「明治十七年的上海爱丽丝」

HiedanoAkyuu、Oneman233、KR12138

2019 年 11 月 23 日

目录

1 字符串 **1**
- 1.1 KMP . . . 1
- 1.2 EX-KMP . . . 1
- 1.3 Manacher . . . 1
- 1.4 串的最小表示 . . . 2
- 1.5 后缀数组 . . . 2
  - 1.5.1 倍增 SA . . . 2
  - 1.5.2 DC3 . . . 2
- 1.6 回文自动机 . . . 2
- 1.7 AC 自动机 . . . 3
  - 1.7.1 多模匹配 . . . 3
  - 1.7.2 自动机上 DP . . . 3
- 1.8 后缀自动机 . . . 4

2 计算几何 **4**
- 2.1 二维几何 . . . 4
- 2.2 三维几何 . . . 6

3 图论 **6**
- 3.1 最短路 . . . 6
  - 3.1.1 Dijkstra . . . 6
  - 3.1.2 SPFA . . . 6
  - 3.1.3 Floyd . . . 6
  - 3.1.4 负环 . . . 6
  - 3.1.5 差分约束 . . . 7
- 3.2 最小生成树 . . . 8
  - 3.2.1 Prim . . . 8
  - 3.2.2 Kruskal . . . 8
  - 3.2.3 最小生成树计数 . . . 9
  - 3.2.4 次小生成树 . . . 9
  - 3.2.5 最小乘积生成树 . . . 10
- 3.3 树的直径 . . . 11
- 3.4 LCA . . . 11
  - 3.4.1 Tarjan 离线 . . . 11
  - 3.4.2 倍增 LCA . . . 12
- 3.5 无向图与有向图联通性 . . . 12
  - 3.5.1 割点 . . . 12
  - 3.5.2 桥 . . . 13
  - 3.5.3 e-DCC . . . 13
  - 3.5.4 v-DCC . . . 13
  - 3.5.5 SCC . . . 13
  - 3.5.6 2-SAT . . . 14
  - 3.5.7 支配树 . . . 15
- 3.6 二分图 . . . 15
  - 3.6.1 最大匹配-匈牙利 . . . 15
  - 3.6.2 带权匹配-KM . . . 15
- 3.7 网络流 . . . 16
  - 3.7.1 最大流-Dinic . . . 16
  - 3.7.2 最小费用最大流-Dij+Dinic . . . 17
  - 3.7.3 最小费用最大流-SPFA+Dinic . . . 17
  - 3.7.4 上下界流 . . . 18
- 3.8 欧拉路 . . . 18
- 3.9 Prufer 序列 . . . 19

4 数据结构 **19**
- 4.1 树状数组 . . . 19
- 4.2 线段树 . . . 19
  - 4.2.1 带优先级线段树 . . . 19
  - 4.2.2 李超线段树 . . . 19
  - 4.2.3 吉司机线段树 . . . 20
  - 4.2.4 线段树维护扫描线 . . . 20
  - 4.2.5 区间图 . . . 21
- 4.3 RMQ . . . 22
  - 4.3.1 一维 . . . 22
  - 4.3.2 两维 . . . 22
- 4.4 树链剖分 . . . 22
  - 4.4.1 点剖分 . . . 22
  - 4.4.2 边剖分 . . . 24
- 4.5 平衡树 . . . 24
  - 4.5.1 Treap . . . 24
  - 4.5.2 Splay . . . 25
  - 4.5.3 红黑树-pbds . . . 25
- 4.6 动态树 . . . 25
- 4.7 主席树 . . . 25
- 4.8 树套树 . . . 26
  - 4.8.1 线段树套 Treap . . . 26
  - 4.8.2 树状数组套线段树 . . . 28
- 4.9 K-D Tree . . . 29
- 4.10 分治 . . . 30
  - 4.10.1 CDQ . . . 30
  - 4.10.2 点分治 . . . 31
  - 4.10.3 dsu on tree . . . 31
  - 4.10.4 整体二分 . . . 31
- 4.11 分块 . . . 31
  - 4.11.1 普通分块 . . . 31
  - 4.11.2 莫队 . . . 32
- 4.12 线性基 . . . 33
- 4.13 珂朵莉树 . . . 33
- 4.14 跳舞链 . . . 34

5 动态规划 **34**
- 5.1 SOS . . . 34
- 5.2 动态 DP . . . 34
- 5.3 插头 DP . . . 34

6 数学 **34**
- 6.1 三分 . . . 34
- 6.2 矩阵类 . . . 35
- 6.3 质数筛 . . . 35
  - 6.3.1 埃筛 . . . 35
  - 6.3.2 线筛 . . . 35
- 6.4 质数判定 . . . 35
  - 6.4.1 Miller Rabin . . . 35
- 6.5 质因数分解 . . . 35
  - 6.5.1 Pollard-Rho . . . 35
- 6.6 逆元 . . . 36
  - 6.6.1 EX-GCD 求逆元 . . . 36
  - 6.6.2 线性筛逆元 . . . 36
  - 6.6.3 阶乘逆元 . . . 36
- 6.7 欧拉函数 . . . 36
  - 6.7.1 欧拉线筛 . . . 36
  - 6.7.2 求单个数的欧拉函数 . . . 36
  - 6.7.3 欧拉降幂 . . . 36
  - 6.7.4 一般积性函数求法 . . . 37
- 6.8 EX-GCD . . . 37
- 6.9 同余方程组 . . . 37
  - 6.9.1 CRT . . . 37
  - 6.9.2 EXCRT . . . 37
- 6.10 N 次剩余 . . . 38
  - 6.10.1 模奇质数的 2 次剩余 . . . 38
  - 6.10.2 N 次剩余 . . . 38
- 6.11 数论分块 . . . 38
- 6.12 高斯消元 . . . 38
  - 6.12.1 普通消元 . . . 38
  - 6.12.2 异或方程组消元 . . . 38
- 6.13 莫比乌斯反演 . . . 38
  - 6.13.1 莫比乌斯函数 . . . 38
  - 6.13.2 杜教筛 . . . 38
  - 6.13.3 洲阁筛 . . . 38
  - 6.13.4 min25 筛 . . . 38

6.14 BSGS 38
6.15 FFT 39
6.16 FFT 39
6.17 DFT 次数优化 FFT 40
6.18 FWT 40
6.19 NTT 40
6.20 NTT 40
6.21 NTT 分治求卷积 41
6.22 NTT 求多项式逆 42
6.23 数值计算 43
6.23.1 辛普森 43
6.23.2 自适应辛普森 43
6.24 康拓展开 43
6.25 卢卡斯定理 44
6.25.1 Lucas(循环或递归实现) 44
6.25.2 EXLucas(分块实现) 44
6.26 博弈论 45
6.26.1 SG 函数 45

7 其他 **45**
7.1 快读快写 45
7.2 高精度 45
7.3 约瑟夫环 49
7.4 悬线法 49
7.5 蔡勒公式 49
7.6 三角公式 49
7.7 海伦公式 49
7.8 匹克定理 49
7.9 组合计数 49
7.9.1 计数原理 49
7.9.2 卡特兰数 49
7.9.3 Polya 49
7.9.4 二项式反演公式 49
7.9.5 斯特林反演公式 49
7.9.6 组合数恒等式 49

1 字符串

1.1 KMP

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

const int MAXN=1000005;
char s1[MAXN],s2[MAXN];
int nxt[MAXN];

/*
    nxt[i] s2[i-x..i-1]=s2[0..x-1]且x最大
    即s2[0..i]的真前缀与真后缀的最大匹配
    "ABAAB\0"=>[-1 0 0 1 1 2]
*/

void get_fail(char *s,int l)
{
    int i=0,j;
    j=nxt[0]=-1;
    while(i<l)
    {
        while(~j&&s[j]!=s[i]) j=nxt[j];
        nxt[++i]=++j;
    }
}

void kmp(char *s1,char *s2,int l1,int l2)
{
    int i=0,j=0;
    get_fail(s2,l2);
    while(i<l1)
    {
        while(~j&&s1[i]!=s2[j]) j=nxt[j];
        i++,j++;
        if(j>=l2); //匹配上了
    }
}

int main()
{
    scanf("%s%s",s1,s2);
    int l1=strlen(s1),l2=strlen(s2);
    kmp(s1,s2,l1,l2);
    for(int i=0;i<=l2;i++)
        printf("%d ",nxt[i]);
    return 0;
}
```

1.2 EX-KMP

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

/*
    ex[i]: s1[i..l1-1]与s2的最大公共前缀长度
    exnext[i]: s2[i..l2-1]与s2的最大公共前缀长度
    get_exnext(s2) 求exnext[]
    exkmp(s1,s2) 求ex[]
*/

const int N=50005;
char s1[N],s2[N];
int ex[N],exnext[N];

void get_exnext(char s[N])
{
    int n=strlen(s),p=1,j,i;
    exnext[0]=n;
    for(i=0;i<n-1&&s[i]==s[i+1];i++);
    exnext[1]=i;
    for(i=2;i<n;i++)
        if(exnext[i-p]+i<p+exnext[p])
            exnext[i]=exnext[i-p];
        else
        {
            j=exnext[p]+p-i;
            if(j<0) j=0;
            while(i+j<n&&s[j]==s[i+j]) j++;
            exnext[i]=j;
            p=i;
        }
}

void exkmp(char s1[N],char s2[N])
{
    int l1=strlen(s1),l2=strlen(s2),p=0,i,j;
    get_exnext(s2);
    for(i=0;i<l1&&i<l2&&s1[i]==s2[i];i++);
    ex[0]=i;
    for(int i=1;i<l1;i++)
    {
        if(exnext[i-p]+i<p+ex[p])
            ex[i]=exnext[i-p];
        else
        {
            j=ex[p]+p-i;
            if(j<0) j=0;
            while(i+j<l1&&s1[i+j]==s2[j]) j++;
            ex[i]=j;
            p=i;
        }
    }
}
```

1.3 Manacher

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

const int N=1000005;
int cnt,len,ans,p[N*2];
char s[N],ss[N*2];

void init() //将每两个字符中插入一个字符
{
    len=strlen(s),cnt=1;
    ss[0]='!',ss[cnt]='#';
    for(int i=0;i<len;i++)
        ss[++cnt]=s[i],ss[++cnt]='#';
}

void manacher()
{
    int pos=0,mx=0;
    for(int i=1;i<=cnt;i++)
    {
```

```
        if(i<mx) p[i]=min(p[pos*2-i],mx-i);
        else p[i]=1;
        while(ss[i+p[i]]==ss[i-p[i]]) p[i]++;
        if(mx<i+p[i]) mx=i+p[i],pos=i;
        ans=max(ans,p[i]-1);
    }
}

int main()
{
    scanf("%s",s);
    init();
    manacher();
    printf("%d\n",ans);
    return 0;
}
```

1.4 串的最小表示

1.5 后缀数组

1.5.1 倍增 SA

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

/*
    str[0..len-1] 原串
    sa[1..len] 排名第i的后缀的下标[1..len]
    Rank[1..len] 从i开始的后缀的排名[1..len]
    height[1..len] 排名第i的后缀与排名第i-1的后缀的lcp
    i开始的后缀与j开始的后缀的lcp (Rank[i]<Rank[j])
    min{height[Rank[i]+1..Rank[j]]}
*/

const int MAXN=100005;
const int inf=0x3f3f3f3f;
int wa[MAXN],wb[MAXN],wv[MAXN],wz[MAXN],sa[MAXN],Rank
    [MAXN],height[MAXN];
char str[MAXN];

inline bool cmp(int *r,int a,int b,int l){return r[a
    ]==r[b]&&r[a+l]==r[b+l];}

void da(const char r[],int sa[],int n,int m)
{
    int i,j,p,*x=wa,*y=wb,*t;
    for(i=0;i<m;i++) wz[i]=0;
    for(i=0;i<n;i++) wz[x[i]=r[i]]++;
    for(i=1;i<m;i++) wz[i]+=wz[i-1];
    for(i=n-1;i>=0;i--) sa[--wz[x[i]]]=i;
    for(j=1,p=1;p<n;j*=2,m=p)
    {
        for(p=0,i=n-j;i<n;i++) y[p++]=i;
        for(i=0;i<n;i++) if(sa[i]>=j) y[p++]=sa[i]-j;
        for(i=0;i<n;i++) wv[i]=x[y[i]];
        for(i=0;i<m;i++) wz[i]=0;
        for(i=0;i<n;i++) wz[wv[i]]++;
        for(i=1;i<m;i++) wz[i]+=wz[i-1];
        for(i=n-1;i>=0;i--) sa[--wz[wv[i]]]=y[i];
        for(t=x,x=y,y=t,p=1,x[sa[0]]=0,i=1;i<n;i++)
            x[sa[i]]=cmp(y,sa[i-1],sa[i],j)?p-1:p++;
    }
}

void calheight(const char *r,int *sa,int n)
{
    int i,j,k=0;
    for(i=1;i<=n;i++) Rank[sa[i]]=i;
    for(i=0;i<n;height[Rank[i++]]=k)
    for(k?k--:0,j=sa[Rank[i]-1];r[i+k]==r[j+k];k++);
    for(int i=n;i>=1;--i) sa[i]++,Rank[i]=Rank[i-1];
}

int main()
{
    scanf("%s",str);
    int len=strlen(str);
    da(str,sa,len+1,130); //字符的值域
    calheight(str,sa,len);
    for(int i=1;i<=len;i++)
        printf("sa[%d] %d\n",i,sa[i]);
    for(int i=1;i<=len;i++)
        printf("Rank[%d] %d\n",i,Rank[i]);
    for(int i=1;i<=len;i++)
        printf("height[%d] %d\n",i,height[i]);
    return 0;
}
```

1.5.2 DC3

1.6 回文自动机

```
#include <bits/stdc++.h>
//jisuanke 41389

/*
    fail[x]: x节点失配之后跳转到不等于自身的最长后缀回文子
        串
    len[x]: 以x结尾的最长回文子串长度
    diff[x]:  与”以x结尾的最长回文子串“本质不同的子串个
        数
    same[x]: 与”以x结尾的最长回文子串“本质相同的子串个数
    (注意上面两个完全相反)
    son[x][c]: 编号为x的节点表示的回文子串在两边添加字符c
        之后变成的回文子串编号
    s[x]: 第x次添加的字符, s数组即原字符串
    tot: 总节点个数, 节点编号由0到tot-1
    last: 最后一个新建立节点的编号
    cur: 当前节点在PAM上的父亲编号
*/

#define int long long
using namespace std;
const int N=1e6+5;

struct PAM
{
    int tot,last,n,cur;
    int fail[N],len[N],same[N],diff[N],son[N][26];
    char s[N];
    int get(int p,int x)
    {
        while(s[x-len[p]-1]!=s[x])
            p=fail[p];
        return p;
    }
    int newnode(int x)
    {
        len[tot]=x;
```

```
        return tot++;
    }
    void build()
    {
        scanf("%s",s+1);
        s[0]=-1,fail[0]=1,last=0;
        newnode(0),newnode(-1);
        for(n=1;s[n];++n)
        {
            s[n]-='a';
            cur=get(last,n);
            if(!son[cur][s[n]])
            {
                int now=newnode(len[cur]+2);
                fail[now]=son[get(fail[cur],n)][s[n]];
                diff[now]=diff[fail[diff[now]]]+1;
                son[cur][s[n]]=now;
            }
            same[last=son[cur][s[n]]]++;
        }
        for(int i=tot-1;i>=0;--i)
            same[fail[i]]+=same[i];
    }
}pam;

int v[26],ans=0;
void dfs(int x,int now)
{
    if(pam.len[x]>0) ans+=pam.same[x]*now;
    for(int i=0;i<26;++i)
    {
        if(pam.son[x][i]!=0)
        {
            if(!v[i])
            {
                v[i]=1;
                dfs(pam.son[x][i],now+1);
                v[i]=0;
            }
            else dfs(pam.son[x][i],now);
        }
    }
}

signed main()
{
    pam.build();
    dfs(0,0);//even string
    dfs(1,0);//odd string
    printf("%lld",ans);
    return 0;
}
```

1.7 AC 自动机

1.7.1 多模匹配

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

/*
    trie静态开点+trie图优化
*/

int sz,hd=1,nxt[1000005][26],fail[1000005],id
    [1000005],n;
char s[1000005];

void trie_clean()
{
    sz=1;
    memset(nxt,0,sizeof(nxt));
    memset(fail,0,sizeof(fail));
    memset(id,0,sizeof(id));
}

void trie_insert(int head,char s[],int len,int idx)
{
    int p=head;
    for(int i=0;i<len;i++)
    {
        int c=s[i]-'a';
        if(!nxt[p][c]) nxt[p][c]=++sz;
        p=nxt[p][c];
    }
    id[p]+=idx;
}

void acatm_build(int head)
{
    int p,tp;
    queue<int> q;
    q.push(head);
    fail[head]=0;
    while(!q.empty())
    {
        p=q.front();
        q.pop();
        for(int i=0;i<26;i++)
            if(nxt[p][i])
            {
                fail[nxt[p][i]]=p==head?head:nxt[fail[p
                    ]][i];
                q.push(nxt[p][i]);
            }
            else
                nxt[p][i]=p==head?head:nxt[fail[p]][i];
    }
}

int acatm_match(int head,char s[],int len)
{
    int p=head,ret=0;
    for(int i=0;i<len;i++)
    {
        int c=(int)s[i]-'a';
        p=nxt[p][c];
        for(int tp=p;tp;tp=fail[tp])
            if(id[tp]) ret++;
    }
    return ret;
}
```

1.7.2 自动机上 DP

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

```

```
/*
    每个串有个权值
    求一个长度为n的串使得每个串的权值乘以出现次数之和最大
*/

int fail[2005],nxt[2005][26],cnt[2005],sz,hd,n,m,dp[55][2005],from[55][2005];
char s[105][15];
string dps[55][2005];

void clear()
{
    sz=hd=1;
    memset(dp,0xc0,sizeof(dp));
    memset(fail,0,sizeof(fail));
    memset(nxt,0,sizeof(nxt));
    memset(cnt,0,sizeof(cnt));
}

void trie_insert(int head,char s[],int len,int idx)
{
    int p=head;
    for(int i=0;i<len;i++)
    {
        int c=s[i]-'a';
        if(!nxt[p][c]) nxt[p][c]=++sz;
        p=nxt[p][c];
    }
    cnt[p]+=idx;
}

void acatm_build(int head)
{
    queue<int> q;
    q.push(head);
    while(!q.empty())
    {
        int p=q.front();
        q.pop();
        for(int i=0;i<26;i++)
            if(nxt[p][i])
            {
                fail[nxt[p][i]]=p==head?head:nxt[fail[p]][i];
                cnt[nxt[p][i]]+=cnt[fail[nxt[p][i]]];
                q.push(nxt[p][i]);
            }
            else
                nxt[p][i]=p==head?head:nxt[fail[p]][i];
    }
}

bool scmp(string a,string b)
{
    if(a.length()==b.length()) return a<b;
    else return a.length()<b.length();
}

void solve()
{
    clear();
    scanf("%d%d",&n,&m);
    for(int i=0;i<m;i++)
        scanf("%s",s[i]);
    for(int i=0;i<m;i++)
    {
        int x;
        scanf("%d",&x);
        trie_insert(hd,s[i],strlen(s[i]),x);
    }
    acatm_build(hd);

    for(int i=0;i<=n;i++)
        for(int j=0;j<=sz;j++)
            dps[i][j]=string("");
    int ans=0;
    string anss;
    queue<pair<int,int> > q;
    dp[0][1]=0;
    for(int i=0;i<n;i++)
        for(int j=1;j<=sz;j++)
            for(int k=0;k<26;k++)
                if(dp[i][j]+cnt[nxt[j][k]]>dp[i+1][nxt[j][k]]
                ||dp[i][j]+cnt[nxt[j][k]]==dp[i+1][nxt[j][k]]&&scmp(dps[i][j]+char('a'+k),dps[i+1][nxt[j][k]]))
                {
                    dps[i+1][nxt[j][k]]=dps[i][j]+char('a'+k);
                    dp[i+1][nxt[j][k]]=dp[i][j]+cnt[nxt[j][k]];
                }
    for(int i=0;i<=n;i++)
        for(int j=1;j<=sz;j++)
            if(dp[i][j]>ans||dp[i][j]==ans&&scmp(dps[i][j],anss))
            {
                ans=dp[i][j];
                anss=dps[i][j];
            }
    for(int i=0;i<anss.length();i++)
        printf("%c",anss[i]);
    printf("\n");
}

int main()
{
    int _;
    scanf("%d",&_);
    while(_--) solve();
    return 0;
}
```

1.8 后缀自动机

2 计算几何

2.1 二维几何

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

#define db double
const db EPS=1e-9;
inline int sign(db a){return a<-EPS?-1:a>EPS;}
inline int cmp(db a,db b){return sign(a-b);}
struct P
{
```

```
    db x,y;
    P(){}
    P(db x,db y):x(x),y(y){}
    P operator+(P p){return {x+p.x,y+p.y};}
    P operator-(P p){return {x-p.x,y-p.y};}
    P operator*(db d){return {x*d,y*d};}
    P operator/(db d){return {x/d,y/d};}
    bool operator<(P p) const
    {
        int c=cmp(x,p.x);
        if(c) return c==-1;
        return cmp(y,p.y)==-1;
    }
    bool operator==(P o) const
    {
        return cmp(x,o.x)==0&&cmp(y,o.y)==0;
    }
    db distTo(P p){return (*this-p).abs();}
    db alpha(){return atan2(y,x);}
    void read(){scanf("%lf%lf",&x,&y);}
    void write(){printf("(%.10f,%.10f)\n",x,y);}
    db abs(){return sqrt(abs2());}
    db abs2(){return x*x+y*y;}
    P rot90(){return P(-y,x);}
    P unit(){return *this/abs();}
    int quad() const {return sign(y)==1||(sign(y)==0&&
        sign(x)>=0);}
    db dot(P p){return x*p.x+y*p.y;}
    db det(P p){return x*p.y-y*p.x;}
    P rot(db an){return {x*cos(an)-y*sin(an),x*sin(an)
        +y*cos(an)};}
};

//For segment
#define cross(p1,p2,p3) ((p2.x-p1.x)*(p3.y-p1.y)-(p3.
    x-p1.x)*(p2.y-p1.y))
#define crossOp(p1,p2,p3) sign(cross(p1,p2,p3))

bool chkLL(P p1,P p2,P q1,P q2) //0:parallel
{
    db a1=cross(q1,q2,p1),a2=-cross(q1,q2,p2);
    return sign(a1+a2)!=0;
}

P isLL(P p1,P p2,P q1,P q2) //crossover point if
    chkLL()
{
    db a1=cross(q1,q2,p1),a2=-cross(q1,q2,p2);
    return (p1*a2+p2*a1)/(a1+a2);
}

bool intersect(db l1,db r1,db l2,db r2)
{
    if(l1>r1) swap(l1,r1);if(l2>r2) swap(l2,r2);
    return !(cmp(r1,l2)==-1||cmp(r2,l1)==-1);
}

bool isSS(P p1,P p2,P q1,P q2)
{
    return intersect(p1.x,p2.x,q1.x,q2.x)&&intersect(
        p1.y,p2.y,q1.y,q2.y)&&
    crossOp(p1,p2,q1)*crossOp(p1,p2,q2)<=0&&crossOp(q1
        ,q2,p1)*crossOp(q1,q2,p2)<=0;
}

bool isSS_strict(P p1,P p2,P q1,P q2)
{
    return crossOp(p1,p2,q1)*crossOp(p1,p2,q2)<0
    &&crossOp(q1,q2,p1)*crossOp(q1,q2,p2)<0;
}

bool isMiddle(db a,db m,db b)
{
    return sign(a-m)==0||sign(b-m)==0||(a<m!=b<m);
}

bool isMiddle(P a,P m,P b)
{
    return isMiddle(a.x,m.x,b.x)&&isMiddle(a.y,m.y,b.y
        );
}

bool onSeg(P p1,P p2,P q)
{
    return crossOp(p1,p2,q)==0&&isMiddle(p1,q,p2);
}

bool onSeg_strict(P p1,P p2,P q)
{
    return crossOp(p1,p2,q)==0&&sign((q-p1).dot(p1-p2)
        )*sign((q-p2).dot(p1-p2))<0;
}

P proj(P p1,P p2,P q)
{
    P dir=p2-p1;
    return p1+dir*(dir.dot(q-p1)/dir.abs2());
}

P reflect(P p1,P p2,P q)
{
    return proj(p1,p2,q)*2-q;
}

db nearest(P p1,P p2,P q)
{
    P h=proj(p1,p2,q);
    if(isMiddle(p1,h,p2))
        return q.distTo(h);
    return min(p1.distTo(q),p2.distTo(q));
}

db disSS(P p1,P p2,P q1,P q2) //dist of 2 segments
{
    if(isSS(p1,p2,q1,q2)) return 0;
    return min(min(nearest(p1,p2,q1),nearest(p1,p2,q2)
        ),min(nearest(q1,q2,p1),nearest(q1,q2,p2)));
}

db rad(P p1,P p2)
{
    return atan2l(p1.det(p2),p1.dot(p2));
}

db area(vector<P> ps)
{
    db ret=0;
    for(int i=0;i<ps.size();i++)
        ret+=ps[i].det(ps[(i+1)%ps.size()]);
    return ret/2;
```

```
}

int contain(vector<P> ps,P p) //2:inside,1:on_seg,0:outside
{
    int n=ps.size(),ret=0;
    for(int i=0;i<n;i++)
    {
        P u=ps[i],v=ps[(i+1)%n];
        if(onSeg(u,v,p)) return 1;
        if(cmp(u.y,v.y)<=0) swap(u,v);
        if(cmp(p.y,u.y)>0||cmp(p.y,v.y)<=0) continue;
        ret^=crossOp(p,u,v)>0;
    }
    return ret*2;
}

vector<P> convexHull(vector<P> ps)
{
    int n=ps.size();if(n<=1) return ps;
    sort(ps.begin(),ps.end());
    vector<P> qs(n*2);int k=0;
    for(int i=0;i<n;qs[k++]=ps[i++])
        while(k>1&&crossOp(qs[k-2],qs[k-1],ps[i])<=0)
            --k;
    for(int i=n-2,t=k;i>=0;qs[k++]=ps[i--])
        while(k>t&&crossOp(qs[k-2],qs[k-1],ps[i])<=0)
            --k;
    qs.resize(k-1);
    return qs;
}

db convexDiameter(vector<P> ps)
{
    int n=ps.size();if(n<=1) return 0;
    int is=0,js=0;
    for(int k=1;k<n;k++) is=ps[k]<ps[is]?k:is,js=ps[js]<ps[k]?js:k;
    int i=is,j=js;
    db ret=ps[i].distTo(ps[j]);
    do{
        if((ps[(i+1)%n]-ps[i]).det(ps[(j+1)%n]-ps[j])>=0) (++j)%=n;
        else (++i)%=n;
        ret=max(ret,ps[i].distTo(ps[j]));
    }while(i!=is||j!=js);
    return ret;
}
```

2.2 三维几何

3 图论

3.1 最短路

3.1.1 Dijkstra

```
#include <bits/stdc++.h>
#define mkp(a,b) make_pair(a,b)
#define fst first
#define snd second
//luogu P4779
using namespace std;
typedef pair<int,int> pii;
const int inf=0x3f3f3f3f;
const int N=1000005;

struct edge
{
    int y,v;
    edge(int Y,int V):y(Y),v(V){}
};
vector<edge> e[N];
void add(int x,int y,int v)
{
    e[x].push_back(edge(y,v));
}

int n,m,s;
int dis[N];
bool vis[N];

void dij(int s)
{
    memset(dis,0x3f,sizeof(dis));
    dis[s]=0;
    priority_queue<pii,vector<pii>,greater<pii>> q;
    q.push(mkp(0,s));
    while(!q.empty())
    {
        int x=q.top().snd;
        q.pop();
        if(vis[x]) continue;
        vis[x]=1;
        for(auto y:e[x])
        {
            if(dis[x]+y.v<dis[y.y])
            {
                dis[y.y]=dis[x]+y.v;
                q.push(mkp(dis[y.y],y.y));
            }
        }
    }
}

int main()
{
    scanf("%d%d%d",&n,&m,&s);
    for(int i=1,x,y,z;i<=m;++i)
    {
        scanf("%d%d%d",&x,&y,&z);
        add(x,y,z);
    }
    dij(s);
    for(int i=1;i<=n;++i)
        printf("%d ",dis[i]==inf?2147483647:dis[i]);
    return 0;
}
```

3.1.2 SPFA

3.1.3 Floyd

3.1.4 负环

```
#include <bits/stdc++.h>
//luogu P3385
using namespace std;
const int N=2005;
```

```

int _,n,m,dis[N],cnt[N];
bool vis[N];
vector<int> e[N],v[N];
void add(int x,int y,int z)
{
    e[x].push_back(y);
    v[x].push_back(z);
}

bool spfa(int s)
{
    queue<int> q;
    memset(dis,0x3f,sizeof(dis));
    memset(vis,0,sizeof(vis));
    memset(cnt,0,sizeof(cnt));
    dis[s]=0;
    vis[s]=cnt[s]=1;
    q.push(1);
    while(!q.empty())
    {
        int f=q.front();
        q.pop();
        vis[f]=0;
        for(int i=0;i<e[f].size();++i)
        {
            int y=e[f][i];
            if(dis[y]>dis[f]+v[f][i])
            {
                dis[y]=dis[f]+v[f][i];
                if(!vis[y])
                {
                    vis[y]=1;
                    q.push(y);
                    cnt[y]++;
                    if(cnt[y]>n) return 1;
                }
            }
        }
    }
    return 0;
}

int main()
{
    scanf("%d",&_);
    while(_--)
    {
        scanf("%d%d",&n,&m);
        for(int i=1;i<=n;++i)
            e[i].clear(),v[i].clear();
        for(int i=1,x,y,z;i<=m;++i)
        {
            scanf("%d%d%d",&x,&y,&z);
            if(z<0) add(x,y,z);
            else add(x,y,z),add(y,x,z);
        }
        if(spfa(1)) puts("YE5");
        else puts("N0");
    }
    return 0;
}
```

3.1.5 差分约束

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
//Gym 102411 E

/*
    差分约束找最小解
    有负环代表无解
    ai-aj<=k表示为j到i的长度为k的边（最大解）
    最小解就交换边的方向
    都跑最短路
*/

int n;
ll l,r,k,a[100005];
vector<int> E[100005];
vector<ll> V[100005];

void add(int x,int y,ll z)
{
    E[x].push_back(y);
    V[x].push_back(z);
}

ll dis[100005];
bool vis[100005];
int cnt[100005];
bool spfa()
{
    for(int i=1;i<=n;++i) dis[i]=0x3f3f3f3f3f3f3f3f;
    dis[0]=0;
    vis[0]=1;
    cnt[0]=0;
    queue<int> q;
    q.push(0);
    while(!q.empty())
    {
        int f=q.front();
        q.pop();
        vis[f]=0;
        for(int i=0;i<E[f].size();++i)
        {
            int e=E[f][i];
            int v=V[f][i];
            if(dis[e]>dis[f]+v)
            {
                dis[e]=dis[f]+v;
                cnt[e]=cnt[f]+1;
                if(cnt[e]>n) return 1;
                if(!vis[e])
                {
                    vis[e]=1;
                    q.push(e);
                }
            }
        }
    }
    return 0;
}

int main()
{
    scanf("%d%lld%lld%lld",&n,&l,&r,&k);
    for(int i=1;i<=n;++i) scanf("%lld",&a[i]);
    for(int i=1;i<n;++i)
```

```
    {
        if(a[i]>a[i+1])
        {
            add(i,i+1,k);
            add(i+1,i,-1);
        }
        else if(a[i]<a[i+1])
        {
            add(i+1,i,k);
            add(i,i+1,-1);
        }
        else if(a[i]==a[i+1])
        {
            add(i+1,i,0ll);
            add(i,i+1,0ll);
        }
    }
    for(int i=1;i<=n;++i) add(i,0,r),add(0,i,-l);//
    if(spfa()) return puts("-1"),0;
    for(int i=1;i<=n;++i) printf("%lld ",-dis[i]);
    return 0;
}
```

3.2 最小生成树

3.2.1 Prim

```
#include <bits/stdc++.h>
//luogu P3366
using namespace std;

/*
    prim/kruskal一定要注意解决重边
*/

const int N=5005;
const int inf=0x3f3f3f3f;

int n,m;
int mp[N][N];
int dis[N];

int prim(int s)
{
    int sum=0;
    int cnt=0;
    for(int i=1;i<=n;++i)
        dis[i]=mp[s][i];
    cnt++;
    while(1)
    {
        int mn=inf;
        int now=-1;
        for(int i=1;i<=n;++i)
        {
            if(dis[i]!=0&&dis[i]<mn)
            {
                mn=dis[i];
                now=i;
            }
        }
        if(now==-1) break;
        sum+=dis[now];
        dis[now]=0;
        cnt++;
        for(int i=1;i<=n;++i)
        {
            if(dis[i]!=0&&mp[now][i]<dis[i])
                dis[i]=mp[now][i];
        }
    }
    if(cnt<n) return -1;
    else return sum;
}

int main()
{
    scanf("%d%d",&n,&m);
    memset(mp,0x3f,sizeof(mp));
    for(int i=1;i<=n;++i)
        mp[i][i]=0;
    for(int i=1,x,y,z;i<=m;++i)
    {
        scanf("%d%d%d",&x,&y,&z);
        mp[x][y]=min(mp[x][y],z);
        mp[y][x]=min(mp[y][x],z);
    }
    int ans=prim(1);
    if(ans==-1) puts("orz");
    else printf("%d",ans);
    return 0;
}
```

3.2.2 Kruskal

```
#include <bits/stdc++.h>
//luogu P3366
using namespace std;

/*
    prim/kruskal一定要注意解决重边
*/

const int N=200005;

int n,m;
struct node
{
    int x,y,z;
}o[N];

bool cmp(node a,node b)
{
    return a.z<b.z;
}

int f[5005];
int _find(int x)
{
    if(x!=f[x]) f[x]=_find(f[x]);
    return f[x];
}
void _merge(int x,int y)
{
    x=_find(x),y=_find(y);
    if(x!=y) f[x]=y;
}

int kk()
```

```
{
    for(int i=1;i<=n;++i)
        f[i]=i;
    sort(o+1,o+1+m,cmp);
    int sum=0;
    for(int i=1;i<=m;++i)
    {
        if(_find(o[i].x)!=_find(o[i].y))
        {
            sum+=o[i].z;
            _merge(o[i].x,o[i].y);
        }
    }
    int tmp=_find(1);
    for(int i=2;i<=n;++i)
        if(_find(i)!=tmp)
            return -1;
    return sum;
}

int main()
{
    scanf("%d%d",&n,&m);
    for(int i=1;i<=m;++i)
        scanf("%d%d%d",&o[i].x,&o[i].y,&o[i].z);
    int ans=kk();
    if(ans==-1) puts("orz");
    else printf("%d",ans);
    return 0;
}
```

3.2.3 最小生成树计数

3.2.4 次小生成树

```
#include <bits/stdc++.h>
#define int long long
using namespace std;
const int N=5e5+5;
const int M=1e6+5;
int inf=1e18;
//luogu P1480

/*
    求解严格小于的次小生成树
*/

struct edge
{
    int to,val;
};
vector<edge> e[N];

struct node
{
    int x,y,v;
}eg[M];

void add(int a,int b,int c)
{
    e[a].push_back({b,c});
    e[b].push_back({a,c});
}

int fa[N][35],dep[N],mx[N][35],mn[N][35];
void dfs(int x,int f)
{
    fa[x][0]=f;
    for(auto i:e[x])
    {
        if(i.to==f) continue;
        dep[i.to]=dep[x]+1;
        mx[i.to][0]=i.val;
        mn[i.to][0]=-inf;
        dfs(i.to,x);
    }
}

int n,m;
void cal()
{
    for(int i=1;i<=30;++i)
    {
        for(int j=1;j<=n;++j)
        {
            fa[j][i]=fa[fa[j][i-1]][i-1];
            mx[j][i]=max(mx[j][i-1],mx[fa[j][i-1]][i
                -1]);
            mn[j][i]=max(mn[j][i-1],mn[fa[j][i-1]][i
                -1]);
            if(mx[j][i-1]>mx[fa[j][i-1]][i-1])
                mn[j][i]=max(mn[j][i],mx[fa[j][i-1]][i
                    -1]);
            else if(mx[j][i-1]<mx[fa[j][i-1]][i-1])
                mn[j][i]=max(mn[j][i],mx[j][i-1]);
        }
    }
}

int lca(int x,int y)
{
    if(dep[x]<dep[y])
        swap(x,y);
    for(int i=30;i>=0;--i)
        if(dep[fa[x][i]]>=dep[y])
            x=fa[x][i];
    if(x==y) return x;
    for(int i=30;i>=0;--i)
        if(fa[x][i]^fa[y][i])
            x=fa[x][i],
            y=fa[y][i];
    return fa[x][0];
}

int qmax(int u,int v,int val)
{
    int ans=-inf;
    for(int i=30;i>=0;--i)
    {
        if(dep[fa[u][i]]>=dep[v])
        {
            if(val!=mx[u][i])
                ans=max(ans,mx[u][i]);
            else
                ans=max(ans,mn[u][i]);
            u=fa[u][i];
        }
    }
    return ans;
}
```

```

bool cmp(node a,node b)
{
    return a.v<b.v;
}

int F[N];
int _find(int x)
{
    if(x!=F[x]) F[x]=_find(F[x]);
    return F[x];
}

bool flg[M];
signed main()
{
    scanf("%lld%lld",&n,&m);
    for(int i=1;i<=m;++i)
    {
        scanf("%lld%lld%lld",&eg[i].x,&eg[i].y,&eg[i].
            v);
    }
    sort(eg+1,eg+1+m,cmp);
    for(int i=1;i<=n;++i)
    {
        F[i]=i;
    }
    int cnt=0;
    for(int i=1;i<=m;++i)
    {
        int fx=_find(eg[i].x);
        int fy=_find(eg[i].y);
        if(fx!=fy)
        {
            cnt+=eg[i].v;
            F[fx]=fy;
            add(eg[i].x,eg[i].y,eg[i].v);
            flg[i]=1;
        }
    }
    mn[1][0]=-inf;
    dep[1]=1;
    dfs(1,-1);
    cal();
    int ans=inf;
    for(int i=1;i<=m;++i)
    {
        if(flg[i]) continue;
        int x=eg[i].x;
        int y=eg[i].y;
        int v=eg[i].v;
        int l=lca(x,y);
        int mxu=qmax(x,l,v);
        int mxv=qmax(y,l,v);
        ans=min(ans,cnt-max(mxu,mxv)+v);
    }
    printf("%lld",ans);
    return 0;
}
```

3.2.5 最小乘积生成树

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
//luogu P5540

/*
    每条边有两个边权
    使得一棵生成树sigma(ai)*sigma(bi)最小
*/

const int N=205;
const int M=10005;
const int inf=0x3f3f3f3f;

struct UFS
{
    int f[N];
    void init(int n)
    {
        for(int i=1;i<=n;++i)
            f[i]=i;
    }
    int find(int x)
    {
        if(x!=f[x]) f[x]=find(f[x]);
        return f[x];
    }
    void merge(int x,int y)
    {
        x=find(x),y=find(y);
        if(x!=y)
            f[x]=y;
    }
    bool check(int x,int y)
    {
        return find(x)==find(y);
    }
};

struct edge
{
    int u,v,w,a,b;
};

bool cmp(edge a,edge b)
{
    return a.w<b.w;
}

struct point
{
    int x,y;
};
point operator - (point a,point b)
{
    return (point){a.x-b.x,a.y-b.y};
}
int cross(point a,point b)
{
    return a.x*b.y-a.y*b.x;
}

int n,m;
UFS s;
point ans=point{inf,inf};
edge e[M];

```

```
point kk()
{
    point res=point{0,0};
    int tot=0;
    sort(e+1,e+1+m,cmp);
    s.init(n);
    for(int i=1;i<=m;++i)
    {
        int u=e[i].u,v=e[i].v,a=e[i].a,b=e[i].b;
        if(s.check(u,v)) continue;
        s.merge(u,v);
        res.x+=a;
        res.y+=b;
        ++tot;
        if(tot==n-1) break;
    }
    ll tmp=1ll*ans.x*ans.y;
    ll now=1ll*res.x*res.y;
    if(tmp>now||(tmp==now&&ans.x>res.x))
        ans=res;
    return res;
}

void solve(point a,point b)
{
    for(int i=1;i<=m;++i)
    {
        e[i].w=e[i].b*(b.x-a.x)+e[i].a*(a.y-b.y);
    }
    point c=kk();
    if(cross(b-a,c-a)>=0) return;
    solve(a,c);
    solve(c,b);
}

int main()
{
    scanf("%d%d",&n,&m);
    for(int i=1;i<=m;++i)
    {
        scanf("%d%d%d%d",&e[i].u,&e[i].v,&e[i].a,&e[i
            ].b);
        e[i].u++;
        e[i].v++;
    }
    for(int i=1;i<=m;++i)
    {
        e[i].w=e[i].a;
    }
    point a=kk();
    for(int i=1;i<=m;++i)
    {
        e[i].w=e[i].b;
    }
    point b=kk();
    solve(a,b);
    printf("%d %d",ans.x,ans.y);
    return 0;
}
```

3.3 树的直径

3.4 LCA

3.4.1 Tarjan 离线

```
#include <bits/stdc++.h>
//luogu P3379
using namespace std;

/*
    tarjan求lca要注意时间复杂度可能会爆炸，模板题开了O2才
        过，并且要小心数组越界
*/

const int N=1000005;

int n,m,s,x,y;
vector<int> e[N],q[N],id[N];
int ans[N*2];
bool vis[N];
int f[N];
int _find(int x)
{
    if(x!=f[x]) f[x]=_find(f[x]);
    return f[x];
}
void _merge(int x,int y)
{
    x=_find(x),y=_find(y);
    if(x!=y) f[x]=y;
}

void tarjan(int u)
{
    vis[u]=1;
    for(auto v:e[u])
    {
        if(!vis[v])
        {
            tarjan(v);
            _merge(v,u);
        }
    }
    for(int i=0;i<q[u].size();++i)
    {
        int v=q[u][i];
        int k=id[u][i];
        if(vis[v]&&ans[k]==0)
            ans[k]=_find(v);
    }
}

int main()
{
    scanf("%d%d%d",&n,&m,&s);
    for(int i=1;i<=n;++i) f[i]=i;
    for(int i=1;i<=n-1;++i)
        scanf("%d%d",&x,&y),e[x].push_back(y),e[y].
            push_back(x);
    for(int i=1;i<=m;++i)
        scanf("%d%d",&x,&y),
        q[x].push_back(y),q[y].push_back(x),
        id[x].push_back(i),id[y].push_back(i);
    tarjan(s);
    for(int i=1;i<=m;++i)
```

```
        printf("%d\n",ans[i]);
    return 0;
}
```

3.4.2 倍增 LCA

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

/*
    预处理 O(nlogn)
    单次查询 O(logn)
*/

const int MAXN=500005;
int n,q,dep[MAXN],s,lg[MAXN],fa[MAXN][32];
vector<int> e[MAXN];

void dfs(int now,int pa)
{
    dep[now]=dep[pa]+1;
    fa[now][0]=pa;
    for(int i=1;(1<<i)<=dep[now];i++)
        fa[now][i]=fa[fa[now][i-1]][i-1];
    for(auto to:e[now])
        if(to!=pa) dfs(to,now);
}

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

int main()
{
    for(int i=1;i<MAXN;i++)
        lg[i]=lg[i-1]+(1<<lg[i-1]==i);
    scanf("%d%d%d",&n,&q,&s);
    for(int i=0,x,y;i<n-1;i++)
    {
        scanf("%d%d",&x,&y);
        e[x].push_back(y),e[y].push_back(x);
    }
    dep[0]=0;
    dfs(s,0);
    for(int i=0,x,y;i<q;i++)
    {
        scanf("%d%d",&x,&y);
        printf("%d\n",lca(x,y));
    }
    return 0;
}
```

3.5 无向图与有向图联通性

3.5.1 割点

```
#include <bits/stdc++.h>
#define int long long
//luogu P3469

/*
    tarjan求割点的算法中，如果不保证连通性，应该使用被注释掉的遍历方法
    part数组储存了被这个割点分成的不同的几块各自的大小
*/

using namespace std;
const int N=100005;

int n,m,x,y;
vector<int> e[N],part[N];
bool is[N];
int dfn[N],low[N],timer=0;
int sz[N];

void tarjan(int u,int f)
{
    dfn[u]=low[u]=++timer;
    sz[u]++;//
    int son=0,tmp=0;
    for(auto v:e[u])
    {
        if(dfn[v]==0)
        {
            tarjan(v,u);
            sz[u]+=sz[v];//
            low[u]=min(low[u],low[v]);
            if(low[v]>=dfn[u]&&u!=f)
            {
                is[u]=1;
                tmp+=sz[v];//
                part[u].push_back(sz[v]);//
            }
            if(u==f) son++;
        }
        low[u]=min(low[u],dfn[v]);
    }
    if(son>=2&&u==f) is[u]=1;//point on the top
    if(is[u]&&n-tmp-1!=0)
        part[u].push_back(n-tmp-1);//
}

signed main()
{
    scanf("%lld%lld",&n,&m);
    for(int i=1;i<=m;++i)
    {
        scanf("%lld%lld",&x,&y);
        e[x].push_back(y),e[y].push_back(x);
    }
    /*
    for(int i=1;i<=n;++i)
        if(!dfn[i]) tarjan(i,i);
    */
    tarjan(1,0);
    for(int i=1;i<=n;++i)
    {
        if(!is[i]) printf("%lld\n",2*(n-1));
        else{
            int tmp=0;
            for(auto j:part[i])
```

```
            tmp+=j*(j-1);
        printf("%lld\n",n*(n-1)-tmp);
    }
}
    return 0;
}
```

3.5.2 桥

```
#include <bits/stdc++.h>
#define mkp make_pair
//uva796
using namespace std;
const int N=1000005;
typedef pair<int,int> pii;
inline int read(){
    char ch=getchar();int s=0,w=1;
    while(ch<48||ch>57){if(ch=='-')w=-1;ch=getchar();}
    while(ch>=48&&ch<=57){s=(s<<1)+(s<<3)+ch-48;ch=
        getchar();}
    return s*w;
}
inline void write(int x){
    if(x<0)putchar('-'),x=-x;
    if(x>9)write(x/10);
    putchar(x%10+48);
}

int n;
int dfn[N],low[N],timer=0;
int fa[N];
vector<int> e[N];
vector<pii> ans;

void tarjan(int u,int f)
{
    fa[u]=f;
    dfn[u]=low[u]=++timer;
    for(auto v:e[u])
    {
        if(!dfn[v])
        {
            tarjan(v,u);
            low[u]=min(low[u],low[v]);
            //if(dfn[u]<low[v]) is[u][v]=1;
            //u is v's father
        }
        else if(v!=f) low[u]=min(low[u],dfn[v]);
    }
}

void init()
{
    timer=0;
    for(int i=0;i<n;++i) dfn[i]=low[i]=fa[i]=0;
    for(int i=0;i<n;++i) e[i].clear();
    ans.clear();
}

void gao()
{
    for(int i=0;i<n;++i)
        if(!dfn[i]) tarjan(i,-1);
    for(int i=0;i<n;++i)
    {
        int F=fa[i];
        if(F!=-1&&dfn[F]<low[i])
            ans.emplace_back(min(F,i),max(F,i));
    }
    sort(ans.begin(),ans.end());
    printf("%d critical links\n",(int)ans.size());
    for(auto i:ans)
        printf("%d - %d\n",i.first,i.second);
    puts("");
}

int main()
{
    while(~scanf("%d",&n))
    {
        if(n==0)
        {
            puts("0 critical links");
            puts("");
            continue;
        }
        init();
        for(int i=0,x,y,z;i<n;++i)
        {
            scanf("%d (%d)",&x,&y);
            for(int i=0;i<y;++i)
                z=read(),
                e[x].push_back(z),
                e[z].push_back(x);
        }
        gao();
    }
    return 0;
}
```

3.5.3 e-DCC

3.5.4 v-DCC

3.5.5 SCC

```
#include <bits/stdc++.h>
//luogu P2341
using namespace std;

/*
    scc表示某标号的强连通分量中的点，co表示某个点属于哪个
        强连通分量
    gao函数是重建图，按照题意寻找有没有链
*/

const int N=10005;

int n,m,x[N*5],y[N*5];
vector<int> e[N],scc[N];
int co[N],color=0;
stack<int> s;
bool vis[N];
int dfn[N],low[N],timer=0;

void tarjan(int u)
{
    dfn[u]=low[u]=++timer;
    s.push(u);
```

```
    vis[u]=1;
    for(auto v:e[u])
    {
        if(!dfn[v])
        {
            tarjan(v);
            low[u]=min(low[u],low[v]);
        }
        else if(vis[v]) low[u]=min(low[u],dfn[v]);
    }
    if(low[u]==dfn[u])
    {
        ++color;
        int t;
        do
        {
            t=s.top();
            s.pop();
            co[t]=color;
            vis[t]=0;
            scc[color].push_back(t);
        }
        while(u!=t);
    }
}

int f[N];
int _find(int x)
{
    if(x!=f[x]) f[x]=_find(f[x]);
    return f[x];
}
void _merge(int x,int y)
{
    x=_find(x),y=_find(y);
    if(x!=y) f[x]=y;
}

int d[N];
void gao()
{
    for(int i=1;i<=color;++i)
        f[i]=i;
    for(int i=1;i<=m;++i)
    {
        if(co[x[i]]!=co[y[i]])
            _merge(co[x[i]],co[y[i]]),
            d[co[x[i]]]++;
    }
    int F=_find(1);
    for(int i=1;i<=color;++i)
        if(_find(i)!=F) {puts("0");return;}
    int ans=0,tmp=0;
    for(int i=1;i<=color;++i)
    {
        if(d[i]==0)
            ans+=scc[i].size(),tmp++;
    }
    if(tmp>1) ans=0;
    printf("%d",ans);
}

int main()
{
    scanf("%d%d",&n,&m);
    for(int i=1;i<=m;++i)
    {
        scanf("%d%d",&x[i],&y[i]);
        e[x[i]].push_back(y[i]);
    }
    for(int i=1;i<=n;++i)
        if(!dfn[i]) tarjan(i);
    gao();
    return 0;
}
```

3.5.6 2-SAT

```
#include <bits/stdc++.h>
using namespace std;
//luogu P4782

/*
    2-SAT用于求解有n个布尔变量x1-xn和m个需要满足的条件
    每个条件形式为xi=0(1)||xj=0(1)，是否有可行解
    注意要开两倍空间建反向边
    2-SAT输出方案的方法为：先把缩点之后的新图进行拓扑排
        序，然后判断每个点i，如果i所在强连通分量的拓扑序在
        i'所在的强连通分量的拓扑序之后，那么第i场游戏使用
        该地图适合的第一种赛车，否则使用第二种赛车。
    但是由于Tarjan求强连通分量就是按拓扑排序的逆序给出的，
        所以直接使用强连通分量编号判断即可。
    即如果bel[]为每个点的所在强连通分量编号，那么判断为：
        如果bel[i]<bel[i']，那么使用该地图适合的第一种赛
        车，否则使用第二种赛车。
*/

const int N=2e6+5;

int n,m,a,va,b,vb;
int dfn[N],low[N],timer=0;
stack<int> s;
bool vis[N];
vector<int> e[N];
int co[N],color=0;

void add(int x,int y)
{
    e[x].push_back(y);
}

void tarjan(int u)
{
    dfn[u]=low[u]=++timer;
    s.push(u);
    vis[u]=1;
    for(auto v:e[u])
    {
        if(!dfn[v])
            tarjan(v),
            low[u]=min(low[u],low[v]);
        else if(vis[v])
            low[u]=min(low[u],dfn[v]);
    }
    if(low[u]==dfn[u])
    {
        int v;
        color++;
        do
```

```
        {
            v=s.top();
            s.pop();
            vis[v]=0;
            co[v]=color;
        }
        while(u!=v);
    }
}

bool solve()
{
    for(int i=1;i<=2*n;++i)
        if(!dfn[i]) tarjan(i);
    for(int i=1;i<=n;++i)
        if(co[i]==co[i+n])
            return 0;
    return 1;
}

int main()
{
    scanf("%d%d",&n,&m);
    for(int i=1;i<=m;++i)
    {
        scanf("%d%d%d%d",&a,&va,&b,&vb);
        int nota=va^1,notb=vb^1;
        add(a+nota*n,b+vb*n);//not a and b
        add(b+notb*n,a+va*n);//not b and a
    }
    if(solve())
    {
        puts("POSSIBLE");
        for(int i=1;i<=n;++i)
            printf("%d ",co[i]>co[i+n]);
    }
    else puts("IMPOSSIBLE");
    return 0;
}
```

3.5.7 支配树

3.6 二分图

3.6.1 最大匹配-匈牙利

```
#include <bits/stdc++.h>
//luogu P3386
using namespace std;

/*
    hungary每一次遍历必须要清空vis数组
*/

const int N=1005;

vector<int> e[N];
bool vis[N];
int match[N],rematch[N];

bool dfs(int u)
{
    for(auto v:e[u])
    {
        if(!vis[v]){
            vis[v]=1;
            if(match[v]==0||dfs(match[v]))
            {
                match[v]=u;
                rematch[u]=v;
                return 1;
            }
        }
    }
    return 0;
}

int n,m,k;

int main()
{
    scanf("%d%d%d",&n,&m,&k);
    for(int i=1,x,y;i<=k;++i)
    {
        scanf("%d%d",&x,&y);
        if(x>n||y>m) continue;
        e[x].push_back(y);
    }
    int ans=0;
    for(int i=1;i<=n;++i)
    {
        memset(vis,0,sizeof(vis));
        if(dfs(i)) ans++;
    }
    printf("%d",ans);
    return 0;
}
```

3.6.2 带权匹配-KM

```
#include <bits/stdc++.h>
//hdu 2255
using namespace std;

/*
    KM仅用于最大带权匹配一定是最大匹配的情况中
*/

const int N=305;
const int inf=0x3f3f3f3f;

int n,mp[N][N];
int la[N],lb[N],delta;
bool va[N],vb[N];
int match[N];

bool dfs(int x)
{
    va[x]=1;
    for(int y=1;y<=n;++y)
    {
        if(!vb[y]){
            if(la[x]+lb[y]==mp[x][y])
            {
                vb[y]=1;
                if(!match[y]||dfs(match[y]))
                {
                    match[y]=x;
                    return 1;
```

```
            }
        }
        else
            delta=min(delta,la[x]+lb[y]-mp[x][y]);
    }
}
return 0;
}

int km()
{
    for(int i=1;i<=n;++i)
    {
        match[i]=0;
        la[i]=-inf;
        lb[i]=0;
        for(int j=1;j<=n;++j)
        {
            la[i]=max(la[i],mp[i][j]);
        }
    }
    for(int i=1;i<=n;++i)
    {
        while(1)
        {
            memset(va,0,sizeof(va));
            memset(vb,0,sizeof(vb));
            delta=inf;
            if(dfs(i)) break;
            for(int j=1;j<=n;++j)
            {
                if(va[j]) la[j]-=delta;
                if(vb[j]) lb[j]+=delta;
            }
        }
    }
    int ans=0;
    for(int i=1;i<=n;++i)
        ans+=mp[match[i]][i];
    return ans;
}

int main()
{
    while(~scanf("%d",&n))
    {
        memset(mp,-0x3f,sizeof(mp));
        for(int i=1;i<=n;++i)
        {
            for(int j=1;j<=n;++j)
            {
                scanf("%d",&mp[i][j]);
            }
        }
        printf("%d\n",km());
    }
    return 0;
}
```

3.7 网络流

3.7.1 最大流-Dinic

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

/*
    s,t 超级源、超级汇
    cur[] 当前弧优化
    时间复杂度 O(n^2*m)
*/

const int MAXN=10005;
const ll inf=0x3f3f3f3f3f3f3f3fll;
int n,m,s,t,tot,dis[MAXN],cur[MAXN];
struct edge
{
    int to,cap,rev;
    edge(){}
    edge(int to,int cap,int rev):to(to),cap(cap),rev(
        rev){}
};
vector<edge> E[MAXN];

inline void add_edge(int x,int y,int f)
{
    E[x].emplace_back(y,f,E[y].size());
    E[y].emplace_back(x,0,E[x].size()-1);
}

int bfs()
{
    for(int i=1;i<=n;i++) dis[i]=0x3f3f3f3f;
    dis[s]=0;
    queue<int> q;
    q.push(s);
    while(!q.empty())
    {
        int now=q.front();q.pop();
        for(int i=0;i<E[now].size();i++)
        {
            edge &e=E[now][i];
            if(dis[e.to]>dis[now]+1&&e.cap)
            {
                dis[e.to]=dis[now]+1;
                if(e.to==t) return 1;
                q.push(e.to);
            }
        }
    }
    return 0;
}

ll dfs(int now,ll flow)
{
    if(now==t) return flow;
    ll rest=flow,k;
    for(int i=cur[now];i<E[now].size();i++)
    {
        edge &e=E[now][i];
        if(e.cap&&dis[e.to]==dis[now]+1)
        {
            cur[now]=i;
            k=dfs(e.to,min(rest,(long long)e.cap));
            e.cap-=k;
            E[e.to][e.rev].cap+=k;
            rest-=k;
        }
    }
```

```
    return flow-rest;
}

ll dinic()
{
    ll ret=0,delta;
    while(bfs())
    {
        for(int i=1;i<=n;i++) cur[i]=0;
        while(delta=dfs(s,inf)) ret+=delta;
    }
    return ret;
}
```

3.7.2 最小费用最大流-Dij+Dinic

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int,int> pii;

/*
    出锅状态 勿用

    第一遍跑的spfa,然后是加上势函数的dij,玄学
    h[] 势函数
    cur[] 当前弧优化
    msmf 最大流时的最小费用
    s,t 超级源、超级汇
    时间复杂度 O(n^2*m)
*/

const int MAXN=2005;
const int inf=0x3f3f3f3f;
int msmf,s,t,cur[MAXN],dis[MAXN],vis[MAXN],h[MAXN];
struct edge
{
    int to,val,cap,rev;
    edge(){}
    edge(int to,int cap,int val,int rev):to(to),cap(
        cap),val(val),rev(rev){}
};
vector<edge> E[MAXN];

inline void add_edge(int x,int y,int f,int cost)
{
    E[x].emplace_back(y,f,cost,E[y].size());
    E[y].emplace_back(x,0,-cost,E[x].size()-1);
}

int dij()
{
    fill(dis,dis+t+1,inf);
    priority_queue<pii,vector<pii>,greater<pii>> q;
    q.emplace(0,s);dis[s]=0;
    while(!q.empty())
    {
        pii p=q.top();q.pop();
        int now=p.second;
        if(dis[now]<p.first) continue;
        for(int i=0;i<E[now].size();i++)
        {
            edge &e=E[now][i];
            if(e.cap>0&&dis[e.to]>p.first+e.val+h[now]-
                h[e.to])
            {
                dis[e.to]=p.first+e.val+h[now]-h[e.to];
                q.emplace(dis[e.to],e.to);
            }
        }
    }
    return dis[t]!=inf;
}

int dfs(int now,int flow)
{
    if(now==t) return flow;
    int rest=flow,k;
    vis[now]=1;
    for(int i=cur[now];i<E[now].size();i++)
    {
        edge &e=E[now][i];
        if(e.cap&&dis[now]+e.val+h[now]-h[e.to]==dis[e
            .to]&&!vis[e.to])
        {
            cur[now]=i;
            k=dfs(e.to,min(e.cap,rest));
            e.cap-=k;
            E[e.to][e.rev].cap+=k;
            rest-=k;
            msmf+=k*e.val;
        }
    }
    vis[now]=0;
    return flow-rest;
}

int dinic()
{
    int ret=0,delta;
    while(dij())
    {
        for(int i=s;i<=t;i++) cur[i]=0;
        while(delta=dfs(s,inf)) ret+=delta;
        for(int i=s;i<=t;i++) h[i]+=(dis[i]==inf)?0:
            dis[i];
    }
    return ret;
}
```

3.7.3 最小费用最大流-SPFA+Dinic

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

/*
    cur[] 当前弧优化
    msmf 最大流时的最小费用
    s,t 超级源、超级汇
    多组记得清边和msmf
    时间复杂度 O(n^2*m)
*/

const ll inf=0x3f3f3f3f3f3f3f3fll;
ll msmf,dis[5005];
int s,t,n,m,cur[5005],vis[5005];
struct edge
{
```

```
    int to,rev;
    ll cap,cost;
    edge(){}
    edge(int to,ll cap,ll cost,int rev):to(to),cap(cap),cost(cost),rev(rev){}
};
vector<edge> E[5005];

inline void add_edge(int x,int y,ll f,ll c)
{
    E[x].emplace_back(y,f,c,E[y].size());
    E[y].emplace_back(x,0,-c,E[x].size()-1);
}

int spfa()
{
    for(int i=0;i<=n;i++) vis[i]=0,dis[i]=inf; //从编号最小的点到最大的点
    dis[s]=0;
    queue<int> q;
    q.push(s);
    while(!q.empty())
    {
        int p=q.front();q.pop();
        vis[p]=0;
        for(auto e:E[p])
            if(e.cap&&dis[p]+e.cost<dis[e.to])
            {
                dis[e.to]=dis[p]+e.cost;
                if(!vis[e.to])
                    vis[e.to]=1,q.push(e.to);
            }
    }
    return dis[t]!=inf;
}

ll dfs(int now,ll flow)
{
    if(now==t) return flow;
    ll rest=flow,k;
    vis[now]=1;
    for(int i=cur[now];i<E[now].size();i++)
    {
        edge &e=E[now][i];
        if(e.cap&&dis[now]+e.cost==dis[e.to]&&!vis[e.to])
        {
            cur[now]=i;
            k=dfs(e.to,min(rest,e.cap));
            e.cap-=k;
            E[e.to][e.rev].cap+=k;
            msmf+=k*e.cost;
            rest-=k;
        }
    }
    vis[now]=0;
    return flow-rest;
}

ll dinic()
{
    msmf=0;
    ll ret=0,delta;
    while(spfa())
    {
        for(int i=0;i<=n;i++) cur[i]=vis[i]=0; //从编号最小的点到最大的点
        while(delta=dfs(s,inf)) ret+=delta;
    }
    return ret;
}

int main()
{
    scanf("%d%d%d%d",&n,&m,&s,&t);
    for(int i=0;i<m;i++)
    {
        int x,y;ll a,b;
        scanf("%d%d%lld%lld",&x,&y,&a,&b);
        add_edge(x,y,a,b);
    }
    ll mxflow=dinic();
    printf("%lld %lld",mxflow,msmf);
    return 0;
}
```

3.7.4 上下界流

3.8 欧拉路

```
#include <bits/stdc++.h>
//luogu P2731
using namespace std;
const int N=505;

/*
    euler_path一定要找到正确的起点
*/

int n;
int mp[N][N];
stack<int> st;
int deg[N];

void dfs(int x)
{
    for(int i=1;i<=500;++i)
    {
        if(mp[x][i])
        {
            mp[x][i]--;
            mp[i][x]--;
            dfs(i);
        }
    }
    st.push(x);
}

int main()
{
    scanf("%d",&n);
    for(int i=1,x,y;i<=n;++i)
    {
        scanf("%d%d",&x,&y);
        mp[x][y]++;
        mp[y][x]++;
        deg[x]++;
        deg[y]++;
    }
    int s=1;
```

```
    for(int i=1;i<=500;++i)
    {
        if(deg[i]%2==1)
        {
            s=i;
            break;
        }
    }
    dfs(s);
    while(!st.empty())
    {
        printf("%d\n",st.top());
        st.pop();
    }
    return 0;
}
```

3.9 Prufer 序列

4 数据结构

4.1 树状数组

4.2 线段树

4.2.1 带优先级线段树

4.2.2 李超线段树

```
#include <bits/stdc++.h>
#define lli long long int
#define debug cout
using namespace std;
const int maxn=4e4+1e2,maxm=1e5+1e2;
const double eps=1e-8;
//luogu P4097

/*
    1、在平面上加入一条线段。记第 i 条被插入的线段的标号为 i
    2、给定一个数 k,询问与直线 x = k 相交的线段中，交点最靠上的线段的编号。
*/

int l[maxn<<3],r[maxn<<3],lson[maxn<<3],rson[maxn<<3],dat[maxn<<3],cnt;
double k[maxm],b[maxm];

inline int dcmp(double x) {
    return fabs(x) <= eps ? 0 : x < 0 ? -1 : 1;
}
inline void build(int pos,int ll,int rr) {
    l[pos] = ll , r[pos] = rr;
    if( ll == rr ) return;
    const int mid = ( ll + rr ) >> 1;
    build(lson[pos]=++cnt,ll,mid);
    build(rson[pos]=++cnt,mid+1,rr);
}
inline double f(int x,int p) {
    return k[x] * p + b[x];
}
inline bool judge(int x,int y,int p) {
    double fx = f(x,p) , fy = f(y,p);
    return dcmp(fx-fy) ? fx < fy : x > y;
}
inline void update(int pos,int ll,int rr,int nw) {
    if( r[pos] < ll || rr < l[pos] ) return;
    if( ll <= l[pos] && r[pos] <= rr ) {
        if( judge(nw,dat[pos],l[pos]) && judge(nw,dat[pos],r[pos]) ) return;
        if( judge(dat[pos],nw,l[pos]) && judge(dat[pos],nw,r[pos]) ) {
            dat[pos] = nw;
            return;
        }
        const int mid = ( l[pos] + r[pos] ) >> 1;
        if( judge(dat[pos],nw,mid) ) swap(dat[pos],nw);
        if( judge(dat[pos],nw,l[pos]) ) update(lson[pos],l[pos],r[pos],nw);
        else update(rson[pos],l[pos],r[pos],nw);
        return;
    }
    update(lson[pos],ll,rr,nw) , update(rson[pos],ll,rr,nw);
}
inline int query(int pos,int x) {
    if( x < l[pos] || r[pos] < x ) return 0;
    if( l[pos] == r[pos] && l[pos] == x ) return dat[pos];
    const int mid = ( l[pos] + r[pos] ) >> 1;
    int ret = x <= mid ? query(lson[pos],x) : query(rson[pos],x);
    if( judge(ret,dat[pos],x) ) ret = dat[pos];
    return ret;
}

int main() {
    static int n,lastans=-1,x0,y0,x1,y1,p,cc;
    build(cnt=1,1,40000);
    scanf("%d",&n);
    for(int i=1,ope;i<=n;i++) {
        scanf("%d",&ope);
        if( ope ) {
            scanf("%d%d%d%d",&x0,&y0,&x1,&y1) , ++cc;
            x0 = ( x0 + lastans + 39989 ) % 39989 + 1 , y0 = ( (lli) y0 + lastans + 1000000000 ) % 1000000000 + 1 ,
            x1 = ( x1 + lastans + 39989 ) % 39989 + 1 , y1 = ( (lli) y1 + lastans + 1000000000 ) % 1000000000 + 1 ;
            if( x1 < x0 ) swap( x1 , x0 ) , swap( y1 , y0 );
            if( x0 == x1 ) k[cc] = 0 , b[cc] = max( y0 , y1 );
            else k[cc] = (double)(y1-y0)/(x1-x0) , b[cc] = y1 - k[cc] * x1;
            update(1,x0,x1,cc);
        } else {
            scanf("%d",&p);
            p = ( p + lastans + 39989 ) % 39989 + 1;
            lastans = query(1,p);
            printf("%d\n",lastans);
            --lastans;
        }
    }
    return 0;
}
```

4.2.3 吉司机线段树

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

/*
    modify 将区间大于x的数变成x
    query 询问区间和
    单次复杂度 O(log^2(n))
*/

const ll INF=0xc0c0c0c0c0c0c0c0ll;
const int MAXN=200005;
ll seg[MAXN<<2],m1[MAXN<<2],m2[MAXN<<2],cnt[MAXN<<2],
    tag[MAXN<<2],a[MAXN];
int n,q;

void pushdown(int rt)
{
    if(!tag[rt]) return;
    ll y=m1[rt];
    if(y<m1[rt<<1])
    {
        tag[rt<<1]=1;
        seg[rt<<1]-=(m1[rt<<1]-y)*cnt[rt<<1];
        m1[rt<<1]=y;
    }
    if(y<m1[rt<<1|1])
    {
        tag[rt<<1|1]=1;
        seg[rt<<1|1]-=(m1[rt<<1|1]-y)*cnt[rt<<1|1];
        m1[rt<<1|1]=y;
    }
    tag[rt]=0;
}

void pushup(int rt)
{
    seg[rt]=seg[rt<<1]+seg[rt<<1|1];
    if(m1[rt<<1]==m1[rt<<1|1])
    {
        m1[rt]=m1[rt<<1];
        cnt[rt]=cnt[rt<<1]+cnt[rt<<1|1];
        m2[rt]=max(m2[rt<<1],m2[rt<<1|1]);
    }
    else if(m1[rt<<1]>m1[rt<<1|1])
    {
        m1[rt]=m1[rt<<1];
        cnt[rt]=cnt[rt<<1];
        m2[rt]=max(m2[rt<<1],m1[rt<<1|1]);
    }
    else
    {
        m1[rt]=m1[rt<<1|1];
        cnt[rt]=cnt[rt<<1|1];
        m2[rt]=max(m2[rt<<1|1],m1[rt<<1]);
    }
}

void build(int rt,int l,int r)
{
    tag[rt]=0;
    if(l==r)
    {
        seg[rt]=m1[rt]=a[l];
        cnt[rt]=1;
        m2[rt]=INF;
        return;
    }
    int m=l+r>>1;
    if(l<=m) build(rt<<1,l,m);
    if(m<r) build(rt<<1|1,m+1,r);
    pushup(rt);
}

void modify(int rt,int l,int r,int L,int R,ll y)
{
    if(y>=m1[rt]) return;
    if(L<=l&&r<=R&&y>m2[rt])
    {
        tag[rt]=1;
        seg[rt]-=(m1[rt]-y)*cnt[rt];
        m1[rt]=y;
        return;
    }
    pushdown(rt);
    int m=l+r>>1;
    if(L<=m) modify(rt<<1,l,m,L,R,y);
    if(m<R) modify(rt<<1|1,m+1,r,L,R,y);
    pushup(rt);
}

ll query(int rt,int l,int r,int L,int R)
{
    if(L<=l&&r<=R) return seg[rt];
    int m=l+r>>1;
    pushdown(rt);
    ll ret=0;
    if(L<=m) ret+=query(rt<<1,l,m,L,R);
    if(m<R) ret+=query(rt<<1|1,m+1,r,L,R);
    pushup(rt);
    return ret;
}
```

4.2.4 线段树维护扫描线

```cpp
#include <bits/stdc++.h>
#define x1 xa
#define x2 xb
#define y1 ya
#define y1 yb
using namespace std;
typedef long long ll;
const int N=500005;

//luogu P5490

/*
上面的define是为了防止某些时候的CE
时刻记得节点[3,3]表示的是3号Y坐标与4号Y坐标构成的线段长度
*/

int n;
ll x1,y1,x2,y2;
vector<ll> Y;

struct line
{
```

```
    ll x,y1,y2;
    int mark;
};
vector<line> v;
bool cmp(line a,line b){return a.x<b.x;}

struct seg
{
    int l,r,cov;
    ll len;
}tr[N<<2];

void build(int p,int l,int r)
{
    tr[p].l=l,tr[p].r=r;
    tr[p].len=tr[p].cov=0;
    if(l==r) return;
    int m=l+r>>1;
    build(p<<1,l,m);
    build(p<<1|1,m+1,r);
}

void up(int p)
{
    int l=tr[p].l,r=tr[p].r;
    if(tr[p].cov>0)
        tr[p].len=Y[r]-Y[l-1];//
    else
        tr[p].len=tr[p<<1].len+tr[p<<1|1].len;
}

void modi(int L,int R,int V,int p=1)
{
    int l=tr[p].l,r=tr[p].r;
    if(L<=l&&r<=R)
    {
        tr[p].cov+=V;
        up(p);
        return;
    }
    int m=l+r>>1;
    if(L<=m) modi(L,R,V,p<<1);
    if(R>m) modi(L,R,V,p<<1|1);
    up(p);
}

int main()
{
    scanf("%d",&n);
    for(int i=1;i<=n;++i)
    {
        scanf("%lld%lld%lld%lld",&x1,&y1,&x2,&y2);
        Y.push_back(y1);
        Y.push_back(y2);
        v.push_back((line){x1,y1,y2,1});
        v.push_back((line){x2,y1,y2,-1});
    }
    sort(v.begin(),v.end(),cmp);
    sort(Y.begin(),Y.end());
    for(auto &i:v)
    {
        i.y1=lower_bound(Y.begin(),Y.end(),i.y1)-Y.
            begin()+1;
        i.y2=lower_bound(Y.begin(),Y.end(),i.y2)-Y.
            begin()+1;
    }
    build(1,1,Y.size());
    ll ans=0;
    for(int i=0;i<=v.size()-2;++i)
    {
        modi(v[i].y1,v[i].y2-1,v[i].mark);//
        ans+=tr[1].len*(v[i+1].x-v[i].x);
    }
    printf("%lld",ans);
    return 0;
}
```

4.2.5 区间图

```
#include <bits/stdc++.h>
//CF 786B

/*
    建立两棵线段树分别表示出区间和入区间
    第1到n标号的节点表示原来的n个点
*/

using namespace std;
const int N=1e6+5;//original N=1e5
typedef long long ll;
const ll inf=0x3f3f3f3f3f3f3f3f;
typedef pair<ll,int> pli;

int n,q,s;
ll ans[N];
vector<int> e[N];
vector<ll> val[N];
bool vis[N];

void add(int x,int y,int z)
{
    e[x].push_back(y);
    val[x].push_back(z);
}

int cnt;
struct seg
{
    int id[N];
    void up(int p,bool flg)
    {
        int u=id[p];
        int v=id[p<<1];
        if(flg) swap(u,v);
        add(u,v,0ll);
        u=id[p];
        v=id[p<<1|1];
        if(flg) swap(u,v);
        add(u,v,0ll);
    }
    void build(int flg,int p=1,int l=1,int r=n)
    {
        id[p]=++cnt;
        if(l==r)
        {
            int u=id[p];
            int v=l;
            if(flg) swap(u,v);
            add(u,v,0ll);
```

```
            return;
        }
        int m=l+r>>1;
        build(flg,p<<1,l,m);
        build(flg,p<<1|1,m+1,r);
        up(p,flg);
    }
    void segadd(int u,ll w,int L,int R,bool flg,int p
        =1,int l=1,int r=n)
    {
        if(L<=l&&r<=R)
        {
            int v=id[p];
            if(flg) swap(u,v);
            add(u,v,w);
            return;
        }
        int m=l+r>>1;
        if(L<=m) segadd(u,w,L,R,flg,p<<1,l,m);
        if(R>m) segadd(u,w,L,R,flg,p<<1|1,m+1,r);
    }
}in,out;

void dij()
{
    priority_queue<pli,vector<pli>,greater<pli>> q;
    q.emplace(0ll,s);
    memset(ans,0x3f3f,sizeof(ans));
    ans[s]=0;
    while(!q.empty())
    {
        int t=q.top().second;q.pop();
        if(vis[t]) continue;
        vis[t]=1;
        for(int i=0;i<e[t].size();++i)
        {
            int y=e[t][i];
            ll v=val[t][i];
            if(ans[t]+v<ans[y])
            {
                ans[y]=ans[t]+v;
                q.emplace(ans[y],y);
            }
        }
    }
    for(int i=1;i<=n;++i)
        printf("%lld ",(ans[i]==inf?-1ll:ans[i]));
}

int main()
{
    scanf("%d%d%d",&n,&q,&s);
    cnt=n;
    //the first n points won't change
    in.build(0);
    out.build(1);
    //in tree no flip,out tree must flip
    for(int i=1;i<=q;++i)
    {
        int t,v,u,l,r;
        ll w;
        scanf("%d",&t);
        if(t==1)
        {
            scanf("%d%d%lld",&v,&u,&w);
            l=r=u;
            t=2;
        }
        else scanf("%d%d%d%lld",&v,&l,&r,&w);
        if(t==2) in.segadd(v,w,l,r,0);
        else if(t==3) out.segadd(v,w,l,r,1);
    }
    dij();
    return 0;
}
```

4.3 RMQ

4.3.1 一维

```
//dp[i][j]表示从a[i]开始，包括a[i]在内的2的j次方个数字中
    的最值
for(int i=1;i<=n;++i)
    dp[i][0]=a[i];
for(int j=1;j<=30;++j){
    for(int i=1;i+(1LL<<(j-1))<=n;++i){
        dp[i][j]=max(dp[i][j-1],dp[i+(1LL<<(j-1))][j
            -1]);//min
    }
}

int ask(int l,int r){
    int k=(int)log2(r-l+1);
    return max(dp[l][k],dp[r-(1LL<<k)+1][k]);//min
}
```

4.3.2 两维

4.4 树链剖分

4.4.1 点剖分

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

/*
    轻重链剖分 单次复杂度 O(log^2(n))
    a[i] 表示dfs标号为i的点的值,而非点i的值
    1 x y z 表示将树从x到y结点最短路径上所有节点值都加上z
    2 x y 表示求树从x到y结点最短路径上所有节点值之和
    3 x z 表示将以x为根节点的子树内所有节点值都加上z
    4 x 表示求以x为根节点的子树内所有节点值之和
*/

const int MAXN=100005;
ll mod,lazy[MAXN<<2],seg[MAXN<<2],a[MAXN],tmp[MAXN];
int n,q,r,cnt,tot,dep[MAXN],top[MAXN],id[MAXN],son[
    MAXN],num[MAXN],fa[MAXN];
vector<int> e[MAXN];

void dfs1(int now,int f)
{
    dep[now]=dep[f]+1;
    fa[now]=f;
    num[now]=1;
    son[now]=0;
    for(auto to:e[now])
    {
```

```
        if(to==f) continue;
        dfs1(to,now);
        num[now]+=num[to];
        if(num[to]>num[son[now]]) son[now]=to;
    }
}

void dfs2(int now,int f)
{
    id[now]=++cnt;
    top[now]=f;
    if(son[now]) dfs2(son[now],f);
    for(auto to:e[now])
        if(to!=fa[now]&&to!=son[now])
            dfs2(to,to);
}

inline void pushdown(int rt,ll lnum,ll rnum)
{
    if(!lazy[rt]) return;
    seg[rt<<1]=(seg[rt<<1]+lazy[rt]*lnum%mod)%mod;
    seg[rt<<1|1]=(seg[rt<<1|1]+lazy[rt]*rnum%mod)%mod;
    lazy[rt<<1]=(lazy[rt<<1]+lazy[rt])%mod;
    lazy[rt<<1|1]=(lazy[rt<<1|1]+lazy[rt])%mod;
    lazy[rt]=0;
}

inline void pushup(int rt)
{
    seg[rt]=(seg[rt<<1]+seg[rt<<1|1])%mod;
}

void build(int rt,int l,int r)
{
    lazy[rt]=0;
    if(l==r)
    {
        seg[rt]=a[l]%mod;
        return;
    }
    int m=l+r>>1;
    if(l<=m) build(rt<<1,l,m);
    if(m<r) build(rt<<1|1,m+1,r);
    pushup(rt);
}

void modify(int rt,int l,int r,int L,int R,ll x)
{
    if(L<=l&&r<=R)
    {
        lazy[rt]=(lazy[rt]+x)%mod;
        seg[rt]=(seg[rt]+x*(r-l+1)%mod)%mod;
        return;
    }
    int m=l+r>>1;
    pushdown(rt,m-l+1,r-m);
    if(L<=m) modify(rt<<1,l,m,L,R,x);
    if(m<R) modify(rt<<1|1,m+1,r,L,R,x);
    pushup(rt);
}

ll query(int rt,int l,int r,int L,int R)
{
    if(L<=l&&r<=R) return seg[rt];
    int m=l+r>>1;
    ll ret=0;
    pushdown(rt,m-l+1,r-m);
    if(L<=m) ret=(ret+query(rt<<1,l,m,L,R))%mod;
    if(m<R) ret=(ret+query(rt<<1|1,m+1,r,L,R))%mod;
    pushup(rt);
    return ret;
}

int main()
{
    scanf("%d%d%d%lld",&n,&q,&r,&mod);
    for(int i=1;i<=n;i++) scanf("%lld",&tmp[i]);
    for(int i=1,x,y;i<n;i++)
    {
        scanf("%d%d",&x,&y);
        e[x].push_back(y),e[y].push_back(x);
    }
    num[0]=0,dep[r]=0;
    dfs1(r,r);
    dfs2(r,r);
    for(int i=1;i<=n;i++) a[id[i]]=tmp[i];
    build(1,1,n);

    while(q--)
    {
        int op,x,y;ll z;
        scanf("%d%d",&op,&x);
        if(op==4)
        {
            printf("%lld\n",query(1,1,n,id[x],id[x]+num
                [x]-1));
            continue;
        }
        if(op==1)
        {
            scanf("%d%lld",&y,&z);z%=mod;
            while(top[x]!=top[y])
            {
                if(dep[top[x]]<dep[top[y]]) swap(x,y);
                modify(1,1,n,id[top[x]],id[x],z);
                x=fa[top[x]];
            }
            if(dep[x]>dep[y]) swap(x,y);
            modify(1,1,n,id[x],id[y],z);
        }
        else if(op==2)
        {
            scanf("%d",&y);
            ll ans=0;
            while(top[x]!=top[y])
            {
                if(dep[top[x]]<dep[top[y]]) swap(x,y);
                ans=(ans+query(1,1,n,id[top[x]],id[x]))%
                    mod;
                x=fa[top[x]];
            }
            if(dep[x]>dep[y]) swap(x,y);
            ans=(ans+query(1,1,n,id[x],id[y]))%mod;
            printf("%lld\n",ans);
        }
        else
        {
            scanf("%lld",&z);z%=mod;
            modify(1,1,n,id[x],id[x]+num[x]-1,z);
        }
```

```
    }
    return 0;
}
```

4.4.2 边剖分

4.5 平衡树

4.5.1 Treap

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

const int MAXN=1e5+5;
const int inf=0x7fffffff;
int n,op,x;

/*
    树内初始化时有无穷大和无穷小两个结点
    _delete(root,x) 删除一个x
    _insert(root,x) 插入一个x
    getRank(root,x) 返回x的排名+1(包含了无穷小)
    getVal(root,x+1) 返回排名为x的数
    getPrev(x) x的前驱
    getNext(x) x的后继
*/

namespace Treap
{
    int tot,root;
    struct node
    {
        int cnt,val,dat,siz,lc,rc;
    }bst[MAXN];

    inline void pushup(int rt)
    {
        bst[rt].siz=bst[rt].cnt;
        if(bst[rt].lc) bst[rt].siz+=bst[bst[rt].lc].siz;
        if(bst[rt].rc) bst[rt].siz+=bst[bst[rt].rc].siz;
    }

    inline void zig(int &rt)
    {
        int p=bst[rt].lc;
        bst[rt].lc=bst[p].rc;
        bst[p].rc=rt;
        rt=p;
        pushup(bst[rt].rc);pushup(rt);
    }

    inline void zag(int &rt)
    {
        int p=bst[rt].rc;
        bst[rt].rc=bst[p].lc;
        bst[p].lc=rt;
        rt=p;
        pushup(bst[rt].lc);pushup(rt);
    }

    int new_node(int val)
    {
        bst[++tot].val=val;
        bst[tot].dat=rand();
        bst[tot].siz=bst[tot].cnt=1;
        bst[tot].lc=bst[tot].rc=0;
        return tot;
    }

    void build()
    {
        new_node(-inf);new_node(inf);
        root=1,bst[1].rc=2;
        pushup(1);
    }

    void _insert(int &rt,int val)
    {
        if(rt==0)
        {
            rt=new_node(val);
            return;
        }
        if(bst[rt].val==val)
        {
            bst[rt].cnt++;
            pushup(rt);
            return;
        }
        if(val<bst[rt].val)
        {
            _insert(bst[rt].lc,val);
            if(bst[rt].dat<bst[bst[rt].lc].dat) zig(rt);
        }
        else
        {
            _insert(bst[rt].rc,val);
            if(bst[rt].dat<bst[bst[rt].rc].dat) zag(rt);
        }
        pushup(rt);
    }

    void _delete(int &rt,int val)
    {
        if(rt==0) return;
        if(bst[rt].val==val)
        {
            if(bst[rt].cnt>1)
            {
                bst[rt].cnt--;
                pushup(rt);
                return;

            }
            if(bst[rt].rc||bst[rt].lc)
            {
                if(bst[rt].rc==0||bst[bst[rt].rc].dat<bst[bst[rt].lc].dat)
                    zig(rt),_delete(bst[rt].rc,val);
                else
                    zag(rt),_delete(bst[rt].lc,val);
                pushup(rt);
            }
            else rt=0;
            return;
        }
```

```
    if(val<bst[rt].val) _delete(bst[rt].lc,val);
    else _delete(bst[rt].rc,val);
    pushup(rt);
}

int getPrev(int val)
{
    int ret=1,rt=root;
    while(rt)
    {
        if(bst[rt].val==val)
        {
            if(bst[rt].lc)
            {
                rt=bst[rt].lc;
                while(bst[rt].rc) rt=bst[rt].rc;
                ret=rt;
            }
            break;
        }
        if(bst[rt].val<val&&bst[rt].val>bst[ret].val) ret=rt;
        if(val<bst[rt].val) rt=bst[rt].lc;
        else rt=bst[rt].rc;
    }
    return bst[ret].val;
}

int getNext(int val)
{
    int ret=2,rt=root;
    while(rt)
    {
        if(bst[rt].val==val)
        {
            if(bst[rt].rc)
            {
                rt=bst[rt].rc;
                while(bst[rt].lc) rt=bst[rt].lc;
                ret=rt;
            }
            break;
        }
        if(bst[rt].val>val&&bst[rt].val<bst[ret].val) ret=rt;
        if(val<bst[rt].val) rt=bst[rt].lc;
        else rt=bst[rt].rc;
    }
    return bst[ret].val;
}

int getRank(int rt,int val)
{
    if(rt==0) return 0;
    if(val==bst[rt].val) return bst[bst[rt].lc].siz+1;
    if(val<bst[rt].val) return getRank(bst[rt].lc,val);
    else return bst[bst[rt].lc].siz+bst[rt].cnt+getRank(bst[rt].rc,val);
}

int getVal(int rt,int k)
{
    if(rt==0) return inf;
    if(bst[bst[rt].lc].siz>=k) return getVal(bst[rt].lc,k);
    if(bst[bst[rt].lc].siz+bst[rt].cnt>=k) return bst[rt].val;
    return getVal(bst[rt].rc,k-bst[bst[rt].lc].siz-bst[rt].cnt);
}
}

int main()
{
    using namespace Treap;
    srand(time(0));
    build();
    scanf("%d",&n);
    while(n--)
    {
        scanf("%d%d",&op,&x);
        if(op==1) _insert(root,x);
        else if(op==2) _delete(root,x);
        else if(op==3) printf("%d\n",getRank(root,x)-1);
        else if(op==4) printf("%d\n",getVal(root,x+1));
        else if(op==5) printf("%d\n",getPrev(x));
        else if(op==6) printf("%d\n",getNext(x));
    }
    return 0;
}
```

4.5.2 Splay

4.5.3 红黑树-pbds

```
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
tree<pii,null_type,less<pii>,rb_tree_tag,
    tree_order_statistics_node_update> t;

t.insert({x,i+1}); //插入x，用独特的正整数i+1标注（因为erase太辣鸡）
t.erase(t.lower_bound({x,0})); //删除x（删除单个元素）
t.order_of_key({x,0})+1; //x的排名（小于x的元素个数+1）
t.find_by_order(x-1)->first; //排名为x的元素（第x小的数）
prev(t.lower_bound({x,0}))->first; //x的前驱（小于x且最大）
t.lower_bound({x+1,0})->first; //x的后继（大于x且最小）
```

4.6 动态树

4.7 主席树

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

/*
    离散化+区间k小
*/

const int MAXN=200005;
int n,m,a[MAXN],tmp[MAXN],org[MAXN],root[MAXN],tot=0;
struct tree
```

```
{
    int cnt,lc,rc;
}seg[30*MAXN];

int build(int l,int r)
{
    int p=tot++;
    if(l==r)
    {
        seg[p].cnt=0;
        return p;
    }
    int m=l+r>>1;
    seg[p].lc=build(l,m);
    seg[p].rc=build(m+1,r);
    seg[p].cnt=seg[seg[p].lc].cnt+seg[seg[p].rc].cnt;
    return p;
}

int modify(int rt,int l,int r,int x)
{
    int p=tot++;
    seg[p]=seg[rt];
    if(l==r)
    {
        seg[p].cnt++;
        return p;
    }
    int m=l+r>>1;
    if(x<=m) seg[p].lc=modify(seg[rt].lc,l,m,x);
    else seg[p].rc=modify(seg[rt].rc,m+1,r,x);
    seg[p].cnt=seg[seg[p].lc].cnt+seg[seg[p].rc].cnt;
    return p;
}

int query(int p,int q,int l,int r,int k)
{
    if(l==r) return l;
    int m=l+r>>1;
    int lcnt=seg[seg[q].lc].cnt-seg[seg[p].lc].cnt;
    if(lcnt>=k) return query(seg[p].lc,seg[q].lc,l,m,k
        );
    else return query(seg[p].rc,seg[q].rc,m+1,r,k-lcnt
        );
}

int main()
{
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++)
        scanf("%d",a+i),tmp[i]=a[i];
    sort(tmp+1,tmp+n+1);
    root[0]=build(1,n);
    for(int i=1;i<=n;i++)
    {
        int k=lower_bound(tmp+1,tmp+n+1,a[i])-tmp;
        org[k]=a[i];
        a[i]=k;
        root[i]=modify(root[i-1],1,n,a[i]);
    }
    while(m--)
    {
        int x,y,k;
        scanf("%d%d%d",&x,&y,&k);
        printf("%d\n",org[query(root[x-1],root[y],1,n,
            k)]);
    }
    return 0;
}
```

4.8 树套树

4.8.1 线段树套 Treap

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

/*
    空间 O(nlogn)
    单点修改,区间rank,前驱后继(不存在则为±2147483647) 单
        次 O(log^2(n))
    区间排名为k的值 单次 O(log^3(n))
*/

const int inf=2147483647;
const int MAXN=50005;
int root[MAXN<<2],n,m,a[MAXN];
struct Treap
{
    int tot;
    struct node
    {
        int lc,rc,dat,val,cnt,siz;
    }bst[MAXN*4*20];

    int newnode(int v)
    {
        bst[++tot].val=v;
        bst[tot].dat=rand();
        bst[tot].siz=bst[tot].cnt=1;
        bst[tot].lc=bst[tot].rc=0;
        return tot;
    }

    void zig(int &rt)
    {
        int p=bst[rt].lc;
        bst[rt].lc=bst[p].rc;
        bst[p].rc=rt;
        rt=p;
        pushup(bst[rt].rc);
        pushup(rt);
    }

    void zag(int &rt)
    {
        int p=bst[rt].rc;
        bst[rt].rc=bst[p].lc;
        bst[p].lc=rt;
        rt=p;
        pushup(bst[rt].lc);
        pushup(rt);
    }

    void pushup(int rt)
    {
        bst[rt].siz=bst[rt].cnt;
        if(bst[rt].lc) bst[rt].siz+=bst[bst[rt].lc].
            siz;
```

```
    if(bst[rt].rc) bst[rt].siz+=bst[bst[rt].rc].siz;
}

int build()
{
    int rt=newnode(-inf);
    bst[rt].rc=newnode(inf);
    pushup(rt);
    return rt;
}

void _delete(int &rt,int x)
{
    if(bst[rt].val==x)
    {
        if(bst[rt].cnt>1)
        {
            bst[rt].cnt--;
            pushup(rt);
            return;
        }
        if(bst[rt].lc||bst[rt].rc)
        {
            if(bst[rt].rc==0||bst[bst[rt].rc].dat<bst[bst[rt].lc].dat)
                zig(rt),_delete(bst[rt].rc,x);
            else
                zag(rt),_delete(bst[rt].lc,x);
            pushup(rt);
        }
        else rt=0;
        return;
    }
    if(x<bst[rt].val) _delete(bst[rt].lc,x);
    else _delete(bst[rt].rc,x);
    pushup(rt);
}

void _insert(int &rt,int x)
{
    if(rt==0)
    {
        rt=newnode(x);
        return;
    }
    if(bst[rt].val==x) bst[rt].cnt++;
    else if(x<bst[rt].val)
    {
        _insert(bst[rt].lc,x);
        if(bst[bst[rt].lc].dat>bst[rt].dat) zig(rt);
    }
    else
    {
        _insert(bst[rt].rc,x);
        if(bst[bst[rt].rc].dat>bst[rt].dat) zag(rt);
    }
    pushup(rt);
}

int get_rank(int rt,int x)
{
    if(!rt) return 1;
    if(bst[rt].val==x) return bst[bst[rt].lc].siz+1;
    if(x<bst[rt].val) return get_rank(bst[rt].lc,x);
    else return get_rank(bst[rt].rc,x)+bst[bst[rt].lc].siz+bst[rt].cnt;
}

int get_num(int rt,int x)
{
    if(!rt) return 0;
    if(bst[rt].val==x) return bst[bst[rt].lc].siz+bst[rt].cnt;
    if(x<bst[rt].val) return get_num(bst[rt].lc,x);
    else return get_num(bst[rt].rc,x)+bst[bst[rt].lc].siz+bst[rt].cnt;
}

int get_prev(int rt,int x)
{
    int ret=-inf;
    while(rt)
    {
        if(bst[rt].val==x)
        {
            if(bst[rt].lc)
            {
                rt=bst[rt].lc;
                while(bst[rt].rc) rt=bst[rt].rc;
                ret=bst[rt].val;
            }
            break;
        }
        if(bst[rt].val<x&&bst[rt].val>ret) ret=bst[rt].val;
        if(x<bst[rt].val) rt=bst[rt].lc;
        else rt=bst[rt].rc;
    }
    return ret;
}

int get_nxt(int rt,int x)
{
    int ret=inf;
    while(rt)
    {
        if(bst[rt].val==x)
        {
            if(bst[rt].rc)
            {
                rt=bst[rt].rc;
                while(bst[rt].lc) rt=bst[rt].lc;
                ret=bst[rt].val;
            }
            break;
        }
        if(bst[rt].val>x&&bst[rt].val<ret) ret=bst[rt].val;
        if(x<bst[rt].val) rt=bst[rt].lc;
        else rt=bst[rt].rc;
    }
    return ret;
}
}treap;
```

```

void build(int rt,int l,int r)
{
    root[rt]=treap.build();
    if(l==r) return;
    int m=l+r>>1;
    build(rt<<1,l,m);
    build(rt<<1|1,m+1,r);
}

void modify(int rt,int l,int r,int x,int v,int y)
{
    if(y==-1) treap._delete(root[rt],v);
    else treap._insert(root[rt],v);
    if(l==r) return;
    int m=l+r>>1;
    if(x<=m) modify(rt<<1,l,m,x,v,y);
    else modify(rt<<1|1,m+1,r,x,v,y);
}

int query(int rt,int l,int r,int op,int L,int R,int x
    )
{
    if(L<=l&&r<=R)
    {
        if(op==1) return treap.get_rank(root[rt],x)-2;
        if(op==2) return treap.get_num(root[rt],x)-1;
        if(op==4) return treap.get_prev(root[rt],x);
        if(op==5) return treap.get_nxt(root[rt],x);
    }
    int m=l+r>>1,ret;
    if(op==1||op==2)
    {
        ret=0;
        if(L<=m) ret+=query(rt<<1,l,m,op,L,R,x);
        if(m<R) ret+=query(rt<<1|1,m+1,r,op,L,R,x);
    }
    if(op==4)
    {
        ret=-inf;
        if(L<=m) ret=max(ret,query(rt<<1,l,m,op,L,R,x)
            );
        if(m<R) ret=max(ret,query(rt<<1|1,m+1,r,op,L,R
            ,x));
    }
    if(op==5)
    {
        ret=inf;
        if(L<=m) ret=min(ret,query(rt<<1,l,m,op,L,R,x)
            );
        if(m<R) ret=min(ret,query(rt<<1|1,m+1,r,op,L,R
            ,x));
    }
    return ret;
}

int main()
{
    srand(time(0));
    scanf("%d%d",&n,&m);
    build(1,1,n);
    for(int i=1;i<=n;i++)
    {
        scanf("%d",a+i);
        modify(1,1,n,i,a[i],1);
    }
    while(m--)
    {
        int op,l,r,k,pos;
        scanf("%d",&op);
        if(op==1)
        {
            scanf("%d%d%d",&l,&r,&k);
            printf("%d\n",query(1,1,n,op,l,r,k)+1);
        }
        else if(op==2)
        {
            scanf("%d%d%d",&l,&r,&k);
            int L=-inf,R=inf,mid;
            while(L<R)
            {
                mid=(L+R+1)>>1;
                if(query(1,1,n,1,l,r,mid)+1>k) R=mid-1;
                    else L=mid;
            }
            printf("%d\n",L);
        }
        else if(op==3)
        {
            scanf("%d%d",&pos,&k);
            modify(1,1,n,pos,a[pos],-1);
            a[pos]=k;
            modify(1,1,n,pos,k,1);
        }
        else
        {
            scanf("%d%d%d",&l,&r,&k);
            printf("%d\n",query(1,1,n,op,l,r,k));
        }
    }
    return 0;
}
```

4.8.2 树状数组套线段树

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

/*
    带单点修区间k小
    用的时候注意下空间 时空 O(nlog^2(n))
    外层 add(pos,x,y) 空间上为pos的点且值域上为x的点加上y
        query(l,r,k) 询问区间[l,r]里k小
    内层 modify 值域线段树动态开点
        query 值域线段树区间k小
    VAL 值域大小
*/

const int MAXN=200005;
int n,a[MAXN],X[MAXN],Y[MAXN],c1,c2,VAL;
struct SEG
{
    int root[MAXN],lc[MAXN*500],rc[MAXN*500],cnt[MAXN
        *500],tot;
    void modify(int &rt,int l,int r,int x,int y)
    {
        if(rt==0) rt=++tot;
        cnt[rt]+=y;
        if(l==r) return;
```

```
        int m=l+r>>1;
        if(x<=m) modify(lc[rt],l,m,x,y);
        else modify(rc[rt],m+1,r,x,y);
    }
    int query(int l,int r,int k)
    {
        if(l==r) return l;
        int sum=0,m=l+r>>1;
        for(int i=0;i<c1;i++) sum-=cnt[lc[X[i]]];
        for(int i=0;i<c2;i++) sum+=cnt[lc[Y[i]]];
        if(sum>=k)
        {
            for(int i=0;i<c1;i++) X[i]=lc[X[i]];
            for(int i=0;i<c2;i++) Y[i]=lc[Y[i]];
            return query(l,m,k);
        }
        else
        {
            for(int i=0;i<c1;i++) X[i]=rc[X[i]];
            for(int i=0;i<c2;i++) Y[i]=rc[Y[i]];
            return query(m+1,r,k-sum);
        }
    }
}seg;

void add(int pos,int x,int y)
{
    for(;pos<=n;pos+=pos&-pos) seg.modify(seg.root[pos
        ],1,VAL,x,y);
}

int query(int l,int r,int k)
{
    c1=c2=0;
    for(int i=l-1;i;i-=i&-i) X[c1++]=seg.root[i];
    for(int i=r;i;i-=i&-i) Y[c2++]=seg.root[i];
    return seg.query(1,VAL,k);
}
```

4.9 K-D Tree

```
#include<bits/stdc++.h>
#define fi first
#define se second
using namespace std;
//HDOJ 4347

/*
    给出 n 个 K 维的点，又给出 t 个查询， 每个查询也给出一
        个 K 维的点，要求查询 n 个点中距离这个点最近的 M
        个点。
*/

const int maxn=5e4+10;
const int maxk=5;
int k,idx;
struct Point
{
    int x[maxk];
    bool operator<(const Point &o)const
    {
        return x[idx]<o.x[idx];
    }
    void print()
    {
        for(int i=0;i<k;i++) printf("%d%c",x[i],(i==k
            -1)?'\n':' ');
    }
}poi[maxn];
typedef pair<double,Point> P;
priority_queue<P> Q;
void clear(priority_queue<P> &Q)
{
    if(Q.empty()) return;
    priority_queue<P> tp;
    swap(Q,tp);
}
struct kdTree
{
    #define sqr(x) ((x)*(x))
    #define ls (rt<<1)
    #define rs (rt<<1|1)
    Point o[maxn<<2];
    int son[maxn<<2];

    void build(int rt,int l,int r,int dep)
    {
        if(l>r) return;
        son[rt]=r-l, son[ls]=son[rs]=-1;
        idx=dep%k;
        int mid=(l+r)>>1;
        nth_element(poi+l,poi+mid,poi+r+1);
        o[rt]=poi[mid];
        build(ls,l,mid-1,dep+1);
        build(rs,mid+1,r,dep+1);
    }
    void query(int rt,Point p,int m,int dep)
    {
        if(son[rt]==-1) return;
        P nd(0,o[rt]);
        for(int i=0;i<k;i++) nd.fi+=sqr(nd.se.x[i]-p.x
            [i]);
        int dim=dep%k, x=ls, y=rs; bool fg=0;
        if(p.x[dim]>=o[rt].x[dim]) swap(x,y);
        if(~son[x]) query(x,p,m,dep+1);
        if(Q.size()<m) Q.push(nd), fg=1;
        else
        {
            if(nd.fi<Q.top().fi) Q.pop(), Q.push(nd);
            if(sqr(p.x[dim]-o[rt].x[dim])<Q.top().fi)
                fg=1;
        }
        if(~son[y] && fg) query(y,p,m,dep+1);
    }
}kdt;

int n;
int main()
{
    while(scanf("%d%d",&n,&k)!=EOF)
    {
        for(int i=0;i<n;i++) for(int j=0;j<k;j++)
            scanf("%d",&(poi[i].x[j]));
        kdt.build(1,0,n-1,0);
        int t; scanf("%d",&t);
        stack<Point> ans;
        while(t--)
        {
            Point qry;
```

```
        for(int i=0;i<k;i++) scanf("%d",&qry.x[i]);
        int m; scanf("%d",&m);
        clear(Q);
        kdt.query(1,qry,m,0);
        printf("the closest %d points are:\n",m);
        while(Q.size()) ans.push(Q.top().se), Q.pop
            ();
        while(ans.size()) ans.top().print(), ans.
            pop();
    }
  }
}
```

4.10 分治

4.10.1 CDQ

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

/*
    严格大于的三维偏序
    无法处理重复的数字，但是注意"大于"的神秘排序方法
*/

const int N=500005;

int n,c[N];
struct node{
    int a,b,c,cnt;
}o[N],t[N];

bool cmp(node a,node b){
    if(a.a!=b.a) return a.a>b.a;
    else return a.c<b.c;
}

inline int lowbit(int x){return x&-x;}
int tr[N];
void add(int x,int y){
    for(;x<=n;x+=lowbit(x)) tr[x]+=y;
}
int sum(int x){
    int res=0;
    for(;x;x-=lowbit(x)) res+=tr[x];
    return res;
}

void cdq(int l,int r){
    if(l==r) return;
    int m=(l+r)/2;
    cdq(l,m);
    cdq(m+1,r);
    int p=l,q=m+1,tot=l;
    while(p<=m&&q<=r){
        if(o[p].b>o[q].b) add(o[p].c,1),t[tot++]=o[p
            ++];
        else o[q].cnt+=sum(n)-sum(o[q].c),t[tot++]=o[q
            ++];
    }
    while(p<=m) add(o[p].c,1),t[tot++]=o[p++];
    while(q<=r) o[q].cnt+=sum(n)-sum(o[q].c),t[tot++]=
        o[q++];
    for(int i=l;i<=m;++i) add(o[i].c,-1);
    for(int i=l;i<=r;++i) o[i]=t[i];
}

int main()
{
    scanf("%d",&n);
    for(int i=1;i<=n;++i) scanf("%d",&o[i].a);
    for(int i=1;i<=n;++i) scanf("%d",&o[i].b);
    for(int i=1;i<=n;++i) scanf("%d",&o[i].c),c[i]=o[i
        ].c;
    sort(c+1,c+1+n);
    for(int i=1;i<=n;++i) o[i].c=lower_bound(c+1,c+1+n
        ,o[i].c)-c;
    sort(o+1,o+1+n,cmp);
    cdq(1,n);
    int ans=0;
    for(int i=1;i<=n;++i) if(o[i].cnt>0) ans++;
    printf("%d",ans);
    return 0;
}
```

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

/*
    解决有等于的三维偏序
    严格小于等于的个数，可以解决重复问题，有离散化
*/

const int maxn=500005;

int n,k;
int cnt[maxn];//save the ans
struct ss{
    int a,b,c,w,ans;
}tmps[maxn],s[maxn];//struct
bool cmp1(ss x,ss y){//sort1
    if(x.a==y.a){
        if(x.b!=y.b) return x.b<y.b;
        else return x.c<y.c;
    }
    else return x.a<y.a;
}
bool cmp2(ss x,ss y){//sort2
    if(x.b!=y.b) return x.b<y.b;
    else return x.c<y.c;
}

struct tree_array{//tree_array
    int tr[maxn+5],n;
    int lowbit(int x){return x&-x;}
    int ask(int x){int ans=0;for(;x;x-=lowbit(x))ans+=
        tr[x];return ans;}
    void add(int x,int y){for(;x<=n;x+=lowbit(x))tr[x
        ]+=y;}
}t;

void cdq(int l,int r){
    if(l==r) return;
    int m=l+r>>1;
    cdq(l,m);
    cdq(m+1,r);
    sort(s+l,s+m+1,cmp2);
    sort(s+m+1,s+r+1,cmp2);//sort2
    int i=l,j=m+1;
    for(;j<=r;++j){
```

```
        while(i<=m&&s[i].b<=s[j].b){//the second dimension
            t.add(s[i].c,s[i].w);//use the tree_array to save the ans
            ++i;
        }
        s[j].ans+=t.ask(s[j].c);//contribution
    }
    for(int j=l;j<i;++j)
        t.add(s[j].c,-s[j].w);//init the first half
}

int main(){
    scanf("%d%d",&n,&k);
    for(int i=1;i<=n;++i)
        scanf("%d%d%d",&tmps[i].a,&tmps[i].b,&tmps[i].c);
    sort(tmps+1,tmps+1+n,cmp1);//sort1
    int now=0,nn=0;
    for(int i=1;i<=n;++i){
        now++;
        if(tmps[i].a!=tmps[i+1].a||tmps[i].b!=tmps[i+1].b
        ||tmps[i].c!=tmps[i+1].c){
            s[++nn]=tmps[i];
            s[nn].w=now;
            now=0;
        }//compress the same
    }
    t.n=maxn;//tree_array on the range
    cdq(1,nn);
    for(int i=1;i<=nn;++i)
        cnt[s[i].ans+s[i].w-1]+=s[i].w;//
    for(int i=0;i<n;++i)
        printf("%d\n",cnt[i]);
    return 0;
}
```

4.10.2 点分治

4.10.3 dsu on tree

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

/*
    统计每颗子树内的出现次数最多的数(们)的和
    复杂度 O(nlogn)
*/

int n,c[100005],cnt[100005],mx,son[100005],siz[100005],hson;
ll ans[100005],sum;
vector<int> e[100005];

void dfs1(int now,int fa)
{
    son[now]=0,siz[now]=1;
    for(auto to:e[now])
    {
        if(to==fa) continue;
        dfs1(to,now);
        siz[now]+=siz[to];
        if(siz[to]>siz[son[now]]) son[now]=to;
    }
}

void cal(int now,int fa,int y)
{
    cnt[c[now]]+=y;
    if(cnt[c[now]]==mx) sum+=c[now];
    else if(cnt[c[now]]>mx) mx=cnt[c[now]],sum=c[now];
    for(auto to:e[now])
        if(to!=fa&&to!=hson) cal(to,now,y);
}

void dfs2(int now,int fa,int keep)
{
    for(auto to:e[now])
    {
        if(to==fa||to==son[now]) continue;
        dfs2(to,now,0);
    }
    if(son[now]) dfs2(son[now],now,1);
    hson=son[now];
    cal(now,fa,1);
    hson=0;
    ans[now]=sum;
    if(!keep) cal(now,fa,-1),sum=0,mx=0;
}

int main()
{
    scanf("%d",&n);
    for(int i=1;i<=n;i++) scanf("%d",c+i);
    for(int i=1,x,y;i<n;i++)
    {
        scanf("%d%d",&x,&y);
        e[x].push_back(y),e[y].push_back(x);
    }
    dfs1(1,1);
    dfs2(1,1,1);
    for(int i=1;i<=n;i++) printf("%lld ",ans[i]);
    return 0;
}
```

4.10.4 整体二分

4.11 分块

4.11.1 普通分块

```
#include <bits/stdc++.h>
//luogu P3203
using namespace std;
const int N=500005;

int n,m,tot;
int a[N],cnt[N],pos[N];
int id[N],from[N],to[N];
int o,x,y;

void modify(int i)
{
    if(i+a[i]>n)
    {
        pos[i]=i;
        cnt[i]=0;
        return;
```

```
    }
    if(id[i]==id[i+a[i]])
    {
        pos[i]=pos[i+a[i]];
        cnt[i]=cnt[i+a[i]]+1;
    }
    else
    {
        pos[i]=i+a[i];
        cnt[i]=1;
    }
}

void ask(int x)
{
    int p=x,res=0;
    while(p!=pos[p])
        res+=cnt[p],
        p=pos[p];
    printf("%d\n",res+1);
}

int main()
{
    scanf("%d",&n);
    tot=(int)sqrt(n);
    for(int i=1;i<=tot;++i)
    {
        from[i]=(i-1)*tot+1;
        to[i]=i*tot;
    }
    if(to[tot]<n)
    {
        tot++;
        from[tot]=to[tot-1];
        to[tot]=n;
    }
    for(int i=1;i<=tot;++i)
    {
        for(int j=from[i];j<=to[i];++j)
            id[j]=i;
    }
    for(int i=1;i<=n;++i)
        scanf("%d",&a[i]);
    for(int i=n;i>=1;--i)
        modify(i);
    scanf("%d",&m);
    while(m--)
    {
        scanf("%d",&o);
        if(o==2)
        {
            scanf("%d%d",&x,&y);
            x++;
            a[x]=y;
            for(int i=x;i>=from[id[x]];--i)
                modify(i);
        }
        else if(o==1)
        {
            scanf("%d",&x);
            x++;;
            ask(x);
        }
    }
    return 0;
}
```

4.11.2 莫队

```
#include <bits/stdc++.h>
//luogu P3203
using namespace std;
const int N=500005;

int n,m,k,a[N];
struct node
{
    int l,r,id,ans;
}q[N];
int cnt[N],ans[N];

bool cmp(node a,node b)
{
    if(a.id==b.id)
    {
        if(a.id%2==0) return a.r<b.r;
        else return a.r>b.r;
    }
    else return a.id<b.id;
}

int now=0;
void del(int x)
{
    now-=cnt[a[x]]*cnt[a[x]];
    cnt[a[x]]--;
    now+=cnt[a[x]]*cnt[a[x]];
}
void add(int x)
{
    now-=cnt[a[x]]*cnt[a[x]];
    cnt[a[x]]++;
    now+=cnt[a[x]]*cnt[a[x]];
}

int main()
{
    scanf("%d%d%d",&n,&m,&k);
    int sz=sqrt(m);
    for(int i=1;i<=n;++i)
        scanf("%d",&a[i]);
    for(int i=1;i<=m;++i)
        scanf("%d%d",&q[i].l,&q[i].r),q[i].ans=i;
    for(int i=1;i<=m;++i)
        q[i].id=(q[i].l+sz-1)/sz;
    sort(q+1,q+1+m,cmp);
    int L,R;
    L=R=q[1].l;
    R--;
    for(int i=1;i<=m;++i)
    {
        while(L<q[i].l) del(L++);
        while(L>q[i].l) add(--L);
        while(R>q[i].r) del(R--);
        while(R<q[i].r) add(++R);
        ans[q[i].ans]=now;
    }
    for(int i=1;i<=m;++i)
```

```
        printf("%d\n",ans[i]);
    return 0;
}
```

4.12 线性基

```

/*
    bool have(int x) 返回线性基中的数字能否表示数字x
    void ins(int x) 插入数字x
    int mn() 返回能表示的最小值
    int mx() 返回能表示的最大值
    void bug() 输出p数组
    int kth(int k) 返回能表示的所有值当中排名第k的
*/

struct LB{
    int p[100],N,tmp[100];
    bool flag;
    LB(){memset(p,0,sizeof(p));memset(tmp,0,sizeof(tmp
        ));flag=0;N=62;}
    void ins(int x){
        for(int i=N;i>=0;--i){
            if(x&(1LL<<i)){
                if(!p[i]) {p[i]=x;return;sz++;}
                else x^=p[i];
            }
        }
        flag=1;
    }
    int mx(){
        int ans=0;
        for(int i=N;i>=0;--i){
            if((ans^p[i])>ans) ans^=p[i];
        }
        return ans;
    }
    int mn(){
        if(flag) return 0;
        for(int i=0;i<=N;++i)
            if(p[i]) return p[i];
    }
    bool have(int x){
        for(int i=N;i>=0;--i){
            if(x&(1LL<<i)){
                if(!p[i]) return 0;
                else x^=p[i];
            }
        }
        return 1;
    }
    int kth(int k){
        int res=0,cnt=0;
        k-=flag;
        if(!k) return 0;
        for(int i=0;i<=N;++i){
            for(int j=i-1;j>=0;--j){
                if(p[i]&(1LL<<j)) p[i]^=p[j];
            }
            if(p[i]) tmp[cnt++]=p[i];
        }
        if(k>=(1LL<<cnt)) return -1;//can't find
        for(int i=0;i<cnt;++i)
            if(k&(1LL<<i)) res^=tmp[i];
        return res;
    }
    void bug(){
        for(int i=0;i<=N;++i) cout<<p[i]<<' ';
        cout<<endl;
    }
}lb;
```

4.13 珂朵莉树

```
#include <bits/stdc++.h>
#define int long long //be careful
//CF896C
using namespace std;

/*
    珂朵莉树的左右split顺序很重要，并且set集合一开始不要为
        空，否则会RE
*/

const int N=1000005;

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

struct node
{
    int l,r;
    mutable int v;
    node(int L,int R=-1,int V=0):l(L),r(R),v(V){}
    bool operator < (const node& o)const{return l<o.l
        ;}
};
set<node> s;
typedef set<node>::iterator it;

it split(int pos)
{
    it i=s.lower_bound(node(pos));
    if(i!=s.end()&&i->l==pos) return i;
    --i;
    int L=i->l,R=i->r,V=i->v;
    s.erase(i);
    s.insert(node(L,pos-1,V));
    return s.insert(node(pos,R,V)).first;
}

void assign(int l,int r,int val)
{
    it ir=split(r+1),il=split(l);
    s.erase(il,ir);
    s.insert(node(l,r,val));
}

void add(int l,int r,int val)
```

```
{
    it ir=split(r+1),il=split(l);
    for(;il!=ir;il++)
        il->v+=val;
}

int rk(int l,int r,int k)
{
    vector<pair<int,int>> v;
    it ir=split(r+1),il=split(l);
    for(;il!=ir;il++)
        v.emplace_back(il->v,il->r-il->l+1);
    sort(v.begin(),v.end());
    for(int i=0;i<v.size();++i)
    {
        k-=v[i].second;
        if(k<=0) return v[i].first;
    }
    return -1; //can't find
}

int sum(int l,int r,int ex,int mod)
{
    it ir=split(r+1),il=split(l);
    int res=0;
    for(;il!=ir;il++)
        res=(res+qpow(il->v,ex,mod)*(il->r-il->l+1)%
            mod)%mod;
    return res;
}

inline int read(){
    char ch=getchar();int s=0,w=1;
    while(ch<48||ch>57){if(ch=='-')w=-1;ch=getchar();}
    while(ch>=48&&ch<=57){s=(s<<1)+(s<<3)+ch-48;ch=
        getchar();}
    return s*w;
}
inline void write(int x){
    if(x<0)putchar('-'),x=-x;
    if(x>9)write(x/10);
    putchar(x%10+48);
}
//Fast I/O

int n,m,seed,vmax,a[N];
int rnd()
{
    int ret=seed;
    seed=(seed*7+13)%1000000007;
    return ret;
}

signed main()
{
    n=read(),m=read(),seed=read(),vmax=read();
    for(int i=1;i<=n;++i)
    {
        a[i]=(rnd()%vmax)+1;
        s.insert(node(i,i,a[i]));
    }
    for(int i=1;i<=m;++i)
    {
        int op=(rnd()%4)+1;
        int l=(rnd()%n)+1;
        int r=(rnd()%n)+1;
        if(l>r) swap(l,r);
        int x,y;
        if(op==3) x=(rnd()%(r-l+1))+1;
        else x=(rnd()%vmax)+1;
        if(op==4) y=(rnd()%vmax)+1;
        switch(op)
        {
            case 1:
                add(l,r,x);break;
            case 2:
                assign(l,r,x);break;
            case 3:
                write(rk(l,r,x)),puts("");break;
            case 4:
                write(sum(l,r,x,y)),puts("");break;
        }

    }
    return 0;
}
```

4.14 跳舞链

5 动态规划

5.1 SOS

```

/*
    (1<<N)-1-a[i]是a[i]的补集
*/

for(int i=0;i<(1<<N);i++) dp[i]=a[i];
for(int i=0;i<N;i++)
    for(int mask=0;mask<(1<<N);mask++)
        if(mask&(1<<i))
            dp[mask]+=dp[mask^(1<<i)];
```

5.2 动态 DP

5.3 插头 DP

6 数学

6.1 三分

```
//答案都取r 浮点数可以取(l+r)/2
//浮点数极小值
while(l+eps<r)
{
    double lm=(l+r)/2,rm=(lm+r)/2;
    if(judge(lm)>judge(rm)) l=lm;
    else r=rm;
}

//整数极小值
while(l+1<r)
{
    int lm=(l+r)>>1,rm=(lm+r)>>1;
    if(judge(lm)>judge(rm)) l=lm;
    else r=rm;
```

```
}

//浮点数极大值
while(l+eps<r)
{
    double lm=(l+r)/2,rm=(lm+r)/2;
    if(judge(lm)>judge(rm)) r=rm;
    else l=lm;
}

//整数极大值
while(l+1<r)
{
    int lm=(l+r)>>1,rm=(lm+r)>>1;
    if(judge(lm)>judge(rm)) r=rm;
    else l=lm;
}
```

6.2 矩阵类

6.3 质数筛

6.3.1 埃筛

6.3.2 线筛

6.4 质数判定

6.4.1 Miller Rabin

$$0 < x < p, p \quad x^2 \equiv 1(\mod p) \Leftrightarrow x = 1\ x = p-1$$

$$0 < a < p, p \quad \Rightarrow a^{p-1} \equiv 1(\mod p)$$

判错概率不超过 $4^{-S}(S\quad)$，时间复杂度 $O\ S * logn$

```
//HDU2138多组输入判断n个数中几个是质数
#include <cstdio>
#include <cstdlib>
#include <ctime>
#include <random>
using namespace std;
typedef long long ll;

inline ll qpow(ll a, ll b, ll P){ //a^b%P
    ll ans=1;
    for(; b; b>>=1,a=a*a%P)
        if(b&1) ans=ans*a%P;
return ans;}

inline bool MR(int tc, ll n){ //用随机测试数据tc测n，返回n是质数?1:0
    ll u=n-1, t=0;
    while(!(u&1)) ++t, u>>=1; //把n-1表示为u*2^t，循环到u为奇数为止
    ll x=qpow(tc,u,n);
    if(x==1) return 1;
    for(int i=1; i<=t; ++i,x=x*x%n)
        if(x!=n-1&&x!=1&&x*x%n==1) return 0;
    return x==1;
}

mt19937_64 rnd(time(0)); //c++11的std的魔法，下文可用rnd()生成[0,2^64)

inline bool isPrime(ll n, int S=10){ //对n测试S次，返回n是质数?1:0
    if(n==2) return 1;
    if(n<2 || !(n&1)) return 0;
// srand(time(0));
    while(S--)
//  if(!MR((ll)rand()*rand()*rand()%(n-1)+1,n)) return 0;
        if(!MR(rnd()%(n-1)+1,n)) return 0;
    return 1;
}

inline void solve(int N){
    int cnt=0;
    while(N--){
        ll n; scanf("%lld",&n);
        if(isPrime(n)) ++cnt;
    }
    printf("%d\n",cnt);
}

int main(int argc, char** argv){
    int _; while(~scanf("%d",&_))
        solve(_);
    return 0;
}
```

6.5 质因数分解

6.5.1 Pollard-Rho

Pollard 设计的求大数因子方法，因为随机找的数会形成环形，因此常称为 ρ 算法大致原理是随机找几个数求差，让差与大数 N 求 gcd，据说复杂度约 $O(N^{1/4})$ 随机找数的方法见 *rho* 函数。若要求质因子，一般需要用一下 $MillerRabin$ 判素数

```
//洛谷P4718求最大质因子
#include <cstdio>
#include <cstdlib>
#include <ctime>
#include <random>
#include <algorithm>
using namespace std;
typedef long long ll;
typedef unsigned long long ull;
typedef long double lb;

ll m, ans; //对每个m，将其最大质因子存进ans

mt19937_64 rnd(time(0)); //c++11的std的魔法，下文可用rnd()生成[0,2^64)

ll read(){
    ll k=0; int f=1;
    char c=getchar();
    while(!isdigit(c)){ if(c=='-') f=-1; c=getchar(); }
    while(isdigit(c)) k= k*10 + c-48, c=getchar();
    return k*f;
}

inline ll Abs(ll x){ return x<0 ? -x : x;} //取绝对值

inline ll qmul(ull x, ull y, ll p){ //O(1)x*y%p
    return (x*y - (ull)((lb)x/p*y)*p + p)%p;
}

inline ll qpow(ll x, ll y, ll p){ //x^y%p
    ll res=1;
```

```
    for(; y; y>>=1, x=qmul(x,x,p))
        if(y&1) res=qmul(res,x,p);
    return res;
}

inline bool MR(int tc, ll p){ //miller rabin判质数，用tc检测p是不是质数
    if(qpow(tc,p-1,p)!=1) return 0; //费马小定理
    ll y=p-1, z;
    while(!(y&1)){ //二次探测
        y>>=1;
        z=qpow(tc,y,p);
        if(z!=1 && z!=p-1) return 0;
        if(z==p-1) return 1;
    }
    return 1;
}

inline bool isPrime(ll x){ //用5个小质数做MR测试，返回n是质数?1:0
    if(x<2) return 0;
    if(x==2 || x==3 || x==5 || x==7 || x==43) return 1;
    return MR(2,x) && MR(3,x) && MR(5,x) && MR(7,x) && MR(43,x);
}

inline ll rho(ll p){ //玄学求出p的非平凡因子
    ll x,y,s,c; //s用来存 (y-x) 的乘积
    for(;;){ //求出一个因子来
        y=x=rnd()%p;
        c=rnd()%p;
        s=1;
        int i=0, I=1;
        while(++i){ //开始玄学倍增生成
            x=(qmul(x,x,p)+c)%p; //以平方再+c的方式生成下一个x
            s=qmul(s,Abs(y-x),p); //将每一次的 (y-x) 都累乘起来
            if(x==y || !s) break; //换下一组，当s=0时，继续下去是没意义的
            if(!(i%127) || i==I){ //每127次求一次gcd，以及按j倍增的求gcd
                ll g=__gcd(s,p);
                if(g>1) return g;
                if(i==I) y=x, I<<=1; //维护倍增正确性，并判环
            }
        }
    }
}

inline void PR(ll p){ //找p的最大质因子，存进全局变量ans
    if(p<=ans) return ; //最优性剪枝
    if(isPrime(p)) return ans=p, void();
    ll pi=rho(p); //玄学求出任一因子p_i
    while(p%pi==0) p/=pi;
    return PR(pi), PR(p); //向下分治
}

int main(){
    int n=read();
    for(int i=0; i<n; ++i){
        ans=1;
        PR( m=read() );
        if(ans==m) puts("Prime");
        else printf("%lld\n",ans);
    }
    return 0;
}
```

6.6 逆元

6.6.1 EX-GCD 求逆元

a P 互质是 a 在模 P 时有乘法逆元的充要条件用扩展欧几里得算法解出的 $ax+by=1$ 的解 x 即为 a 在模 P 时的乘法逆元顺带一提，有乘法逆元时，根据欧拉定理，逆元一定为 $a^{\phi(P)-1} \mod P$

```

//洛谷P1082
#include <bits/stdc++.h>
typedef long long ll;

ll a,P,x,y;
inline ll exgcd(ll a, ll b, ll &x, ll &y){
    if(!b) return x=1, y=0, a;
    ll g=exgcd(b, a%b, x, y);
    ll z=x; x=y; y=z-a/b*y;
    return g;
}
inline ll exinv(int a,int P){ //用exgcd求a模P的逆元，无解时返回-1
    ll x,y;
    if(exgcd(a,P,x,y)!=1) return -1;
    else return (x%P+P)%P;
}

int main(){
    scanf("%lld%lld",&a,&P);
    printf("%lld",exinv(a,P));
    return 0;
}
```

6.6.2 线性筛逆元

6.6.3 阶乘逆元

6.7 欧拉函数

6.7.1 欧拉线筛

6.7.2 求单个数的欧拉函数

6.7.3 欧拉降幂

费马小定理：p 为质数时，

$$a^{p-1} \equiv 1 \pmod p$$

欧拉定理：a,p **互质时**，

$$a^{\phi(p)} \equiv 1 \pmod p$$

扩展欧拉定理：a,p **不互质时**，

$$a^b \equiv \begin{cases} a^{b \bmod \phi(p)+\phi(p)} & b \geq \phi(p) \\ a^b & 0 \leq b < \phi(p) \end{cases} \pmod p$$

```cpp
//SP10050:用扩展欧拉定理求乘方塔a^a^a……(b个a)的后九位数
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

struct ST{
    int v;
    bool ge; //大于等于模数与否
    ST(int v=0,bool g=0): v(v), ge(g) {}
};

ST qpow(ll a,ll b,int p){ //快速幂过程中取模过p iff 返回值.ge==1
    ll ans=1ll;
    bool ge=0;
    while(b){
        if(b&1)
            ans*=a,
            ge |= ans>=p,
            ans%=p;
        b>>=1;
        if(!b) break; //防止被没有乘到的a更新ge
        a*=a;
        ge |= a>=p, //注意ans*取余后的a可能更新不了ge，在这也要更新
        a%=p;
    }
    return ST(ans,ge);
}

map<int,int>eu;
ll euler(ll n){ //欧拉函数值
    if(eu[n]) return eu[n];
    ll n0=n, ans=n, ed=sqrt(n);
    for(int i=2; i<=ed; ++i)
        if(n%i==0){
            ans-=ans/i;
            while(n%i==0) n/=i;
        }
    if(n>1) ans-=ans/n;
    return eu[n0]=ans;
}

ST tower(ll a,ll b,int p){ //计算b层a取余p的值
    if(p==1) return ST(0,1); //特判取模1的特殊情况
    if(a==1) return ST(1,0); //特判不取模1但底为1的特殊情况
    if(b==1) return a<p? ST(a,0): ST(a%p,1); //递归终点
    int phip=euler(p);
    ST ans=tower(a,b-1,phip); //递归计算取余phip后的指数
    if(ans.ge) ans.v+=phip; //扩展欧拉定理
    return qpow(a, ans.v, p);
}

void solve(){
    ll a,b; scanf("%lld%lld",&a,&b);
    if(b==0) return printf("%d\n",1), void(0);
    else if(a==0) return printf("%d\n", b%2? 0: 1), void(0);
    ST ans=tower(a,b,1000000000);
    if(ans.ge) printf("...%09d\n",ans.v);
    else printf("%d\n",ans.v);
}

int main(int argc, char** argv) {
    int _; scanf("%d",&_); while(_--)
        solve();
    return 0;
}
```

6.7.4 一般积性函数求法

6.8 EX-GCD

6.9 同余方程组

6.9.1 CRT

6.9.2 EXCRT

求解 $x \mod m_i = a_i$ 方程组，其中 m_i 不一定为质数和 CRT 本身没啥关系，是用数学归纳法求解齐次同余方程组的

```cpp
//洛谷P4777
#include <bits/stdc++.h>
typedef long long ll;
const int MN = 3e5 + 5;

ll a[MN],m[MN];

inline ll exgcd(ll a, ll b, ll &x, ll &y){
    if(!b) return x=1, y=0, a;
    ll g=exgcd(b, a%b, x, y);
    ll z=x; x=y; y=z-a/b*y;
return g;}

ll smul(ll a,ll b,ll p){ //记得传参时先给ab取余一发p
    ll ans=0;
    for(;b;b>>=1){
        if(b&1) ans= (ans+a)%p;
        a= (a<<1)%p;}
return ans;}

//ll qmul(ll a,ll b,ll p){ //玄学高精度乘法，备用，可能可以代替上一个
// a%=p, b%=p;
// ll t=(long double)a*b/p;
// ll ans=a*b-t*p;
//return ans<0? ans+p: ans;}

ll excrt(int n){ //解[0,n)
    ll X, Y, M=m[0], ans=a[0];
    for(int i=1; i<n; ++i){
        ll A=M, B=m[i];
        ll c=(a[i]-ans%B+B)%B; //新同余方程的右部
        ll g=exgcd(A,B,X,Y);
        if(c%g!=0) return -1;
        X=smul(X,c/g,B/g);
        ans+=X*M;
        M*=B/g;
        ans=(ans%M+M)%M;}
return (ans%M+M)%M;}

int main(){
    int n; scanf("%d",&n);
    for(int i=0; i<n; ++i) scanf("%lld%lld",m+i,a+i);
```

```
    printf("%lld",excrt(n));
    return 0;
}
```

6.10 N 次剩余

6.10.1 模奇质数的 2 次剩余

```
/*
求解x^2 % P = n，当P为奇质数时，可用类似虚数平方的方法求解，
类似实数域一元二次方程，答案可能为两不同解，两相同解或无实数解，详见solve()
*/
//洛谷P5491求二次剩余
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

int P; //模数，为了避免传参而放在全局变量

inline ll qpow(ll a,int b){ //a^b%P
    ll ans=1;
    for(; b; b>>=1,a=a*a%P)
        if(b&1) ans=ans*a%P;
return ans;}

ll t,tt; //tt为t的平方。注意！！！复数类中每次乘法都要用到tt！！！
struct CP{ //求解二次剩余专用的魔改复数类
    ll x,y;
    CP(ll x=0,ll y=0):x(x),y(y){}
    CP operator*=(const CP&r){ //乘以r，模数P为全局变量
        return *this = CP((x*r.x%P+y*r.y%P*tt%P)%P,(y*r.x%P+x*r.y%P)%P);
    }
    CP qpow(int n){ //n次幂，模数P为全局变量
        CP rt = CP(1,0);
        for(; n; n>>=1){
            if(n&1) rt *= *this;
            *this *= *this;
        }
        return *this = rt;
    }
};

int cipolla(int n){ //求x*x%P=n的一个解，P是奇质数，无解时return-1
    if(n==0) return 0;
    if(qpow(n,(P-1)>>1)==P-1) return -1; //无解
    srand(time(0)); //初始化随机数种子
    for(;;){ //随机找到一个满足break条件的t即可
        t=rand()%P;
        tt=(t*t%P-n+P)%P;
        if(qpow(tt,(P-1)>>1)==P-1) break;
    }
    CP rt=CP(t,1).qpow((P+1)>>1);
    return rt.x;
}

inline void solve(){
    int n; scanf("%d%d",&n,&P); //注意P是全局变量，之后就不传参了
    int ans=cipolla(n);
    int ans2=(P-ans)%P; //此处取模纯粹是为了避免ans为0时ans2为P
    if(ans==-1) return puts("Hola!"), void();
    if(ans2<ans) swap(ans,ans2);
    if(ans!=ans2) printf("%d %d\n",ans,ans2);
    else printf("%d\n",ans);
}

int main(int argc, char** argv){
    int _; scanf("%d",&_); while(_--)
        solve();
    return 0;
}
```

6.10.2 N 次剩余

6.11 数论分块

6.12 高斯消元

6.12.1 普通消元

6.12.2 异或方程组消元

6.13 莫比乌斯反演

6.13.1 莫比乌斯函数

6.13.2 杜教筛

6.13.3 洲阁筛

6.13.4 min25 筛

6.14 BSGS

求解 $a^x \bmod p = b$：当**模数 p 是质数**时可直接使用 $baby_step_giant_step$，设 p 的正平方根为 s，BSGS 的原理是将 a 的 p 次幂分块成 s 个，

预处理好大约 $s+1$ 个 $b*a^i(i\in[0,s])$，再用遍历各分块，查找是否存在 a^{s*i} 等于预处理的 $b*a^i$；当**模数 p 不是质数**时，在 EXBSGS 先约分 b,p 到 a,p 互质，再调用 BSGS。

```
//洛谷P4195
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

inline ll exgcd(ll a, ll b, ll &x, ll &y){
    if(!b) return x=1, y=0, a;
    ll g=exgcd(b, a%b, x, y);
    ll z=x; x=y; y=z-a/b*y;
return g;}

inline ll exinv(int a,int P){ //用exgcd求a模P的逆元，无解时返回-1
    ll x,y;
    if(exgcd(a,P,x,y)!=1) return -1;
    else return (x%P+P)%P;
}

inline ll qpow(ll a,int b,int P){ //a^b%P
    ll ans=1;
    for(;b;b>>=1,a=a*a%P)
        if(b&1) ans=ans*a%P;
return ans;}

```

```cpp
int bsgs(int a, int b, int p){ //a^x=b%p的最小非负x, 无解时返回-1
    unordered_map<int,int>hsh;
    a%=p, b%=p;
    int s=sqrt(p)+1, bai=b;
    for(int i=0; i<s; ++i) hsh[bai]=i, bai=ll(bai)*a%p;
    int as=qpow(a,s,p);
    if(as==0) return b==0? 1: -1;
    int asi=1;
    for(int i=0; i<=s; ++i){
        int t=hsh.find(asi)==hsh.end()? -1 : hsh[asi];
        if(t>=0&&s*i>=t) return s*i-t;
        asi=ll(asi)*as%p;
    }
    return -1;
}

int exbsgs(int a, int b, int p){ //a^x=b%p的最小非负x, 无解时返回-1
    a%=p, b%=p;
    if(b==1) return 0;
    int k=1, cnt=0, d;
    while((d=__gcd(a,p))!=1){
        if(b%d) return -1;
        p/=d, b/=d, k=ll(a)/d*k%p, ++cnt;
        if(b==k) return cnt;
    }
    int ans=bsgs(a,ll(b)*exinv(k,p)%p,p);
    if(ans>=0) ans+=cnt;
    return ans;
}

inline int solve(){ //模板题中以全0为输入结尾
    int a,b,p;
    cin>>a>>p>>b;
    if(!a && !b && !p) return 0;
    int x=exbsgs(a,b,p);
    if(x==-1) cout<<"No Solution\n";
    else cout<<x<<'\n';
    return 1;
}

int main(int argc, char** argv){
    ios::sync_with_stdio(0);
    while(solve()) ;
    return 0;
}
```

6.15 FFT

6.16 FFT

```cpp
/*
大致原理：O(NlogN)地将一个N次多项式系数表达式转化成关于(1的N次复数根)的N个点值
对两个多项式的点值做乘法，得到新的点值表达式，再O(NlogN)地转回系数表达式
利用R数组和“蝴蝶定理”可以将递归FFT过程转为循环过程，优化复杂度的常数
写得很乱，欢迎有空的同学来更新个好看点的
*/
//洛谷P3803裸的多项式乘积
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef double db;

const int MN = 3e6 + 5;
const db pi = acos(-1);

struct CP{
    db x,y; //实部，虚部
    CP (db x=0, db y=0): x(x), y(y) {}
    CP operator+(CP &t){ return CP(x+t.x, y+t.y); }
    CP operator-(CP &t){ return CP(x-t.x, y-t.y); }
    CP operator*(CP &t){ return CP(x*t.x - y*t.y, x*t.y + y*t.x); }
}a[MN],b[MN];

int read(){
    int k=0, f=1;
    char c=getchar();
    while(!isdigit(c)){ if(c=='-') f=-1; c=getchar(); }
    while(isdigit(c)) k= k*10 + c-48 , c=getchar();
    return k*f;
}
void write(int x){
    if(x<0){ putchar('-'); x=~(x-1); }
    int s[20],top=0;
    while(x){ s[++top]=x%10; x/=10; }
    if(!top) s[++top]=0;
    while(top) putchar(s[top--]+'0');
}

int N,M,n,ln; //左多项式次数，右多项式次数，fft次数及其位数
int R[MN]; //01串逆转后对应的下标
inline void calR(){ //通过NM初始化以上数据
    n=1;
    while(n<=N+M) n<<=1, ++ln;
    for(int i=1; i<n; ++i) R[i] = (R[i>>1] >>1) | ((i&1)<< ln-1);
}
inline void init(){
    N=read(); M=read();
    for(int i=0; i<=N; ++i) a[i].x=read();
    for(int i=0; i<=M; ++i) b[i].x=read();
    calR();
}
void fft(CP c[], int f=1){ //f取-1时是逆变换
    for(int i=0; i<n; ++i) if(i<R[i]) swap(c[i], c[R[i]]);
    for(int j=1; j<n; j<<=1){
        CP wn(cos(pi/j), f*sin(pi/j));
        for(int k=0; k<n; k+=(j<<1)){
            CP t(1, 0);
            for(int l=0; l<j; ++l){
                CP cl=c[k+l], cr=t*c[j+k+l];
                c[k+l]=cl+cr;
                c[j+k+l]=cl-cr;
                t=t*wn;
            }
        }
    }
```

```
}

int main(int argc, char** argv) {
// ios::sync_with_stdio(0);
    init();
    fft(a), fft(b);
    for(int i=0; i<=n; ++i) a[i]=a[i]*b[i];
    fft(a,-1);
    for(int i=0; i<=N+M; ++i) write(int(a[i].x/n +
        0.5)), putchar(' ');
    return 0;
}
```

6.17 DFT 次数优化 FFT

```
/*
大致原理：O(NlogN)地将一个N次多项式系数表达式转化成关于(1
    的N次复数根)的N个点值
对两个多项式的点值做乘法，得到新的点值表达式，再O(NlogN)
    地转回系数表达式
利用R数组和“蝴蝶定理”可以将递归FFT过程转为循环过程，优化
    复杂度的常数
写得很乱，欢迎有空的同学来更新个好看点的
*/
//洛谷P3803裸的多项式乘积，利用了(a+bi)^2 = (a^2-b^2) +
    2abi减少变换次数
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef double db;

const int MN = 3e6 + 5;
const db pi = acos(-1);

struct CP{
    db x,y; //实部，虚部
    CP (db x=0, db y=0): x(x), y(y) {}
    CP operator+(CP &t){ return CP(x+t.x, y+t.y); }
    CP operator-(CP &t){ return CP(x-t.x, y-t.y); }
    CP operator*(CP &t){ return CP(x*t.x - y*t.y, x*t.
        y + y*t.x); }
}a[MN];

int read(){
    int k=0, f=1;
    char c=getchar();
    while(!isdigit(c)){ if(c=='-') f=-1; c=getchar();
        }
    while(isdigit(c)) k= k*10 + c-48 , c=getchar();
    return k*f;
}
void write(int x){
    if(x<0){ putchar('-'); x=~(x-1); }
    int s[20],top=0;
    while(x){ s[++top]=x%10; x/=10; }
    if(!top) s[++top]=0;
    while(top) putchar(s[top--]+'0');
}

int N,M,n,ln; //左多项式次数，右多项式次数，fft次数及其
    位数
int R[MN]; //01串逆转后对应的下标
inline void calR(){ //通过NM初始化以上数据
    n=1;
    while(n<=N+M) n<<=1, ++ln;
    for(int i=1; i<n; ++i) R[i] = (R[i>>1] >>1) | ((i
        &1)<< ln-1);
}
inline void init(){
    N=read(); M=read();
    for(int i=0; i<=N; ++i) a[i].x=read();
    for(int i=0; i<=M; ++i) a[i].y=read();
    calR();
}
void fft(CP c[], int f=1){ //f取-1时是逆变换
    for(int i=0; i<n; ++i) if(i<R[i]) swap(c[i], c[R[i
        ]]);
    for(int j=1; j<n; j<<=1){
       CP wn(cos(pi/j), f*sin(pi/j));
       for(int k=0; k<n; k+=(j<<1)){
          CP t(1, 0);
          for(int l=0; l<j; ++l){
             CP cl=c[k+l], cr=t*c[j+k+l];
             c[k+l]=cl+cr;
             c[j+k+l]=cl-cr;
             t=t*wn;
          }
       }
    }
}

int main(int argc, char** argv) {
// ios::sync_with_stdio(0);
    init();
    fft(a);//, fft(b);
    for(int i=0; i<=n; ++i) a[i]=a[i]*a[i];
    fft(a,-1);
    for(int i=0; i<=N+M; ++i) write(int(a[i].y/n/2 +
        0.5)), putchar(' ');
    return 0;
}
```

6.18 FWT

6.19 NTT

6.20 NTT

```
/*
大致原理：令P=998244353，g=3，g的n次幂模P可得到P种值，称
    g为模P的原根
O(NlogN)地将一个N次多项式系数表达式转化成关于模P的原根的g
    的次幂的N个点值
对两个多项式的点值做乘法，得到新的点值表达式，再O(NlogN)
    地转回系数表达式
利用R数组和“蝴蝶定理”可以将递归NTT过程转为循环过程，优化
    复杂度的常数
写得很乱，欢迎有空的同学来更新个好看点的
顺带一提求逆和分治里用的是细节不同的另外两种实现的NTT，不
    过也都很乱
*/
//洛谷P3803裸的多项式乘积，在大数据下的表现可能反而不如
    FFT
#include <bits/stdc++.h>
```

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

const int MN = 3e6 + 5;
const int P = 998244353;

int qpow(ll a,int b){
    ll ans=1;
    for(;b;b>>=1){
        if(b&1)
            ans=ans*a%P;
        a=a*a%P;}
return ans;}

int read(){
    int k=0, f=1;
    char c=getchar();
    while(!isdigit(c)){ if(c=='-') f=-1; c=getchar(); }
    while(isdigit(c)) k= k*10 + c-48 , c=getchar();
    return k*f;
}
void write(int x){
    if(x<0){ putchar('-'); x=~(x-1); }
    int s[20],top=0;
    while(x){ s[++top]=x%10; x/=10; }
    if(!top) s[++top]=0;
    while(top) putchar(s[top--]+'0');
}

int a[MN],b[MN],c[MN];
int N,M,n,ln; //左多项式次数，右多项式次数，fft次数及其位数
int R[MN]; //01串逆转后对应的下标
inline void calR(){ //通过NM初始化以上数据
    n=1;
    while(n<=N+M) n<<=1, ++ln;
    for(int i=1; i<n; ++i) R[i] = (R[i>>1] >>1) | ((i&1)<< ln-1);
}
inline void init(){
    N=read(); M=read();
    for(int i=0; i<=N; ++i) a[i]=read();
    for(int i=0; i<=M; ++i) b[i]=read();
    calR();
}
void ntt(int c[], int f=1){ //f取-1时是逆变换
    for(int i=0; i<n; ++i) if(i<R[i]) swap(c[i], c[R[i]]);
    for(int j=1; j<n; j<<=1){
        int j2=j<<1;
        int rt=qpow(3,(P-1)/j2);
        if(f==-1) rt=qpow(rt, P-2); //逆变换时除以rt，即乘以rt逆元
        for(int k=0; k<n; k+=j2){
            int t=1;
            for(int l=0; l<j; ++l){
                int tl=c[k+l], tr=ll(t)*c[j+k+l]%P;
                c[k+l]=(tl+tr)%P;
                c[j+k+l]=((tl-tr)%P+P)%P;
                t=ll(t)*rt%P;
            }
        }
    }
}

int main(int argc, char** argv) {
// ios::sync_with_stdio(0);
    init();
    ntt(a), ntt(b);
    for(int i=0; i<=n; ++i) c[i]=ll(a[i])*b[i]%P;
    ntt(c,-1);
    ll mi=qpow(n,P-2);
    for(int i=0; i<=N+M; ++i) write((c[i]*mi)%P), putchar(' ');
    return 0;
}
```

6.21 NTT 分治求卷积

```cpp
//大致原理是cdq分治套NTT求f_i = f和g的前i项卷积
//洛谷P4721求f_i = \sigma_{j=1}^i f_{i-j} * g_j
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

const int P = 998244353;
const int G = 3;
const int MN = 2e5 + 5;

inline int read(){
    int k=0, f=1;
    char c=getchar();
    while(!isdigit(c)){ if(c=='-') f=-1; c=getchar(); }
    while(isdigit(c)) k=k*10+c-48, c=getchar();
    return k*f;}

int qpow(ll a, int b){
    ll ans=1ll;
    for(;b;b>>=1){
        if(b&1)
            ans=ans*a%P;
        a=a*a%P;}
    return ans;}

inline int inv(ll value){ return qpow(value, P-2); }

namespace NTT{
    int R[MN]; //01串逆转后对应的下标
    void calcR(int len){ //计算len次多项式的R数组
        int loglen = R[0] = 0;
        while(1<<loglen < len) ++loglen;
        for(int i=1; i<len; ++i) R[i] = (R[i>>1] >>1) | ((i&1)<< loglen-1);
    }
    void ntt(int *a, int n, int f=1){ //f取-1时是逆变换
        for(int i=0; i<n; ++i) if(R[i] < i) swap(a[i], a[R[i]]);
        int baseW = qpow(G, (P-1) / n);
        if(f==-1) baseW = inv(baseW);
        for(int len = 2; len <= n; len <<= 1){
            int mid = len >>1;
            int wn = qpow(baseW, n / len);
            for(int *pos = a; pos != a+n; pos += len){
                int w = 1;
                for(int i=0; i<mid; ++i) {
                    int x = pos[i], y = (ll)pos[mid + i] * w % P;
```

```
                pos[i] = ((ll)x + y) % P;
                pos[mid + i] = ((ll)x - y + P) % P;
                w = (ll)w * wn % P;
            }
        }
    }
}
void mult(int *a, int *b, int len){ //封装好的len次多项式乘法，答案存进a
    calcR(len);
    ntt(a,len);
    ntt(b,len);
    for(int i=0; i<len; ++i) a[i] = (ll)a[i] * b[i] % P;
    ntt(a,len,-1);
    int x = inv(len);
    for(int i=0; i<len; ++i) a[i] = (ll)a[i] * x % P;
}
}
using namespace NTT;

int f[MN],g[MN],A[MN],B[MN]; //输出输入及其副本
int n; //多项式长度

void cdq(int l, int r){ //分治[l,r]
    if(l==r) return;
    int mid = l+r >> 1, len = 1;
    while(len < r-l) len <<= 1;
    cdq(l,mid);
    for(int i=0; i<len; ++i) A[i] = 0, B[i] = 0;
    for(int i=0; i<=mid-l; ++i) A[i] = f[l+i]; //l开始的当前区间的左半f
    for(int i=0; i<r-l; ++i) B[i] = g[i+1]; //1开始的当前区间长度的g
    mult(A,B,len); //卷积求出左半端的贡献
    for(int i=mid+1; i<=r; ++i) f[i] = (f[i] + A[i-l-1]) % P;
    cdq(mid+1,r);
}

int main(){
    n = read();
    for(int i=1; i<n; ++i) g[i] = read();
    f[0] = 1; //题目给的初值
    cdq(0,n-1);
    for(int i=0; i<n; ++i) printf("%d ",f[i]);
    return 0;
}
```

6.22 NTT 求多项式逆

```
//洛谷P4238倍增求模x^n的多项式，即相乘后忽略n次及更高次只剩1
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

const int P = 998244353;
const int G = 3;
const int MN = 4e5 + 5;

inline int read(){
    int k=0, f=1;
    char c=getchar();
    while(!isdigit(c)){ if(c=='-') f=-1; c=getchar(); }
    while(isdigit(c)) k=k*10+c-48, c=getchar();
    return k*f;}

int qpow(ll a, int b){
    ll ans=1ll;
    for(;b;b>>=1){
        if(b&1)
            ans=ans*a%P;
        a=a*a%P;}
    return ans;}

inline int inv(ll value){ return qpow(value, P-2); }

//namespace NTT{
    int R[MN]; //01串逆转后对应的下标
    void calcR(int len){ //计算len次多项式的R数组
        int loglen = R[0] = 0;
        while(1<<loglen < len) ++loglen;
        for(int i=1; i<len; ++i) R[i] = (R[i>>1] >>1) | ((i&1)<< loglen-1);
    }
    void ntt(int *a, int n, int f=1){ //f取-1时是逆变换
        for(int i=0; i<n; ++i) if(R[i] < i) swap(a[i], a[R[i]]);
        int baseW = qpow(G, (P-1) / n);
        for(int len = 2; len <= n; len <<= 1){
            int mid = len >>1;
            int wn = qpow(baseW, n / len);
            for(int *pos = a; pos != a+n; pos += len){
                int w = 1;
                for(int i=0; i<mid; ++i){
                    int x = pos[i], y = (ll)pos[mid + i] * w % P;
                    pos[i] = ((ll)x + y) % P;
                    pos[mid + i] = ((ll)x - y + P) % P;
                    w = (ll)w * wn % P;
                }
            }
        }
        if(f==-1){
            int miv = inv( ll(n) );
            reverse(a+1,a+n);
            for(int i=0; i<n; ++i) a[i] = ll(a[i]) * miv % P;
        }
    }
    void mult(int *a, int *b, int len){ //len次多项式乘法，答案存进a
        calcR(len);
        ntt(a,len);
        ntt(b,len);
        for(int i=0; i<len; ++i) a[i] = (ll)a[i] * b[i] % P;
        ntt(a,len,-1);
        int x = inv(len);
        for(int i=0; i<len; ++i) a[i] = (ll)a[i] * x % P;
    }
//}
//using namespace NTT;

int f[MN],g[MN],A[MN],B[MN]; //输出输入及其副本
```

```
int n; //多项式长度

void inv(int *a, int *b, int len){ //封装好的len次多项式求逆，答案存进b
    if(len==1) return b[0] = inv(ll(a[0])), void();
    inv(a, b, len+1 >>1);
    int LEN = 1; //扩展到2的整数次幂的len
    while(LEN < (len<<1)) LEN <<= 1;
    calcR(LEN);
    for(int i=0; i<len; ++i) A[i] = a[i];
    for(int i=len; i<LEN; ++i) A[i] = 0;
    ntt(A,LEN), ntt(b,LEN);
    for(int i=0; i<LEN; ++i) b[i] = ll(2 - ll(A[i]) * b[i] % P + P) % P * b[i] % P;
    ntt(b,LEN,-1);
    for(int i=len; i<LEN; ++i) b[i] = 0;
}

int main(){
    n = read();
    for(int i=0; i<n; ++i) g[i] = read();
    inv(g,f,n);
    for(int i=0; i<n; ++i) printf("%d ",f[i]);
    return 0;
}
```

6.23 数值计算

6.23.1 辛普森

6.23.2 自适应辛普森

```
#include <bits/stdc++.h>
using namespace std;
const double eps=1e-12;

/*
    调用 asr(l,r,simpson(l,r))
*/

inline double f(double x)
{
    return x; //被积函数
}

double simpson(double l,double r)
{
    double mid=(l+r)/2;
    return (f(l)+4*f(mid)+f(r))*(r-l)/6;
}

double asr(double l,double r,double ans)
{
    double mid=(l+r)/2;
    double l1=simpson(l,mid),r1=simpson(mid,r);
    if(fabs(l1+r1-ans)<eps) return l1+r1;
    return asr(l,mid,l1)+asr(mid,r,r1);
}

int main()
{

    return 0;
}
```

6.24 康拓展开

```
//contor展开
int bit[maxn];
void add(int x,int y)
{
    for(;x<=n;x+=lowbit(x)) bit[x]+=y;
}

int sum(int x)
{
    int res=0;
    for(;x>0;x-=lowbit(x)) res+=bit[x];
    return res;
}

int contor(vector& p)
{
    int ans=0;
    rre(i,p.size()-1,0)
    {
        add(p[i],1);
        int cnt=sum(p[i]-1);
        ans=(ans+cnt*fac[p.size()-i-1]%MOD)%MOD;
    }
    return ans+1;
}

//逆contor展开
int k,s[50005];
int tr[200005];

void up(int p) {tr[p]=tr[ll(p)]+tr[rr(p)];}

void build(int p=1,int l=1,int r=k)
{
    if(l==r)
    {
        tr[p]=1;
        return;
    }
    int m=mm(l,r);
    build(ll(p),l,m);
    build(rr(p),m+1,r);
    up(p);
}

int ask(int cnt,int p=1,int l=1,int r=k)
{
    if(l==r)
    {
        tr[p]=0;
        return l;
    }
    int m=mm(l,r);
    int ans=-1;
    if(cnt<=tr[ll(p)]) ans=ask(cnt,ll(p),l,m);
    else ans=ask(cnt-tr[ll(p)],rr(p),m+1,r);
    up(p);
    return ans;
}

vei recontor()
{
    vei v(k+1);
```

```
    re(i,1,k) v[i]=ask(s[i]+1);
    return v;
}
```

6.25 卢卡斯定理

6.25.1 Lucas(循环或递归实现)

$$C_N^m \equiv C_{N \bmod p}^{m \bmod p} * C_{N/p}^{m/p} (\bmod p)$$

$$\binom{N}{m} \equiv \binom{N \bmod p}{m \bmod p} \cdot \binom{N/p}{m/p} (\bmod p)$$

可理解为将 N 和 m 表示为 p 进制数（形如 $\Sigma N_i p^i$），对每一位的 N_i 和 m_i 分别求组合数，再累乘，注意此处的 p **必须为质数**

```
//洛谷P3807
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int MN = 5e6 + 5;

inline ll qpow(ll a,ll b,int P){ //a^b%P
    ll ans=1;
    for(;b;b>>=1,a=a*a%P)
        if(b&1) ans=ans*a%P;
return ans;}

ll fct[MN],fi[MN]; //阶乘及其逆元
inline void init(int k,int P){ //打表模P的[1,k]阶乘及其逆元
    fct[0]=1;
    for(int i=1; i<=k; ++i) fct[i]=fct[i-1]*i%P;
    if(k<P){
        fi[k]=qpow(fct[k],P-2,P);
        for(int i=k; i>=1; --i) fi[i-1]=fi[i]*i%P;
    }else{ //k阶乘为0，会把所有逆元都变成0，应从P-1开始
        fi[P-1]=qpow(fct[P-1],P-2,P);
        for(int i=P-1; i>=1; --i) fi[i-1]=fi[i]*i%P;
    }
}

inline int C(int N,int m,int P){ //C_N^m % P
    if(m>N) return 0;
    return fct[N]*fi[m]%P*fi[N-m]%P;
}

//ll lucas(int N,int m,int P){ //递归求C_N^m % P
// if(!m) return 1;
// return C(N%P,m%P,P)*lucas(N/P,m/P,P)%P;
//}

int lucas(int N,int m,int P){ //循环求C_N^m % P
    ll rt=1;
    while(N&&m)
        (rt*=C(N%P,m%P,P))%=P,
        N/=P, m/=P;
    return rt;
}

void solve(){
    int n,m,p; scanf("%d%d%d",&n,&m,&p);
    init(n+m,p);
    printf("%lld\n",lucas(n+m,m,p));
}

int main(int argc, char** argv){
    int _; scanf("%d",&_); while(_--)
        solve();
    return 0;
}
```

6.25.2 EXLucas(分块实现)

模数 P 不是质数时不能使用卢卡斯定理计算组合数!
可对 P 做质因子分解，对各质因子分别求解组合数，得到同余方程组，再用 CRT 求解。
具体方法是将 P 分解成 Σp^k 的形式，再计算模 p^k 意义的阶乘。
由于 $n > p$ 时暴力算 $n!\mod p$ 为 0，因此计算阶乘时需先不断递归分块给阶乘除以质因子 p，算完阶乘后再把除掉的 p 乘回去，才能计算出模 p^k 意义的组合数。

```
//洛谷P4720
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

inline ll qpow(ll a,ll b,int P){ //a^b%P，此题中b可能爆int
    ll ans=1;
    for(;b;b>>=1,a=a*a%P)
        if(b&1) ans=ans*a%P;
return ans;}

inline ll exgcd(ll a, ll b, ll &x, ll &y){
    if(!b) return x=1, y=0, a;
    ll g=exgcd(b, a%b, x, y);
    ll z=x; x=y; y=z-a/b*y;
return g;}

inline ll exinv(int a,int P){ //用exgcd求a模P的逆元
    ll x,y;
    if(exgcd(a,P,x,y)!=1) return -1;
    else return (x%P+P)%P;
}

inline int g(ll n,int p){ //n!中质因子p的次数
    if(n<p) return 0;
    return n/p+g(n/p,p);
}

int f(ll n,int p,int pk){ //n!/(p^x) % pk，其中x=g(n,p)
    if(n==0) return 1;
    ll s=1, s2=1; //<=pk的分块乘积，>pk的块外乘积
    for(ll i=1; i<=pk; ++i)
        if(i%p) s=s*i%pk;
    s=qpow(s,n/pk,pk);
    for(ll i=n/pk*pk; i<=n; ++i)
        if(i%p) s2=i%pk*s2%pk;
    return f(n/p,p,pk)*s%pk*s2%pk;
}

inline ll c(ll N,ll m,int p,int pk){ //C^m_N % (p^k)
    ll rt=f(N,p,pk);
    (rt*=qpow(f(m,p,pk),pk/p*(p-1)-1,pk))%=pk;
// (rt*=exinv(f(m,p,pk),pk))%=pk;
    (rt*=qpow(f(N-m,p,pk),pk/p*(p-1)-1,pk))%=pk;
```

```
// (rt*=exinv(f(N-m,p,pk),pk))%=pk;
    (rt*=qpow(p,g(N,p)-g(m,p)-g(N-m,p),pk))%=pk;
    return rt;
}

inline ll crt(ll ai,int p,int pk,int P){ //x%(pi^ki)=ai, pk乘积=P
    return ai*(P/pk)%P*exinv(P/pk,pk)%P;
// return ai*(P/pk)%P*qpow(P/pk,pk/p*(p-1)-1,pk)%P;
}

int exlucas(ll N,ll m,int P){ //C^m_N % P
    ll rt=0, P2=P;
    int ed=sqrt(P)+1;
    for(int p=2; p<=ed; ++p){
        int pk=1;
        while(P2%p==0) pk*=p, P2/=p;
        if(pk>1) (rt+=crt(c(N,m,p,pk),p,pk,P))%=P;
    }
    if(P2>1) (rt+=crt(c(N,m,P2,P2),P2,P2,P))%=P;
    return rt;
}

int main(int argc, char** argv){
    ios::sync_with_stdio(0);
    ll N,m,P; cin>>N>>m>>P;
    cout<<exlucas(N,m,P);
    return 0;
}
```

6.26 博弈论

6.26.1 SG 函数

```
#include <bits/stdc++.h>
//Gym 101128 G
using namespace std;
const int N=1005;

int p,k;
vector<int> v(N);
int sg[N];
bool vis[N];

void gao(int n)
{
    memset(sg,0,sizeof(sg));
    for(int i=1;i<=n;++i)//由小到大枚举
    {
        memset(vis,0,sizeof(vis));
        for(int j=0;j<=min(i,k);++j)
        {
            if(i==j) continue;
            int tmp=i-j-v[i-j];//枚举子状态
            if(tmp>=0)
                vis[sg[tmp]]=1;
        }
        for(int j=0;;j++)
            if(!vis[j])
            {
                sg[i]=j;
                break;
            }
    }
}

int main()
{
    int ans=0;
    scanf("%d%d",&p,&k);
    for(int i=1,n;i<=p;++i)
    {
        scanf("%d",&n);
        for(int j=1;j<=n;++j)
            scanf("%d",&v[j]);
        gao(n);
        ans^=sg[n];//多堆求每一堆的异或值即可
    }
    if(ans==0) puts("Bob will win.");//先手必败
    else puts("Alice can win.");
    return 0;
}
```

7 其他

7.1 快读快写

```
inline int read()
{
    char ch=getchar();int s=0,w=1;
    while(ch<48||ch>57){if(ch=='-')w=-1;ch=getchar();}
    while(ch>=48&&ch<=57){s=(s<<1)+(s<<3)+ch-48;ch=getchar();}
    return s*w;
}

inline void write(int x)
{
    if(x<0)putchar('-'),x=-x;
    if(x>9)write(x/10);
    putchar(x%10+48);
}
```

7.2 高精度

```
#include <bits/stdc++.h>
#define MAXN 9999
#define MAXSIZE 1000
#define DLEN 4
using namespace std;

class BigNum
{
private:
    int a[MAXSIZE];
    int len;
public:
    BigNum(){ len = 1;memset(a,0,sizeof(a)); }
    void XD();
    BigNum(const int);
    BigNum(const long long int);
    BigNum(const char*);
    BigNum(const string &);
    BigNum(const BigNum &);
    BigNum &operator = (const BigNum &);
    BigNum &operator = (const int &);
    BigNum &operator = (const long long int &);
```

```
23
24      friend istream& operator >> (istream&, BigNum&);
25      friend ostream& operator << (ostream&, BigNum&);
26
27      template<typename T> BigNum operator << (const T
            &) const;
28      template<typename T> BigNum operator >> (const T
            &) const;
29
30      BigNum operator + (const BigNum &) const;
31      BigNum operator - (const BigNum &) const;
32      BigNum operator * (const BigNum &) const;
33      bool operator > (const BigNum& b)const;
34      bool operator < (const BigNum& b) const;
35      bool operator == (const BigNum& b) const;
36      template<typename T> BigNum operator / (const T &)
             const;
37      template<typename T> BigNum operator ^ (const T &)
             const;
38      template<typename T> T operator % (const T &)
            const;
39
40      template<typename T> BigNum operator + (const T& b
            ) const {BigNum t = b; t = *this + t; return
            t;}
41      template<typename T> BigNum operator - (const T& b
            ) const {BigNum t = b; t = *this - t; return
            t;}
42      template<typename T> BigNum operator * (const T& b
            ) const {BigNum t = b; t = (*this) * t;
            return t;}
43      template<typename T> bool operator < (const T& b)
            const {BigNum t = b; return ((*this) < t);}
44      template<typename T> bool operator > (const T& b)
            const {BigNum t = b; return ((*this) > t);}
45      template<typename T> bool operator == (const T& b)
             const {BigNum t = b; return ((*this) == t);}
46
47      bool operator <= (const BigNum& b) const {return
            (*this) < b || (*this) == b;}
48      bool operator >= (const BigNum& b) const {return
            (*this) > b || (*this) == b;}
49      bool operator != (const BigNum& b) const {return
            !((*this) == b);}
50
51      template<typename T> bool operator >= (const T& b)
             const {BigNum t = b; return !((*this) < t);}
52      template<typename T> bool operator <= (const T& b)
             const {BigNum t = b; return !((*this) > t);}
53      template<typename T> bool operator != (const T& b)
             const {BigNum t = b; return !((*this) == t)
            ;}
54
55      BigNum& operator += (const BigNum& b) {*this = *
            this + b; return *this;}
56      BigNum& operator -= (const BigNum& b) {*this = *
            this - b; return *this;}
57      BigNum& operator *= (const BigNum& b) {*this = *
            this * b; return *this;}
58      template<typename T> BigNum& operator /= (const T&
             b) {*this = *this/b; return *this;}
59      template<typename T> BigNum& operator %= (const T&
             b) {*this = *this%b; return *this;}
60      template<typename T> BigNum& operator += (const T&
             b) {*this = *this+b; return *this;}
61      template<typename T> BigNum& operator -= (const T&
             b) {*this = *this-b; return *this;}
62      template<typename T> BigNum& operator *= (const T&
             b) {*this = *this*b; return *this;}
63      template<typename T> BigNum& operator ^= (const T&
             b) {*this = *this^b; return *this;}
64
65      BigNum operator ++ (int) {BigNum t = *this; *this
            += 1; return t;}
66      BigNum operator -- (int) {BigNum t = *this; *this
            -= 1; return t;}
67      BigNum& operator -- () {*this -= 1; return *this;}
68      BigNum& operator ++ () {*this += 1; return *this;}
69
70      template<typename T> BigNum& operator <<= (const T
            & b) {*this = *this << b; return *this;}
71      template<typename T> BigNum& operator >>= (const T
            & b) {*this = *this >> b; return *this;}
72
73      template<typename T> BigNum friend operator + (
            const T& a, const BigNum& b) {BigNum t = a; t
             = t + a; return t;}
74      template<typename T> BigNum friend operator - (
            const T& a, const BigNum& b) {BigNum t = a; t
             = t - b; return t;}
75      template<typename T> BigNum friend operator * (
            const T& a, const BigNum& b) {BigNum t = a; t
             = t * b; return t;}
76      template<typename T> friend bool operator < (const
             T& a, const BigNum& b) {return b > a;}
77      template<typename T> friend bool operator > (const
             T& a, const BigNum& b) {return b < a;}
78      template<typename T> friend bool operator <= (
            const T& a, const BigNum& b) {return b >= a;}
79      template<typename T> friend bool operator >= (
            const T& a, const BigNum& b) {return b <= a;}
80      template<typename T> friend bool operator == (
            const T& a, const BigNum& b) {return b == a;}
81      template<typename T> friend bool operator != (
            const T& a, const BigNum& b) {return b != a;}
82
83      void print();
84      int Size();
85      int the_first();
86      int the_last();
87      int to_int();
88      long long int to_long();
89      string to_String();
90  };
91
92  BigNum::BigNum(const int b)
93  {
94      int c,d = b;
95      len = 0;
96      memset(a,0,sizeof(a));
97      while(d > MAXN){
98          c = d - (d / (MAXN+1)) * (MAXN+1);
99          d = d / (MAXN+1);
100         a[len++] = c;
101     }
102     a[len++] = d;
103 }
104 BigNum::BigNum(const long long int b)
105 {
106     long long int c,d = b;
```

```
    len = 0;
    memset(a,0,sizeof(a));
    while(d > MAXN){
        c = d - (d / (MAXN+1)) * (MAXN+1);
        d = d / (MAXN+1);
        a[len++] = c;
    }
    a[len++] = d;
}
BigNum::BigNum(const string& s)
{
    int t,k,index,l,i;
    memset(a,0,sizeof(a));
    l = s.size();
    len = l/DLEN;
    if(l%DLEN)
        len++;
    index = 0;
    for(i = l-1; i >=0 ;i -= DLEN){
        t = 0;
        k = i-DLEN+1;
        if(k < 0) k = 0;
        for(int j = k; j <= i; j++)
            t = t*10 + s[j]-'0';
        a[index++] = t;
    }
}
BigNum::BigNum(const char* s)
{
    int t,k,index,l,i;
    memset(a,0,sizeof(a));
    l = strlen(s);
    len = l/DLEN;
    if(l%DLEN)
        len++;
    index = 0;
    for(i = l-1; i >= 0; i -= DLEN){
        t = 0;
        k = i - DLEN + 1;
        if(k < 0) k = 0;
        for(int j = k; j <= i; j++)
            t = t*10 + s[j] - '0';
        a[index++] = t;
    }
}
BigNum::BigNum(const BigNum & b) : len(b.len)
{
    memset(a,0,sizeof(a));
    for(int i = 0 ; i < len ; i++)
        a[i] = b.a[i];
}
BigNum & BigNum::operator = (const BigNum& n)
{
    len = n.len;
    memset(a,0,sizeof(a));
    for(int i = 0 ; i < len ; i++)
        a[i] = n.a[i];
    return *this;
}
BigNum & BigNum::operator = (const int& num)
{
    BigNum t(num);
    *this = t;
    return *this;
}
BigNum & BigNum::operator = (const long long int& num
    )
{
    BigNum t(num);
    *this = t;
    return *this;
}
istream& operator >> (istream & in, BigNum & b)
{
    char ch[MAXSIZE*4];
    int i = -1;
    in>>ch;
    int l = strlen(ch);
    int cnt = 0, sum = 0;
    for(i = l-1; i >= 0; ){
        sum = 0;
        int t = 1;
        for(int j = 0; j < 4 && i >= 0; j++,i--,t *=
            10)
            sum += (ch[i]-'0')*t;
        b.a[cnt] = sum;
        cnt++;
    }
    b.len = cnt++;
    return in;

}
ostream& operator << (ostream& out, BigNum& b)
{
    int i;
    cout << b.a[b.len - 1];
    for(i = b.len - 2 ; i >= 0 ; i--){
        cout.width(DLEN);
        cout.fill('0');
        cout << b.a[i];
    }
    return out;
}

template<typename T> BigNum BigNum::operator << (
    const T& b) const
{
    T temp = 1;
    for(int i = 0; i < b; i++)
        temp *= 2;
    BigNum t = (*this) * temp;
    return t;
}
template<typename T> BigNum BigNum::operator >> (
    const T& b) const
{
    T temp = 1;
    for(int i = 0; i < b; i++)
        temp *= 2;
    BigNum t = (*this) / temp;
    return t;
}

BigNum BigNum::operator + (const BigNum& b) const
{
    BigNum t(*this);
    int i,big;
    big = b.len > len ? b.len : len;
    for(i = 0 ; i < big ; i++){
        t.a[i] += b.a[i];
```

```
        if(t.a[i] > MAXN){
            t.a[i + 1]++;
            t.a[i] -=MAXN+1;
        }
    }
    if(t.a[big] != 0)
        t.len = big + 1;
    else
        t.len = big;
    return t;
}
BigNum BigNum::operator - (const BigNum& b) const
{
    int i,j,big;
    bool flag;
    BigNum t1,t2;
    if(*this>b){
        t1 = *this;
        t2 = b;
        flag = 0;
    }
    else{
        t1 = b;
        t2 = *this;
        flag = 1;
    }
    big = t1.len;
    for(i = 0 ; i < big ; i++){
        if(t1.a[i] < t2.a[i]){
            j = i + 1;
            while(t1.a[j] == 0)
                j++;
            t1.a[j--]--;
            while(j > i)
                t1.a[j--] += MAXN;
            t1.a[i] += MAXN + 1 - t2.a[i];
        }
        else
            t1.a[i] -= t2.a[i];
    }
    t1.len = big;
    while(t1.a[t1.len - 1] == 0 && t1.len > 1){
        t1.len--;
        big--;
    }
    if(flag)
        t1.a[big-1] = 0-t1.a[big-1];
    return t1;
}

BigNum BigNum::operator * (const BigNum& b) const
{
    BigNum ret;
    int i,j,up;
    int temp,temp1;
    for(i = 0 ; i < len ; i++){
        up = 0;
        for(j = 0 ; j < b.len ; j++){
            temp = a[i] * b.a[j] + ret.a[i + j] + up;
            if(temp > MAXN){
                temp1 = temp - temp / (MAXN + 1) * (MAXN
                    + 1);
                up = temp / (MAXN + 1);
                ret.a[i + j] = temp1;
            }
            else{
                up = 0;
                ret.a[i + j] = temp;
            }
        }
        if(up != 0) ret.a[i + j] = up;
    }
    ret.len = i + j;
    while(ret.a[ret.len - 1] == 0 && ret.len > 1)
        ret.len--;
    return ret;
}
template<typename T> BigNum BigNum::operator / (const
     T& b) const
{
    BigNum ret;
    T i,down = 0;
    for(i = len - 1 ; i >= 0 ; i--){
        ret.a[i] = (a[i] + down * (MAXN + 1)) / b;
        down = a[i] + down * (MAXN + 1) - ret.a[i] * b
            ;
    }
    ret.len = len;
    while(ret.a[ret.len - 1] == 0 && ret.len > 1)
        ret.len--;
    return ret;
}
template<typename T> T BigNum::operator % (const T& b
    ) const
{
    T i,d=0;
    for (i = len-1; i>=0; i--){
        d = ((d * (MAXN+1))% b + a[i])% b;
    }
    return d;
}


template<typename T> BigNum BigNum::operator^(const T
    & n) const
{
    BigNum t,ret(1);
    int i;
    if(n < 0) return 0;
    if(n == 0)
        return 1;
    if(n == 1)
        return *this;
    int m = n;
    while(m > 1){
        t =* this;
        for(i = 1; (i<<1) <= m;i <<= 1)
            t = t*t;
        m-=i;
        ret=ret*t;
        if(m == 1) ret = ret * (*this);
    }
    return ret;
}

bool BigNum::operator > (const BigNum& b) const
{
    int tot;
    if(len > b.len)
        return true;
```

```
    else if(len == b.len){
        tot = len - 1;
        while(a[tot] == b.a[tot] && tot >= 0)
            tot--;
        if(tot >= 0 && a[tot] > b.a[tot])
            return true;
        else
            return false;
    }
    else
        return false;
}

bool BigNum::operator < (const BigNum& b) const
{
    int tot;
    if(len > b.len)
        return false;
    else if(len == b.len){
        tot = len - 1;
        while(a[tot] == b.a[tot] && tot >= 0)
            tot--;
        if(tot >= 0 && a[tot] > b.a[tot])
            return false;
        else
            return true;//
    }
    else
        return true;
}

bool BigNum::operator == (const BigNum& b) const
{
    int tot = len-1;
    if(len != b.len)
        return false;
    while(a[tot] == b.a[tot] && tot >= 0)
        tot--;
    if(tot < 0)
        return true;
    return false;
}

void BigNum::print()
{
    int i;
    cout << a[len - 1];
    for(i = len-2; i >= 0; i--){
        cout.width(DLEN);
        cout.fill('0');
        cout << a[i];
    }
    cout << endl;
}
int BigNum::Size()
{
    int t = a[len-1],cnt = 0;
    while(t){ t /= 10; cnt++; }
    cnt += (len-1)*4;
    return cnt;
}
int BigNum::the_first()
{
    int t = a[len-1];
    while(t > 10){ t /= 10;}
    return t;
}
int BigNum::the_last()
{
    int t = a[0];
    return t%10;
}
int BigNum::to_int()
{
    int i,num;
    num = a[len-1];
    for(i = len-2; i >= 0; i--)
        num = num*(MAXN+1) + a[i];
    return num;
}
long long int BigNum::to_long()
{
    int i;
    long long int num;
    num = a[len-1];
    for(i = len-2; i >= 0; i--)
        num = num*(MAXN+1) + a[i];
    return num;
}

int main()
{
    BigNum a,b;
    cin>>a>>b;
    (a+b).print();
    return 0;
}
```


7.3 约瑟夫环

7.4 悬线法

7.5 蔡勒公式

7.6 三角公式

7.7 海伦公式

7.8 匹克定理

7.9 组合计数

7.9.1 计数原理

7.9.2 卡特兰数

7.9.3 Polya

7.9.4 二项式反演公式

7.9.5 斯特林反演公式

7.9.6 组合数恒等式