**头文件**
**DP**
- 数位DP

**字符串**
- hash
- KMP
- Trie树
- EXKMP
- AC自动机
- 后缀自动机
- Manacher
- 最小表示法

**数学**
- 快速幂
- 矩阵乘法
- gcd
- exgcd
- exgcd求乘法逆元
- exgcd求解线性同余方程
- 线性筛
- 大质数分解
- 原根
- 莫比乌斯反演
- 生成函数
- NTT
- MTT
- 组合数
- Lagrange插值
- 高斯消元

**数据结构**
- ST表
- 树状数组
  - 单点修改，区间查询
  - 区间修改，区间求和
- 线段树
  - 单点修改，区间查询
  - 区间修改，区间查询
- 线段树维护区间绝对众数
- 可持久化权值线段树
- 平衡树
- KD树
- LCT

**图论**
- 最小生成树
- LCA
- 单源最短路
- 树链剖分
- 启发式合并
- 树哈希
- 2-SAT
- 虚树

kruskal重构树
**网络流**
最大流
费用流
**计算几何**
计算两圆交点

# 头文件

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
char *p1,*p2,buf[100000];
const int N = 3e6 + 10;
const int mod = 1e9 + 7;
const ll inf = 1e18;
#define nc() (p1==p2 && (p2=(p1=buf)+fread(buf,1,100000,stdin),p1==p2)?EOF:*p1++)
int read()
{
    int x=0,f=1;
    char ch=nc();
    while(ch<48||ch>57)
    {
        if(ch=='-')
            f=-1;
        ch=nc();
    }
    while(ch>=48&&ch<=57)
        x=x*10+ch-48,ch=nc();
    return x*f;
}
void write(int x)
{
    if(x<0)
        putchar('-'),x=-x;
    if(x>9)
        write(x/10);
    putchar(x%10+'0');
    return;
}
#pragma GCC optimize(2)
```

# DP

## 数位DP

```
int a[N];
int digit;//当前需要统计的数字
// f[pos][cnt][0/1]：当最高位已经填了（没有前导0时）在[pos + 1, len]中digit填了cnt个，[1, pos]任意填，digit出现的次数
// 第三维0表示digit=0,第三维为1表示digit!=1
// 当limit=false时，容易知道填1-9的数量是相同的
ll f[N][N][2];

ll dfs(int pos, bool limit, bool lead0, int cnt)
{
    if (!pos) // 递归边界
        return cnt;
    auto &now = f[pos][cnt][digit != 0];
    if (!limit && !lead0 && ~now) // 没限制并且dp值已搜索过
        return now;
    int up = limit ? a[pos] : 9;
    ll res = 0;
    for(int i = 0; i <= up; i ++)
    {
        //填0的时候需注意是否为前导
        //前导零不算入0的个数
        int tmp = cnt + (i == digit);
        if(lead0 && digit == 0 && i == 0)
            tmp = 0;
        res += dfs(pos - 1, limit && i == up, lead0 && i == 0, tmp);
    }
    if (!limit && !lead0) //无限制并且没有前导0
        now = res;
    return res;
}

ll ans[10];
void solve(ll x, int f){
    int len = 0;
    while (x > 0)
    {
        a[++len] = x % 10;
        x /= 10;
    }
    for(int i = 0; i <= 9; i ++)
    {
        digit = i;
        ans[i] += f * dfs(len, true, true, 0);
    }
}

int main()
{
    memset(f, -1, sizeof f);
    ll l, r;
    cin >> l >> r;
    solve(r, 1); //加上[1, r]
```

```
    solve(l - 1, -1); //扣除掉[1, l-1]
    for(int i = 0; i <= 9; i ++)
        cout << ans[i] << ' ';
}
```

# 字符串

## hash

```
char s[N];
inline bool check(int l1, int r1, int l2, int r2)
{
    ll t1 = (mod + hs[r1] - hs[l1 - 1] * p[r1 - l1 + 1] % mod) % mod;
    ll t2 = (mod + hs[r2] - hs[l2 - 1] * p[r2 - l2 + 1] % mod) % mod;
    return t1 == t2;
}
inline void del(char *s,int len)//[1,n]
{
    p[0] = 1;
    for (int i = 1; i <= len; i++)
    {
        f[i] = f[i - 1] * P + s[i] - 'a' + 1;
        p[i] = p[i - 1] * P;
    }
}
```

## KMP

```
char a[N],b[N];
int lena,lenb;
ll nex[N];
signed main() {
    cin>>b+1,cin>>a+1;
    lena=strlen(a+1),lenb=strlen(b+1);
    for(int i=2,j=0; i<=lena; i++) {
        while(j&&a[i]!=a[j+1])j=nex[j];
        if(a[i]==a[j+1])j++;
        nex[i]=j;
    }
    for(int i=1,j=0; i<=lenb; i++) {
        while(j>0&&b[i]!=a[j+1])j=nex[j];
        if(b[i]==a[j+1])j++;
        if(j==lena) {
            //匹配成功操作
            j=nex[j];
        }
    }
    return 0;
```

```
}
```

## Trie树

```
int nex[100000][26], cnt;
bool exist[100000];  // 该结点结尾的字符串是否存在

void insert(char *s, int l) {  // 插入字符串
    int p = 0;
    for (int i = 0; i < l; i++) {
        int c = s[i] - 'a';
        if (!nex[p][c]) nex[p][c] = ++cnt;  // 如果没有，就添加结点
        p = nex[p][c];
    }
    exist[p] = 1;
}

bool find(char *s, int l) {  // 查找字符串
    int p = 0;
    for (int i = 0; i < l; i++) {
        int c = s[i] - 'a';
        if (!nex[p][c]) return 0;
        p = nex[p][c];
    }
    return exist[p];
}
```

## EXKMP

```
//###### 求Next[i]
//nex[*i*] 表示“模式串 c” 与 “模式串c[i]开头的后缀” 的 “最长公共前缀” 的长度。
int p = 1, k = 2, l;
int len = c.size();
c = " "+c;
nex[1] = len;//1开头的最长后缀是它本身
while(p<len&&c[p] == c[p+1])p++;
nex[2] = p-1;
for(int i=3;i<=len;i++)
{
    p = k + nex[k] - 1;
    l = nex[i - k + 1];
    if(i + l <= p)nex[i] = l;
    else
    {
        int j =max(0,p-i);
        while(i+j<=len&&c[i+j] == c[j+1])j++;
        nex[i] = j;
        k = i;
    }
}
```

```
//###### 求ext[i]
//ext[i]表示“模式串c” 与 “文本串 a 以 a[i] 开头的后缀” 的 “最长公共前缀” 的长度。

int p = 1, k = 1, l;
int lena = a.size();
a=" " +a;
while(p<=lena&&p<=len&&a[p] == c[p])p++;
ext[1] = p - 1;
for(int i=2;i<=lena;i++)
{
    p = k + ext[k] -1;
    l = nex[i - k + 1];
    if(i+l<=p)ext[i] = l;
    else
    {
        int j =max(0,p-i);
        while(i+j<=lena&&j+1<=len&&a[i+j] == c[j+1])j++;
        ext[i] = j;
        k = i;
    }
}
```

# AC自动机

```
int trie[N][26];//字典树
int cntword[N];//记录该单词出现次数
int fail[N];//失败时的回溯指针,同next[]数组
int cnt = 0;
void insertWords(string s)//下标从0开始
{
    int root = 0, len= s.size();
    for (int i = 0; i < len; i++)
    {
        int t = s[i] - 'a';
        if (!trie[root][t])
            trie[root][t] = ++cnt;
        root = trie[root][t];
    }
    cntword[root]++;//当前节点单词数+1
}
void getFail()
{
    queue<int>q;
    for (int i = 0; i < 26; i++) //将第二层所有出现了的字母扔进队列
        if (trie[0][i])
        {
            fail[trie[0][i]] = 0;
            q.push(trie[0][i]);
        }
    while (!q.empty())
```

```
    {
        int now = q.front();
        q.pop();
        for (int i = 0; i < 26; i++)
        {
            int t=trie[now][i];
            if (t)
            {
                //如果有这个子节点为字母i+'a',则
                //让这个子节点的失败指针指向(((他父亲节点)的失败指针所指向的那个节点)的下一个节点)
                //有点绕,为了方便理解特意加了括号
                fail[t] = trie[fail[now]][i];
                q.push(t);
            }
            //否则就让当前节点的这个子节点,指向当前节点fail指针的这个子节点
            else trie[now][i] = trie[fail[now]][i];
        }
    }
}
int query(string s)
{
    int now = 0, ans = 0, len=s.size();
    for (int i = 0; i < len; i++)//遍历文本串
    {
        now = trie[now][s[i] - 'a'];//从s[i]点开始寻找
        for (int j = now; j && cntword[j] != -1; j = fail[j])
        {
            //注意j = now
            //一直向下寻找,直到匹配失败(失败指针指向根或者当前节点已找过).
            ans += cntword[j];
            cntword[j] = -1;//将遍历过后的节点标记,防止重复计算
        }
    }
    return ans;
}
```

# 后缀自动机

```
int num_sub[N];//前i的本质不同子串
struct NODE {
    int ch[26]; // 每个节点添加一个字符后到达的节点
    int len;  // 每个节点代表的状态中的最大串长
    int fa; // 每个节点在parent树上的父亲节点, 即该节点link边指向的节点
    int sz; // 每个节点对应的endpos集合的大小（即串的出现次数）, 等于所有parent树上儿子的大小
    NODE() {
        memset(ch,0,sizeof(ch));
        len=0;
    }
} dian[N<<1]; // 节点数开串长的两倍
```

```
int n; // 串长
int las=1; // las: 上一个用到的节点编号
int tot=1; // tot: 当前用到的节点编号
// 向SAM中插入一个字符c
void add(int c,int pos) {
    if(pos>=1) num_sub[pos]=num_sub[pos-1];
    int p=las; // 上一个状态的节点
    int np=las=++tot; // 要加入的状态的节点
    dian[np].sz=1; // 叶子节点endpos大小为1
    dian[np].len=dian[p].len+1; // 新状态比上一个装填多一个首字符
    for(; p&&!dian[p].ch[c]; p=dian[p].fa)dian[p].ch[c]=np; // 指针p沿link边回跳,直至找到一个节点包含字符c的出边，无字符c的出边则将出边指向新状态的节点
    if(!p)dian[np].fa=1;// 以上为case 1，指针p到SAM的起点的路径上的节点都无字符c的出边，将新节点作为SAM的起点的一个儿子节点
    else { // 节点p包含字符c的出边
        int q=dian[p].ch[c]; // 节点p的字符c的出边指向的节点
        if(dian[q].len==dian[p].len+1)dian[np].fa=q;// 以上为case 2，节点p和q代表的状态的最大串长相差1
        else { // 节点p和q代表的状态的最大串长相差>1
            int nq=++tot; // 新建一个节点nq
            dian[nq]=dian[q]; // 节点nq克隆节点q的信息
            dian[nq].sz=0; // nq产生时，是一个分支节点，需要从后续儿子节点里更新获取sz
            dian[nq].len=dian[p].len+1; // 保证节点p和nq代表的状态的最大串长相差1
            num_sub[pos]-=dian[q].len-dian[dian[q].fa].len;
            dian[q].fa=dian[np].fa=nq;
            num_sub[pos]+=dian[q].len-dian[dian[q].fa].len;
            num_sub[pos]+=dian[nq].len-dian[dian[nq].fa].len;
            for(; p&&dian[p].ch[c]==q; p=dian[p].fa)dian[p].ch[c]=nq; // 以上为case 3，将节点p到SAM的起点的路径上的所有节点的字符c的出边指向的节点替换为nq
        }
    }
    num_sub[pos]+=dian[np].len-dian[dian[np].fa].len;
}
void init(string s){
    n=s.size();
    for(int i=0; i<n; i++){
        add(s[i]-'a',i);
    }
}
int b[N<<1],a[N<<1]; // b: 用于基数排序 a: 用于记录点号
// 按长度基数排序，短的在前长的在后
// 另一种方法是用vector直接建出parent树，对parent树直接dfs
void base_sort() {
    for(int i=1; i<=tot; ++i)b[dian[i].len]++;
    for(int i=1; i<=tot; ++i)b[i]+=b[i-1];
    for(int i=1; i<=tot; ++i)a[b[dian[i].len]--]=i;
}
// 用于逆拓扑序遍历求出sz
void get_sz() {
    for(int i=tot; i>=1; --i) {
        int u=a[i],fa=dian[u].fa;
        dian[fa].sz+=dian[u].sz;
```

```
    }
}
void build(string s){
    init(s);
    base_sort();
    get_sz();
}
/*求模式串
 有没有在字符串
 中出现过。
非常简单，只要根据
 的字符在SAM上进行转移，如果在某一步无法转移则没有出现过。例如在上面的图中，当
 为"aba" 时，走1->2->6->5可以处理完整个模式串，所以该模式串出现过；
 为"baa"时，走1->3->5后便无法再转移，所以该模式串未出现过。容易发现这个方法还可以求出出现过的最长前缀。

求模式串
 在字符串
 中出现的次数。
其实就是
 对应节点的
 集合的大小，这个直接在parent树上dp就可以了。注意那些原字符串的前缀对应的节点，划分后
 集合会丢失一个元素，所以这些节点的dp值要比子节点dp值之和多1。这个直接在建SAM的时候预处理就行，直接令dp[cur] = 1。

求字符串
 的最小表示法（循环同构字符串中字典序最小的一个）
这个巨简单，把
 重复一遍，建出其SAM，然后从开始节点往下走
 步，每次都走最小字符对应的边，得到的显然便是最小表示法。

求
 的本质不同子串个数
从parent tree的角度，每个节点代表了
 个子串，且每个节点代表的子串各不相同，所以只要对每个节点计算这个值然后相加即可。这个甚至可以在建SAM途中动态维护。

最长公共子串
为
 建SAM，对
 每一个前缀求它在
 中出现过的最长后缀。这个可以用类似KMP的方法，先尽量转移，转移不动则跳到parent tree上的父亲节点继续尝试转移，直到跳到根节点。整个过程中要维护当前最长后缀的长度。*/
```

# Manacher

```
char t[12000000],s[24000000];
int ans=0;
cin>>t;
int n=strlen(t);
for(int i=0; i<2*n+1; i++)
```

```
    s[i]=(i&1)?t[(i-1)/2]:'#';
n=strlen(s);
vector<int>d1(n);
for(int i = 0, l = 0, r = -1; i < n; i++) {
    int k = (i>r)?1:min(d1[l+r-i],r-i+1);
    while(i-k>=0&&i+k<n&&s[i-k]==s[i+k]) {
        k++;
    }
    d1[i]=k--;
    if(i+k>r) {
        l=i-k;
        r=i+k;
    }
    ans=max(ans,k);
}
cout<<s<<endl;
cout<<d1;
cout<<ans;
//``````````````````````````````````````````````````````````
char Ma[MAXN*2];
int Mp[MAXN*2],sum[MAXN];
void Manacher(char s[],int len)
{
    int l=0;
    Ma[l++]='$';
    Ma[l++]='#';
    for (int i=0;i<len;i++) Ma[l++]=s[i],Ma[l++]='#';
    Ma[l]=0;
    int mx=0,id=0;
    for (int i=0;i<l;i++)
    {
        Mp[i]=mx>i? min(Mp[2*id-i],mx-i):1;
        while (Ma[i+Mp[i]]==Ma[i-Mp[i]]) Mp[i]++;
        if (i+Mp[i]>mx) mx=i+Mp[i],id=i;
    }
}
```

# 最小表示法

```
string Necklace(string a){
    int I=1,J=2,len=a.length();
    string re=a;
    a=" "+a+a;
    while(J<=len){
        for(int k=0;k<len;k++){
            if(a[I+k]<a[J+k]){
                J+=k;
                break;
            }
```

```
            if(a[I+k]>a[J+k]){
                int temp=I;
                I=J;
                J=max(J,temp+k);
                break;
            }
        }
        J++;
    }
    for(int t=0;t<len;t++){
        re[t]=a[I+t];
    }
    return re;
}
```

# 数学

## 快速幂

```
int qPower(int base, int exponent) {
    int sum = 1;
    while (exponent != 0) {
        if ((exponent & 1) != 0) {
            sum *= base;
        }
        exponent = exponent >> 1;  // 对指数进行移位
        base *= base;               // 让base的次幂以2的倍数增长
    }
    return sum;
}
```

## 矩阵乘法

```
const int maxn = 25;
struct Matrix {
    int n,m;
    ll v[maxn][maxn];
    Matrix(int n,int m):n(n),m(m) {}
    void init() {                   //初始化矩阵
        memset(v,0,sizeof v);
    }
    ll* operator[] (int x){
        return v[x];
    }
    Matrix operator* (Matrix B) const {
        Matrix C(n,B.m);            //用来存放答案
        C.init();
        for(int i = 0; i < n; i++)
```

```
            for(int j = 0; j < B.m; j++)
                for(int g = 0; g < m; g++)      {
                    C[i][j] += v[i][g]*B[g][j];
                    C[i][j] %= mod;
                }

        return C;
    }
    Matrix qpow(int k){
        Matrix C(n,n),D(n,n);
        C.init();D.init();
        for(int i = 0; i < n; i++)
            C[i][i]=1;
        for(int i = 0; i < n; i++)
            for(int j = 0; j < n; j++)
                D[i][j]=v[i][j];
        while(k){
            if(k & 1) C = C * D;
            D = D * D;
            k >>= 1;
        }
        return C;
    }
    void print() { //输出该矩阵，用来测试
        for(int i = 0; i < n; i++) {
            for(int j = 0; j < m; j++)
                cout << v[i][j] << " ";
            cout << endl;
        }
    }
};
```

# gcd

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

# exgcd

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

## exgcd求乘法逆元

```
//拓欧求乘法逆元
ll exgcd(ll a,ll b,ll &x,ll &y) {
    if(b==0) {
        x=1;
        y=0;
        return a;
    }
    ll d=exgcd(b,a%b,y,x);
    y=y-a/b*x;
    return d;
}
ll inv(ll a,ll mod) {
    ll x,y;
    exgcd(a,mod,x,y);
    return (x%mod+mod)%mod;
}
```

## exgcd求解线性同余方程

```
ll exgcd(ll a,ll b,ll &x,ll &y) {
    if(b==0) {
        x=1;
        y=0;
        return a;
    }
    ll d=exgcd(b,a%b,y,x);
    y=y-a/b*x;
    return d;
}
bool liEu(ll a, ll n, ll b, ll& x, ll& k) {
  ll d = exgcd(a, n, x, k);
  if (b % d != 0) return 0;
  ll h = b / d;
  x *= h;
  //最小正整数解
  ll t = n / d;
  x = (x % t + t) % t;
  return 1;
}
```

## 线性筛

```
void init(int n) {
  for (int i = 2; i <= n; ++i) {
    if (!vis[i]) {
      pri[cnt++] = i;
    }
```

```
    for (int j = 0; j < cnt; ++j) {
      if (1ll * i * pri[j] > n) break;
      vis[i * pri[j]] = 1;
      if (i % pri[j] == 0) {
        // i % pri[j] == 0
        // 换言之，i 之前被 pri[j] 筛过了
        // 由于 pri 里面质数是从小到大的，所以 i乘上其他的质数的结果一定会被
        // pri[j]的倍数筛掉，就不需要在这里先筛一次，所以这里直接 break
        // 掉就好了
        break;
      }
    }
  }
}
```

# 大质数分解

```
ll ksm(ll a,ll b,ll p){
    ll ret=1;
    for(;b;b>>=1,a=(__int128)a*a%p)
        if(b&1)ret=(__int128)ret*a%p;
    return ret;
}
bool Miller_Rabin(ll p){
    if(p<2)return 0;
    if(p==2||p==3)return 1;
    ll d=p-1,r=0;
    while(!(d&1))++r,d>>=1;
    for(ll k=0;k<10;++k){
        ll a=rand()%(p-2)+2;
        ll x=ksm(a,d,p);
        if(x==1||x==p-1)continue;
        for(int i=0;i<r-1;++i){
            x=(__int128)x*x%p;
            if(x==p-1)break;
        }
        if(x!=p-1)return 0;
    }
    return 1;
}
ll Pollard_Rho(ll x){
    ll s=0,t=0;
    ll c=(ll)rand()%(x-1)+1;
    ll val=1;
    for(int goal=1;;goal<<=1,s=t,val=1){
        for(int step=1;step<=goal;step++){
            t=((__int128)t*t+c)%x;
            val=(__int128)val*abs(t-s)%x;
            if(step%127==0){
                ll d=__gcd(val,x);
                if(d>1)return d;
```

```
            }
        }
        ll d=__gcd(val,x);
        if(d>1)return d;
    }
}
```

# 原根

# 莫比乌斯反演

# 生成函数

# NTT

```
const ll P = 998244353, G = 3, Gi = 332748118;//通常情况下的模数 4179340454199820289
inline ll fastpow(ll a, ll k)
{
    ll base = 1;
    while (k)
    {
        if (k & 1) base = (base * a) % P;
        a = (a * a) % P;
        k >>= 1;
    }
    return base % P;
}
const int N = 1e6 + 10;//1≤n,m≤10^6
int limit = 1, r[N], len = 0;
ll a[N], b[N], g[N];
inline void NTT(ll* A, int type)
{
    for (int i = 0; i < limit; i++)
        if (i < r[i]) swap(A[i], A[r[i]]);
    for (int mid = 1; mid < limit; mid <<= 1)
    {
        ll Wn = fastpow(type ? G : Gi, (P - 1) / (mid << 1));
        for (int j = 0; j < limit; j += (mid << 1))
        {
            ll w = 1;
            for (int k = 0; k < mid; k++, w = (w * Wn) % P)
            {
                ll x = A[j + k], y = w * A[j + k + mid] % P;
                A[j + k] = (x + y) % P;
                A[j + k + mid] = (x - y + P) % P;
            }
        }
```

```
    }
}

void solve()
{
    int n,m;
    cin>>n>>m;
    while (limit <= m + n) limit <<= 1, len++;
    for (int i = 0; i < limit; i++)r[i] = (r[i >> 1] >> 1) | ((i & 1) << (len - 1));
    for (int i = 0; i < n; i++)//1 2 4 1
        cin>>a[i];
    for (int i = 0; i < m; i++)//1 2 4 1
        cin>>b[i];
    NTT(a,1);NTT(b,1);
    for (int i = 0; i < limit; i++)a[i] = a[i] * b[i] % P;
    NTT(a, 0);
    ll inv = fastpow(limit, P - 2);
    for (int i = 0; i < limit; i++) a[i] = a[i] * inv % P;
    for(int i=0;i<limit;i++)
    cout<<a[i]<<" ";
}
```

# MTT

```
//Luogu O2
#include <cmath>
#include <cstdio>
#include <cctype>
#include <cstring>
#include <algorithm>
#define rint register int
typedef long long ll;
typedef long double ld;

//Having A Daydream...

char In[1<<20],*p1=In,*p2=In,Ch;
#define Getchar (p1==p2&&(p2=(p1=In)+fread(In,1,1<<20,stdin),p1==p2)?EOF:*p1++)
inline int Getint()
{
    register int x=0;
    while(!isdigit(Ch=Getchar));
    for(;isdigit(Ch);Ch=Getchar)x=x*10+(Ch^48);
    return x;
}

char Out[22222222],*Outp=Out,St[22],*Tp=St;
inline void Putint(int x)
{
    do *Tp++=x%10^48;while(x/=10);
    do *Outp++=*--Tp;while(St!=Tp);
```

```
}

const double Eps=1e-8,Pi=std::acos(-1),e=std::exp(1);
struct Complex
{
    ld x,y;
    inline Complex operator+(const Complex &o)const{return (Complex){x+o.x,y+o.y};}
    inline Complex operator-(const Complex &o)const{return (Complex){x-o.x,y-o.y};}
    inline Complex operator*(const Complex &o)const{return (Complex){x*o.x-
y*o.y,x*o.y+y*o.x};}
    inline Complex operator/(const ld k)const{return (Complex){x/k,y/k};}
    inline Complex Conj(){return (Complex){x,-y};}
}Ome[1<<18],Inv[1<<18],I=(Complex){0,1};

int r[1<<18];
namespace Poly
{
    void Pre(int n)
    {
        for(rint i=0,l=(int)log2(n);i<n;++i)r[i]=(r[i>>1]>>1)|((i&1)<<(l-1));
        for(rint i=0;i<n;++i)
        {
            ld x=std::cos(2*Pi*i/n),y=std::sin(2*Pi*i/n);
            Ome[i]=(Complex){x,y},Inv[i]=(Complex){x,-y};
        }
    }

    void FFT(int n,Complex *A,Complex *T)
    {
        for(rint i=0;i<n;++i)if(i<r[i])std::swap(A[i],A[r[i]]);
        for(rint i=2;i<=n;i<<=1)
            for(rint j=0,h=i>>1;j<n;j+=i)
                for(rint k=0;k<h;++k)
                {
                    Complex Tmp=A[j+h+k]*T[n/i*k];
                    A[j+h+k]=A[j+k]-Tmp,A[j+k]=A[j+k]+Tmp;
                }
    }

    Complex P[1<<18],Q[1<<18];
    void Double_DFT(int n,Complex *A,Complex *B,Complex *T)
    {
        for(rint i=0;i<n;++i)P[i]=A[i]+B[i]*I,Q[i]=A[i]-B[i]*I;
        FFT(n,P,T);
        for(rint i=0;i<n;++i)Q[i]=(i?P[n-i]:P[0]).Conj();
        for(rint i=0;i<n;++i)A[i]=(P[i]+Q[i])/2,B[i]=(P[i]-Q[i])*I/-2;
    }

    Complex A1[1<<18],B1[1<<18],A2[1<<18],B2[1<<18];
    Complex A[1<<18],B[1<<18],C[1<<18];
    void MTT(int n,int p,int *F,int *G,int *S)
    {
```

```
        //这里为了方便直接设M=2^15=32768
        for(rint i=0;i<n;++i)
        {
            A1[i].x=F[i]>>15,B1[i].x=F[i]&0x7FFF;
            A2[i].x=G[i]>>15,B2[i].x=G[i]&0x7FFF;
        }
        //FFT(n,A1,Ome),FFT(n,B1,Ome),FFT(n,A2,Ome),FFT(n,B2,Ome);
        Double_DFT(n,A1,B1,Ome),Double_DFT(n,A2,B2,Ome);
        for(rint i=0;i<n;++i)
        {
            A[i]=A1[i]*A2[i];
            B[i]=A1[i]*B2[i]+A2[i]*B1[i];
            C[i]=B1[i]*B2[i];
        }
        FFT(n,A,Inv),FFT(n,B,Inv),FFT(n,C,Inv);
        //Double_DFT(n,A,B,Inv),FFT(n,C,Inv);//IDFT怎么合并?
        for(rint i=0;i<n;++i)
        {
            ll Av=(ll)round(A[i].x/n),Bv=(ll)round(B[i].x/n),Cv=(ll)round(C[i].x/n);
            S[i]=((Av%p<<30)+(Bv%p<<15)+Cv)%p;
        }
    }
}

int n,m,p,F[1<<18],G[1<<18],S[1<<18];

int main()
{
    n=Getint(),m=Getint(),p=Getint();
    for(rint i=0;i<=n;++i)F[i]=Getint();
    for(rint i=0;i<=m;++i)G[i]=Getint();
    for(m=n+m,n=1;n<=m;n<<=1);
    Poly::Pre(n),Poly::MTT(n,p,F,G,S);
    for(rint i=0;i<=m;++i)Putint(S[i]),*Outp++=i==m?'\n':' ';
    return fwrite(Out,1,Outp-Out,stdout),0;
}
```

# 组合数

```
const int N = 1e6 + 7, mod = 1e9 + 7;
int n, m, k, t;
int inv[N];
int fact[N], infact[N];

void init (int n)
{
    fact[0] = infact[0] = inv[0] = inv[1] = 1;
    for (int i = 2; i <= n; ++ i)
```

```
        inv[i] = 1ll * (mod - mod / i) * inv[mod % i] % mod;
    for (int i = 1; i <= n; ++ i) {
        fact[i] = 1ll * fact[i - 1] * i % mod;
        infact[i] = 1ll * infact[i - 1] * inv[i] % mod;
    }
}

int C(int n, int m)
{
    if(n < m) return 0;
    if(m == 0 || n == m) return 1;
    return 1ll * fact[n] * infact[m] % mod * infact[n - m] % mod;
}
```

# Lagrange插值

```
ll Lagrange(ll* g, ll k, ll n) //g是关于n的k次多项式，需要准备1~k+1处的值，求g[n] 即0~k次方的系数
{
    pre[0] = 1;
    suf[k + 2] = 1;
    for (int i = 1; i <= k + 2; i++)
        pre[i] = pre[i - 1] * (n - i) % mod;
    for (int i = k + 1; i >= 1; i--)
        suf[i] = suf[i + 1] * (n - i) % mod;
    ll res = 0;
    ll MOD = mod;
    for (int i = 1; i <= k + 1; i++)
    {
        ll t = g[i] * pre[i - 1] % mod * suf[i + 1] % mod * inv(fact[i - 1]) % mod * inv(fact[k + 1 - i]) % mod;
        if ((k + 1 - i) & 1)t = (mod - t) % mod;
        res = (res + t % mod) % mod;
    }
    return res;
}
```

# 高斯消元

```
void Gauss_Jordan()
{
    for (int i = 1; i <= n; ++i)//枚举列（项）
    {
        int max = i;
        for (int j = i + 1; j <= n; ++j)//选出该列最大系数
            if (fabs(a[j][i]) > fabs(a[max][i]))
                max = j;

        for (int j = 1; j <= n + 1; ++j)//交换
            swap(a[i][j], a[max][j]);
```

```
        if (!a[i][i])//最大值等于0则说明该列都为0，肯定无解
        {
            puts("No Solution");
            return;
        }
        for (int j = 1; j <= n; ++j)//每一项都减去一个数（即加减消元）
            if (j != i)
            {
                double temp = a[j][i] / a[i][i];
                for (int k = i + 1; k <= n + 1; ++k)
                {
                    a[j][k] -= a[i][k] * temp;
                    //a[j][k]-=a[j][i]*a[i][k]/a[i][i];
                }
            }
    }
    //上述操作结束后，矩阵会变成这样
    /*
    k1*a=e1
    k2*b=e2
    k3*c=e3
    k4*d=e4
    */
}
```

# 数据结构

## ST表

```
int  pre[N][30],a[N];
int find(int l,int r) {
    int k=log2(r-l+1);
    return max(pre[l][k],pre[r-(1<<k)+1][k]);
}
void init() {
    for(int i=1; i<=n; i++) pre[i][0]=a[i];
    for(int i=1; i<=20; i++)
        for(int t=1; t+(1<<i)-1<=n; t++)
            pre[t][i]=max(pre[t][i-1],pre[t+(1<<(i-1))][i-1]);
}
```

## 树状数组

## 单点修改，区间查询

```
int  t[N];
//区间最值
void add_max(int x,int k){
    for(;x<=n;x+=-x&x) t[x]=max(t[x],k);
}
int find_max(int x,int y){
    int re=0;
    while(x<=y){
        re=max(re,a[y]);
        y--;
        for(;y-(-y&y)>=x;y-=-y&y){
            re=max(re,t[y]);
        }
    }
    return re;
}
//区间和
void add_sum(int x,int k){
    for(;x<=n;x+=-x&x) t[x]+=k;
}
int find_sum(int x){
    int re=0;
    for(;x;x-=-x&x) re+=t[x];
    return re;
}
void clear(int x){
    for(;x<=n;x+=-x&x) t[x]=0;
}
```

## 区间修改，区间求和

```
ll c[2][N],sum[N];int n;
inline void add_inter(int k,int x,int d)
{
    for(;x<=n;x+=x&-x)c[k][x]+=d;
}
inline void add(int d,int l,int r)
{
    add_inter(0,l,d);
    add_inter(0,r+1,-d);
    add_inter(1,l,l*d);
    add_inter(1,r+1,-(r+1)*d);
}

inline ll ask(int k,int x)
{
    ll res = 0;
    for(;x;x-=x&-x)res += c[k][x];
```

```
    return res;
}
inline ll query(int l,int r)
{
    ll res = sum[r] + (r+1)*ask(0,r) - ask(1,r);
    res -= sum[l-1] + l*ask(0,l-1) - ask(1,l-1);
    return res;
}
```

# 线段树

## 单点修改，区间查询

```
int a[N],s[N];
struct Node {
    int ind,val;
    Node operator+(const Node& a) {
        Node ant;
        ant.ind=this->val<=a.val?this->ind:a.ind;
        ant.val=this->val<=a.val?this->val:a.val;
        return ant;
    }
} tr[N];
void build(int now,int l,int r) {
    if(l==r) {
        tr[now].ind=l;
        tr[now].val=s[l];
    } else {
        int mid=(l+r)>>1;
        build(now<<1,l,mid);
        build(now<<1|1,mid+1,r);
        tr[now]=tr[now<<1]+tr[now<<1|1];
    }
}
void update(int now,int l,int r,int aim,int k) {
    if(aim==0)return;
    if(l==r&&r==aim) {
        tr[now].val=k;
    } else {
        int mid=(l+r)>>1;
        if(aim<=mid)
            update(now<<1,l,mid,aim,k);
        else
            update(now<<1|1,mid+1,r,aim,k);
        tr[now]=tr[now<<1]+tr[now<<1|1];
    }
}
Node query(int now,int l,int r,int L,int R) {
    if(L<=l&&r<=R)return tr[now];
    int mid=(l+r)>>1;
    if(R<=mid)return query(now<<1,l,mid,L,R);
```

```
    if(L>=mid+1)return query(now<<1|1,mid+1,r,L,R);
    return query(now<<1,l,mid,L,R)+query(now<<1|1,mid+1,r,L,R);
}
```

## 区间修改，区间查询

```
const int M=1e9+7;
ll n,m,q,a[N];
struct tree{
    ll l,r,sum,lza,lzm;
}t[N<<2];
void build(ll p,ll l,ll r){
    t[p].l=l,t[p].r=r;
    t[p].lzm=1,t[p].lza=0;
    if(l==r){
        t[p].sum=a[l];
        return;
    }
    ll s=0;
    ll mid=l+r>>1;
    build(p<<1,l,mid);
    build(p<<1|1,mid+1,r);
    t[p].sum=(t[p<<1].sum+t[p<<1|1].sum)%q;
}
void pushdown(ll p){
    t[p<<1].sum=(t[p<<1].sum*t[p].lzm+t[p].lza*(t[p<<1].r-t[p<<1].l+1))%q;
    t[p<<1|1].sum=(t[p<<1|1].sum*t[p].lzm+t[p].lza*(t[p<<1|1].r-t[p<<1|1].l+1))%q;
    t[p<<1].lzm=(t[p].lzm*t[p<<1].lzm)%q;
    t[p<<1].lza=(t[p].lzm*t[p<<1].lza+t[p].lza)%q;
    t[p<<1|1].lzm=(t[p].lzm*t[p<<1|1].lzm)%q;
    t[p<<1|1].lza=(t[p].lzm*t[p<<1|1].lza+t[p].lza)%q;
    t[p].lza=0;
    t[p].lzm=1;
}
void add(ll p,ll l,ll r,ll k){
    if(t[p].r<l||t[p].l>r)
        return;
    if(t[p].l>=l&&t[p].r<=r){
        t[p].sum=((t[p].r-t[p].l+1)*k+t[p].sum)%q;
        t[p].lza=(t[p].lza+k)%q;
        return;
    }
    pushdown(p);
    if(t[p<<1].r>=l)
        add(p<<1,l,r,k);
    if(t[p<<1|1].l<=r)
        add(p<<1|1,l,r,k);
    t[p].sum=(t[p<<1].sum+t[p<<1|1].sum)%q;
}
void mul(ll p,ll l,ll r,ll k){
    if(t[p].r<l||t[p].l>r)
```

```
        return;
    if(t[p].l>=l&&t[p].r<=r){
        t[p].sum=(t[p].sum*k)%q;
        t[p].lza=(t[p].lza*k)%q;
        t[p].lzm=(t[p].lzm*k)%q;
        return;
    }
    pushdown(p);
    if(t[p<<1].r>=l)
        mul(p<<1,l,r,k);
    if(t[p<<1|1].l<=r)
        mul(p<<1|1,l,r,k);
    t[p].sum=(t[p<<1].sum+t[p<<1|1].sum)%q;
}
ll search(ll p,ll l,ll r){
    if(t[p].r<l||t[p].l>r)
        return 0;
    if(t[p].r<=r&&t[p].l>=l)
        return t[p].sum;
    pushdown(p);
    ll sum=0;
    if(t[p<<1].r>=l)
        sum+=search(p<<1,l,r);
    if(t[p<<1|1].l<=r)
        sum+=search(p<<1|1,l,r);
    return sum;
}
```

## 线段树维护区间绝对众数

```
pair <int,int> merge(pair<int,int> a,pair<int,int> b){
    if(a.first==b.first) return {a.first,a.second+b.second};
    else{
        if(a.second>=b.second) return {a.first,a.second-b.second};
        else return {b.first,b.second-a.second};
    }
}
void build(ll p,ll l,ll r){
    t[p].l=l,t[p].r=r;
    if(l==r){
        if(a[l]==1||a[l]==-1) t[p].pa={a[l],0};
        else t[p].pa={a[l],1};
        return;
    }
    ll s=0;
    ll mid=l+r>>1;
    build(p<<1,l,mid);
    build(p<<1|1,mid+1,r);
    t[p].pa=merge(t[p<<1].pa,t[p<<1|1].pa);
}
pair<int,int> find_1(ll p,ll l,ll r){
```

```
    if(t[p].r<l||t[p].l>r)
        return {0,0};
    if(t[p].r<=r&&t[p].l>=l)
        return t[p].pa;
    ll sum=0;
    pair<int,int> re={0,0};
    if(t[p<<1].r>=l){
        pair<int,int> temp=find_1(p<<1,l,r);
        re=merge(re,temp);
    }
    if(t[p<<1|1].l<=r){
        pair<int,int> temp=find_1(p<<1|1,l,r);
        re=merge(re,temp);
    }
    return re;
}
```

## 可持久化权值线段树

```
const int N=200005;
int n,n1,m,a[N],a1[N],tim,root[N],l,r,k;
struct node{
    int l,r,v;
}t[N<<5];
int build(int l,int r){
    int id=++tim;
    if(l==r)
        return id;
    int mid=(l+r)>>1;
    t[id].l=build(l,mid);
    t[id].r=build(mid+1,r);
    return id;
}
int update(int p,int l,int r,int idx){
    int id=++tim;
    t[id]=t[p];
    t[id].v++;
    if(l==r) return tim;
    int mid=(l+r)>>1;
    if(idx<=mid)
        t[id].l=update(t[id].l,l,mid,idx);
    else
        t[id].r=update(t[id].r,mid+1,r,idx);
    return id;
}
int query(int p1,int p2,int l,int r,int k){
    int re,mid=(l+r)>>1,x=t[t[p2].l].v-t[t[p1].l].v;
    if(l==r)
        return l;
    if(x<k)
        return query(t[p1].r,t[p2].r,mid+1,r,k-x);
```

```
    else
        return query(t[p1].l,t[p2].l,l,mid,k);
}
```

## 平衡树

## KD树

## LCT

# 图论

## 最小生成树

```
ll ans = 0,n,m;
struct Edge {
    ll a,b,value;
    friend bool operator< (Edge n1, Edge n2) {
        return n1.value > n2.value;
    }
};
priority_queue<Edge>qn;

//--------------------------

int f[N];

int find (int x) { //并查集的find
    return f[x] == x ? x : f[x] = find(f[x]);
}

void make(int a, int b) {
    int x = find(a);
    int y = find(b);
    f[y] = x;
}

void init() {
    for(int i=1; i<=n; i++) {
        f[i]=i;
    }
}
//---------------------------
ll cnt=0;
void Kruskal() {
    while(qn.size()) {
        Edge x=qn.top();
        qn.pop();
        if(find(x.a)!=find(x.b)) {
//      cout<<"!";
```

```
            make(x.a,x.b);
            ans+=x.value;
            cnt++;
        }
    }
}
void solve() {
    cin>>n>>m;
    init();
    while(m--) {
        Edge start;
        int a,b,v;
        cin>>a>>b>>v;
        start.a=a;
        start.b=b;
        start.value=v;
        qn.push(start);
    }
    Kruskal();
    if(cnt==n-1) {
        cout<<ans<<endl;
    }
}
```

## LCA

```
vector<int>ed[N];
int depth[500001], fa[500001][22], lg[500001];
void dfs(int now, int fath) {
    fa[now][0] = fath;
    depth[now] = depth[fath] + 1;
    for(int i = 1; i <= lg[depth[now]]; ++i)
        fa[now][i] = fa[fa[now][i-1]][i-1];
    for(int i = 0; i<ed[now].size(); i ++)
        if(ed[now][i] != fath) dfs(ed[now][i], now);
}
int LCA(int x, int y) {
    if(depth[x] < depth[y]) swap(x, y);
    while(depth[x] > depth[y])
        x = fa[x][lg[depth[x]-depth[y]] - 1];
    if(x == y) return x;
    for(int k = lg[depth[x]] - 1; k >= 0; --k)
        if(fa[x][k] != fa[y][k])
            x = fa[x][k], y = fa[y][k];
    return fa[x][0];
}
void solve() {
    cin>>n>>m>>s;
    for(int i=1; i<=n-1; i++) {
        cin>>x>>y;
        ed[x].push_back(y);
```

```
        ed[y].push_back(x);
    }
    for(int i = 1; i <= n; ++i)
        lg[i] = lg[i-1] + (1 << lg[i-1] == i);
    dfs(s, 0);
}
//--------------------------------------------------------------------

int dep[N],lg[N<<1],f[N<<1][21];
int dfn[N<<1],que[N<<1],idx;
void dfs(int u,int pa)
{
    dfn[u]=++idx,que[idx]=u;
    dep[u]=dep[pa]+1;
    for(int i=head[u];i;i=e[i].nxt){
        int v=e[i].to;
        if(v==pa)continue;
        dfs(v,u);
        que[++idx]=u;
    }
}
void buildst()
{
    For(i,1,idx)f[i][0]=que[i];
    For(j,1,20)
        for(int i=1;i+(1<<j)<=idx;++i){
            int f1=f[i][j-1],f2=f[i+(1<<j-1)][j-1];
            f[i][j]=dep[f1]<dep[f2]?f1:f2;
        }
    lg[0]=-1;
    For(i,1,idx)lg[i]=lg[i>>1]+1;
}
```

# 单源最短路

```
bool vis[N];
ll n,m,s;
ll d[N];
struct Edge {
    int to,value;
};
vector<Edge>ed[N];
struct Node {
    int u;
    ll value;
    friend bool operator< (Node n1, Node n2) {
        return n1.value > n2.value;
    }
};
priority_queue <Node> qn;
void dijkstra(int v0) {
```

```
    for(int i = 1; i<=n; i++) {
        d[i] = Maxn;
        vis[i] = 0;
    }

    qn.push((Node) {
        v0,0
    });
    d[v0]=0;
    while(qn.size()) {
        Node x=qn.top();
        qn.pop();
        int now=x.u,value=x.value;
        if(vis[now])continue;
        vis[now] = 1;
        for(int i=0; i<ed[now].size(); i++) {
            int to = ed[now][i].to;
            if(d[now]+ed[now][i].value<d[to]) {
                d[to]=d[now]+ed[now][i].value;
                qn.push((Node) {
                    to,d[to]
                });
            }
        }
    }
}
void solve() {
    cin>>n>>m>>s;
    for(int i=0; i<m; i++) {
        int x,y,v;
        cin>>x>>y>>v;
        Edge start;
        start.to=y;
        start.value=v;
        ed[x].push_back(start);
    }
    dijkstra(s);
    for(int i=1; i<=n; i++) {
        cout<<d[i]<<" ";
    }
}
```

# 树链剖分

```
int sz[M],son[M],top[M],dep[M];
int Dfn[M],Low[M],tot_dfs;
int fa[M];
void dfs(int now){
    Dfn[now]=++tot_dfs;
    sz[now]=1;son[now]=0;
    for(int i=head[now];i;i=edge[i].nx){
```

```
        int nxt=edge[i].to;
        if(nxt==fa[now])continue;
        fa[nxt]=now;
        dep[nxt]=dep[now]+1;
        dfs(nxt);
        sz[now]+=sz[nxt];
        if(sz[son[now]]<sz[nxt])son[now]=nxt;
    }
    Low[now]=tot_dfs;
}
void dfs_top(int now){
    if(son[now]){
        top[son[now]]=top[now];
        dfs_top(son[now]);
    }
    for(int i=head[now];i;i=edge[i].nx){
        int nxt=edge[i].to;
        if(nxt==fa[now]||nxt==son[now])continue;
        top[nxt]=nxt;
        dfs_top(nxt);
    }
}
int LCA(int a,int b){
    while(top[a]!=top[b]){
        if(dep[top[a]]<dep[top[b]])b=fa[top[b]];
        else a=fa[top[a]];
    }
    return dep[a]<dep[b]?a:b;
}
bool In(int x,int y){
    return Dfn[y]<=Dfn[x]&&Dfn[x]<=Low[y];
}
```

# 启发式合并

# 树哈希

```
int head[N],flag,base=114514,size[N],d[N],mx=1e9,n;
struct node1{
    int to,next;
}p[N];
void add(int x,int y){
    p[++flag].to=y;
    p[flag].next=head[x];
    head[x]=flag;
}
//ll Has(int x,int fa){
//  ll i,y,res=23533;
//  vector<ll > t;
//  for(i=head[x];i;i=p[i].next){
//      y=p[i].to;if(y==fa)continue;
```

```
//      t.push_back(Has(y,x));
//  }
//  sort(t.begin(),t.end());
//  for(i=0;i<t.size();++i)res=((res*base)^t[i])%mod;
//  return res;
//}
ll hsh[N];
mt19937 rnd(random_device{}());
uniform_int_distribution<ll> dist(0,ULLONG_MAX);
const ll s=dist(rnd);
ll xorshift(ll x) {
    x^=x<<13;
    x^=x>>7;
    x^=x<<17;
    return x;
}
ll Has(int x,int fa) {
    ll res=s;
    for (int i=head[x];i;i=p[i].next)
    {
        int y=p[i].to;
        if (y==fa) continue;
        res+=xorshift(Has(y,x));
    }
    return res;
}
void dfs_barycenter(int x,int fa){
    size[x]=1;
    int res=0;
    for(int i=head[x];i;i=p[i].next){
        if(p[i].to==fa) continue;
        dfs_barycenter(p[i].to,x);
        size[x]+=size[p[i].to];
        res=max(res,size[p[i].to]);
    }
    res=max(res,n-size[x]);
    d[x]=res;
    mx=min(mx,res);
}
pair<int,int> find_barycenter(int x,int fa){
    pair<int,int> pos={-1,-1};
    for(int j=1;j<=n;j++){
        if(d[j]==mx){
            if(pos.first==-1) pos.first=j;
            else pos.second=j;
        }
    }
    return pos;
}
```

## 2-SAT

```
int dfsClock = 0, sccCnt = 0;
int low[2 * N], dfn[2 * N], ins[2 * N], color[2 * N], n;
stack<int>stk;
void tarjan(int x) //找环
{
    low[x] = dfn[x] = ++dfsClock;
    stk.push(x); ins[x] = true;
    for (int i = head[x]; i; i = nex[i])
    {
        int y = ver[i];
        if (!dfn[y])tarjan(y), low[x] = min(low[x], low[y]);
        else if (ins[y])low[x] = min(low[x], dfn[y]);
    }
    if (low[x] == dfn[x])
    {
        ++sccCnt;
        do {
            color[x] = sccCnt;
            x = stk.top(); stk.pop(); ins[x] = false;
        } while (low[x] != dfn[x]);
    }
}
int main()//a能推出b 就连一条a到b的有向边
{
    for (int i = 1; i <= 2 * n; ++i)    // 使用 Tarjan 找环，得到的 color[x] 是 x 所在的
scc 的拓扑逆序。
        if (!dfn[i]) tarjan(i);
    for (int i = 1; i <= n; ++i)
        if (color[i] == color[i + n])
        { // x 与 -x 在同一强连通分量内，一定无解
            return;
        }
    for (int i = 1; i <= n; ++i)    //有解
    {
        //所在的强连通分量的拓扑序大的为1
        cout << (color[i] < color[i + n]) << " ";// 如果不使用 Tarjan 找环，请改成大于号
    }
}
```

# 虚树

节点多查询少

# kruskal重构树

```
ll n,m,q,flag,head[N],siz[N],Fa[N][30],pre[N],w[N];
struct node{
    int x,y,v;
}a[N];
```

```
struct node1{
    int to,next;
}p[N];
int cmp(node a,node b){
    return a.v<b.v;
}
int find(int x){
    if(pre[x]==x) return x;
    return pre[x]=find(pre[x]);
}
void add(int x,int y){
    int x1=find(x);
    int y1=find(y);
    pre[x1]=y1;
}
void add_t(int x,int y){
    p[++flag].to=y;
    p[flag].next=head[x];
    head[x]=flag;
}
int kruskal(int n,int m){
    int cnt=n;
    for(int i=1;i<=n;i++) pre[i]=i;
    sort(a+1,a+1+m,cmp);
    for(int i=1;i<=m;i++){
        int x=find(a[i].x),y=find(a[i].y);
        if(x==y) continue;
        ++cnt;
        pre[cnt]=cnt;
        w[cnt]=a[i].v;
        add(x,cnt),add(y,cnt);
        add_t(cnt,x),add_t(x,cnt),add_t(cnt,y),add_t(y,cnt);
    }
    return cnt;
}
void dfs(int x,int fa){
    Fa[x][0]=fa;
    for(int i=1;i<=20;i++){
        Fa[x][i]=Fa[Fa[x][i-1]][i-1];
    }
    for(int i=head[x];i;i=p[i].next){
        if(fa==p[i].to) continue;
        dfs(p[i].to,x);
        siz[x]+=siz[p[i].to];
    }
}
```

# 网络流

## 最大流

```cpp
template <class T>
struct Flow {
    struct edge {
        int to;
        T cost;
    };
    vector<edge> edges;
    vector<vector<int>> e;
    vector<int> cur, h;
    int n;
    Flow(int _n) : n(_n), e(_n + 1) {}

    bool bfs(int s, int t) {
        h.assign(n + 1, -1);
        h[s] = 0;
        queue<int> q;
        q.push(s);
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (int i : e[u]) {
                int v = edges[i].to;
                T c = edges[i].cost;
                if (c > 0 && h[v] == -1) {
                    h[v] = h[u] + 1;
                    if (v == t) return true;
                    q.push(v);
                }
            }
        }
        return false;
    }
    T dfs(int u, int t, T f) {
        if (u == t) return f;
        auto r = f;
        for (int &i = cur[u]; i < int(e[u].size()); ++i) {
            int j = e[u][i];
            int v = edges[j].to;
            T c = edges[j].cost;
            if (c > 0 && h[v] == h[u] + 1) {
                auto a = dfs(v, t, min(r, c));
                edges[j].cost -= a;
                edges[j ^ 1].cost += a;
                r -= a;
                if (r <= 0) return f;
            }
        }
        return f - r;
    }
    inline void addEdge(int u, int v, T c) {
        e[u].push_back(edges.size());
        edges.push_back({v, c});
```

```
        e[v].push_back(edges.size());
        edges.push_back({u, 0});//无向图{u, c}
    }
    T maxflow(int s, int t) {
        T ans = 0;
        while (bfs(s, t)) {
            cur.assign(n + 1, 0);
            ans += dfs(s, t, numeric_limits<T>::max());//!!double
        }
        return ans;
    }
};
```

# 费用流

```
int tot = 1;
template<class T>
struct Flow {
    const int INF = 0x3f3f3f3f;
    static const int N = 5e3 + 10, M = 1e5 + 10;
    int ver[M], nex[M];
    int head[N], dep[N], last[N], pre[N];
    bool vis[N];
    T cap[M], cost[M];
    T flow[N], dis[N];
    T max_flow = 0, min_cost = 0;
    inline bool spfa(int s, int t)
    {
        memset(dis, 0x3f, sizeof dis);
        memset(flow, 0x3f, sizeof flow);
        memset(vis, 0, sizeof vis);
        queue<int>q;
        q.push(s); vis[s] = 1; dis[s] = 0; pre[t] = -1;
        while (!q.empty())
        {
            int x = q.front();
            q.pop();
            vis[x] = 0;
            for (int i = head[x]; i; i = nex[i])
            {
                int y = ver[i];
                T _cost = cost[i];
                if (cap[i] > 0 && dis[y] > dis[x] + _cost)
                {
                    dis[y] = dis[x] + _cost;
                    pre[y] = x;
                    last[y] = i;
                    flow[y] = min(flow[x], cap[i]);
                    if(!vis[y])
                    {
                        vis[y] = 1;
```

```
                        q.push(y);
                    }
                }
            }
        }
        return pre[t] != -1;
    }
    inline void addEdge(int u, int v, T c, T dis) {
        ver[++tot] = v; nex[tot] = head[u]; head[u] = tot; cap[tot] = c; cost[tot] =
dis;
        ver[++tot] = u; nex[tot] = head[v]; head[v] = tot; cap[tot] = 0; cost[tot] =
-dis;
    }
    inline void MCMF(int s, int t) {
        while (spfa(s, t)) {
            int now = t;
            max_flow += flow[t];
            min_cost += flow[t] * dis[t];
            while (now != s)
            {
                cap[last[now]] -= flow[t];
                cap[last[now] ^ 1] += flow[t];//注意异或1的位置
                now = pre[now];
            }
        }
    }
};
```

# 计算几何

## 计算两圆交点

```
const double Pi=acos(-1);
const double eps=1e-18;
struct Point {
    double x,y;
    Point(double x=0,double y=0):x(x),y(y) {};
} p1,p2,p3;
Point operator +(Point A,Point B) {
    return Point(A.x+B.x,A.y+B.y);
}
Point operator -(Point A,Point B) {
    return Point(A.x-B.x,A.y-B.y);
}
Point operator *(Point A,double B) {
    return Point(A.x*B,A.y*B);
}
Point operator /(Point A,double B) {
    return Point(A.x/B,A.y/B);
}
```

```
int dcmp(double x) {
    if(fabs(x)<eps)return 0;
    return x<0?-1:1;
}
bool operator<(const Point&a,const Point&b) {
    return a.x<b.x||(a.x==b.x&&a.y<b.y);
}
bool operator == (const Point &a,const Point &b) {
    return dcmp(a.x-b.x)==0&&dcmp(a.y-b.y)==0;
}
double Dot(Point A,Point B) {
    return A.x*B.x+A.y*B.y;
}
double Length(Point A) {
    return sqrt(Dot(A,A));
}
double Cross(Point A,Point B) {
    return A.x*B.y-A.y*B.x;
}
double Angle(Point A,Point B) {//利用点积 求出向量 A B 的夹角 cos值
    return acos(Dot(A,B)/Length(A)/Length(B));
}
double Angle(Point A) {
    return atan2(A.y, A.x);
}
struct Circle {   //圆
    Point c;
    double r;
    Circle(Point c = Point(0, 0), double r = 0):c(c),r(r) {}
    Point point(double a) {
        return Point(c.x+cos(a)*r,c.y+sin(a)*r);
    }
};
double dis(Point A,Point B) {
    return sqrt((A.x-B.x)*(A.x-B.x)+(A.y-B.y)*(A.y-B.y));
}
int circle_circle(Circle C1,Circle C2,vector<Point>&sol) { //圆与圆的交点，核心思想就是利用余弦定理求出两交点与圆心连线的夹角，显然可以根据已知距离，和圆心坐标，求出交点坐标，不同去画图，高中几何知识
    double d=Length(C1.c-C2.c);
    if(dcmp(d)==0) {
        if(dcmp(C1.r-C2.r)==0)return -1;
        return 0;
    }
    if(dcmp(C1.r+C2.r-d)<0)return 0;
    if(dcmp(fabs(C1.r-C2.r)-d)>0)return 0;
    double a=Angle(C2.c-C1.c);
    double da=acos((C1.r*C1.r+d*d-C2.r*C2.r)/(2*d*C1.r));
    Point p1=C1.point(a-da),p2=C1.point(a+da);
    sol.push_back(p1);
    if(p1==p2)return 1;
    sol.push_back(p2);
```

```
    return 2;
}
```