++){
        scanf("%d%d%d",&a,&b,&c);
            ac e;
            e.to=b;
            e.len=c;
            head[a].push_back(e);
            e.to=a;
            h[b].push_back(e);
        }
        memset(d,0x3f,sizeof(d));
        memset(f,-1,sizeof(f));
        queue<int>q;
        q.push(1);
        d[1]=0;
        while(!q.empty()){
            int x=q.front();
            q.pop();
            for(int i=0;i<head[x].size();i++){
                if(d[head[x][i].to]>d[x]+head[x][i].len){
                    d[head[x][i].to]=d[x]+head[x][i].len;
                    q.push(head[x][i].to);
                }
            }
        }
        int ans=0,flag=1;
        for(int i=0;i<=k;i++){
            int val=dp(n,i);
            if(val==-1){
                flag=0;
                break;
            }
            ans=(ans+val)%p;
        }
        if(flag)
            printf("%d\n",ans);
        else
            printf("-1\n");
    }
```

```
    return 0;
}
```

3.3 图的连通性问题

3.3.1 维护连通性

对于每个连通块维护一棵有根树，F(x) 表示点 x 的父亲则假设我们要添加一条边 U,V，首先求出 U,V 所在连通块的有根树树根 X,Y，然后令 F(X)=Y GetRoot(X)：若 F(X)=0 则 return X，否则 return F(X)=GetRoot(F(X))//并查集思想

3.3.2 桥

对于一个连通无向图，定义一条边 (u,v) 是桥，当且仅当断开这条边后图变得不连通图的大致形状：O-O，则中间的边就是桥强连通分量：没有桥的连通块

3.3.2.1 Tarjan 算法

对整个图进行 dfs，设 dfn[x] 表示点 x 是第几个被搜到的 low[x] 表示：x 通过非返祖边，且至多通过一条非树边能到达的最小 dfn

```
void tarjan(int now){
    dfn[now]=low[now]=++cnt;
    stack[++t]=now;
    v[now]=1;
    for(int i=f[now];i!=-1;i=e[i].next)
        if(!dfn[e[i].v]){
            tarjan(e[i].v);
            low[now]=min(low[now],low[e[i].v]);
        }
        else if(v[e[i].v])
            low[now]=min(low[now],dfn[e[i].v]);
    if(dfn[now]==low[now]){
        int cur;
        do{
            cur=stack[t--];
            v[cur]=false;
        }while(now!=cur);
    }
}
```

```
struct Edge{
    int nxt,to;
#define nxt(x) e[x].nxt
#define to(x) e[x].to
}e[N<<1];
int dfn[N],low[N],tot=1;//储存边的编号，由于要用^1找反向边，从1开始
int bridge[N],head[N];
```

```cpp
int n,m,cnt;
void addedge(int f,int t){
    nxt(++tot)=head[f];to(tot)=t;head[f]=tot;
}
void tarjan(int x,int in_edge){//in_edge表示递归进入每个节点的边的编号
    dfn[x]=low[x]=++cnt;
    for(int i=head[x];i;i=nxt(i)){
        int y=to(i);
        if(!dfn[y]){
            tarjan(y,i);
            low[x]=min(low[x],low[y]);//在搜索树上的边
            if(low[y]>dfn[x])//桥判定法则
                bridge[i]=bridge[i^1]=1;//这条边和它的反向边都是桥
        }else if(i!=(in_edge^1))
            low[x]=min(low[x],dfn[y]);//不在搜索树上的边
    }
}
void run(int n){
    for(int i=1;i<=n;i++)
        if(!dfn[i])tarjan(i,0);
}
int main(){
    scanf("%d%d",&n,&m);
    for(int i=1;i<=m;i++){
        scanf("%d%d",&x,&y);
        addedge(x,y);addedge(y,x);
    }
```

-Tarjan 算法求桥非树边一定不是桥对于树边 (U,V)，它是桥等价于 low[V]>dfn[U] 去掉所有的桥后，剩下的每个连通分量都是强连通分量 -有向图的强连通分量在做 Tarjan 算法时，如果 tarjan(x) 后发现 dfn[x]=low[x]，则 x 的子树里的剩下的所有点构成一个强连通分量

有 N 个人，给你 M 对整数 (a,b)，表示第 a 个人认为 b 很厉害而这种关系具备传递性，也就是如果 a 认为 b 厉害，且 b 认为 c 厉害，则 a 认为 c 厉害求有多少人被所有人都觉得很厉害 N,M<=10^5 等价于求有几个点，使得他们能到达所有点可以发现每个强连通分量中的人都是互相认为对方很厉害的，所以可以把它们看成一个人求出强连通分量并缩点，之后得到一个无环图就能做了

给定一个有向图，N 个点，求：1) 至少要选几个顶点，才能做到从这些顶点出发，可以到达全部顶点 2) 至少要加多少条边，才能使得从任何一个顶点出发，都能到达全部顶点按照套路，Tarjan 算法求 SCC，并缩点建图。那么对于问题 1，新的图中入度为 0 点的点数即是答案。对于问题 2，答案为 max（入度为 0 的点的点数，出度为 0 的点数），因为对于每个入度或出度为 0 的点，需要连一条边来解决，那么将出度为 0 的点连向入度为 0 的点是最优的。

```cpp
#include<stdio.h>
#include<iostream>
#include<string.h>
#include<vector>
using namespace std;
#define maxn 105
struct edge
{
```

```cpp
    int to,nxt;
}e[maxn*maxn];
int head[maxn],low[maxn],dfn[maxn],sta[maxn],belong[maxn],vis[maxn],num[maxn],rd[
     maxn],cd[maxn];
int en,top,scc,n,m,index;
void addedge(int u,int v)
{
    e[en].to=v;
    e[en].nxt=head[u];
    head[u]=en++;
}
void tarjan(int x)
{
    int v;
    dfn[x]=low[x]=++index;
    sta[top++]=x;
    vis[x]=1;
    for(int i=head[x];i!=-1;i=e[i].nxt)
    {
        v=e[i].to;
        if(!dfn[v])
        {
            tarjan(v);
            low[x]=min(low[x],low[v]);
        }
        else
            if(vis[v])
            {
                low[x]=min(low[x],dfn[v]);
            }
    }
    if(low[x]==dfn[x])
    {
        scc++;
        do
        {
            v=sta[--top];
            vis[v]=0;
            belong[v]=scc;
            num[scc]++;
        }while(x!=v);
    }
}
int main()
{
    std::ios::sync_with_stdio(false);
    cin>>n;
    memset(head,-1,sizeof(head));
    memset(low,0,sizeof(low));
    memset(dfn,0,sizeof(dfn));
    memset(vis,0,sizeof(vis));
    memset(rd,0,sizeof(rd));
    memset(cd,0,sizeof(cd));
    memset(num,0,sizeof(num));
    top=0;
    index=0;
    scc=0;
```

```
    for(int i=1;i<=n;i++)
    {
        while(1)
        {
            int v;
            cin>>v;
            if(v==0) break;
            addedge(i,v);
        }
    }
    for(int i=1;i<=n;i++)
        if(!dfn[i]) tarjan(i);
    for(int u=1;u<=n;u++)
        for(int i=head[u];i!=-1;i=e[i].nxt)
        {
            int v=e[i].to;
            if(belong[u]!=belong[v])
            {
                cd[belong[u]]++;
                rd[belong[v]]++;
            }

        }
    int ans1=0,ans2=0;
    for(int i=1;i<=scc;i++)
        if(rd[i]==0) ans1++;
    for(int i=1;i<=scc;i++)
        if(cd[i]==0) ans2++;
    if(scc==1) cout<<1<<endl<<0<<endl;
    else cout<<ans1<<endl<<max(ans1,ans2);
    return 0;
}
```

3.4 欧拉回路问题

给定一张有向图，如何求出一条经过每条边恰好一次的回路必要条件：1. 这张图是个强连通分量 2. 每个点的出度等于入度可以发现这两个条件同时也是充分的

3.4.1 圈套圈算法

任选一个起点，从起点开始 dfs，每条边只能被走一遍，当没有边可以走的时候把 x 压入答案的队列中，最后的答案是反着的欧拉回路

3.4.2 二进制

有 n 个灯泡，灯泡有两种状态：开和关，每次可以操作某个灯泡，使得它状态取反定义一个局面为每个灯泡的状态以及最后一个被操作的灯泡的编号，可以将

一个局面看成一个二元组 (0...2^N-1,0...N-1) 现在你可以选定任何初始局面，求最少几步遍历所有局面 N<=20 可以建立 2^N 个点，将每个局面看成边，求欧拉回路

3.5 树

树：N 个点 N-1 条边的连通无向图，分为有根树和无根树树的叶子：度数为 1 的点

3.6 最小生成树

3.6.1 Kruskal 算法

1）把图中的所有边按代价从小到大排序 2）把图中的 n 个顶点看成独立的 n 棵树组成的森林 3）按权值从小到大选择边，所选的边连接的两个顶点 ui,viui,vi, 应属于两颗不同的树，则成为最小生成树的一条边，并将这两颗树合并作为一颗树 4）重复 3), 直到所有顶点都在一颗树内或者有 n-1 条边为止

3.6.2 Prim 算法

选定一个点做为一个集合 a ，剩下的点为另一个集合 b 将横跨两个集合且权重在其中最小的边加入最小生成树将刚刚加入最小生成树的边中不在集合 a 中的点加入集合 a，直到所有的点加入集合 a

```
int prim(int e[][maxx],int n)
{
    int ans=0;
    memset(book,false,sizeof(book));
    book[0]=true;
    for(int i=1; i<n; i++)
        dis[i]=e[0][i];
    for(int i=1; i<n; i++)
    {
        int minn=inf,u=-1;
        for(int j=0; j<n; j++)
        {
            if(!book[j]&&dis[j]<minn)
            {
                minn=dis[j];
                u=j;
            }
        }
        if(ans==inf)
            return -1;
        ans+=minn;
        book[u]=true;
        for(int v=0; v<n; v++)
```

```
            if(!book[v])
                dis[v]=min(dis[v],e[u][v]);
    }
    return ans;
}
```

3.6.3 最小瓶颈路

最小生成树定义，x 到 y 的最小瓶颈路上的最大边权等于最小生成树上 x 到 y 路径上的最大边权。虽然最小生成树不唯一，但是每种最小生成树 x 到 y 路径的最大边权相同且为最小值。也就是说，每种最小生成树上的 x 到 y 的路径均为最小瓶颈路。最小生成树上两个点之间的简单路径上边权最大值 = Kruskal 重构树上两点之间的 LCA 的权值。

```
const int maxn = 500*500+10;
int n,m,q;
int cnt = 0;
int a[550][550];
int fa[maxn*2];

int getfa(int x)
{
    if(fa[x] == x)
    {
        return x;
    }
    return fa[x] = getfa(fa[x]);
}
struct E{int x,y,w;} e[maxn*4];
void add(int x,int y,int w)
{
    e[cnt].x = x;
    e[cnt].y = y;
    e[cnt].w = w;
    cnt++;
}
bool cmp(E a,E b)
{
    return a.w < b.w;
}
int lg[maxn*2];

int val[maxn*2];
VI G[maxn*2];
int pa[maxn*2][24];
int dep[maxn*2];


void dfs(int u,int f)
{
    dep[u] = dep[f] + 1;
    pa[u][0] = f;
    for(int i = 1; (1 << i) <= dep[u] ;i++)
    {
```

```
            pa[u][i] = pa[pa[u][i-1]][i-1];
        }
        for(auto &v:G[u])
        {
            if(v != f)
            {
                dfs(v,u);
            }
        }
}

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

int id(int x,int y) { return (x-1)*m+y-1;}
int main(int argc, char const *argv[])
{
#ifdef indiewar
    freopen("in.txt","r",stdin);
#endif
    ios_base::sync_with_stdio(false), cin.tie(0), cout.tie(0);
    cin >> n >> m >> q;
    rep(i,1,n*m*2+10)
    {
        lg[i] = lg[i-1];
        if(i == 1 << lg[i-1]) lg[i]++;
    }
    rep(i,0,n)
    {
        rep(j,0,m)
        {
            cin >> a[i][j];
        }
    }
    rep(i,0,n*m*2+10)
    {
        fa[i] = i;
    }
    rep(i,0,n)
    {
        rep(j,0,m)
        {
            if(i != 0)
            {
                add(id(i+1,j+1),id(i,j+1),max(a[i][j],a[i-1][j]));
            }
            if(j != 0)
```

```cpp
            {
                add(id(i+1,j+1),id(i+1,j),max(a[i][j],a[i][j-1]));
            }
            if(i != n-1)
            {
                add(id(i+1,j+1),id(i+2,j+1),max(a[i][j],a[i+1][j]));
            }
            if(j != m-1)
            {
                add(id(i+1,j+1),id(i+1,j+2),max(a[i][j],a[i][j+1]));
            }
        }
    }
    sort(e,e+cnt,cmp);
    int tmp = id(n,m);
    rep(i,0,cnt)
    {

        int fx = getfa(e[i].x);
        int fy = getfa(e[i].y);

        if(fx != fy)
        {
            tmp++;
            fa[fx] = fa[fy] = tmp;
            val[tmp] = e[i].w;
            G[fx].pb(tmp);
            G[fy].pb(tmp);
            G[tmp].pb(fx);
            G[tmp].pb(fy);
        }
    }
    dfs(tmp,0);
    int x,y,xx,yy;
    rep(i,0,q)
    {
        cin >> x >> y >> xx >> yy;
        if(x == xx && y == yy)
        {
            cout << a[x-1][y-1] << endl;
        } else{
            cout << val[lca(id(x,y),id(xx,yy))] << endl;
        }
    }
    return 0;
}
```

3.7 Prufer 序列

将一棵树变成一个序列：每次选择树上标号最小的叶子，删掉它，将与它相连的那个点的标号加到序列里，直到只剩下 2 个点可以证明：任意一个长度为 n-2 的 1...n 的序列都是某棵树的 Prufer 序列所以可以推出：n 个点的无根树个数为 n^(n-2) - 无根树转化为 prufer 序列 1. 找到编号最小的度数为 1 的点 2. 删

除该节点并在序列中添加与该节点相连的节点的编号 3. 重复 1,2 操作，直到整棵树只剩下两个节点 - prufer 序列转化为无根树 1. 每次取出 prufer 序列中最前面的元素 u 2. 在点集中找到编号最小的没有在 prufer 序列中出现的元素 v 3. 给 u,v 连边然后分别删除 4. 最后在点集中剩下两个节点，给它们连边

给出标号为 1 到 N 的点, 以及某些点最终的度数, 允许在任意两点间连线, 可产生多少棵度数满足要求的树? 一个含有 n 个节点的 prufer 序列有 n−2 个数字。Purfer Sequence 中的每个数是 1~n 中的一个数。而每一个 prufer 序列和一棵树是唯一对应的。

对于`cnt`个度数限制为`d_i`的点，那么它会在 prufer 序列中出现`d_i-1`次

故 Prufer Sequence 的总长度`sum=\sum_{i=1}^{cnt}(d_i-1)`

所以 Prufer Sequence 不同的排列总数为

$$C_{n-2}^{sum} \times \frac{sum!}{\prod_{i=1}^{cnt}(d_i-1)!}$$

而剩下 (n-2-sum) 个位置可以随意排列 (n-cnt) 个点

所以

$$ans = C_{n-2}^{sum} \times \frac{sum!}{\prod_{i=1}^{cnt}(d_i-1)!} \times (n-cnt)^{(n-2-sum)}$$

化简得

$$ans = \frac{(n-2)!}{(n-2-sum)! \times \prod_{i=1}^{cnt}(d_i-1)!} \times (n-cnt)^{(n-2-sum)}$$

```cpp
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <cmath>
#include <cstring>
#include <cctype>
using namespace std;
int read()
{
    int x=0,f=1;char ch=getchar();
    while (!isdigit(ch)){if (ch=='-') f=-1;ch=getchar();}
    while (isdigit(ch)){x=x*10+ch-48;ch=getchar();}
    return x*f;
}
int n,cnt,d[1050],sum,tot;
struct BigInt
```

```
{
    int num[10000],len;

    BigInt()
    {
        memset(num,0,sizeof(num));
        len=1;
    }

    BigInt operator* (const int &rhs) const
    {
        BigInt ans;
        ans.len=len+6;
        for (int i=1;i<=len;i++)
            ans.num[i]+=num[i]*rhs;
        for (int i=1;i<ans.len;i++)
            if (ans.num[i]>9)
            {
                ans.num[i+1]+=ans.num[i]/10;
                ans.num[i]%=10;
            }
        while (!ans.num[--ans.len])
        return ans;
    }

    BigInt operator/ (const int &rhs) const
    {
        BigInt ans=*this;
        ans.len++;
        for (int i=ans.len;i;i--)
        {
            ans.num[i-1]+=ans.num[i]%rhs*10;
            ans.num[i]/=rhs;
        }
        while (!ans.num[--ans.len]);
        return ans;
    }
}ans;
int main()
{
    n=read();
    for (int i=1;i<=n;i++)
    {
        d[i]=read();
        if (!d[i])
        {
            puts("0");
            return 0;
        }
        if (d[i]!=-1)
        {
            sum+=d[i]-1;
            cnt++;
        }
    }
    if (sum>2*n-2)
    {
```

```
        puts("0");
        return 0;
    }
    ans.num[1]=1;
    for (int i=n-1-sum;i<n-1;i++)
        ans=ans*i;
    for (int i=1;i<=n-2-sum;i++)
        ans=ans*(n-cnt);
    for (int i=1;i<=n;i++)
    {
        for (int j=2;j<=d[i]-1;j++)
            ans=ans/j;
    }
    for (int i=ans.len;i;i--)
        printf("%d",ans.num[i]);
    return 0;
}
```

```
from math import factorial


def C(n, m):
    if m > n:
        return 0
    return factorial(n) // factorial(m) // factorial(n - m)


def pow(x, k):
    res = 1
    while k > 0:
        if k & 1:
            res *= x
        x *= x
        k >>= 1
    return res


n = int(input())
k, Sum = 0, 0
a = []

for i in range(n):
    x = int(input())
    if x == -1:
        k += 1
    else:
        x -= 1
        Sum += x
        a.append(x)

ans = C(n - 2, Sum) * factorial(Sum)
for x in a:
    ans = ans // factorial(x)
ans *= pow(k, n - 2 - Sum)

print(ans)
```

3.8 二分图

可以分成两部分，使得这两部分内部没有边的图一个图是二分图等价于该图没有奇环

3.8.1 二分图匹配

给定一张二分图，求它的最大匹配匈牙利算法：每次找一条增广路

```
bool find(int x){
    int i,j;
    for (j=1;j<=m;j++){
        if (line[x][j]==true && used[j]==false)
        {
            used[j]=1;
            if (girl[j]==0 || find(girl[j])) {
                girl[j]=x;
                return true;
            }
        }
    }
    return false;
}
```

最小顶点覆盖：选最少的点覆盖所有边 | 二分图最小顶点覆盖 |=| 二分图最大匹配 | 最大独立集：选最多的点使得它们两两没边相连 | 二分图最大独立集 |= 总点数-| 二分图最小顶点覆盖 |

3.8.2 KM

```
struct KM {
    typedef long long cost_t;
    static const int N = 222;
    static const cost_t inf = 1e9;
    cost_t lx[N], ly[N], sl[N];
    int px[N],py[N],sy[N],fa[N],n;
    void aug(int v) {
        sy[v]=py[v]; if (px[sy[v]]!=-2) aug(px[sy[v]]);
    }
    bool find(int v, const cost_t w[][N]) {
        for (int i=0;i<n;++i) if (!~py[i]) {
                if (sl[i]>lx[v]+ly[i]-w[v][i]) {
                    sl[i]=lx[v]+ly[i]-w[v][i]; fa[i] = v;
                }
                if (lx[v]+ly[i]==w[v][i]) {
                    py[i]=v; if (!~sy[i]) {aug(i); return 1;}
                    if (~px[sy[i]]) continue;
                    px[sy[i]]=i; if (find(sy[i],w)) return 1;
                }
            }
        return 0;
```

```
    }
    cost_t gao(int _n, const cost_t w[][N], cost_t m=inf) {
        n=_n; std::fill(sy,sy+n,-1); std::fill(ly,ly+n,0);
        for (int i=0;i<n;++i) lx[i]=*std::max_element(w[i],w[i]+n);
        for (int i(0),flag;i<n;++i) {
            for (int j=0;j<n;++j)px[j]=py[j]=-1,sl[j]=inf;
            px[i]=-2; if (find(i,w)) continue;
            for (flag=0,m=inf; !flag; m=inf) {
                for (int j=0;j<n;++j) if (!~py[j]) m=std::min(m,sl[j]);
                for (int j=0;j<n;++j) {
                    if (~px[j]) lx[j]-=m;
                    if (~py[j]) ly[j]+=m;
                    else sl[j]-=m;
                }
                for (int j=0;j<n;++j) if (!~py[j]&&!sl[j]) {
                        py[j]=fa[j]; if (!~sy[j]) {aug(j);flag=1;break;}
                        px[sy[j]]=j; if (find(sy[j],w)) {flag=1;break;}
                    }
            }
        }
        cost_t ret(0);
        for (int i=0;i<n;++i) ret+=w[sy[i]][i];
        return ret;
    }
};
```

```
const int N=405;
const ll inf=~0U>>1;
int n,nl,nr,m,z,py,x,y,i,j,p,lk[N],pre[N];
bool vy[N];
int lx[N],ly[N],d,w[N][N],slk[N];ll ans;
int main(){
  scanf("%d%d%d",&nl,&nr,&m);
  n=max(nl,nr);
  while(m--)scanf("%d%d%d",&x,&y,&z),w[y][x]=max(w[y][x],z);
  for(i=1;i<=n;i++)for(j=1;j<=n;j++)lx[i]=max(lx[i],w[i][j]);
  for(i=1;i<=n;i++){
    for(j=1;j<=n;j++)slk[j]=inf,vy[j]=0;
    for(lk[py=0]=i;lk[py];py=p){
      vy[py]=1;d=inf;x=lk[py];
      for(y=1;y<=n;y++)if(!vy[y]){
        if(lx[x]+ly[y]-w[x][y]<slk[y])slk[y]=lx[x]+ly[y]-w[x][y],pre[y]=py;
        if(slk[y]<d)d=slk[y],p=y;
      }
      for(y=0;y<=n;y++)if(vy[y])lx[lk[y]]-=d,ly[y]+=d;else slk[y]-=d;
    }
    for(;py;py=pre[py])lk[py]=lk[pre[py]];
  }
  for(i=1;i<=n;i++)ans+=lx[i]+ly[i];
  printf("%lld\n",ans);
  for(i=1;i<=nl;i++)printf("%d ",w[lk[i]][i]?lk[i]:0);
}
```

3.9 三元环计数

求无向图的三元环个数将点按照度数为第一关键字，标号为第二关键字从小到大排序，定义排序后每个点的序为 pos[x] 对于每条无向边，变成有向边：pos 较小的点连向 pos 较大的点这样连完后每个点的出度都最多只有 $N^{0.5}$ 枚举三元环中 pos 最小的点 x，然后枚举 x 的出边 y，再枚举 y 的出边 z 如果 z 也在 x 的出边中的话就得到一个三元环时间复杂度：$O(N^{1.5})$

3.10 网络流

- dinic

```
const int maxn = 4e3+100;
const int maxm = 1e5+100;
const int inf = 0x7f7f7f7f;

typedef struct Dinic
{
    typedef struct Edge
    {
        int u,v,w,nxt;
    } Edge;
    int head[maxn],hcnt;
    int dep[maxn];
    int cur[maxn];
    Edge e[maxm];
    int S,T,N;
    void init()
    {
        memset(head,-1,sizeof head);
        hcnt = 0;
        S = T = N = 0;
    }
    void adde(int u,int v,int w)
    {
        e[hcnt].u = u,e[hcnt].v = v,e[hcnt].w = w;
        e[hcnt].nxt = head[u];head[u] = hcnt++;
        e[hcnt].u = v,e[hcnt].v = u,e[hcnt].w = 0;
        e[hcnt].nxt = head[v];head[v] = hcnt++;
    }
    int bfs()
    {
        rep(i,0,N)
        {
            dep[i] = inf;
        }
        queue<int> q;
        q.push(S); dep[S] = 0;
        while(!q.empty())
        {
            int u = q.front();q.pop();
            for(int i = head[u];~i;i = e[i].nxt)
            {
                int v = e[i].v,w = e[i].w;
                if(w > 0 && dep[u] + 1 < dep[v])
```

```
                {
                    dep[v] = dep[u] + 1;
                    if(v == T)
                    {
                        return 1;
                    }
                    q.push(v);
                }
            }
        }
        return dep[T] != inf;
    }
    int dfs(int s,int mw)
    {
        if(s == T) return mw;
        for(int i = cur[s];~i;i=e[i].nxt)
        {
            cur[s] = i;
            int v = e[i].v,w=e[i].w;
            if(w <= 0 || dep[v] != dep[s] + 1)
            {
                continue;
            }
            int cw = dfs(v,min(w,mw));
            if(cw <= 0)
                continue;
            e[i].w -= cw;
            e[i^1].w += cw;
            return cw;
        }
        return 0;
    }
    ll dinic()
    {
        ll res = 0;
        while(bfs())
        {
            rep(i,0,N)
            {
                cur[i] = head[i];
            }
            while(int d = dfs(S,inf))
            {
                res += 1ll * d;
            }
        }
        return res;
    }
} Dinic;
```

- MCMF1

```
namespace mincostflow {
    const int INF=0x3f3f3f3f;
    struct node {
        int to; int cap,cost; int rev;
        node(int t=0,int c=0,int _c=0,int n=0):
            to(t),cap(c),cost(_c),rev(n) {};
```

```
};  vector<node> edge[maxn];
void addedge(int from,int to,int cap,int cost) {
    edge[from].push_back(node(to,cap,cost,edge[to].size()));
    edge[to].push_back(node(from,0,-cost,edge[from].size()-1));
}
int dis[maxn];
bool mark[maxn];
void spfa(int s,int t,int n) {
    memset(dis+1,0x3f,n*sizeof(int));
    memset(mark+1,0,n*sizeof(bool));
    static int Q[maxn],ST,ED;
    dis[s]=0; ST=ED=0; Q[ED++]=s;
    while (ST!=ED) {
        int v=Q[ST]; mark[v]=0;
        if ((++ST)==maxn) ST=0;
        for (node &e:edge[v]) {
            if (e.cap>0&&dis[e.to]>dis[v]+e.cost) {
                dis[e.to]=dis[v]+e.cost;
                if (!mark[e.to]) {
                    if (ST==ED||dis[Q[ST]]<=dis[e.to]) {
                        Q[ED]=e.to,mark[e.to]=1;
                        if ((++ED)==maxn) ED=0;
                    } else {
                        if ((--ST)<0) ST+=maxn;
                        Q[ST]=e.to,mark[e.to]=1;
                    }
                }
            }
        }
    }
} int cur[maxn];
int dfs(int x,int t,int flow) {
    if (x==t||!flow) return flow;
    int ret=0; mark[x]=1;
    for (int &i=cur[x];i<(int)edge[x].size();i++) {
        node &e=edge[x][i];
        if (!mark[e.to]&&e.cap) {
            if (dis[x]+e.cost==dis[e.to]) {
                int f=dfs(e.to,t,min(flow,e.cap));
                e.cap-=f; edge[e.to][e.rev].cap+=f;
                ret+=f; flow-=f;
                if (flow==0) break;
            }
        }
    } mark[x]=0;
    return ret;
}
pair<int,int> min_costflow(int s,int t,int n) {
    int ret=0,ans=0;
    int flow = INF;
    while (flow) {
        spfa(s,t,n); if (dis[t]==INF) break;
        memset(cur+1,0,n*sizeof(int));
        int len=dis[t],f;
        while ((f=dfs(s,t,flow))>0)
            ret+=f,ans+=len*f,flow-=f;
    } return make_pair(ret,ans);//最大流，最小费用
```

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

- MCMF2

```
const int maxn = 2e4+10;
namespace MCMF {
    const int inf=0x3f3f3f3f;
    struct Edge {
        int to; int cap,cost; int rev;
        Edge(int t=0,int c=0,int _c=0,int n=0):
            to(t),cap(c),cost(_c),rev(n) {};
    };
    vector<Edge> edge[maxn];
    void adde(int from,int to,int cap,int cost)
    {
        edge[from].push_back(Edge(to,cap,cost,edge[to].size()));
        edge[to].push_back(Edge(from,0,-cost,edge[from].size()-1));
    }

    int dis[maxn];
    bool mark[maxn];

    void spfa(int s,int t,int n)
    {
        memset(dis,0x3f,sizeof dis);
        memset(mark,0,sizeof mark);
        static int Q[maxn],ST,ED;
        dis[s]=0; ST=ED=0; Q[ED++]=s;
        while (ST!=ED)
        {
            int v=Q[ST]; mark[v]=0;
            if ((++ST)==maxn) ST=0;
            for (Edge &e:edge[v])
            {
                if (e.cap>0&&dis[e.to]>dis[v]+e.cost)
                {
                    dis[e.to]=dis[v]+e.cost;
                    if (!mark[e.to])
                    {
                        if (ST==ED||dis[Q[ST]]<=dis[e.to])
                        {
                            Q[ED]=e.to,mark[e.to]=1;
                            if ((++ED)==maxn) ED=0;
                        }
                        else
                        {
                            if ((--ST)<0) ST+=maxn;
                            Q[ST]=e.to,mark[e.to]=1;
                        }
                    }
                }
            }
        }
    }
```

```
    int cur[maxn];
    int dfs(int x,int t,int flow)
    {
        if (x==t||!flow) return flow;
        int ret=0; mark[x]=1;
        for (int &i=cur[x];i<(int)edge[x].size();i++)
        {
            Edge &e=edge[x][i];
            if (!mark[e.to]&&e.cap)
            {
                if (dis[x]+e.cost==dis[e.to])
                {
                    int f=dfs(e.to,t,min(flow,e.cap));
                    e.cap-=f; edge[e.to][e.rev].cap+=f;
                    ret+=f; flow-=f;
                    if (flow==0) break;
                }
            }
        }
        mark[x]=0;
        return ret;
    }
    pair<int,ll> mc(int s,int t,int n)
    {
        int ret=0;
        ll ans=0;
        int flow = inf;
        while(flow)
        {
            spfa(s,t,n); if (dis[t]==inf) break;
            memset(cur,0,sizeof cur);
            int len=dis[t],f;
            while ((f=dfs(s,t,flow))>0)
                ret+=f,ans+=(ll)len*(ll)f,flow-=f;
        }
        return make_pair(ret,ans);//最大流，最小费用
    }
    void init(int n)
    {
        for(int i = 1; i <= n; i++) edge[i].clear();
    }
}
```

4 数学

4.1 BSGS

- 用于计算方程 $ a^x = b (% p)$

```
int BSGS(int a,int b,int p)
{
    map<int,int > hash;
    b %= p;
```

```
    int t = (int)sqrt(p) + 1;
    for(int j = 0; j < t; j ++){
        int val = 1ll * b * fpow(a,j,p) % p;
        hash[val] = j;
    }
    a = fpow(a,t,p);
    if(!a) return b == 0 ? 1 : -1;
    for(int i = 0; i <= t;++i){
        int val = fpow(a,i,p);
        int j = hash.find(val) == hash.end() ? -1 : hash[val];
        if(j >= 0 && i * t - j > 0) return i * t - j;
    }
    return -1;
}
```

4.2 扩展中国剩余定理

4.2.1 值在 __int128 以内

```
typedef __int128 ll;
void exgcd(ll a,ll b,ll &x,ll &y)
{
    if(!b){
        x = 1;y = 0;
        return ;
    }
    exgcd(b,a % b,y,x);
    y -= a / b * x;
}
ll inv(ll a,ll p)
{
    ll x,y;
    exgcd(a,p,x,y);
    if(x < 0) x += p;
    return x;
}
ll ex_crt()
{
    bool flag = true;
    ll m1,m2,c1,c2,tmp;
    for(int i = 2;i <= n;++i){//m[i]为模,c[i]为余数
        m1 = m[i - 1];m2 = m[i];
        c1 = c[i - 1];c2 = c[i];
        tmp = gcd(m1,m2);
        if((c2 - c1) % tmp != 0) {
            flag = false;
            return -1;
        }
        m[i] = m1 / tmp * m2;
        c[i] = inv(m1 / tmp,m2 / tmp) * (c2 - c1) / tmp % (m2 / tmp) * m1 + c1;
        c[i] = (c[i] % m[i] + m[i]) % m[i];
    }
    return c[n];
}
```

4.2.2 值在 __int128 以外

```python
import sys
def exgcd(a, b):
    if b == 0:
        return (a, 1, 0)
    q = a // b
    g, y, x = exgcd(b, a - q * b)
    y -= q * x
    return (g, x, y)

def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)

N, lim = map(int, input().split())
M, ans = map(int, input().split())
f = 0
for i in range(0, N - 1):
    m, r = map(int, input().split())
    A = M
    B = m
    C = ((r - ans) % m)
    if (ans - r) % gcd(M, m) != 0:
        f = 1
    g, x, y = exgcd(A, B)
    x = (x * C // g) % (B // g)
    ans += x * M
    M *= B // g
    ans %= M

if ans < 0:
    ans += M
if f == 1:
    ans = -1
    print(ans)
else:
    print(ans)
# print ans
```

4.3 扩展卢卡斯

```cpp
ll n, m, p;
ll exgcd(ll a, ll b, ll &x, ll &y){
    if(!b){
        x = 1, y = 0;
        return a;
    }
    ll res = exgcd(b, a % b, x, y);
    ll t = x;
    x = y;
    y = t - a / b * y;
```

```
    return res;
}

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

ll inv(ll a, ll p){
    ll x, y;
    exgcd(a, p, x, y);
    if(x + p > p) return x;
    return x + p;
}

inline ll crt(ll n, ll mod)
{
    return n * (p / mod) % p * inv(p / mod, mod) % p;
}

ll fac(ll n, ll p, ll k){   //k = p^x
       if(!n) return 1;
       ll ans = 1;
    for(int i = 2; i <= k;++i)
        if(i % p) ans = ans * i % k;
    ans = fpow(ans, n / k, k);
    for(int i = 2; i <= n % k;++i)
        if(i % p) ans = ans * i % k;
    return ans * fac(n / p, p, k) % k;
}

ll C(ll n, ll m, ll p, ll k){ //k = p^x
       if(n < m) return 0;
    ll a = fac(n,p,k), b = fac(m,p,k), c = fac(n - m,p,k);
    ll cnt = 0;
    for(ll i = p; i <= n; i *= p) cnt += n / i;
    for(ll i = p; i <= m; i *= p) cnt -= m / i;
    for(ll i = p; i <= n-m; i *= p) cnt -= (n - m) / i;
    return a*inv(b, k) % k * inv(c, k) % k * fpow(p, cnt, k) % k ;
}

ll ex_Lucas()
{
    ll t = p, ans =i;
            t /= i;
        }
        ans = (ans + crt(C(n, m, i, tmp), tmp))%p;
    }
    if(t > 1) ans = (ans + crt(C(n, m, t, t), t))%p;
    return ans%p;
}
```

4.4 fft

- 多项式快速计算

```
const double PI = acos(-1.0);
struct Complex{
   double x, y; // 实部和虚部 x+yi
   Complex(double _x = 0.0, double _y = 0.0) {
      x = _x;
      y = _y;
   }
   Complex operator-(const Complex &b) const {
      return Complex(x - b.x, y - b.y);
   }
   Complex operator+(const Complex &b) const {
      return Complex(x + b.x, y + b.y);
   }
   Complex operator*(const Complex &b) const {
      return Complex(x * b.x - y * b.y, x * b.y + y * b.x);
   }
};
/*
* 进行 FFT 和 IFFT 前的反转变换。
* 位置 i 和 (i 二进制反转后位置)互换
* len 必须去 2 的幂
*/
void change(Complex y[], int len){
   for (int i = 1, j = len / 2,k;i < len - 1;++i){
      if (i < j)swap(y[i], y[j]);
      //交换互为小标反转的元素,i<j 保证交换一次
      //i 做正常的+1,j 左反转类型的+1,始终保持 i 和 j 是反转的
      k = len / 2;
      while(j >= k){
         j -= k;
         k >>= 1;
      }
      if (j < k) j += k;
   }
}
/*
* 做 FFT
* len 必须为 2^k 形式,
* on==1 时是 DFT,on==-1 时是 IDFT
*/
void fft(Complex y[], int len, int on) {
   change(y, len);
   for (int h = 2; h <= len; h <<= 1) {
      Complex wn(cos(-on * 2 * PI / h), sin(-on * 2 * PI / h));
      for (int j = 0; j < len; j += h) {
         Complex w(1, 0);
         for (int k = j; k < j + h / 2; k++) {
            Complex u = y[k];
            Complex t = w * y[k + h / 2];
            y[k] = u + t;
            y[k + h / 2] = u - t;
            w = w * wn;
```

```
            }
        }
    }
    if (on == -1)
        for (int i = 0; i < len; i++) y[i].x /= len;
}
```

use by indiewar

```
using int64 = long long;

namespace fft {
    const int N = 1 << 21, M = 31768;
    struct Complex {
        double x , y;
        Complex () {x = y = 0;}
        Complex (double _x , double _y) {x = _x , y = _y;}
        Complex operator + (const Complex& r) const {
            return Complex(x + r.x , y + r.y);
        }
        Complex operator - (const Complex& r) const {
            return Complex(x - r.x , y - r.y);
        }
        Complex operator * (const double k) const {
            return Complex(x * k, y * k);
        }
        Complex operator / (const double k) const {
            return Complex(x / k, y / k);
        }
        Complex operator * (const Complex& r) const {
            return Complex(x * r.x - y * r.y , x * r.y + y * r.x);
        }
        int operator = (const int a) {
            *this = Complex(a , 0);
            return a;
        }
        Complex conj() const {
            return Complex(x, -y);
        }
    };
    const double pi = acos(-1.0);
    Complex w[N];
    int rev[N];
    void init(int L) {
        int n = 1 << L;
        for (int i = 0 ; i < n ; ++ i) {
            double ang = 2 * pi * i / n;
            w[i] = Complex(cos(ang) , sin(ang));
            rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (L - 1));
        }
    }
    void trans(Complex P[], int n, int oper) {
        for (int i = 0 ; i < n ; i ++) {
            if (i < rev[i]) {
                std::swap(P[i], P[rev[i]]);
            }
        }
        for (int d = 0; (1 << d) < n; d++) {
```

```
            int m = 1 << d, m2 = m * 2 , rm = n / m2;
            for (int i = 0; i < n; i += m2) {
                for (int j = 0; j < m; j++) {
                    Complex &P1 = P[i + j + m], &P2 = P[i + j];
                    Complex t = w[rm * j] * P1;
                    P1 = P2 - t;
                    P2 = P2 + t;
                }
            }
        }
        if (oper == -1) {
            for (int i = 0; i < n; ++i) {
                P[i] = P[i] / n;
            }
        }
    }
    Complex A[N] , B[N] , C1[N] , C2[N];
    std::vector<int64> conv(const std::vector<int> &a, const std::vector<int> &b) {
        int n = a.size(), m = b.size(), L = 0, s = 1;
        while (s <= n + m - 2) s <<= 1, ++L;
        init(L);
        for (int i = 0; i < s; ++i) {
            A[i] = i < n ? Complex(a[i], 0) : Complex();
            B[i] = i < m ? Complex(b[i], 0) : Complex();
        }
        trans(A, s, 1);
        trans(B, s, 1);
        for (int i = 0; i < s; ++i) {
            A[i] = A[i] * B[i];
        }
        for (int i = 0; i < s; ++i) {
            w[i] = w[i].conj();
        }
        trans(A, s, -1);
        std::vector<int64> res(n + m - 1);
        for (int i = 0; i < n + m - 1; ++i) {
            res[i] = (int64)(A[i].x + 0.5);
        }
        return res;
    }
    std::vector<int64> fast_conv(const std::vector<int> &a, const std::vector<int> &b) {
        int n = a.size(), m = b.size(), L = 0, s = 1;
        for (; s <= n + m - 2; s <<= 1, ++L);
        s >>= 1, --L;
        init(L);
        for (int i = 0; i < s; ++i) {
            A[i].x = (i << 1) < n ? a[i << 1] : 0;
            B[i].x = (i << 1) < m ? b[i << 1] : 0;
            A[i].y = (i << 1 | 1) < n ? a[i << 1 | 1] : 0;
            B[i].y = (i << 1 | 1) < m ? b[i << 1 | 1] : 0;
        }
        trans(A, s, 1); trans(B, s, 1);
        for (int i = 0; i < s; ++i) {
            int j = (s - i) & (s - 1);
            C1[i] = (Complex(4, 0) * (A[j] * B[j]).conj() - (A[j].conj() - A[i]) *
```

```
                (B[j].conj() - B[i]) * (w[i] + Complex(1, 0))) * Complex(0, 0.25);
        }
        std::reverse(w + 1, w + s);
        trans(C1, s, -1);
        std::vector<int64> res(n + m);
        for (int i = 0; i <= (n + m - 1) / 2; ++i) {
            res[i << 1] = int64(C1[i].y + 0.5);
            res[i << 1 | 1] = int64(C1[i].x + 0.5);
        }
        res.resize(n + m - 1);
        return res;
    }
    // arbitrary modulo convolution
    void conv(int a[], int b[], int n, int m, int mod, int res[]) {
        int s = 1, L = 0;
        while (s <= n + m - 2) s <<= 1, ++L;
        init(L);
        for (int i = 0; i < s; ++i) {
            A[i] = i < n ? Complex(a[i] / M, a[i] % M) : Complex();
            B[i] = i < m ? Complex(b[i] / M, b[i] % M) : Complex();
        }
        trans(A, s, 1); trans(B, s, 1);
        for (int i = 0; i < s; ++i) {
            int j = i ? s - i : i;
            Complex a1 = (A[i] + A[j].conj()) * Complex(0.5, 0);
            Complex a2 = (A[i] - A[j].conj()) * Complex(0, -0.5);
            Complex b1 = (B[i] + B[j].conj()) * Complex(0.5, 0);
            Complex b2 = (B[i] - B[j].conj()) * Complex(0, -0.5);
            Complex c11 = a1 * b1 , c12 = a1 * b2;
            Complex c21 = a2 * b1 , c22 = a2 * b2;
            C1[j] = c11 + c12 * Complex(0, 1);
            C2[j] = c21 + c22 * Complex(0, 1);
        }
        trans(C1, s, -1); trans(C2, s, -1);
        for (int i = 0 ; i < n + m - 1; ++i) {
            int x = int64(C1[i].x + 0.5) % mod;
            int y1 = int64(C1[i].y + 0.5) % mod;
            int y2 = int64(C2[i].x + 0.5) % mod;
            int z = int64(C2[i].y + 0.5) % mod;
            res[i] = ((int64)x * M * M + (int64)(y1 + y2) * M + z) % mod;
        }
    }
}

int n,m;
VI a,b;
int main()
{
#ifdef indiewar
    freopen("in.txt","r",stdin);
    freopen("out.txt","w",stdout);
#endif
    int x;
    cin >> n >> m;
    rep(i,0,n+1)
    {
        cin >> x;a.pb(x);
```

```
    }
    rep(i,0,m+1)
    {
        cin >> x;b.pb(x);
    }
    vector<ll> res;
    if(a.size() >= 2 && b.size()>=2)
         res = fft::fast_conv(a,b);
    else
         res = fft::conv(a,b);
    for(auto &v:res) cout << v << " ";
    return 0;
}
```

4.5 fwt

- 计算集合卷积

```
void fwt(ll *a, int n)
{
    for(int d = 1;d < n;d <<= 1){
        for(int m = d << 1,i = 0;i < n;i += m){
            for(int j = 0;j < d;++j){
                ll x = a[i + j],y = a[i + j + d];
                a[i + j] = (x + y),a[i + j + d] = (x - y);
                //xor:a[i + j] = x + y,a[i + j + d] = (x - y + mod) % mod;
                //and:a[i + j] = x + y;
                //or:a[i + j + d] = x + y;
            }
}
void ufwt(ll *a, int n)
{
    for(int d = 1;d < n;d <<= 1){
        for(int m = d << 1,i = 0;i < n;i += m){
            for(int j = 0;j < d;++j){
                ll x = a[i + j],y = a[i + j + d];
                a[i + j] = 1LL * (x + y) / 2,a[i + j + d] = (1LL * (x - y) / 2);
                //xor:a[i + j] = (x + y) / 2,a[i + j + d] = (x - y) / 2;
                //and:a[i + j] = x - y;
                //or:a[i + j + d] = y - x;
            }
}
void work(ll *a,ll *b, int n)
{
    fwt(a,n);
    fwt(b,n);
    for(int i = 0;i < n;++i) a[i] *= b[i];
    ufwt(a,n);
}
```

4.6 polya

- m 种颜色，n 个长度

```
const int N = 1e5 + 7;
int euler(int n){
   int ans = n;
   for(int i = 1;i <= cnt && prime[i] * prime[i] <= n;++i)
      if(n % prime[i] == 0){
         ans -= ans / prime[i];
         while(n % prime[i] == 0) n /= prime[i];
      }
   if(n > 1) ans -= ans / n;
   return ans;
}
int n,m;
int main()
{
   getprime();
   while(scanf("%d%d",&m,&n) && (n || m)){
      ll ans = 0;
      for(int i = 1; i * i <= n;++i){
         if(n % i == 0)
            ans += i * i == n ? euler(i) * fpow(m,n / i) : euler(i) * fpow(m,n
                 / i) + euler(n / i) * fpow(m,i);
      }
      ans += n & 1 ? n * fpow(m,n / 2 + 1) : (fpow(m,n >> 1) + fpow(m,n / 2 + 1)
          ) * (n >> 1);
      printf("%lld\n",ans / (n << 1LL));
   }
   return 0;
}
```

4.7 大素数

- 包含分解，判断过程为 $O(n^{\frac{1}{4}})$

```
typedef long long ll;
int pri[] = {2,3,5,7,11,13,17,19,23,29,31};
ll minfacotr;
ll mult(ll a, ll b, ll mod) // 大数乘法
{
   ll ans = 0;
   while(b)
   {
      if (b&1)
      {
         ans+= a;
         if (ans >= mod)
            ans -= mod;
      }
      b >>= 1;
      a <<= 1;
```

```
        if (a >= mod)
            a-= mod;
    }
    return ans;
}
ll qpow(ll x, ll n, ll mod)
{
    ll ans = 1;
    while(n)
    {
        if (n&1)
            ans = mult(ans, x, mod);
        x = mult(x, x, mod);
        n >>= 1;
    }
    return ans;
}
bool wintness(ll n, ll a)
{
    ll p = qpow(a, n-1, n);
    if(p != 1)
        return false;
    ll s = n - 1;
    while(!(s&1)&& p == 1)
    {
        s >>= 1;
        p = qpow(a, s, n);
    }
    if (p == 1 || p == n - 1)
        return true;typedef long long ll;
int pri[] = {2,3,5,7,11,13,17,19,23,29,31};
ll minfacotr;
ll mult(ll a, ll b, ll mod) // 大数乘法
{
    ll ans = 0;
    while(b)
    {
        if (b&1)
        {
            ans+= a;
            if (ans >= mod)
                ans -= mod;
        }
        b >>= 1;
        a <<= 1;
        if (a >= mod)
            a-= mod;
    }
    return ans;
}
ll qpow(ll x, ll n, ll mod)
{
    ll ans = 1;
    while(n)
    {
        if (n&1)
            ans = mult(ans, x, mod);
```

```
        x = mult(x, x, mod);
        n >>= 1;
    }
    return ans;
}
bool wintness(ll n, ll a)
{
    ll p = qpow(a, n-1, n);
    if(p != 1)
        return false;
    ll s = n - 1;
    while(!(s&1)&& p =
    return false;
}
bool miller_rabin(ll n) // 判断素数
{
    if (n < 32)
    {
        for(int i = 0; i < 11; ++i)
            if(n == pri[i])
                return true;
        return false;
    }
    for(int i = 0 ; i < 10; ++i)
        if(!wintness(n,pri[i]))
            return false;
    return true;
}
ll gcd(ll a, ll b)
{
    return b ? gcd(b, a % b) : a;
}
ll pollard_rho(ll n, ll c) // Pollard_rho 大数分解
{
    ll x= rand() % n, y = x, i = 1, k = 2, d;
    while(1)
    {
        i++;
        x =(mult(x, x, n) + c) % n;
        d = gcd(y-x+n,n);
        if(d > 1 && d < n)
            return d;
        if(y == x)
            return n;
        if (i == k)
        {
            y = x;
            k <<= 1;
        }
    }
}
ll fac[maxn];
int cnt = 0;
void findfactor(ll n)
{
    if(miller_rabin(n))
    {
```

```
        minfacotr = min(minfacotr,n);// 求最小素数因子
        fac[++cnt] = n;//储存素数
        return ;
    }
    ll p = n;
    while(p >= n)
        p = pollard_rho(n, rand() % (n - 1) + 1);
    findfactor(p);
    findfactor(n / p);
}
```

4.8 线性基

```
struct L_B{
    ll d[64],p[64];
    int cnt;
    void init(){// 初始化
        memset(d,0,sizeof d);
        memset(p,0,sizeof p); // 单位基
        cnt = 0; // 单位基中1的数量
    }
    bool insert(ll val){ // 插入 如果x的1数位不存在则插入val，存在则赋值为0
        for(int i = 63;i >= 0;--i)
            if(val & (1LL << i)){
                if(!d[i]){
                    d[i] = val;
                    break;
                }
                val ^= d[i];
            }
        return val > 0;
    }
    ll query_max(){ // 查询最大异或和
        ll ret = 0;
        for(int i = 63;i >= 0;--i)
            if((ret ^ d[i]) > ret)
                ret ^= d[i];
        return ret;j >= 0;--j)
                if(d[i] & (1LL << j)) d[i] ^= d[j];
        for(int i = 0;i <= 63;++i)
            if(d[i]) p[cnt++] = d[i];
    }
    ll kth_query(ll k){ // 查询第k小值
        int ret = 0;
        if(k >= (1LL << cnt))
            return -1;
        for(int i = 63;i >= 0;--i)
            if(k & (1LL << i)) ret ^= p[i];
        return ret;
    }
}
L_B merge(const L_B &n1,const L_B &n2) // 暴力合并两个线性基
{
    L_B ret = n1;
```

```
    for(int i = 63;i >= 0;--i)
        if(n2.d[i]) ret.insert(n1.d[i]);
    return ret;
}
```

4.9 洲阁筛

4.10 计算素数 k 次幂前缀和

```
const int N = 1e6 + 7;
ll n,k,sqrtn;
int m;
ll a[N << 1],cnt;
int pos1[N],pos2[N];
ull g[N << 1]; // 构造函数
int pri[N],pcnt;
bool ispri[N];
void get_pri()
{
   for(int i = 2;i < N;++i){
      if(!ispri[i]) pri[++pcnt] = i;
      for(int j = 1;j <= pcnt && i * pri[j] < N;++j){
         ispri[i * pri[j]] = 1;
         if(i % pri[j] == 0) break;
      }
   }
}

inline int get_pos(ll x)
{
    return x <= sqrtn ? pos1[x] : pos2[n / x];
}
void Discretization(){ // 离散化处理
    for(ll i = 1,j;i <= n;i = j + 1){
        a[++cnt] = n / i;
        j = n / (n / i);
    }
    reverse(a + 1,a + 1 + cnt);
    for(int i = 1;i <= cnt;++i)
        if(a[i] <= sqrtn) pos1[a[i]] = i;
        else pos2[n / a[i]] = i;
}
void calc_g() // 预处理构造函数
{
    for(int i = 1;i <= cnt;++i) g[i] = a[i] - 1;
    for(int i = 1;i <= m;++i)
        for(int j = cnt;j >= 1 && a[j] >= pri[i] * pri[i];--j)
            g[j] -= g[get_pos(a[j] / pri[i])] - g[get_pos(pri[i] - 1)];
}
ull calc_h(ll i,ll j){ // 计算递归函数
    if(i <= 1) return 0;
    ull sum = 0;
    int res;
```

```
    for(res = j;res <= m && pri[res] * pri[res] <= i;++res)
        for(ll o = pri[res],e = 1;o <= i;o *= pri[res],++e)
            sum += (ull)(e * k + 1) * (calc_h(i / o,res + 1) + 1);
    if(pri[res - 1] <= i)
        sum += (ull)(k + 1) * (g[get_pos(i)] - g[get_pos(pri[res - 1])]);
    return sum;
}
int t;
int main()
{
    get_pri();
    scanf("%lld%lld",&n,&k);
    sqrtn = (ll)sqrt(n);cnt = 0;
    m = upper_bound(pri + 1,pri + 1 + pcnt,sqrtn) - pri - 1;
    Discretization();
    calc_g();
    ull ans = (ull)(k + 1) * (g[get_pos(n)] - m);
    for(int i = 1;i <= m;++i)
        for(ll j = pri[i],e = 1;j <= n;j *= pri[i],e++)
            ans += (ull)(e * k + 1) * (calc_h(n / j,i + 1) + 1);
    ans++;
    printf("%ull\n",ans);
    return 0;
}
```

4.11 自然幂数和

```
void init()
{
//预处理组合数
    for(int i = 0; i < N;++i){
        C[i][0] = C[i][i] = 1;
        if (i == 0)
            continue;
        for (int j = 1; j < i;++j)
            C[i][j] = (C[i - 1][j] % mod + C[i - 1][j - 1] % mod) % mod;
    }
//预处理逆元
    inv[1] = 1;
    for (int i = 2; i < N; ++i)
        inv[i] = (mod - mod / i) * inv[mod % i] % mod;
//预处理伯努利数
    B[0] = 1;
    for (int i = 1; i < N; ++i)
    {
        ll ans = 0;
        if (i == N - 1)
            break;
        for (int j = 0; j < i; ++j)
        {
            ans += C[i + 1][j] * B[j];
            ans %= mod;
        }
        ans *= -inv[i + 1];
```

```
        ans = (ans % mod + mod) % mod;
        B[i] = ans;
    }
}
ll Work(int k)
{
    ll ans = inv[k + 1];
    ll sum = 0;
    for(int i=1; i<=k+1; ++i)
    {
        sum += C[k + 1][i] * tmp[i] % mod * B[k + 1 - i] % mod;
        sum %= mod;
    }
    ans *= sum;
    ans %= mod;
    return ans;
}
int main()
{
    int t;
    init();
    scanf("%d", &t);
    while(t--)
    {
        int k;
        scanf("%lld %d", &n, &k);
        n %= mod;
        tmp[0] = 1;
        for(int i = 1; i < N; ++i)
            tmp[i] = tmp[i - 1] * (n + 1) % mod;
        printf("%lld\n", Work(k));
    }
    return 0;
}
```

4.12 组合数打表

```
ll f[N],inv[N];
inline ll C(ll a,ll b){return a < 0 || b < 0 || a < b ? 0 : f[a] * inv[b] % mod *
      inv[a - b] % mod;}
void get_table(int index)
{
    f[0] = inv[0] = 1;
    for(int i = 1;i <= index;++i) f[i] = f[i - 1] * i % mod;
    inv[index] = fpow(f[index],mod - 2);
    for(;index;--index)
        inv[index - 1] = inv[index] * index % mod;
}
```

4.13 杜教筛

```
const int N = 4e6 + 7;
const int mod = 1e9 + 7;
int prime[N],cnt;
ll g[N],phi[N];
bool isprime[N];
ll n,m;
inline ll gao(ll x)
{
    x %= mod;
    return (x + 1) % mod * x % mod * inv2 % mod;
}
void get_phi() // 预处理phi或者Mobius,n^(2/3)的前缀和
{
   phi[1] = 1;
   for(int i = 2;i < N;++i){
       if(!isprime[i]){
            prime[++cnt] = i;
            phi[i] = i - 1;
        }
       for(int j = 1;j <= cnt && i * prime[j] < N;++j)
       {
          isprime[i * prime[j]] = 1;
          if(i % prime[j] == 0){
             phi[i * prime[j]] = phi[i] * prime[j];
             break;
          }
          phi[i * prime[j]] = phi[i] * (prime[j] - 1);
       }
   }
   for(int i = 1;i < N;++i) g[i] = (g[i - 1] + phi[i] * i % mod) % mod;
}

inline ll cal(ll x) //
{
    x %= mod;
    return x * (2 * x + 1) % mod * (x + 1) % mod * inv6 % mod;
}
map<ll ,ll>mp; // 存已到达情况
ll solve(ll pos) // 递归
{
    if(pos < N) return g[pos];
    if(mp[pos]) return mp[pos];
    ll res = cal(pos),last;
    for(ll i = 2;i <= pos;i = last + 1){
        last = pos / (pos / i);
        res = ((res - (gao(last) - gao(i - 1)) * solve(pos / i) % mod) % mod + mod
             ) % mod;
    }
    mp[pos] = res;
    return res;
}
ll work(ll pos)
{
    ll last,ans = 0;
    for(ll i = 1;i <= pos;i = last + 1){
        last = pos / (pos / i);
        ans = (ans + (last - i + 1) % mod * solve(pos / i) % mod) % mod + mod;
```

```
        ans %= mod;
    }
    return (ans + pos) % mod * inv2 % mod;
}
int main()
{
    get_phi();
    scanf("%lld%lld",&n,&m);
    printf("%lld\n",((work(m) - work(n - 1)) % mod + mod) % mod);
    return 0;
}
```

4.14 二次剩余

4.14.1 解 $x^2 = a(mod\, p)$

```
struct T{
    ll p, d;
};
ll w;
//二次域乘法
T multi_er(T a, T b, ll m)
{
    T ans;
    ans.p = (a.p * b.p % m + a.d * b.d % m * w % m) % m;
    ans.d = (a.p * b.d % m + a.d * b.p % m) % m;
    return ans;
}
//二次域上快速幂
T power(T a, ll b, ll m)
{
    T ans;
    ans.p = 1;
    ans.d = 0;
    while(b){
        if(b & 1)
        ans = multi_er(ans, a, m);
        b >>= 1;
        a = multi_er(a, a, m);
    }
    return ans;
}
//求勒让德符号
ll Legendre(ll a, ll p)
{
    return qpow(a, (p-1)>>1, p);
}
ll mod(ll a, ll m)
{
    a %= m;
    if(a < 0) a += m;
    return a;
}
```

```
ll Solve(ll n,ll p)
{
    if(p == 2) return 1;
    if (Legendre(n, p) + 1 == p)
        return -1;
    ll a = -1, t;
    while(true) {
        a = rand() % p;
}
int main()
{
    int t;
    scanf("%d", &t);
    while(t--){
        int n, p;
        scanf("%d %d",&n,&p);
        n %= p;
        int a = Solve(n, p);//x * x = n % p
        if(a == -1) {
            puts("No root");
            continue;
        }
        int b = p - a;
        if(a > b) swap(a, b);
        if(a == b)
            printf("%d\n",a);
        else printf("%d %d\n",a,b);
    }
    return 0;
}
```

4.15 高精度

```
struct BigInteger {
    typedef unsigned long long ll;

    static const int BASE = 100000000;
    static const int WIDTH = 8;
    vector<int> s;

    BigInteger& clean(){while(!s.back()&&s.size()>1)s.pop_back(); return *this;}
    BigInteger(ll num = 0) {*this = num;}
    BigInteger(string s) {*this = s;}
    BigInteger& operator = (long long num) {
        s.clear();
        do {
            s.push_back(num % BASE);
            num /= BASE;
        } while (num > 0);
        return *this;
    }
    BigInteger& operator = (const string& str) {
        s.clear();
        int x, len = (str.length() - 1) / WIDTH + 1;
        for (int i = 0; i < len; i++) {
```

```
            int end = str.length() - i*WIDTH;
            int start = max(0, end - WIDTH);
            sscanf(str.substr(start,end-start).c_str(), "%d", &x);
            s.push_back(x);
        }
        return (*this).clean();
    }

    BigInteger operator + (const BigInteger& b) const {
        BigInteger c; c.s.clear();
        for (int i = 0, g = 0; ; i++) {
            if (g == 0 && i >= s.size() && i >= b.s.size()) break;
            int x = g;
            if (i < s.size()) x += s[i];
            if (i < b.s.size()) x += b.s[i];
            c.s.push_back(x % BASE);
            g = x / BASE;
        }
        return c;
    }
    BigInteger operator - (const BigInteger& b) const {
        assert(b <= *this); // 减数不能大于被减数
        BigInteger c; c.s.clear();
        for (int i = 0, g = 0; ; i++) {
            if (g == 0 && i >= s.size() && i >= b.s.size()) break;
            int x = s[i] + g;
            if (i < b.s.size()) x -= b.s[i];
            if (x < 0) {g = -1; x += BASE;} else g = 0;
            c.s.push_back(x);
        }
        return c.clean();
    }
    BigInteger operator * (const BigInteger& b) const {
        int i, j; ll g;
        vector<ll> v(s.size()+b.s.size(), 0);
        BigInteger c; c.s.clear();
        for(i=0;i<s.size();i++) for(j=0;j<b.s.size();j++) v[i+j]+=ll(s[i])*b.s[j];
        for (i = 0, g = 0; ; i++) {
            if (g ==0 && i >= v.size()) break;
            ll x = v[i] + g;
            c.s.push_back(x % BASE);
            g = x / BASE;
        }
        return c.clean();
    }
    BigInteger operator / (const BigInteger& b) const {
        assert(b > 0); // 除数必须大于0
        BigInteger c = *this;   // 商:主要是让c.s和(*this).s的vector一样大
        BigInteger m;           // 余数:初始化为0
        for (int i = s.size()-1; i >= 0; i--) {
            m = m*BASE + s[i];
            c.s[i] = bsearch(b, m);
            m -= b*c.s[i];
        }
        return c.clean();
    }
    BigInteger operator % (const BigInteger& b) const { //方法与除法相同
```

```
        BigInteger c = *this;
        BigInteger m;
        for (int i = s.size()-1; i >= 0; i--) {
            m = m*BASE + s[i];
            c.s[i] = bsearch(b, m);
            m -= b*c.s[i];
        }
        return m;
    }

    int bsearch(const BigInteger& b, const BigInteger& m) const{
        int L = 0, R = BASE-1, x;
        while (1) {
            x = (L+R)>>1;
            if (b*x<=m) {if (b*(x+1)>m) return x; else L = x;}
            else R = x;
        }
    }
    BigInteger& operator += (const BigInteger& b) {*this = *this + b; return *this;}
    BigInteger& operator -= (const BigInteger& b) {*this = *this - b; return *this;}
    BigInteger& operator *= (const BigInteger& b) {*this = *this * b; return *this;}
    BigInteger& operator /= (const BigInteger& b) {*this = *this / b; return *this;}
    BigInteger& operator %= (const BigInteger& b) {*this = *this % b; return *this;}

    bool operator < (const BigInteger& b) const {
        if (s.size() != b.s.size()) return s.size() < b.s.size();
        for (int i = s.size()-1; i >= 0; i--)
            if (s[i] != b.s[i]) return s[i] < b.s[i];
        return false;
    }
    bool operator >(const BigInteger& b) const{return b < *this;}
    bool operator<=(const BigInteger& b) const{return !(b < *this);}
    bool operator>=(const BigInteger& b) const{return !(*this < b);}
    bool operator!=(const BigInteger& b) const{return b < *this || *this < b;}
    bool operator==(const BigInteger& b) const{return !(b < *this) && !(b > *this);}
};

ostream& operator << (ostream& out, const BigInteger& x) {
    out << x.s.back();
    for (int i = x.s.size()-2; i >= 0; i--) {
        char buf[20];
        sprintf(buf, "%08d", x.s[i]);
        for (int j = 0; j < strlen(buf); j++) out << buf[j];
    }
    return out;
}

istream& operator >> (istream& in, BigInteger& x) {
    string s;
    if (!(in >> s)) return in;
    x = s;
```

```
    return in;
}
```

4.16 反素数

4.16.1 求小于 n 并且因子个数最多的那个数

```
int pri[]= {2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61};
ll n, ans, tmp;
const ll INF = 0x3f3f3f3f;
void dfs(int step, ll sum, ll num) //num 为因子个数
{
    if(step == 16) return;
    if(num > tmp){
        ans = sum;
        tmp = num;
    }
    if(sum > n) return ;

    if(num == tmp && sum < ans)//因子个数一样,选取较小的那个数
        ans = sum;
    for(int i = 1; i <= 63; ++i){
        if(n / pri[step] < sum) break;
        dfs(step + 1, sum *= pri[step], num * (i + 1));
    }
}

//求一个最小的正整数,使得它的因子个数为 n
int p[16] = {2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53};
void dfs(int dept,ull tmp,int num)
{
    if(num > n) return;
    if(num == n && ans > tmp)
        ans = tmp;
    for(int i=1; i <= 63;++i){
        if(ans / p[dept] < tmp)
            break;
        dfs(dept+1,tmp *= p[dept],num * (i + 1));
    }
}
```

4.17 求区间 [1,N] 的质数的个数 (1≤N≤10^11)

```
ll f[340000],g[340000],n;
    void init(){
        ll i,j,m;
        for(m=1;m*m<=n;++m)f[m]=n/m-1;
        for(i=1;i<=m;++i)g[i]=i-1;
        for(i=2;i<=m;++i){
            if(g[i]==g[i-1])continue;
```

```
        for(j=1;j<=min(m-1,n/i/i);++j){
            if(i*j<m)f[j]-=f[i*j]-g[i-1];
            else f[j]-=g[n/i/j]-g[i-1];
        }
        for(j=m;j>=i*i;--j)g[j]-=g[j/i]-g[i-1];
    }
}
int main(){
    while(scanf("%I64d",&n)!=EOF){
        init();
        cout<<f[1]<<endl;
    }
    return 0;
}
```

O(n^(2/3))

```
const int N = 5e6 + 2;
bool np[N];
int prime[N], pi[N];
int getprime()
{
    int cnt = 0;
    np[0] = np[1] = true;
    pi[0] = pi[1] = 0;
    for(int i = 2; i < N; ++i)
    {
        if(!np[i]) prime[++cnt] = i;
        pi[i] = cnt;
        for(int j = 1; j <= cnt && i * prime[j] < N; ++j)
        {
            np[i * prime[j]] = true;
            if(i % prime[j] == 0) break;
        }
    }
    return cnt;
}
const int M = 7;
const int PM = 2 * 3 * 5 * 7 * 11 * 13 * 17;
int phi[PM + 1][M + 1], sz[M + 1];
void init()
{
    getprime();
    sz[0] = 1;
    for(int i = 0; i <= PM; ++i) phi[i][0] = i;
    for(int i = 1; i <= M; ++i)
    {
        sz[i] = prime[i] * sz[i - 1];
        for(int j = 1; j <= PM; ++j) phi[j][i] = phi[j][i - 1] - phi[j / prime[i
            ]][i - 1];
    }
}
int sqrt2(LL x)
{
    LL r = (LL)sqrt(x - 0.1);
    while(r * r <= x) ++r;
    return int(r - 1);
}
```

```
int sqrt3(LL x)
{
    LL r = (LL)cbrt(x - 0.1);
    while(r * r * r <= x) ++r;
    return int(r - 1);
}
LL getphi(LL x, int s)
{
    if(s == 0) return x;
    if(s <= M) return phi[x % sz[s]][s] + (x / sz[s]) * phi[sz[s]][s];
    if(x <= prime[s]*prime[s]) return pi[x] - s + 1;
    if(x <= prime[s]*prime[s]*prime[s] && x < N)
    {
        int s2x = pi[sqrt2(x)];
        LL ans = pi[x] - (s2x + s - 2) * (s2x - s + 1) / 2;
        for(int i = s + 1; i <= s2x; ++i) ans += pi[x / prime[i]];
        return ans;
    }
    return getphi(x, s - 1) - getphi(x / prime[s], s - 1);
}
LL getpi(LL x)
{
    if(x < N) return pi[x];
    LL ans = getphi(x, pi[sqrt3(x)]) + pi[sqrt3(x)] - 1;
    for(int i = pi[sqrt3(x)] + 1, ed = pi[sqrt2(x)]; i <= ed; ++i) ans -= getpi(x / prime[i]) - i + 1;
    return ans;
}
LL lehmer_pi(LL x)
{
    if(x < N) return pi[x];
    int a = (int)lehmer_pi(sqrt2(sqrt2(x)));
    int b = (int)lehmer_pi(sqrt2(x));
    int c = (int)lehmer_pi(sqrt3(x));
    LL sum = getphi(x, a) +(LL)(b + a - 2) * (b - a + 1) / 2;
    for (int i = a + 1; i <= b; i++)
    {
        LL w = x / prime[i];
        sum -= lehmer_pi(w);
        if (i > c) continue;
        LL lim = lehmer_pi(sqrt2(w));
        for (int j = i; j <= lim; j++) sum -= lehmer_pi(w / prime[j]) - (j - 1);
    }
    return sum;
}
int main()
{
    init();
    LL n;
    while(~scanf("%lld",&n))
    {
        printf("%lld\n",lehmer_pi(n));
    }
    return 0;
}
```

4.18 高斯消元

```
const int N = 307;
int x[N],a[N][N];// x[N]解集, a[N][N]系数
bool free_x[N];
int gcd(int a,int b){return b ? gcd(b,a % b) : a;}
int lcm(int a,int b){return a / gcd(b,a % b) * b;}
int Gauss(int equ,int var)//equ个方程, var个变元
{
    int free_x_num,i,j,row,max_r,col;//row表示行,col表示列,max_r表示列最大的行,free_x_num变元数量
    int free_index,LCM,ta,tb,temp;// free_index变元下标
    for(i = 0;i <= var;++i){
        x[i] = 0;
        free_x[i] = true;//第i个元素是否是变元
    }
    for(row = 0,col = 0;row < equ && col < var;++row,++col){
        max_r = row;
        //找到col最大的行, 进行交换 (除法时减小误差)
        for(i = row + 1;i < equ;++i) if(abs(a[i][col]) > abs(a[max_r][col])) max_r = i;
        //与第row行交换
        if(max_r != row) for(j = row;j < var + 1;++j) swap(a[row][j],a[max_r][j]);
        if(a[row][col]==0){
            //说明该col列第row行以下全是0了, 则处理当前行的下一列.
            row--;
            continue;
        }
        for(i = row + 1;i < equ;++i)//枚举被删行
            if(a[i][col]){
                LCM = lcm(abs(a[i][col]),abs(a[row][col]));
                ta = LCM / abs(a[i][col]);
                tb = LCM / abs(a[row][col]);
                if(a[i][col] * a[row][col] < 0)tb = -tb;//异号的情况是相加
                for(j = col;j < var + 1;++j)
                    a[i][j] = a[i][j] * ta - a[row][j] * tb;
            }
        /*求解小数解,防止溢出
        for(int i = row + 1; i < equ; ++i)
            if(fabs(a[i][col]) > eps){
                double t1 = a[i][col]/a[row][col];
                for(int j = col; j <= var;++j) a[i][j] -= a[row][j] * t1;
            }*/
    }
    for (i = row;i < equ;++i) if(a[i][col]) return -1; // 无解
    if (row < var){// 多解
        for(i = row - 1;i >= 0;--i){
            free_x_num = 0;
            for (j = 0;j < var;++j)
                if(a[i][j] && free_x[j]) free_x_num++,free_index = j;
            if (free_x_num > 1) continue; // 无法求解出确定的变元.
            temp = a[i][var];
            for (j = 0;j < var;++j) if (a[i][j] && j != free_index) temp -= a[i][j] * x[j];
            x[free_index] = temp / a[i][free_index]; //求出该变元.
```

```
            free_x[free_index] = 0; //该变元是确定的.
        }
        return var - row; //自由变元有 var - row 个.
    }
    for (i = var - 1; i >= 0;--i){// 唯一解
        temp = a[i][var];
        for (j = i + 1;j < var;++j)
            if (a[i][j]) temp -= a[i][j] * x[j];
        if (temp % a[i][i]) return -2; // 说明有浮点数解,但无整数解.
        x[i] = temp / a[i][i];
    }
    return 0;
}
```

4.19 高斯消元解异或方程

```
int Guass_xor(int equ,int var)
{
    int row,col;
    for(row = 0, col = 0; row < equ && col < var; ++row, ++col){
        int maxr = row;
        for(int i = row; i < equ; ++i)
            if(a[i][col] > a[maxr][col]) maxr = i;
            if(maxr != row) for(int j = col; j <= var; ++j) swap(a[maxr][j],a[row][j]);
            if(!a[row][col]){
                row--;
                continue;
            }
            for(int i = row + 1; i < equ; ++i)
                if(a[i][col])
                    for(int j = col; j <= var; ++j) a[i][j] ^= a[row][j];
    }
    for(int i = row; i < equ; ++i) if(a[i][col]) return -1;//无可行解
    if(row < var) // 存在多解
            for(int i = row - 1; i >= 0; --i){
                int num = 0, index;
                for(int j = 0; j < var; ++j)
                    if(a[i][j] && vis[j]) num++, index = j;
                if(num > 1) continue;
                for(int j = 0; j < var; ++j)
                    if(j != index) x[index] ^= (a[i][j] && x[j]);
                vis[index] = false;
            }
    for(int i = var - 1; i >= 0; --i){ //唯一解
        x[i] = a[i][var];
        for(int j = i + 1; j < var; ++j) x[i] ^=(a[i][j] && x[j]);
    }
    return 0;
}
```

4.20 矩阵类（快速幂）

```
int sz;
struct Matrix{
    ll a[10][10];
    Matrix(){
        for(int i = 0;i < sz;++i)
            for(int j = 0;j < sz;++j)
                if(i == j) a[i][j] = 1;
                else a[i][j] = 0;
    }
    void init(){
        memset(a,0,sizeof a);
    }
    void show(){
        for(int i = 0;i < sz;++i)
            for(int j = 0;j < sz;++j)
                dd(i),dd(j),de(a[i][j]);
    }
    void Relation_matrix(ll *s){
        init();
        for(int i = 0;i < sz;++i){
            if(i + 1 < sz) a[i][i + 1] = 1;
            a[sz - 1][i] = s[i];
        }
    }
    Matrix operator *(const Matrix& tmp) const{
        Matrix res;
        res.init();
        for(int i = 0;i < sz;++i)
            for(int j = 0;j < sz;++j)
                for(int k = 0;k < sz;++k)
                    res.a[i][j] = (res.a[i][j] + a[i][k] * tmp.a[k][j] % mod) % mod;
        return res;
    }
    Matrix operator +(Matrix tmp){
        for(int i = 0;i < sz;++i)
            for(int j = 0;j < sz;++j)
                a[i][j] += tmp.a[i][j];
    }
    friend Matrix fpow(Matrix a,ll b){
        Matrix res;
        while(b){
            if(b & 1) res = res * a;
            a = a * a;
            b >>= 1;
        }
        return res;
    }
    ll get_fib(){
        Matrix tmp;
        tmp.init();
        for(int i = 0;i < sz;++i)
            tmp.a[i][0] = 1;
        Matrix ans = *this;
        ans = ans * tmp;
        return ans.a[0][0];
```

```
    }
};
```

4.21 类欧几里得

- 求解 $f(a,b,c,n)=\sum_{i=0}^{n}\lfloor\frac{a*i+b}{c}\rfloor$

```
ll f(ll a,ll b,ll c,ll n)
{
    if(a == 0)
        return (b / c) * (n + 1) % mod;
    if(a >= c)
        return (n * (n + 1) / 2 % mod * (a / c) % mod + f(a % c,b,c,n)) % mod;
    if(b >= c)
        return ((b / c) * (n + 1) % mod + f(a,b % c,c,n)) % mod;
    //ll m = (a * n + b) / c % mod;
    ll m = (n / c * a + (n % c * a + b) / c);
    return ((m % mod) * (n % mod) % mod - f(c,c - b - 1,a,m - 1) + mod) % mod;
}
```

4.22 牛顿迭代

```
#define f(x) () //原函数
#define f2(x) () //导函数
double Newton(double x){
    for(int i = 0;i < 100;++i){
        x = x - f(x) / f2(x); //牛顿迭代法
        if(fabs(f(x)) < eps) return x;
    }
    return -1;
}
```

4.23 母函数

- n 个人分成任意组, 每组人数必须为素数, 有多少种分法,n < 150;

```
void solve()
{
    memset(a, 0, sizeof a);
    for(int i = 0; i <= 160; i += 2)
        a[i] = 1;
    memset(b, 0, sizeof b);
    for(int i = 1; pri[i] <= 150;++i){
        for(int j = 0; j <= 150;++j)
            for(int k = 0;k + j <= 150;k += pri[i])
                b[k + j] += a[j];
        for(int j = 0; j <= 150; ++j){
            a[j] = b[j];
```

```
            b[j] = 0;
        }
    }
}
```

4.24 莫比乌斯反演

- 求有多少个数对 (x,y) , 满足 a ≤ x ≤ b , c ≤ y ≤ d , 且 gcd(x,y) = 1

```
void Mobius()
{
    cnt = 0;
    mu[1] = 1;sum[1] = 1;
    for(int i = 2; i < maxn; ++i){
        if(!vis[i]){
            prime[++cnt] = i;
            mu[i] = -1;
        }
        for(int j = 1; j <= cnt; ++j){
            if(i * prime[j] >= maxn) break;
            vis[i * prime[j]] = true;
            if(i % prime[j]== 0){
                mu[i * prime[j]] = 0;
                break;
            }
            mu[i * prime[j]] = -mu[i];
        }
        sum[i] = sum[i-1] + mu[i];
    }
}
ll solve(int b, int d)
{
    if(b > d) swap(b, d);
    ll ans = 0,pos;
    for(int i = 1; i <= b; i = pos+1){
        pos = min(b / (b / i),d / (d / i));
        ans += 1LL * (b / i) * (d / i) * (sum[pos] - sum[i - 1]);
    }
    return ans;
}
```

4.25 斐波那契广义循环节

- $f(n) = af(n-1) + bf(n-2) \quad f(1) = c \quad f(2) = d$

```
const int N = 2;
const ll mod = 1000000007;

ll fac[2][505];
int cnt,ct;

ll pri[6] = {2, 3, 7, 109, 167, 500000003};
```

```
ll num[6] = {4, 2, 1, 2, 1, 1};

struct Matrix
{
    ll m[N][N];
} ;

Matrix A;
Matrix I = {1, 0, 0, 1};

Matrix multi(Matrix a,Matrix b)
{
    Matrix c;
    for(int i=0; i<N; i++)
    {
        for(int j=0; j<N; j++)
        {
            c.m[i][j] =0;
            for(int k=0; k<N; k++)
            {
                c.m[i][j] += a.m[i][k] * b.m[k][j];
                c.m[i][j] %= mod;
            }
        }
    }
    return c;
}

Matrix power(Matrix A,ll n)
{
    Matrix ans = I, p = A;
    while(n)
    {
        if(n & 1)
        {
            ans = multi(ans,p);
            n--;
        }
        n >>= 1;
        p = multi(p,p);
    }
    return ans;
}

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

```
    }
    return ans;
}

ll Legendre(ll a,ll p)
{
    ll t = quick_mod(a,(p-1)>>1);
    if(t == 1) return 1;
    return -1;
}

void dfs(int dept,ll product = 1)
{
    if(dept == cnt)
    {
        fac[1][ct++] = product;
        return;
    }
    for(int i=0; i<=num[dept]; i++)
    {
        dfs(dept+1,product);
        product *= pri[dept];
    }
}

bool OK(Matrix A,ll n)
{
    Matrix ans = power(A,n);
    return ans.m[0][0] == 1 && ans.m[0][1] == 0 &&
           ans.m[1][0] == 0 && ans.m[1][1] == 1;
}

int main()
{
    fac[0][0] = 1;
    fac[0][1] = 2;
    fac[0][2] = 500000003;
    fac[0][3] = 1000000006;
    ll a,b,c,d;
    while(cin>>a>>b>>c>>d)
    {
        ll t = a * a + 4 * b;
        A.m[0][0] = a;
        A.m[0][1] = b;
        A.m[1][0] = 1;
        A.m[1][1] = 0;
        if(Legendre(t,mod) == 1)
        {
            for(int i=0; i<4; i++)
            {
                if(OK(A,fac[0][i]))
                {
                    cout<<fac[0][i]<<endl;
                    break;
                }
            }
        }
```

```
        else
        {
            ct = 0;
            cnt = 6;
            dfs(0,1);
            sort(fac[1],fac[1]+ct);
            for(int i=0;i<ct;i++)
            {
                if(OK(A,fac[1][i]))
                {
                    cout<<fac[1][i]<<endl;
                    break;
                }
            }
        }
    }
    return 0;
}
```

4.26 robenius-equations

use by indiewar

```
//sum_{i=1}^{n} a[i] * x[i] = b, x[i] >= 0
//given a[i] and b, check solution existing
typedef std::pair<LL, int> PLI;
int get_vals(int n, int a[]) {
  std::priority_queue<PLI, std::vector<PLI>, std::greater<PLI>> Q;
  int X = *std::min_element(a, a + n);
  std::fill(dis, dis + X, inf);
  dis[0] = 0; Q.push(PLI(0, 0));
  while (!Q.empty()) {
    PLI nw = Q.top(); Q.pop();
    int u = nw.second;
    LL val = nw.first;
    if (dis[u] != val) continue;
    for (int i = 0; i < n; ++ i) {
      LL tmp = val + a[i];
      int v = tmp % X;
      if (dis[v] > tmp) {
        dis[v] = tmp;
        Q.push(PLI(tmp, v));
      }
    }
  }
  return X;
}
bool ok(int n, int a[], int b) {
  int X = get_vals(n, a);
  return dis[b % X] <= b;
}
```

4.27 logarithm

```
// 求离散对数，即 a^x % m = r的最小非负整数解x，返回-1说明无解
int64 log(int64 a, int64 r, int64 m){
  if (r >= m) return -1;
  int64 i, g, x, c = 0, at = int(2 + sqrt(m));
  for (i = 0, x = 1 % m; i < 50; ++i) {
    if (x == r) return i;
    x = int64(x) * a % m;
  }
  for (g = x = 1; gcd(int64(x) * a % m, m) != g; ++c) {
    x = x * a % m;
    g = gcd(x, m);
  }
  if (r % g) return -1;
  if (x == r) return c;
  std::unordered_map<int64, int64> u;
  g = euler_phi(m / g), u[x] = 0;
  g = pow_mod(a, g - at % g, m);
  for (i = 1; i < at; ++i){ // Baby Step
    x = x * a % m;
    u.emplace(x, i);
    if (x == r) return c + i;
  }
  for (i = 1; i < at; ++i) { // Giant Step
    r = r * g % m;
    auto t = u.find(r);
    if (t != u.end()) return c + i * at + t->second;
  }
  return -1;
}
```

4.28 拉格朗日插值

```
const int N = 3010, P = 998244353;

int n;
int po[N], pr[N];

int norm(int x) { return x >= P ? x - P : x; }

void exGcd(int a, int b, int& x, int& y) {
    if (!b) {
        x = 1;
        y = 0;
        return;
    }
    exGcd(b, a % b, y, x);
    y -= a / b * x;
}

int inv(int a) {
    int x, y;
    exGcd(a, P, x, y);
```

```
    return norm(x + P);
}

int eval(int a[], int n, int x) {
    int ret = 0;
    for (int i = n; i >= 0; --i) ret = (ret * (ll)x + a[i]) % P;
    return ret;
}

int main() {
    int s,n, k, x, y;
    scanf("%d%d%d%d", &s, &k, &x, &y);
    x %= P;
    n = 1;
    po[0] = y;
    pr[1] = 1;
    pr[0] = norm(P - x);
    rep(_,0,s-1)
    {
        scanf("%d%d",&x,&y);
        int v = (y + P - eval(po, n - 1, x)) * (ll)inv(eval(pr, n, x)) % P;
        for (int i = 0; i <= n; ++i) po[i] = (po[i] + pr[i] * (ll)v) % P;
        for (int i = ++n; i; --i) pr[i] = (pr[i - 1] + pr[i] * (ll)(P - x)) % P;
        pr[0] = pr[0] * (ll)(P - x) % P;
    }
    printf("%d\n", eval(po, n - 1, k));
    return 0;
}
```

- 给 f(0),f(1)...f(n), 求 f(m),f(m+1)...,f(m+n)

```
ll pow_mod(ll x, int k) {
    ll ans = 1;
    while (k) {
        if (k & 1)
            ans = ans * x % MOD;
        x = x * x % MOD;
        k >>= 1;
    }
    return ans;
}

ll facd[200005], facv[200005];
ll upd[200005], upv[200005];

void pre(int n, int m) {
    facd[0] = upd[0] = 1;
    for (int i = 1; i <= n; i++) {
        facd[i] = facd[i - 1] * i % MOD;
        upd[i] = upd[i - 1] * (m + i) % MOD;
    }
    facv[n] = pow_mod(facd[n], MOD - 2);
    upv[n] = pow_mod(upd[n], MOD - 2);
    for (int i = n - 1; i >= 0; i--) {
        facv[i] = facv[i + 1] * (i + 1) % MOD;
        upv[i] = upv[i + 1] * (m + i + 1) % MOD;
    }
}
```

```
const int Maxn = 1 << 19;

ll *w[19];

void ntt_init() {
    for (int i = 2, t = 0; i <= Maxn; i <<= 1, t++) {
        w[t] = new ll[i >> 1];
        ll wn = pow_mod(3, (MOD - 1) / i);
        w[t][0] = 1;
        for (int j = 1; j < (i >> 1); j++) w[t][j] = w[t][j - 1] * wn % MOD;
    }
}

void rev(ll *p, int len) {
    int j = len >> 1;
    for (int i = 1; i < len - 1; i++) {
        if (i < j)
            swap(p[i], p[j]);
        int k = len >> 1;
        while (j >= k) {
            j -= k;
            k >>= 1;
        }
        if (j < k)
            j += k;
    }
}

void ntt(ll *p, int len, int check) {
    rev(p, len);
    for (int i = 2, t = 0; i <= len; i <<= 1, t++)
        for (int j = 0; j < len; j += i)
            for (int k = j; k < j + (i >> 1); k++) {
                ll u = p[k];
                ll v = w[t][k - j] * p[k + (i >> 1)];
                p[k] = (u + v) % MOD;
                p[k + (i >> 1)] = (u - v) % MOD;
            }
    if (check == -1) {
        reverse(p + 1, p + len);
        ll nev = pow_mod(len, MOD - 2);
        for (int i = 0; i < len; i++) p[i] = (p[i] + MOD) * nev % MOD;
    }
}

int num[100005];
ll p[Maxn], q[Maxn];

int main() {
    ntt_init();
    int n, m;
    scanf("%d%d", &n, &m);
    pre(n << 1, m - n);
    for (int i = 0; i <= n; i++) scanf("%d", &num[i]);
    int len = 1;
    while (len < ((n + 1) << 1)) len <<= 1;
```

```cpp
    for (int i = 0; i <= n; i++) {
        p[i] = num[i] * facv[i] % MOD;
        q[i] = ((i & 1) ? MOD - 1 : 1) * facv[i] % MOD;
    }
    for (int i = n + 1; i < len; i++) p[i] = q[i] = 0;
    ntt(p, len, 1);
    ntt(q, len, 1);
    for (int i = 0; i < len; i++) p[i] = p[i] * q[i] % MOD;
    ntt(p, len, -1);
    len <<= 1;
    for (int i = n + 1; i < len; i++) p[i] = 0;
    for (int i = 0; i <= (n << 1); i++) q[i] = upv[i];
    for (int i = (n << 1) + 1; i < len; i++) q[i] = 0;
    ntt(p, len, 1);
    ntt(q, len, 1);
    for (int i = 0; i < len; i++) p[i] = p[i] * q[i] % MOD;
    ntt(p, len, -1);
    for (int i = 0; i <= n; i++) printf("%lld ", p[n + i] * upd[n + i] % MOD);
    printf("\n");
    return 0;
}
```

4.29 Cantor_expansion

```cpp
int n, m, c[1000005], a[1000005], ans = 1;
#define lowbit(x) x & -x
void add(int x, int v) {
    for (; x <= n; x += lowbit(x)) c[x] += v;
}
int sum(int x) {
    int ans = 0;
    for (; x; x -= lowbit(x)) ans += c[x];
    return ans;
}
int main() {
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> a[i];
    for (int i = n, fac = 1; i >= 1; i--, fac = 1ll * fac * (n - i) % mod) {
        ans = (ans + 1ll * sum(a[i] - 1) * fac % mod) % mod;
        add(a[i], 1);
    }
    cout << ans << endl;
}
```

4.30 无名小定理

- 1. 设 $x = gcd(\sum_{i=1}^{n-1} C(n,i))$
- 当 n 是素数是 x = n (根据 1.)
- 当 n 有多个素因子 x = 1(根据 1.)

- n 只有一个素因子，答案为该素因子 (根据 1.)
- 当 a > b , gcd(a,b) = 1, 那么 $ gcd(a^m − b^m, a^n − b^n) = a^{gcd(m,n)} − b^{gcd(n,m)}$
- $(n+1)*lcm(\sum_{i=0}^{n}C(n,i)) = lcm(\sum_{i=1}^{n+1}i)$
- 判断组合数 C(n,m) 的奇偶性, 当 n & m == m 为奇数, 反之就是偶数.
- 三角形求圆半径: 边长为 a, b, c 的三角形面积为 S, 则外接圆半径为 $a*b*c/(4S)$, 内切圆半径为 $2S/(a+b+c)$

4.31 自然常数

e= 2.7182818284590452353602874713526624

4.32 欧拉常数

C = 0.57721566490153286060651209

4.33 错排公式

$D_1 = 0$
$D_2 = 1$
$D_n = (n-1)(D_{n-1} + D_{n-2})$

4.34 伯努利数

$ F_0 = 1, F_n = _{i=0}^{n-1}C(n,i)F_i$

4.35 自然幂数和

$\sum_{i=1}^{n} i^k = \frac{1}{k+1}\sum_{i=1}^{k+1}C(k+1,i)*F_{k+1-i}*(n+1)^i$

4.36 Catalan

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

$h(n)=h(n-1)\times(4\times n2)/(n+1)h(n)=C(2n,n)/(n+1)(n=0,1,2,...)$

$h(n)=c(2n,n)-c(2n,n-1)(n=0,1,2,...) $

- 将 n + 2 边形沿弦切割成 n 个三角形的不同切割数
- n + 1 个数相乘, 给每两个元素加上括号的不同方法数
- n 个节点的不同形状的二叉树数
- 从 n * n 方格的左上角移动到右下角不升路径数

4.37 pick 定理

计算点阵中顶点在格点上的多边形面积公式, 该公式可以表示为: $2*S=2*a+b-2$, 其中 a 表示多边形内部的点数,b 表示多边形边界上的点数,S 表示多边形的面积.

4.38 正多边形外接圆半径

$R=\frac{a}{2sin(\frac{pi}{n})}$

4.39 Fibonacci

- $-F(n+m)=F(n+1)F(m)+F(n)F(m-1)$
- $F(n)^2=(-1)^{n+1}+F(n-1)F(n+1)$
- 前 n 项有 $\lceil\frac{2n}{3}\rceil$ 个奇数项
- 如果 fib(k) 能被 x 整除，则 fib(k*i) 都可以被 x 整除。
- $ f(0)+f(1)+f(2)+...+f(n)=f(n+2)-1$
- $f(1)+f(3)+f(5)+...+f(2n-1)=f(2n)$
- $f(2)+f(4)+f(6)+...+f(2n)=f(2n+1)-1$
- $[f(0)]^2+[f(1)]^2+...+[f(n)]^2=f(n)*f(n+1)$
- $f(0)-f(1)+f(2)-...+(-1)^n$
- $f(n)=(-1)^n*[f(n+1)-f(n)]+1$
- $f(n+m)=f(n+1)*f(m)+f(n)*f(m-1)$
- $[f(n)]^2=(-1)^{n-1}+f(n-1)*f(n+1)*f(2n-1)=[f(n)]^2-[f(n-2)]^2$

- $3f(n)=f(n+2)+f(n-2)$
- $f(2n-2m-2)[f(2n)+f(2n+2)]=f(2m+2)+f(4n-2m) [n>m>=-1, 且 n>=1] $
- $Fn=\frac{1}{\sqrt{5}}*(\frac{1+\sqrt{5}}{2})-\frac{1}{\sqrt{5}}*(\frac{1-\sqrt{5}}{2})$

5 计算几何

5.1 处理平面内所有直线围成的所有多边形

```
const int MAXN=1e6+10;
const double eps=1e-8;
const double pi=acos(-1.0);
const ll INF=0x3f3f3f3f3f3f3f3f;

inline int dcmp(double x){
    if(fabs(x)<eps) return 0;
    return (x>0? 1: -1);
}

inline double sqr(double x){ return x*x; }

struct Point{
    double x,y;
    Point(){ x=0,y=0; }
    Point(double _x,double _y):x(_x),y(_y){}
    void input(){ scanf("%lf%lf",&x,&y); }
    void output(){ printf("%.2f %.2f\n",x,y); }
    friend istream &operator >>(istream &os,Point &b){
        os>>b.x>>b.y;
        return os;
    }
    friend ostream &operator <<(ostream &os,Point &b){
        os<<b.x<<' '<<b.y;
        return os;
    }
    bool operator ==(const Point &b)const{
        return (dcmp(x-b.x)==0&&dcmp(y-b.y)==0);
    }
    bool operator !=(const Point &b)const{
        return !((dcmp(x-b.x)==0&&dcmp(y-b.y)==0));
    }
    bool operator <(const Point &b)const{
        return (dcmp(x-b.x)==0? dcmp(y-b.y)<0 : x<b.x);
    }
    double operator ^(const Point &b)const{ //叉积
        return x*b.y-y*b.x;
    }
    double operator *(const Point &b)const{ //点积
        return x*b.x+y*b.y;
    }
    Point operator +(const Point &b)const{
        return Point(x+b.x,y+b.y);
```

```
    }
    Point operator -(const Point &b)const{
        return Point(x-b.x,y-b.y);
    }
    Point operator *(double a){
        return Point(x*a,y*a);
    }
    Point operator /(double a){
        return Point(x/a,y/a);
    }
    double len2(){ //长度平方
        return sqr(x)+sqr(y);
    }
    double len(){ //长度
        return sqrt(len2());
    }
    double polar(){ //向量的极角
        return atan2(y,x); //返回与x轴正向夹角(-pi~pi]
    }
    Point change_len(double r){ //转化为长度为r的向量
        double l=len();
        if(dcmp(l)==0) return *this; //零向量
        return Point(x*r/l,y*r/l);
    }
    Point rotate_left(){ //逆时针旋转90度
        return Point(-y,x);
    }
    Point rotate_right(){ //顺时针旋转90度
        return Point(y,-x);
    }
    Point rotate(Point p,double ang){ //绕点p逆时针旋转ang度
        Point v=(*this)-p;
        double c=cos(ang),s=sin(ang);
        return Point(p.x+v.x*c-v.y*s,p.y+v.x*s+v.y*c);
    }
    Point normal(){  //单位化，逆时针旋转90°
        return Point(-y/len(),x/len());
    }
};

inline double cross(Point a,Point b){ //叉积
    return a.x*b.y-a.y*b.x;
}

inline double dot(Point a,Point b){ //点积
    return a.x*b.x+a.y*b.y;
}


double rad(Point a,Point b){ //两个向量的夹角
    return fabs(atan2(fabs(cross(a,b)),dot(a,b)));
}

bool is_parallel(Point a,Point b){ //判断向量是否平行
    double p=rad(a,b);
    return dcmp(p)==0||dcmp(p-pi)==0;
```

```
}

struct Line{
    Point s,e;
    Line(){}
    Line(Point _s,Point _e):s(_s),e(_e){} //两点确定直线
    Line(Point p,double ang){ //一个点和斜率(弧度制)确定直线
        s=p;
        if(dcmp(ang-pi/2)==0){
            e=s+Point(0,1);
        }
        else{
            e=s+Point(1,tan(ang));
        }
    }
    Line(double a,double b,double c){ //ax+by+c=0
        if(dcmp(a)==0){
            s=Point(0,-c/b);
            e=Point(1,-c/b);
        }
        else if(dcmp(b)==0){
            s=Point(-c/a,0);
            e=Point(-c/a,1);
        }
        else{
            s=Point(0,-c/b);
            e=Point(1,(-c-a)/b);
        }
    }
    void input(){
        s.input();
        e.input();
    }
    void adjust(){
        if(e<s) swap(e,s);
    }
    double polar(){ //极角
        return atan2(e.y-s.y,e.x-s.x); //返回与x轴正向夹角(-pi~pi]
    }
    double angle(){  //倾斜角
        double k=atan2(e.y-s.y,e.x-s.x);
        if(dcmp(k)<0) k+=pi;
        if(dcmp(k-pi)==0) k-=pi;
        return k;
    }
    Point operator &(const Line &b)const{ //求两直线交点
        Point res=s;
        double t=((s-b.s)^(b.s-b.e))/((s-e)^(b.s-b.e));
        res.x+=(e.x-s.x)*t;
        res.y+=(e.y-s.y)*t;
        return res;
    }
};

double polygon_area(vector<Point> p){ //多边形的有向面积，加上绝对值就是面积 正值表示输入点按照逆时针 否则为顺时针
```

```cpp
    int n=p.size(); double area=0;
    for(int i=1;i<n-1;i++) area+=cross(p[i]-p[0],p[i+1]-p[0]);
    return fabs(area/2);
}

struct PSLG{ //平面直线图 处理平面内所有直线围成的所有多边形 传入直线交点之间的每条线段
    struct Edge{
        int from,to;
        double ang;
        Edge(){ ang=from=to=0; }
        Edge(int s,int t,double a){ from=s,to=t,ang=a; }
    };
    int n,m,face_cnt; //平面个数 包括外面最大的多边形
    double area[MAXN]; //每个多边形面积
    Point point[MAXN]; //平面内所有的点
    vector<Edge>edge;
    vector<int>G[MAXN];
    vector<vector<Point> >face;
    int vis[2*MAXN],left[2*MAXN],pre[2*MAXN]; //left表示这条边的左侧属于哪个面
    void Init(){
        face.clear();
        edge.clear();
        for(int i=0;i<n;i++) G[i].clear();
        n=m=0;
    }
    PSLG(){ Init(); }
    void AddEdge(int from, int to){      //需要建立反向边帮助寻找下一条边
       edge.pb(Edge(from,to,(point[to]-point[from]).polar()));
       edge.pb(Edge(to,from,(point[from]-point[to]).polar()));
       m=edge.size();
       G[from].pb(m-2);
       G[to].pb(m-1);
    }
    void Build(){
        for(int u=0;u<n;u++){
            int d=G[u].size();
            for(int i=0;i<d;i++)
                for(int j=i+1;j<d;j++)
                    if(edge[G[u][i]].ang>edge[G[u][j]].ang)
                        swap(G[u][i],G[u][j]);
            for(int i=0;i<d;i++) pre[G[u][(i+1)%d]]=G[u][i]; //从u出发的i条边顺时针旋转的第一条边是pre[i]
        }
        face_cnt=0; memset(vis,0,sizeof(vis));
        for(int u=0;u<n;u++){
            for(int i=0;i<G[u].size();i++){
                int e=G[u][i];
                if(!vis[e]){
                    face_cnt++;
                    vector<Point> polygon;
                    while(1){
                        vis[e]=1;
                        left[e]=face_cnt;
                        int from=edge[e].from;
                        polygon.pb(point[from]);
```

```cpp
                        e=pre[e^1];     //逆时针旋转最多的一条边即为顺时针转动的第一条边
                        if(e==G[u][i]) break;
                    }
                    face.pb(polygon);
                }
            }
        }
        for(int i=0;i<face_cnt;i++) area[i]=polygon_area(face[i]);
    }
    vector<pair<double,int> >tmp[MAXN];
    void Insert(Line *line,int m){
        for(int i=0;i<m;i++){
            for(int j=i+1;j<m;j++){
                if(!is_parallel(line[i].e-line[i].s,line[j].e-line[j].s)){
                    Point inter=line[i]&line[j];
                    point[n++]=inter;
                    tmp[i].pb({dot(inter-line[i].s,line[i].e-line[i].s),n-1});
                    tmp[j].pb({dot(inter-line[j].s,line[j].e-line[j].s),n-1});
                }
            }
            sort(tmp[i].begin(),tmp[i].end());
            for(int j=1;j<tmp[i].size();j++) AddEdge(tmp[i][j-1].se,tmp[i][j].se);
        }
        Build();
    }
}pslg;

Line line[MAXN];

int main(void){
    int n; scanf("%d",&n);
    for(int i=0;i<n;i++) line[i].input();
    pslg.Insert(line,n);
    sort(pslg.area,pslg.area+pslg.face_cnt);
    printf("%d %.6f %.6f\n",pslg.face_cnt-1,pslg.area[pslg.face_cnt-2],pslg.area[0]);
    int q; scanf("%d",&q);
    while(q--){
        int p; scanf("%d",&p);
        if(p>=pslg.face_cnt) puts("Invalid question");
        else   printf("%.6f\n",pslg.area[pslg.face_cnt-p-1]);
    }
    return 0;
}
```

6 字符串

6.1 kmp

- border

```cpp
void get_fail(int f[],char s[])
```

```
{
    int j = f[0] = 0;
    int n = strlen(s);
    rep(i,1,n)
    {
        while(j && s[i] != s[j]) j = f[j-1];
        f[i] = j += s[i] == s[j];
    }
}
```

- kmp

```
void kmp(int f[],char p[],char s[])
{
    int n = strlen(s);
    int m = strlen(p);
    int j = 0;
    rep(i,0,n)
    {
        while(j && s[i] != p[j]) j = f[j-1];
        if(s[i] == p[j]) j++;
        if(j == m)
        {
            cout << i - j + 2 << endl;
            j = f[j-1];
        }
    }
}
```

6.2 SA

```
const int N=4e5+100;
const int maxn = 2e5+100;
const int inf=1e9+9;

namespace SA {
    char s[N];
    int sa[N],x[N],y[N],hep[N],height[N],n,m;
    void init()
    {
        n = 0;
    }
    void add(char c)
    {
        // c -= 'a';
        n++;
        s[n]=c;
    }
    void Sort() {
        for(int i=0;i<=m;++i) hep[i]=0;
        for(int i=1;i<=n;++i) ++hep[x[i]];
        for(int i=1;i<=m;++i) hep[i]+=hep[i-1];
        for(int i=n;i>=1;--i) sa[hep[x[y[i]]]--]=y[i];
    }
```

```
    void Pre_sa() {
        for(int i=1;i<=n;++i) x[i]=s[i],y[i]=i;
        m=223;Sort();
        for(int w=1,p=0;m=p,p<n;w<<=1) {
            p=0;
            for(int i=1;i<=w;++i) y[++p]=n-w+i;
            for(int i=1;i<=n;++i) if(sa[i]>w) y[++p]=sa[i]-w;
            Sort(),swap(x,y),x[sa[1]]=p=1;
            for(int i=2;i<=n;++i)
                x[sa[i]]=(y[sa[i]]==y[sa[i-1]]&&y[sa[i]+w]==y[sa[i-1]+w])?p:++p;
        }return;
    }
    ll Pre_height() {
        for(int i=1;i<=n;++i) x[sa[i]]=i;
        int k=0,res=0;
        for(int i=1;i<=n;++i) {
            k-=k>0;
            int j=sa[x[i]-1];
            while(i+k<=n&&j+k<=n&&s[i+k]==s[j+k]) ++k;
            height[x[i]]=k,res+=k;
        }return res;//直接返回height数组的和
    }
    ll solve()
    {
        /**
        给你一个长为N的字符串，求不同的子串的个数?
        对于一个后缀sa[i]，它产生了n-sa[i]个前缀，减去height[i]个相同的前缀(与前一个比较)，
        则产生了n-sa[i]-height[i]个子串。累加后即结果。
        */
        ll ans = 0;
        for (int i = 1; i <= n; i++)
        {
            ans += n + 1 - sa[i] - height[i];
        }
        return ans;
    }
    ll gao()
    {
        Pre_sa();
        Pre_height();
        return solve();
    }
}
```

6.3 回文树 1

```
struct Pal
{
    int ch[maxn][26],f[maxn],len[maxn],s[maxn];
    int cnt[maxn];
    int num[maxn];
    int last,sz,n;

```

```
    int newnode(int x)
    {
        memset(ch[sz],0,sizeof(ch[sz]));
        cnt[sz] = num[sz] = 0, len[sz] = x;
        return sz++;
    }
    void init()
    {
        sz = 0;
        newnode(0),newnode(-1);
        last = n = 0,s[0] = -1,f[0] = 1;
    }

    int get_fail(int u)
    {
        while(s[n - len[u] - 1] != s[n])
            u = f[u];
        return u;
    }

    void add(int c)
    {
        c -= 'a';
        s[++n] = c;
        int u = get_fail(last);
        if(!ch[u][c])
        {
            int np = newnode(len[u] + 2);
            f[np] = ch[get_fail(f[u])][c];
            num[np] = num[f[np]] + 1;
            ch[u][c] = np;
        }
        last = ch[u][c];
        cnt[last]++;
    }

    void count()
    {
        for(int i = sz - 1;~i;i--)
        cnt[f[i]] += cnt[i];
    }
} pa;
```

6.4 回文树 2

```
struct Palindromic_Tree {
    int son[N][26]; //转移边
    int fail[N]; //fail 指针
    int cnt[N]; //当前节点表示的回文串在原串中出现了多少次
    int num[N]; //当前节点 fail 可以向前跳多少次
    int len[N]; //当前节点表示的回文串的长度
    int S[N]; //插入的字符串
    int last; //最后一次访问到的节点，类似 SAM
    int n; //插入的字符串长度
```

```
    long long p; //自动机的总状态数

    int newnode(int l) {
        memset(son[p], 0, sizeof(son[p]));
        cnt[p] = 0;
        num[p] = 0;
        len[p] = l;
        return p++;
    }

    void init() {
        p = 0;
        newnode(0);
        newnode(-1);
        last = 0;
        n = 0;
        S[n] = -1;
        fail[0] = 1;
    }

    int get_fail(int x) {
        while (S[n - len[x] - 1] != S[n]) x = fail[x];
        return x;
    }

    void add(int c) {
        c -= 'a';
        S[++n] = c;
        int cur = get_fail(last); //通过上一次访问的位置去扩展
        if (!son[cur][c]) { //如果没有对应的节点添加一个新节点
            int now = newnode(len[cur] + 2);
            fail[now] = son[get_fail(fail[cur])][c]; //通过当前节点的 fail 去扩展出新的 fail
            son[cur][c] = now;
            num[now] = num[fail[now]] + 1; //记录 fail 跳多少次
        }
        last = son[cur][c];
        cnt[last]++; //表示当前节点访问了一次
    }
    void count() {
        //如果某个节点出现一次，那么他的 fail 也一定会出现一次，并且在插入的时候没有计数
        for (int i = p - 1; i >= 0; i--) cnt[fail[i]] += cnt[i];
    }
} AUT;
```

6.5 MANACHER

```
struct MANACHER{
#define STRSIZE 2010000
  char str[STRSIZE*2+5];
  int radius[STRSIZE*2+5];
  int str_len;
  MANACHER(){}
```

```cpp
  MANACHER(char *tar){solver(tar);}
  int solver(char *tar);          //执行MANACHER算法
  int getMaxPalSize();            //最长回文串长度
  long long getPalNum();          //不同回文串个数
};
int MANACHER::solver(char *tar){
    //(1)construct new string
    int tar_len=strlen(tar);
    str_len=tar_len*2+2;
    str[0]='~';
    str[1]='!';
    for (int i=0;i<tar_len;i++){
      str[i*2+2]=tar[i];
      str[i*2+3]='!';
    }
    str[str_len]=0;
    //(2)calculate radius
    int max_idx=-1;
    int center=-1;
    for (int i=0;i<str_len;i++){
      radius[i]=1;
      if (max_idx>i)
        radius[i]=min(max_idx-i+1,radius[center*2-i]);
      while(str[i+radius[i]]==str[i-radius[i]]) radius[i]++;
      //update max_idx
      if (max_idx<i+radius[i]-1){
        max_idx=i+radius[i]-1;
        center=i;
      }
    }
    return 0;
}
int MANACHER::getMaxPalSize(){
    int ans=0;
    for (int i=1;i<str_len;i++)
       ans=max(ans,radius[i]-1);
    return ans;
}
long long MANACHER::getPalNum(){
    long long ans=0;
    for (int i=1;i<str_len;i++){
      ans+=radius[i]/2;
    }
    return ans;
}
MANACHER x;
```

7 杂项

7.1 退火

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

```cpp

using namespace std;

const int maxn = 1e5 + 10;
const double eps = 1e-8;
const double delta = 0.98;
const double inf = 1e18;

struct Point { double x, y; } p[maxn];

double dis(Point A, Point B) { return sqrt((A.x - B.x) * (A.x - B.x) + (A.y - B.y
    ) * (A.y - B.y)); };

double Simulate_Annea(int n)
{
    Point S;
    S.x = S.y = 0;
    double t = 1000;
    double res = inf;
    while(t > eps)
    {
        int k = 0;
        for(int i = 0; i < n; i ++) if(dis(S, p[i]) > dis(S, p[k])) k = i;
        double d = dis(S, p[k]);
        res = min(res, d);
        S.x += (p[k].x - S.x) / d * t;
        S.y += (p[k].y - S.y) / d * t;
        t *= delta;
    }
    return res;
}

int main()
{
    int n;
    scanf("%d", &n);
    for(int i = 0; i < n; i ++) scanf("%lf%lf", &p[i].x, &p[i].y);
    printf("%.3f\n", Simulate_Annea(n));
    return 0;
}
```

7.2 博弈

7.2.1 Bash

只有一堆 n 个物品, 两个人轮流从这堆物品中取物, 规定每次至少取一个, 最多取 m 个. 最后取光者得胜. 考虑 n = k*(1+m)+r 即可 r = 0, 先手必败，无论先手怎么取（设取的是 a），后手都能凑出 1+m-a

7.2.2 Wythoff

有两堆火柴棍, 每次可以从某一堆取至少 1 根火柴棍 (无上限)，或者从两堆取相同的火柴棍数。最后取完的是胜利者。考虑状态 (a,b) a[i] = i*(1+√5)/2 b[i] = a[i]+i; 因为有浮点数参与运算，范围过大要考虑手写高精度

7.2.3 Fibonacci’s Game

有一堆个数为 n 的石子，游戏双方轮流取石子，满足：1) 先手不能在第一次把所有的石子取完；2) 之后每次可以取的石子数介于 1 到对手刚取的石子数的 2 倍之间（包含 1 和对手刚取的石子数的 2 倍）。先手胜当且仅当 n 不是 Fibonacci 数。

7.2.4 staircase nim

每层有若干个石子，每次可以选择任意层的任意个石子将其移动到该层的下一层。最后不能操作的人输把所有奇数阶梯看成 N 堆石子做 nim。把石子从奇数堆移动到偶数堆可以理解为拿走石子，就相当于几个奇数堆的石子在做 Nim。

7.2.5 anti-nim

正常的 nim 游戏是取走最后一颗的人获胜，而反 nim 游戏是取走最后一颗的人输。一个状态为必胜态，当且仅当：1) 所有堆的石子个数为 1，且 NIM_sum（异或和）=0 2）至少有一堆的石子个数大于 1，且 NIM_sum≠0

7.2.6 约数博弈

游戏初始状态包含 1-n, 这 n 个正整数。甲乙两个人轮流玩这个游戏. 每轮游戏中，游戏者任意选择一个还存在的数，然后删掉它和它所有的约数。第一个删掉所有数的人获胜。先手必胜反证法 - 考虑一个新的规则 “不准写数字 1”。依然先手必胜证明：如果在新规则下后写者必胜，则原游戏中的先写者写下数字 1，然后他就变成了新规则下的后写者。

7.2.7 约数和倍数博弈

游戏初始状态包含 2-n, 这 n-1 个正整数。甲乙两个人轮流玩这个游戏，每轮游戏中，游戏者任意选择一个还存在的数，然后删掉它和它所有的约数、倍数。第一个删掉所有数的人获胜。先手必败

7.2.8 Chomp 博弈

有一个 n * m 的棋盘，每次可以取走一个方格并拿掉它右边和上面的所有方格。拿掉左下角的格子 (1,1) 者输。- 除了 1*1 的棋盘，先手必败 - 对于其他大小的棋盘，先手必胜。

7.2.9 树上删边游戏

给出一个有 N 个点的树，有一个点作为树的根节点。游戏者轮流从树中删去边，删去一条边后，不与根节点相连的部分将被移走。谁无法移动谁输。叶子节点的 SG 值为 0；中间节点的 SG 值为它的所有子节点的 SG 值加 1 后的异或和。HDU 3590 N<=100 颗树, 每棵数有 m<=100 个点, 双方每次可以选择一棵树的一条边删去, 并且把不与根相连的边一并删去, 不能操作者赢先手必胜当且仅当: 1) 游戏的 SG 函数不为 0 且游戏中某个单一游戏的 SG 函数大于 1
2) 游戏的 SG 函数为 0 且游戏中没有单一游戏的 SG 函数大于 1

```
VI v[100005];
int getsg(int x,int pre)
{
    int ans = 0;
    rep(i,0,v[x].size())
        if(v[x][i] != pre)
        {
            ans ^= (1+getsg(v[x][i],x));
        }
    return ans;
}

int main()
{
    int t,n;
    while(scanf("%d",&t)!=EOF)
    {
        int sum = 0,ans = 0;
        while(t--)
        {
            scanf("%d",&n);
            rep(i,1,n+1)
            {
                v[i].clear();
            }
            rep(i,1,n)
            {
                int x,y;
                scanf("%d%d",&x,&y);
                v[x].push_back(y);
                v[y].push_back(x);
            }
            int s = getsg(1,-1);
            if(s > 1)
                ans++;
            sum ^= s;
        }
```

```
        if((sum == 0&&ans==0)||(sum!=0&&ans>=1))
        {
            cout<<"PP"<<endl;
        }
        else
        {
            cout<<"QQ"<<endl;
        }
    }
    return 0;
}
```

7.2.10 SG

```
define MAX 1005
/* 计算从1-n范围内的SG值。
   Array(存储可以走的步数，Array[0]表示可以有多少种走法)
   Array[]需要从小到大排序 */
/*HDU1847博弈SG函数
1.可选步数为1-m的连续整数，直接取模即可，SG(x) = x % (m+1);
2.可选步数为任意步，SG(x) = x;
3.可选步数为一系列不连续的数，用GetSG(计算) */
int SG[MAX], hash[MAX];
void GetSG(int Array[],int n = MAX-1)
{
    memset(SG, 0, sizeof(SG));
    for(int i = 0; i <= n; ++i)
    {
        memset(hash, 0, sizeof(hash));
        for(int j = 1; j <= Array[0]; ++j)
        {
            if(i < Array[j])
                break;
            hash[SG[i - Array[j]]] = 1;
        }
        for(int j = 0; j <= n; ++j)
            if(!hash[j])
            {
                SG[i] = j;
                break;
            }
    }
}
```

S-Nim 游戏仅仅是限制了每一次从每一堆中选取的个数，依旧用 sg 函数计算即可。经典的 Nim 游戏中 sg(x) = x，所以结果就是每一堆的状态直接 xor 即可。S-Nim 游戏先计算每一堆的 sg 函数值，然后判断方法依旧是用 xor.

```
const int maxn = 10100;
const int N = 110;
int SG[maxn],f[N],s[maxn];

int k,n,m,t;
```

```
void getsg(int n,int nn)
{
    SG[0] = 0;
    int tt = 0;
    for(int i = 1;i<=10000;i++)
    {
        tt++;
        //memset(s,0,sizeof s);
        for(int j = 0;f[j]<=i&&j<nn;j++)
        {
            s[SG[i-f[j]]] = tt;
        }
        for(int j = 0;;j++)
            if(s[j]!=tt)
        {
            SG[i] = j;
            break;
        }
    }
}

int main()
{
    while(scanf("%d",&k)!=EOF&&k)
    {
        for(int i =0;i<k;i++)
            scanf("%d",&f[i]);
        sort(f,f+k);
        scanf("%d",&n);
        getsg(10000,k);
        for(int i = 0;i<n;i++)
        {
            int ans = 0;
            scanf("%d",&m);
            for(int i = 0;i<m;i++)
                {
                    scanf("%d",&t);
                    ans^=SG[t];
                }
                if(ans)
                    cout<<"W";
                else
                    cout<<"L";
        }

        cout<<endl;
    }
}
```

7.3 indiewar 的私人题

7.3.1 半圆概率

在一个圆内任取 n 个点，则这 n 个点出现在同一个半圆内的概率是多少转化为 2n 条半径的古典概型问题：

$P_n = 2n/(2^n) = n/2^{n-1}$

7.3.2 百囚徒挑战

监狱决定给关押的 100 名囚徒一次特赦的机会，条件是囚徒通过一项挑战。

所有囚徒被编号为 1-100，对应他们编号的 100 个号码牌被打乱顺序放在了 100 个抽屉里。

每个囚徒需要从所有抽屉里打开至多半数 (50 个)，并从中找出对应自己编号的号码牌。

如果找到了则该名囚徒的任务成功。所有囚徒会依次单独进入挑战室完成任务，并且从第一个囚徒进入挑战室开始，直到所有囚徒结束挑战为止囚徒之间任何形式的交流都是禁止的。

当一名囚徒完成任务后，挑战室会被恢复为他进入之前的样子（号码牌当然也放回原来的抽屉里）。

在这 100 名囚徒中，任意一名囚徒的失败都会导致整个挑战失败，只有当所有囚徒全部成功完成任务时，他们才会统一得到特赦的机会。最后，在开始挑战之前，监狱给了所有囚徒一个月时间商量对策。那么，囚徒究竟有多大的几率得到释放？$p(n) = 1 - \sum_{m=n+1}^{2n} \frac{1}{m}$

7.3.3 投针问题

假定有一块画着许多等距的平行线的地板，现往板上投放一根细针，求细针与平行线相交的概率。假设平行线间距为 a ，针长为 l 且满足 l < a $p = \frac{2l}{a*pi}$

l >= a $p = 1 - \frac{1}{pi}(2arcsin\frac{a}{l} - \frac{2l}{a} + \frac{2}{a}\sqrt{l^2 - a^2})$

7.3.4 打怪兽

假设你打一下怪, 怪掉总血量的 0% 到 100% 之间的一个随机数,(包括 [0,1] 区间内的所有实数诸如 0.5%) 那么, 求你把怪打死的一共要打几次的数学期望. 不妨设怪物血量为 1, 你每次攻击输出 [公式] 间均匀的随机伤害. 假设 f(x) 为打死

血量为 x 的怪物所用的期望次数. 这个比较反常识, 虽然我们是要求 f(1), 但是, 我们却选择求看起来似乎更难的 f(x) $f(x) = e^x$

7.3.5 互素

随机地取两个正整数，他们互素的概率为 $p = \frac{6}{pi*pi}$

7.3.6 半球

在一个球内任取 n 个点，则这 n 个点落在同一个半球内的概率 $p = \frac{n^2-n+2}{2^n}$

圆 $p = \frac{2n}{2^n}$

7.3.7 杨表

1 到 N 的排列，最长上升子序列（LIS）长度的期望

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int mo = 998244353;
const int MAXN = 1e3 + 10;
ll inv[MAXN], a[MAXN], n;
inline ll fpm(ll x, ll y)
{
    ll res = 1;
    while(y)
    {
        if(y & 1) res = res * x % mo;
        x = x * x % mo;
        y >>= 1;
    }
    return res;
}
ll ans;
inline void dfs(int x, int y)
{
    if(!x)
    {
        ll res = 1;
        for(int i = 1; i <= n; ++i)
            res = res * i % mo;
        for(int i = 1; i < y; ++i)
            for(int j = 1; j <= a[i]; ++j)
            {
                int ct = a[i] - j;
                for(int k = i; k < y; ++k)
                    if(a[k] >= j) ++ct;
                res = res * inv[ct] % mo;
            }
```

```
        ans = (ans + res * res % mo * a[1] % mo) % mo;
    }
    for(int i = 1; i <= x; ++i)
    {
        if(y != 1 && i > a[y - 1]) continue;
        a[y] = i;
        dfs(x - i, y + 1);
    }
}
int main()
{
    cin >> n;
    for(int i = 1; i <= n; ++i)
        inv[i] = fpm(i, mo - 2);
    dfs(n, 1);
    for(int i = 1; i <= n; ++i)
        ans = ans * inv[i] % mo;
    cout << ans << endl;
    return 0;
}
```

7.4 网格问题

转化问题, 变成从 (0,0) 到 (n+m,n+m) 的网格问题, 再减去不合法的情况设 x 坐标表示 A 的值,y 坐标表示 B 的值则约束条件转化为 - $x <= y + n$ - $y <= x + m$

即不能穿过两条直线，关于不合法的计算，就是平移，对称的技巧运用例如下面的直线 $y = x - n$，平移 1，得到 $y = x - n - 1$（平移是因为要穿过直线，在直线上也是合法情况），对称点为 $(n + 1, -(n + 1))$，这样 $(0, 0)$ 到 $(2 * (n + m), n + m)$ 穿过 $y = x - n$ 的情况就变成 $(n + 1, -(n + 1))$ 到 $(2 * (n + m), n + m)$ 的情况

结果为 $C(2 * (n + m), n + m) - C(2 * (n + m), m - 1) - C(2 * (n + m), n - 1)$

```
const int maxn = 4e3+100;
int n,m;

ll fac[maxn],invf[maxn];

ll C(ll n, ll m) { // n >= m >= 0
    return n < m || m < 0 ? 0 : fac[n] * invf[m] % mod * invf[n - m] % mod;
}

void init()
{
     fac[0]=invf[0]=1;
    rep(i,1,4001) fac[i]=fac[i-1]*i%mod,invf[i]=powmod(fac[i],mod-2);
}

int main(int argc, char const *argv[])
```

```
{
    // ios_base::sync_with_stdio(false), cin.tie(0), cout.tie(0);
    init();
    while(scanf("%d%d",&n,&m)!=EOF)
    {
        ll ans = 0;
        ans = C(2*(n+m),n+m);
        ans -= C(2*(n+m),n-1);
        ans -= C(2*(n+m),m-1);
        ans = (ans%mod+mod)%mod;
        printf("%lld\n",ans);
    }
    return 0;
}
```

8 DP

8.1 背包

8.1.1 01 背包

- normal $f_i = max(f_i, f_i - v + w)$ 需要按照 i 从大到小的顺序更新，确保每个物品只会选一次

```
memset(dp,0xcf,sizeof dp);
dp[0] = 0;
rep(i,0,n)
{
    cin >> v[i] >> w[i];
    per(j,v[i],m+1)
    {
        dp[j] = max(dp[j],dp[j-v[i]]+w[i]);
    }
}
```

- 计数不超过 m 的方案数 $f_i += f_i - v$
- 删除加入物品的顺序不影响结果，假设被删除的物品是最后一次加入的，那么倒过来还原即可。$f_i -= f_i - v$ 需要按照 i 从小到大的顺序更新. 给定 ai 和 bi，表示第 i 个商店有 ai 个商品可以买，单价为 bi 元，给出 m 个询问，问用 c 元在 l~r 商店买东西的方案数一种物品的背包可以看成 $\sum_{i=0}^{a} x^{ib} = \frac{1-x^{(a+1)b}}{1-x^b}$，所以可以先用 (a+1)b 去做一个 01 背包 (系数为负)，再除以一个 x^b 的 (系数为负)01 背包。从生成函数来看，$\frac{1}{1-x^b} = \sum_{i=0}^{\infty} x^{bi}$，即做一遍完全背包就可以等效

然后对可逆背包的预处理，由于 $\frac{1-x^b}{1-x^{(a+1)*b}} = (1-x^b) * \sum_{i=0}^{\infty} x^{i*(a+1)b}$，于是反过来对 x^b 做 01 背包，对 (a+1)b 做完全背包就可以

```cpp
int n,m;
int a[maxn*10],b[maxn*10],f[maxn*10][maxn],g[maxn*10][maxn];

void gao(int *dp,int w)
{
    per(i,w,maxn)
    {
        dp[i] = (dp[i] - dp[i-w]+mod)%mod;
    }
}

void gao2(int *dp,int w)
{
    rep(i,w,maxn)
    {
        dp[i] = (dp[i] + dp[i-w]+mod)%mod;
    }
}

int main(int argc, char const *argv[])
{
    ios_base::sync_with_stdio(false), cin.tie(0);
    // cout.tie(0);
    int T,cas;
    cin >> T;
    cas = 1;
    f[0][0] = 1;
    rep(i,0,maxn)
        g[0][i] = 1;
    while(T--)
    {
        prr(cas++);
        cin >> n >> m;
        rep(i,1,n+1)
        {
            cin >> a[i] >> b[i];
            a[i] = (a[i] + 1) * b[i];
        }
        rep(i,1,n+1)
        {
            memcpy(f[i],f[i-1],sizeof(f[i]));
            memcpy(g[i],g[i-1],sizeof(g[i]));
            gao(f[i],a[i]);gao2(f[i],b[i]);
            gao(g[i],b[i]);gao2(g[i],a[i]);
        }
        int ans = 0;
        rep(i,0,m)
        {
            int l,r,c;
            cin >> l >> r >> c;
            l = (l+ans)%n+1;
            r = (r+ans)%n+1;
            if(l > r)
            {
                swap(l,r);
            }
            ans = 0;
```

```
        rep(i,0,c+1)
        {
            ans = (ans + 1ll*f[r][i]*g[l-1][c-i])%mod;
        }
        printf("%d\n",ans);
    }
  }
  return 0;
}
```

8.1.2 完全背包

- normal 需要按照 i 从小到大的顺序更新，意为要么停止选，要么接着多选一个。

```
dp[0] = 0;
rep(i,0,n)
{
    cin >> v >> w;
    rep(j,v,m+1)
    {
        dp[j] = max(dp[j],dp[j-v]+w);
    }
}
cout << dp[m] << endl;
```

- 计数 same
- 删除 same

8.1.3 多组背包

- 二进制拆分二进制拆分，将一个物品拆成 O(logk) 个 01 背包的物品。eg:$10 = 1 + 2 + 4 + 3$，可以表示 $1 - 10$ O(nmlog(k))

```
memset(dp,0,sizeof dp);
    rep(i,0,n)
    {
        cin >> v[i] >> w[i] >> s[i];
    }
    int cnt = 0;
    rep(i,0,n)
    {
        for(int j = 1;j < s[i];j <<= 1)
        {
            ww[cnt] = j * w[i];
            vv[cnt] = j * v[i];
            s[i] -= j;
            cnt++;
        }
        if(s[i])
        {
```

```
            ww[cnt] = s[i] * w[i];
            vv[cnt++] = s[i] * v[i];
        }
    }
    rep(i,0,cnt)
    {
        per(j,vv[i],m+1)
        {
            dp[j] = max(dp[j],dp[j-vv[i]] + ww[i]);
        }
    }
    cout << dp[m] << endl;
```

- 单调队列按%v 的余数分组，每组滑窗求区间最大值 O(nm), 但不见得比上面快

8.1.4 分组背包

n 个物品，每个物品只能选一个，体积为 vi，种类为 ki。求总体积恰好 m 的情况下能拿走物品种类数的最大值。将所有物品按 k 分组状态：$f_{i,j,k,s}$ 表示考虑前 i 组，这一组内考虑了前 j 个物品，总体积为 k，第 i 组物品是否被选择的情况为 s 时，种类数的最大值。

8.1.5 树形依赖背包

以 1 为根的树上有 n 个节点，每个节点有一个物品，体积 vi，价值 wi。选了一个点就必须选它的父亲。求总体积不超过 m 的情况下能拿走物品总价值的最大值。

按照 DFS 的顺序进行 DP。往下搜的时候，强行将儿子选入背包中。往上回溯的时候，可以选择要这棵子树的 DP 值，或者不要。

8.2 LIS

```cpp
std::vector<int> lis(int n, int a[]) {
  std::vector<int> g, res(n);
  for (int i = 0; i < n; ++ i) {
    res[i] = std::lower_bound(g.begin(), g.end(), a[i]) - g.begin();
    if (res[i] == g.size()) g.push_back(a[i]);
    else g[res[i]] = a[i];
  }
  return res;
}
```

8.3 数位 DP

```
int a[20];
ll dp[20][state];
ll dfs(int pos,/*state变量*/,bool lead/*前导零*/,bool limit)
{

    if(pos==-1) return 1;
    //第二个就是记忆化(在此前可能不同题目还能有一些剪枝)
    if(!limit && !lead && dp[pos][state]!=-1) return dp[pos][state];
    int up=limit?a[pos]:9;
    ll ans=0;
    for(int i=0;i<=up;i++)
    {
        if() ...
        else if()...
        ans+=dfs(pos-1,/*状态转移*/,lead && i==0,limit && i==a[pos]);
    }
    //计算完，记录状态
    if(!limit && !lead) dp[pos][state]=ans;
    return ans;
}
ll solve(ll x)
{
    int pos=0;
    while(x)
    {
        a[pos++]=x%10;
        x/=10;
    }
    return dfs(pos-1/*从最高位开始枚举*/,/*一系列状态 */,true,true);
}
```

8.3.1 数位上不能有 4 也不能有连续的 62

dp[pos][sta] 表示当前第 pos 位，前一位是否是 6 的状态，这里 sta 只需要去 0 和 1 两种状态就可以了，不是 6 的情况可视为同种，不会影响计数

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<string>
using namespace std;
typedef long long ll;
int a[20];
int dp[20][2];
int dfs(int pos,int pre,int sta,bool limit)
{
    if(pos==-1) return 1;
    if(!limit && dp[pos][sta]!=-1) return dp[pos][sta];
    int up=limit ? a[pos] : 9;
    int tmp=0;
    for(int i=0;i<=up;i++)
    {
```

```
        if(pre==6 && i==2)continue;
        if(i==4) continue;//都是保证枚举合法性
        tmp+=dfs(pos-1,i,i==6,limit && i==a[pos]);
    }
    if(!limit) dp[pos][sta]=tmp;
    return tmp;
}
int solve(int x)
{
    int pos=0;
    while(x)
    {
        a[pos++]=x%10;
        x/=10;
    }
    return dfs(pos-1,-1,0,true);
}
int main()
{
    int le,ri;
    //memset(dp,-1,sizeof dp);可优化
    while(~scanf("%d%d",&le,&ri) && le+ri)
    {
        memset(dp,-1,sizeof dp);
        printf("%d\n",solve(ri)-solve(le-1));
    }
    return 0;
}
```

8.3.2 优化

memset(dp,-1,sizeof dp); 放在多组数据外面使用的条件是：约束条件是每个数自身的属性，而与输入无关。1. 求数位和是 10 的倍数的个数, 这里简化为数位 sum%10 这个状态，即 dp[pos][sum] 这里 10 是与多组无关的，所以可以 memset 优化，不过注意如果题目的模是输入的话那就不能这样了。

2. 求二进制 1 的数量与 0 的数量相等的个数，这个也是数自身的属性。

8.3.3 相减

约束就是一个数的二进制中 0 的数量要不能少于 1 的数量 dp[pos][num], 到当前数位 pos,0 的数量减去 1 的数量不少于 num 的方案数，一个简单的问题，中间某个 pos 位上 num 可能为负数 (这不一定是非法的，因为我还没枚举完嘛，只要最终的 num>=0 才能判合法，中途某个 pos 就不一定了)，7 最小就-32 吧 (好像), 直接加上 32，把 32 当 0 用。

lead 的用法，显然我要统计 0 的数量，前导零是有影响的。

```
#pragma comment(linker, "/STACK:10240000,10240000")
#include<iostream>
#include<cstdio>
```

```
#include<cstring>
#include<string>
#include<queue>
#include<set>
#include<vector>
#include<map>
#include<stack>
#include<cmath>
#include<algorithm>
using namespace std;
const double R=0.5772156649015328606065120900;
const int N=1e5+5;
const int mod=1e9+7;
const int INF=0x3f3f3f3f;
const double eps=1e-8;
const double pi=acos(-1.0);
typedef long long ll;
int dp[35][66];
int a[66];
int dfs(int pos,int sta,bool lead,bool limit)
{
    if(pos==-1)
        return sta>=32;
    if(!limit && !lead && dp[pos][sta]!=-1) return dp[pos][sta];
    int up=limit?a[pos]:1;
    int ans=0;
    for(int i=0;i<=up;i++)
    {
        if(lead && i==0) ans+=dfs(pos-1,sta,lead,limit && i==a[pos]);//有前导零就不统计在内
        else ans+=dfs(pos-1,sta+(i==0?1:-1),lead && i==0,limit && i==a[pos]);
    }
    if(!limit && !lead ) dp[pos][sta]=ans;
    return ans;
}
int solve(int x)
{
    int pos=0;
    while(x)
    {
        a[pos++]=x&1;
        x>>=1;
    }
    return dfs(pos-1,32,true,true);
}
int main()
{
    memset(dp,-1,sizeof dp);
    int a,b;
    while(~scanf("%d%d",&a,&b))
    {
        printf("%d\n",solve(b)-solve(a-1));
    }
    return 0;
}
```

8.3.4 计数转求和

要求数的平方和。先考虑求和的问题，一个区间，数位 dp 能在一些约束下计数，现在要这些数的和。其实组合数学搞搞就可以了：如现在枚举的某一位 pos, 我统计了这一位枚举 i 的满足条件的个数 cnt，其实只要算 i 对总和的贡献就可以了，对于一个数而言第 pos 位是 i，那么对求和贡献就是 i*10^pos , 就是十进制的权值，然后有 cnt 个数都满足第 pos 位是 i，最后 $sum = cnt * i * 10^{pos}$.

原理就是这样平方和可以看做 $(a * 10^{pos} + b)^2$ a 是你当前 pos 位要枚举的，b 其实是个子问题，就是 pos 之后的位的贡献值，把这个平方展开.

```
typedef long long ll;
ll fact[20];
void init()
{
    fact[0]=1;
    for(int i=1;i<20;i++)
        fact[i]=fact[i-1]*10%mod;
}
struct node
{
    ll cnt,sum,sqr;
    node(ll cnt=-1,ll sum=0,ll sqr=0):cnt(cnt),sum(sum),sqr(sqr){}
}dp[20][7][7];
int a[20];
ll fac(ll x)
{
    return x*x%mod;
}
ll dfs(int pos,ll num,ll val,ll&cnt,ll&sum,bool limit)
{
    if(pos==-1) {
        if(num==0 || val==0)
            return 0;
        cnt=1;
        return 0;
    }
    if(!limit && dp[pos][num][val].cnt!=-1) {
            cnt=dp[pos][num][val].cnt;
            sum=dp[pos][num][val].sum;
            return dp[pos][num][val].sqr;
    }
    int up=limit?a[pos]:9;
    ll sq=0;
    for(int i=0;i<=up;i++)
    if(i!=7)
    {
        ll cn=0,su=0;
        ll tmp=dfs(pos-1,(num+i)%7,(val*10+i)%7,cn,su,limit && i==a[pos]);
        ll tm=i*fact[pos]%mod;
        tmp=(tmp+fac(tm)*cn%mod+(tm*su%mod)*2%mod)%mod;//计数之后要更新sum,sqr
        sum=(sum+su+(i*fact[pos]%mod)*cn%mod)%mod;
        cnt=(cnt+cn)%mod;
        sq=(sq+tmp)%mod;
    }
    if(!limit) dp[pos][num][val]=node(cnt,sum,sq);
```

```
    return sq;
}
ll solve(ll x)
{
    int pos=0;
    while(x)
    {
        a[pos++]=x%10;
        x/=10;
    }
    ll t1=0,t2=0;
    return dfs(pos-1,0,0,t1,t2,true);
}
bool judge(ll x)
{
    int sum=0;
    int pos=0;
    if(x%7==0) return false;
    while(x)
    {
        if(x%10==7) return false;
        sum+=x%10;
        x/=10;
    }
    sum%=7;
    return sum!=0;
}
int main()
{
    init();
    for(int i=0;i<20;i++)
        for(int j=0;j<7;j++)
        for(int k=0;k<7;k++)//memset
    {
        dp[i][j][k].cnt=-1;
        dp[i][j][k].sum=0;
        dp[i][j][k].sqr=0;
    }
    int T_T;
    scanf("%d",&T_T);
    while(T_T--)
    {
        ll le,ri;
        scanf("%I64d%I64d",&le,&ri);
        ll ans=solve(ri)-solve(le-1);
        ans=(ans%mod+mod)%mod;
        printf("%I64d\n",ans);
    }
    return 0;
}
```

9 FINALE

9.1 Keep Calm

1. **n = 0 or n = 1**(各种模拟题?)
2. **array bounds** 每一个数组的大小
3. **int overflow**
4. 多组是否清空 (vector?)
5. 多组清空的复杂度
6. 位运算的优先级，打括号 (状压 dp?)
7. 上取整以及 GCD 小心**负数**
8. think twice and code once
9. **do sth instead of nothing and stay organized**