Nantong University ICPC Team Notebook (2018-19)

thirtiseven wanggann Simon

Oct 31 2018

目录

第一章　输入输出

1.1　取消同步

```
std::ios::sync_with_stdio(false);
std::cin.tie(0);
```

1.2　浮点数输出格式

```
//include <iomanip>

std::cout << std::fixed << std::setprecision(12) << ans << std::endl;
```

1.3　整型快速输入

```
//整型
//若读入不成功，返回false
//ios::sync_with_stdio(true)
//#include <cctype>
bool quick_in(int &x) {
    char c;
    while((c = getchar()) != EOF && !isdigit(c));
    if(c == EOF) {
        return false;
    }
    x = 0;
    do {
        x *= 10;
        x += c - '0';
    } while((c = getchar()) != EOF && isdigit(c));
    return true;
}

//带符号整型
//直接=返回值
//#include <cctype>
int read() {
    int x = 0, l = 1; char ch = getchar();
    while (!isdigit(ch)) {if (ch=='-') l=-1; ch=getchar();}
```

```
    while (isdigit(ch)) x=x*10+(ch^48),ch=getchar();
    return x*1;
}

template <class T>
inline bool Read(T &ret) {
    char c; int sgn;
    if(c=getchar(),c==EOF) return 0; //EOF
    while(c!='-'&&(c<'0'||c>'9')) c=getchar();
    sgn=(c=='-') ?-1:1 ;
    ret=(c=='-') ?0:(c -'0');
    while(c=getchar(),c>='0'&&c<='9')
        ret=ret*10+(c-'0');
    ret*=sgn;
    return 1;
}
```

1.4 字符串快速输入

```
bool quick_in(char *p) {
    char c;
    while((c = getchar()) != EOF && (c == ' ' || c == '\n'));
    if(c == EOF) {
        return false;
    }
    do {
        *p++ = c;
    } while((c=getchar()) != EOF && c != ' ' && c != '\n');
    *p = 0;
    return true;
}
```

1.5 整型快速输出

```
void quick_out(int x) {
    char str[13];
    if(x) {
        int i;
        for(i = 0; x; ++i) {
            str[i] = x % 10 + '0';
            x /= 10;
        }
        while(i--) {
            putchar(str[i]);
        }
    } else {
        putchar('0');
    }
}
```

1.6 字符串快速输出

```
void quick_out(char *p) {
    while(*p) {
        putchar(*p++);
    }
}
```

1.7 python 输入

```
a, b, c =map(int,input().split('␣'))
```

1.8 java 输入

```
Scanner cin=new Scanner(System.in);// 读入
```

1.9 int128 输入输出

```
std::ostream& operator<<(std::ostream& os, __int128 T) {
    if (T<0) os<<"-";if (T>=10 ) os<<T/10;if (T<=-10) os<<(-(T/10));
    return os<<( (int) (T%10) >0 ? (int) (T%10) : -(int) (T%10) ) ;
}

void scan(__int128 &x) {
    x = 0;
    int f = 1;
    char ch;
    if((ch = getchar()) == '-') f = -f;
    else x = x*10 + ch-'0';
    while((ch = getchar()) >= '0' && ch <= '9')
        x = x*10 + ch-'0';
    x *= f;
}

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

第二章　动态规划

2.1　背包问题

```cpp
const int maxn=100005;
int w[maxn],v[maxn],num[maxn];
int W,n;
int dp[maxn];

void ZOP(int weight, int value) {
    for(int i = W; i >= weight; i--) {
        dp[i]=std::max(dp[i],dp[i-weight]+value);
    }
}

void CP(int weight, int value){
    for(int i = weight; i <= W; i++) {
        dp[i] = std::max(dp[i], dp[i-weight]+value);
    }
}

void MP(int weight, int value, int cnt){
    if(weight*cnt >= W) {
         CP(weight, value);
    } else {
        for(int k = 1; k < cnt; k <<= 1) {
            ZOP(k*weight, k*value), cnt -= k;
        }
        ZOP(cnt*weight, cnt*value);
    }
}
```

2.2　最长单调子序列（nlogn）

```cpp
int arr[maxn], n;

template<class Cmp>
int LIS (Cmp cmp) {
    static int m, end[maxn];
    m = 0;
    for (int i=0; i<n; i++) {
        int pos = lower_bound(end, end+m, arr[i], cmp)-end;
        end[pos] = arr[i], m += pos==m;
```

```
    }
    return m;
}

bool greater1(int value) {
    return value >=1;
}

/*********
    std::cout << LIS(std::less<int>()) << std::endl;          //严格上升
    std::cout << LIS(std::less_equal<int>()) << std::endl;    //非严格上升
    std::cout << LIS(std::greater<int>()) << std::endl;       //严格下降
    std::cout << LIS(std::greater_equal<int>()) << std::endl;//非严格下降
    std::cout << count_if(a,a+7,std::greater1) << std::endl; //计数
**********/
```

2.3 最长公共子序列

```
int dp[maxn][maxn];

void LCS(int n1, int n2, int A[], int B[]) {
    for(int i=1; i<=n1; i++) {
        for(int j=1; j<=n2; j++) {
            dp[i][j] = dp[i-1][j];
            if (dp[i][j-1] > dp[i][j]) {
                dp[i][j] = dp[i][j-1];
            }
            if (A[i] == B[j] && dp[i-1][j-1] + 1 > dp[i][j]) {
                dp[i][j] = dp[i-1][j-1] + 1;
            }
        }
    }
}
```

2.4 单调队列优化 DP

```
//单调队列求区间最小值
int a[maxn], q[maxn], num[maxn] = {0};
int Fmin[maxn];
int k, n, head, tail;

void DPmin() {
    head = 1, tail = 0;
    for (int i = 1; i <= n; i++) {
        while (num[head] < i-k+1 && head <= tail) head++;
        while (a[i] <= q[tail] /*区间最大值此处改为>=*/ && head <= tail) tail--;
        num[++tail] = i;
        q[tail] = a[i];
        Fmin[i] = q[head];
```

```
    }
}
```

2.5 单调队列 DP

```
int head,tail;
struct node
{
    int val,k;
}q[maxm*2];
void right(int i,int j) {
    while(head<tail&&dp[i-1][j]-sum[i][j-1]>=q[tail-1].val) tail--;//dp[i-1][k]-sum[i][k]
    q[tail].val=dp[i-1][j]-sum[i][j-1];
    q[tail++].k=j;
    while(head==-1||(head<tail&&j-q[head].k>t)) head++;//保证步数不大于限制条件
    dp[i][j]=max(dp[i][j],q[head].val+sum[i][j]);//dp[i][j]=max(dp[i][j],dp[i-1][k]-sum[i][k-1]+sum[i][j])
}
void left(int i,int j) {
    while(head<tail&&dp[i-1][j]+sum[i][j]>=q[tail-1].val) tail--;//dp[i-1][k]+sum[i][k];
    q[tail].val=dp[i-1][j]+sum[i][j];
    q[tail++].k=j;
    while(head==-1||(head<tail&&q[head].k-j>t)) head++;//保证步数不大于限制条件
    dp[i][j]=max(dp[i][j],q[head].val-sum[i][j-1]);//dp[i][j]=max(dp[i][j],dp[i-1][j]+sum[i][k]-sum[i][j-1])
}
void solve() {
    for(int i=1;i<=n;i++) for(int j=1;j<=m;j++) dp[i][j]=-INF;
    for(int i=1;i<=m;i++){//第一行初始化
        if(abs(x-i)>t) continue;
        if(i<=x) dp[1][i]=sum[1][x]-sum[1][i-1];
        else dp[1][i]=sum[1][i]-sum[1][x-1];
    }
    for(int i=2;i<=n;i++) {
        head=tail=0;
        for(int j=m;j>=1;j--) left(i,j);
        head=tail=0;
        for(int j=1;j<=m;j++) right(i,j);
    }
}
```

2.6 区间 DP

```
for (int x = 0; x < n; x++){//枚举长度
    for (int i = 1; i + x <= n; i++){//枚举起点
        dp[i][i] = 1;
        int j = x + i;//终点
        dp[i][j] = dp[i + 1][j] + 1;
        for (int k = i + 1; k <= j; k++) {
```

```
            if (a[i] == a[k])
                dp[i][j] = min(dp[i][j], dp[i][k - 1] + dp[k + 1][j]);
        }
    }
}
```

2.7 时间段 DP

```
int t=maxn-1;
memset(dp,-1,sizeof(dp));dp[0][0]=0;
for(int i=0;i<t;i++){//枚举每一个起点
    for(int j=0;j<(1<<n);j++) dp[i+1][j]=max(dp[i+1][j],dp[i][j]); //更新所有状态的值
    for(int j=0;j<g[i].size();j++){//枚举当前起点的所有终点
        for(int k=0;k<(1<<n);k++){
            if(dp[i][k]!=-1){
                dp[g[i][j].fin][k|(1<<g[i][j].num)]=max(dp[g[i][j].fin][k|(1<<g[i][j].num)],dp
                    [i][k]+g[i][j].val);
                //枚举所有结束时间，结束时间由起始时间转移过来
            }
        }
    }
}
```

2.8 数位 DP

```
typedef long long ll;
int a[20];
ll dp[20][state];//不同题目状态不同
ll dfs(int pos,/*state变量*/,bool lead/*前导零*/,bool limit/*数位上界变量*/)//不是每个题都要判
    断前导零
{
    //递归边界，既然是按位枚举，最低位是0，那么pos==-1说明这个数我枚举完了
    if(pos==-1) return 1;/*这里一般返回1，表示你枚举的这个数是合法的，那么这里就需要你在枚举时
        必须每一位都要满足题目条件，也就是说当前枚举到pos位，一定要保证前面已经枚举的数位是合
        法的。不过具体题目不同或者写法不同的话不一定要返回1 */
    //第二个就是记忆化(在此前可能不同题目还能有一些剪枝)
    if(!limit && !lead && dp[pos][state]!=-1) return dp[pos][state];
    /*常规写法都是在没有限制的条件记忆化，这里与下面记录状态是对应，具体为什么是有条件的记忆化
        后面会讲*/
    int up=limit?a[pos]:9;//根据limit判断枚举的上界up;这个的例子前面用213讲过了
    ll ans=0;
    //开始计数
    for(int i=0;i<=up;i++)//枚举，然后把不同情况的个数加到ans就可以了
    {
        if() ...
        else if()...
        ans+=dfs(pos-1,/*状态转移*/,lead && i==0,limit && i==a[pos]) //最后两个变量传参都是这
            样写的
        /*这里还算比较灵活，不过做几个题就觉得这里也是套路了
```

```
        大概就是说，我当前数位枚举的数是i，然后根据题目的约束条件分类讨论
        去计算不同情况下的个数，还有要根据state变量来保证i的合法性，比如题目
        要求数位上不能有62连续出现，那么就是state就是要保存前一位pre，然后分类，
        前一位如果是6那么这意味就不能是2，这里一定要保存枚举的这个数是合法*/
    }
    //计算完，记录状态
    if(!limit && !lead) dp[pos][state]=ans;
    /*这里对应上面的记忆化，在一定条件下时记录，保证一致性，当然如果约束条件不需要考虑lead，这
        里就是lead就完全不用考虑了*/
    return ans;
}

ll solve(ll x)
{
    int pos=0;
    while(x)//把数位都分解出来
    {
        a[pos++]=x%10;//个人老是喜欢编号为[0,pos)，看不惯的就按自己习惯来，反正注意数位边界就行
        x/=10;
    }
    return dfs(pos-1/*从最高位开始枚举*/,/*一系列状态 */,true,true);//刚开始最高位都是有限制并
        且有前导零的，显然比最高位还要高的一位视为0嘛
}

int main()
{
    ll le,ri;
    while(~scanf("%lld%lld",&le,&ri))
    {
        //初始化dp数组为-1，这里还有更加优美的优化，后面讲
        printf("%lld\n",solve(ri)-solve(le-1));
    }
}
```

2.9 数位 DP

```
int a[maxn],bit[maxn];//a为分解整数数组，bit数组为10^(i-1)
pair<int,int>dp[maxn][2000];//first=满足条件的数个数，second=满足条件的数的和
bool vis[maxn][2000];
pair<int,int> dfs(int pos,int sta,int num,bool lead,bool limit){//求满足条件的所有数的和
    if(pos==0) return make_pair(1,0);//计数
    if(!limit&&!lead&&vis[pos][sta]) return dp[pos][sta];
    if(!limit&&!lead) vis[pos][sta]=1;
    int up=limit?a[pos]:9,t,tt; pair<int,int>tmp,ans;
    for(int i=0;i<=up;i++){
        if(num>=k&&!(sta&(1<<i))) continue;//不满足条件，跳出
        if(lead&&!i) t=0,tt=0;
        else t=(sta|(1<<i)),tt=(sta&(1<<i))?num:num+1;
        tmp=dfs(pos-1,t,tt,lead&&!i,limit&&i==up);
        ans.first+=tmp.first;ans.first%=mod;//满足条件的数的个数
        ans.second+=tmp.first*bit[pos]%mod*i%mod+tmp.second;ans.second%=mod;//满足条件的数的和
            10+11+...+19=9*(10*1)+45
```

```
    }
    if(!limit&&!lead) dp[pos][sta]=ans;
    return ans;
}
```

2.10 上升子序列最大和 (nlogn)

```
//二维带权最大上升子序列
//written by Simon

#include<bits/stdc++.h>
using namespace std;
typedef int Int;
#define int long long
#define INF 0x3f3f3f3f
#define maxn 100005
int bin[maxn];//用于二分
struct cmp_key{//map排序辅助结构体,第一维升序, 第二维降序
    bool operator ()(const pair<int,int>&a,const pair<int,int>&b){
        if(a.first==b.first){
            return a.second>b.second;
        }
        else return a.first<b.first;
    }
};
map<pair<int,int>,int,cmp_key>p;
int binary_search(int v,int n){
    int left=0,right=n,mid;
    while(left<=right){
        mid=(left+right)>>1;
        if(bin[mid]<v){
            left=mid+1;
        }
        else right=mid-1;
    }
    return left;
}
int tree[maxn];
int lowbit(int x){
    return (x&(-x));
}
void update(int x,int v,int len){
    for(int i=x;i<=len;i+=lowbit(i)){
        tree[i]=max(tree[i],v);
    }
}
int get(int x,int len){
    int sum=0;
    for(int i=x;i>=1;i-=lowbit(i)){
        sum=max(sum,tree[i]);
    }
    return sum;
```

```
}
Int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    int n;
    scanf("%lld",&n);
    for(int i=1;i<=n;i++){
        int x,y,v;
        scanf("%lld%lld%lld",&x,&y,&v);
        p[make_pair(x,y)]+=v;//去重（关键）
        bin[i]=y;
    }
    sort(bin+1,bin+n+1);
    auto len=unique(bin+1,bin+n+1)-(bin+1);//去重
    for(auto v:p){
        int x=binary_search(v.first.second,len);
        int val=get(x-1,len);
        update(x,val+v.second,len);
    }
    printf("%lld\n",get(len,len));
    cin.get(),cin.get();
    return 0;
}
```

第三章　数论

3.1　暴力判素数

```cpp
bool is_prime(int  u) {
    if(u ==  0 || u ==  1) return false;
    if(u ==  2)      return true;
    if(u%2 == 0)      return false;
    for(int i=3; i <=  sqrt(u) ; i+=2)
        if(u%i==0)      return false;
    return true;
}
```

3.2　米勒罗宾素性检测

```cpp
using ll = long long;

ll prime[5] = {2, 3, 5, 233, 331};

ll pow_mod(ll a, ll n, ll mod) {
    ll ret = 1;
    while (n) {
        if (n&1) ret = ret * a % mod;
        a = a * a % mod;
        n >>= 1;
    }
    return ret;
}

int isPrime(ll n) {
    if (n < 2 || (n != 2 && !(n&1))) return 0;
    ll s = n - 1;
    while (!(s&1)) s >>= 1;
    for (int i = 0; i < 5; ++i) {
        if (n == prime[i]) return 1;
        ll t = s, m = pow_mod(prime[i], s, n);
        while (t != n-1 && m != 1 && m != n-1) {
            m = m * m % n;
            t <<= 1;
        }
        if (m != n-1 && !(t&1)) return 0;
    }
    return 1;
```

```
}
```

3.3 埃氏筛

```
bool prime_or_not[maxn];
for (int i = 2; i <= int(sqrt(maxn)); i++) {
    if (!prime_or_not[i]) {
        for (int j = i * i; j <= maxn; j = j+i) {
            prime_or_not[j] = 1;
        }
    }
}
```

3.4 欧拉筛

```
#include <iostream>

const int maxn = 1234;
int flag[maxn], primes[maxn], totPrimes;

void euler_sieve(int n) {
    totPrimes = 0;
    memset(flag, 0, sizeof(flag));
    for (int i = 2; i <= n; i++) {
        if (!flag[i]) {
            primes[totPrimes++] = i;
        }
        for (int j = 0; i * primes[j] <= n; j++) {
            flag[i * primes[j]] = true;
            if (i % primes[j] == 0)
            break;
        }
    }
}
```

3.5 分解质因数

```
int cnt[maxn];//存储质因子是什么
int num[maxn];//该质因子的个数
int tot = 0;//质因子的数量
void factorization(int x)//输入x，返回cnt数组和num数组
{
    for(int i=2;i*i<=x;i++)
    {
        if(x%i==0)
        {
            cnt[tot]=i;
```

```
            num[tot]=0;
            while(x%i==0)
            {
                x/=i;
                num[tot]++;
            }
            tot++;
        }
    }
    if(x!=1)
    {
        cnt[tot]=x;
        num[tot]=1;
        tot++;
    }
}
```

3.6 最大公约数

```
ll gcd(ll a, ll b) {
    ll t;
    while(b != 0) {
        t=a%b;
        a=b;
        b=t;
    }
    return a;
}
```

3.7 最小公倍数

```
ll lcm(ll a, ll b) {
    return a * b / gcd(a, b);
}
```

3.8 扩展欧几里得

```
//如果GCD(a,b) = d, 则存在x, y, 使d =  ax +  by
// extended_euclid(a, b) = ax + by
int extended_euclid(int a, int b, int &x, int &y) {
    int d;
    if(b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    d = extended_euclid(b, a % b, y, x);
```

```
    y -= a / b * x;
    return d;
}
```

3.9 中国剩余定理

```
LL Crt(LL *div, LL *rmd, LL len) {
    LL sum = 0;
    LL lcm = 1;
    //lcm为除数们的最小公倍数, 若div互素, 则如下一行计算lcm
    for (int i = 0; i < len; ++i)
        lcm *= div[i];
    for (int i = 0; i < len; ++i) {
        LL bsn = lcm / div[i];
        LL inv = Inv(bsn, div[i]);
        // dvd[i] = inv[i] * bsn[i] * rmd[i]
        LL dvd = MulMod(MulMod(inv, bsn, lcm), rmd[i], lcm);
        sum = (sum + dvd) % lcm;
    }
    return sum;
}
```

3.10 扩展 CRT

```
// ---
// $X = r_i (mod m_i)$; $m_i$可以不两两互质\\
// 引用返回通解$X = re + k * mo$; 函数返回是否有解
// ---
bool excrt(ll r[], ll m[], ll n, ll &re, ll &mo) {
    ll x, y;
    mo = m[0], re = r[0];
    for (int i = 1; i < n; i++) {
        ll d = exgcd(mo, m[i],  x, y);
        if ((r[i] - re) % d != 0) return 0;
        x = (r[i] - re) / d * x % (m[i] / d);
        re += x * mo;
        mo = mo / d * m[i];
        re %= mo;
    }
    re = (re + mo) % mo;
    return 1;
}
```

3.11 欧拉函数

```
LL EulerPhi(LL n){
    LL m = sqrt(n + 0.5);
```

```
    LL ans = n;
    for(LL i = 2; i <= m; ++i)
    if(n % i == 0) {
        ans = ans - ans / i;
    while(n % i == 0)
        n/=i;
    }
    if(n > 1)
        ans = ans - ans / n;
    return ans;
}
```

3.12 求逆元

```
LL Inv(LL a, LL n){
    return PowMod(a, EulerPhi(n) - 1, n);
    //return PowMod(a,n-2,n); //n为素数
}

int Inv(int a, int n) {
    int d, x, y;
    d = extended_euclid(a, n, x, y);
    if(d == 1)  return (x%n + n) % n;
    else     return -1; // no solution
}
```

3.13 快速乘法取模

```
//by sevenkplus
#define ll long long
#define ld long double
ll mul(ll x,ll y,ll z){return (x*y-(ll)(x/(ld)z*y+1e-3)*z+z)%z;}

//by Lazer2001
inline long long mmul (long long a, long long b, const long long& Mod) {
    long long lf = a * (b >> 25LL) % Mod * (1LL << 25) % Mod;
    long long rg = a * ( b & ( ( 1LL << 25 ) - 1 ) ) % Mod ;
    return (lf + rg) % Mod ;
}
```

3.14 快速幂取模

```
using LL = long long;

LL PowMod(LL a, LL b, const LL &Mod) {
    a %= Mod;
    LL ans = 1;
```

```
    while(b) {
        if (b & 1){
            ans = (ans * a) % Mod;
        }
        a = (a * a) % Mod;
        b >>= 1;
    }
    return ans;
}
```

3.15 互质对数计数

```
//Written by Simon
//求r以内与n不互质的数的个数
int solve(int r) {
    int sum=0;
    for(int i=1;i<(1<<fac.size());i++) {//枚举质因数的每一种组合
        int ans=1,num=0;
        for(int j=0;j<fac.size();j++) {//求当前组和的积
            if(i&(1<<j)) {
                ans *= fac[j];
                num++;
            }
        }
        if(num&1) sum+=r/ans;//如果当前组合个数为奇数个，加上r以内能被ans整除的数的个数
        else sum-=r/ans;//否则减去r以内能被ans整除的数的个数
    }
    return sum;
}
```

3.16 BSGS

```
//Author: Simon
#include <algorithm>
#include <cmath>
#include <cstring>
using ll = long long;
const int maxn = 1000005;
const ll mod = 611977;

struct HashMap {
    ll head[mod+5], key[maxn], value[maxn], nxt[maxn], tol;
    inline void clear() {
        tol=0;
        memset(head,-1,sizeof(head));
    }
    HashMap() {
        clear();
    }
```

```
18      inline void insert(ll k,ll v) {
19          ll idx = k % mod;
20          for(ll i = head[idx]; ~i; i = nxt[i]) {
21              if(key[i] == k) {
22                  value[i] = std::min(value[i], v);
23                  return ;
24              }
25          }
26          key[tol] = k;
27          value[tol] = v;
28          nxt[tol] = head[idx];
29          head[idx] = tol++;
30      }
31      inline ll operator [](const ll &k) const {
32          ll idx = k % mod;
33          for(ll i=head[idx]; ~i; i=nxt[i]) {
34              if(key[i]==k) return value[i];
35          }
36          return -1;
37      }
38  }mp;
39
40  inline ll fpow(ll a, ll b, ll mod) {
41      a %= mod;
42      ll ans = 1;
43      while (b) {
44          if(b&1) ans = ans * a % mod;
45          a = a * a % mod;
46          b >>= 1;
47      }
48      return ans;
49  }
50  inline ll exgcd(ll a,ll b,ll &x,ll &y) {
51      if (b==0) {
52          x=1, y=0;
53          return a;
54      }
55      ll ans = exgcd(b, a%b, y, x);
56      y -= a/b*x;
57      return ans;
58  }
59
60  inline ll Bsgs(ll a,ll b,ll mod) {
61      a %= mod, b %= mod;
62      if (b==1) return 0;
63      ll m = ceil(sqrt(mod)), inv, y;
64      exgcd(fpow(a, m, mod), mod, inv, y);
65      inv = (inv % mod + mod) % mod;
66      mp.insert(1, 1);
67      for(ll i=1, e=1; i<m; i++) {
68          e = e * a % mod;
69          if(mp[e] == -1) mp.insert(e, i+1);
70      }
71      for(ll i = 0; i <= m; i++) {
```

```
        if(mp[b] != -1) {
            ll ans = mp[b]-1;
            return ans + i * m;
        }
        b = b * inv % mod;
    }
    return -1;
}

inline ll gcd(ll a, ll b) {
    return b==0 ? a : gcd(b, a%b);
}

inline int exBsgs(int a,int b,int mod) {//扩展BSGS, 处理a, mod不互质的情况
    if(b==1) return 0;
    for(int g=gcd(a,mod),i=0;g!=1;g=gcd(a,mod),i++)  {
        if(b%g) return -1;//保证g为a,b,mod的最大公约数
        mod/=g;
    }
    return Bsgs(a,b,mod);
}
```

3.17 二分分数树（Stern-Brocot Tree）

```
//Author:CookiC
//未做模板调整, 请自行调整
#include <cmath>
#define LL long long
#define LD long double

void SternBrocot(LD X, LL &A, LL &B) {
    A=X+0.5;
    B=1;
    if(A==X)
        return;
    LL la=X, lb=1, ra=X+1, rb=1;
    long double C=A, a, b, c;
    do {
        a = la+ra;
        b = lb+rb;
        c = a/b;
        if(std::abs(C-X) > std::abs(c-X)) {
            A=a;
            B=b;
            C=c;
            if(std::abs(X-C) < 1e-10) {
                break;
            }
        }
        if(X<c) {
            ra=a;
            rb=b;
```

```
        } else {
            la=a;
            lb=b;
        }
    } while(lb+rb<=1e5);
}
```

3.18 二次剩余

```
//求解X^2==n(mod p)
ll P;

inline ll Pow(ll a,ll b){
    ll ret=1;
    for (;b;b>>=1,a=a*a%P)
        if (b&1)
            ret=ret*a%P;
    return ret;
}

inline ll legendre(ll a){
    return Pow(a,(P-1)>>1);
}

struct abcd{
    ll a,b,w; //a+b*sqrt(w)
    abcd(ll a=0,ll b=0,ll w=0):a(a),b(b),w(w) { }
    friend abcd operator *(abcd A,abcd B){
        return abcd((A.a*B.a%P+A.b*B.b%P*A.w%P)%P,(A.a*B.b%P+A.b*B.a%P)%P,A.w);
    }
};

inline abcd Pow(abcd a,int b){
    abcd ret=abcd(1,0,a.w);
    for (;b;b>>=1,a=a*a)
        if (b&1)
            ret=ret*a;
    return ret;
}

inline ll Solve(ll n,ll p){
    P=p;
    if (P==2) return 1;
    if (legendre(n)==P-1) return -1;
    ll a,w;
    while (1){
        a=rand()%P;
        w=((a*a-n)%P+P)%P;
        if (legendre(w)==P-1) break;
    }
    return Pow(abcd(a,1,w),(P+1)>>1).a;
}
```

3.19 计算莫比乌斯函数

```
const int n=1<<20;
int mu[n];
int getMu() {
    for(int i=1;i<=n;i++) {
        int target=i==1?1:0;
        int delta=target-mu[i];
        mu[i]=delta;
        for(int j=i+i;j<=n;j+=i) {
            mu[j]+=delta;
        }
    }
}
```

3.20 杜教筛

```
int DuJiao(int n)// 杜教筛——欧拉函数之和
{
    if(n<maxn) return Phi[n]; //欧拉函数前缀和
    if(mp[n]!=-1) return mp[n];
    int sum=0,z=n%mod;
    // for(int l=2,r;l<=n;l=r+1) // #version 1
    // {
    //     r=n/(n/l);
    //     sum+=DuJiao(n/l)*(r-l+1);
    //     sum%=mod;
    // }
    for(int i=1;i*i<=n;i++) // #vsesion 2————对每一个i=[2...n]求sum[phi(1)+...+phi(n/i)]
    {
        sum+=DuJiao(i)*(n/i-n/(i+1));
        sum%=mod;
        int x=n/i; //x为值，枚举i求x;
        if(x==i||i==1) continue;
        sum+=DuJiao(x)*(n/x-n/(x+1));
        sum%=mod;
    }
    sum=((z*(z+1)%mod*inv2%mod)%mod-sum%mod+mod)%mod; //等差数列前n项和-sum
    mp.insert(n,sum);//加入HashMap
    return sum%mod;
}
```

3.21 常用公式

1. 约数定理：若 $n=\prod_{i=1}^{k} p_i^{a_i}$，则

   (a) 约数个数 $f(n)=\prod_{i=1}^{k}(a_i+1)$

(b) 约数和 $g(n) = \prod_{i=1}^{k}(\sum_{j=0}^{a_i} p_i^j)$

2. 小于 n 且互素的数之和为 $n\varphi(n)/2$
3. 若 $gcd(n,i) = 1$，则 $gcd(n, n-i) = 1(1 \le i \le n)$
4. 错排公式：$D(n) = (n-1)(D(n-2) + D(n-1)) = \sum_{i=2}^{n} \frac{(-1)^k n!}{k!} = [\frac{n!}{e} + 0.5]$
5. 威尔逊定理：$p\ is\ prime\ \Rightarrow (p-1)! \equiv -1(mod\ p)$
6. 欧拉定理：$gcd(a,n) = 1 \Rightarrow a^{\varphi(n)} \equiv 1(mod\ n)$
7. 欧拉定理推广：$gcd(n,p) = 1 \Rightarrow a^n \equiv a^{n\%\varphi(p)}(mod\ p)$
8. 素数定理：对于不大于 n 的素数个数 $\pi(n)$，$\lim\limits_{n\to\infty} \pi(n) = \frac{n}{\ln n}$
9. 位数公式：正整数 x 的位数 $N = log10(n) + 1$
10. 斯特灵公式 $n! \approx \sqrt{2\pi n}(\frac{n}{e})^n$
11. 设 $a > 1, m, n > 0$, 则 $gcd(a^m - 1, a^n - 1) = a^{gcd(m,n)} - 1$
12. 设 $a > b, gcd(a,b) = 1$, 则 $gcd(a^m - b^m, a^n - b^n) = a^{gcd(m,n)} - b^{gcd(m,n)}$

$$G = gcd(C_n^1, C_n^2, ..., C_n^{n-1}) = \begin{cases} n, & n \text{ is prime} \\ 1, & n \text{ has multy prime factors} \\ p, & n \text{ has single prime factor } p \end{cases}$$

$gcd(Fib(m), Fib(n)) = Fib(gcd(m,n))$

13. 若 $gcd(m,n) = 1$, 则:
    (a) 最大不能组合的数为 $m * n - m - n$
    (b) 不能组合数个数 $N = \frac{(m-1)(n-1)}{2}$
14. $(n+1)lcm(C_n^0, C_n^1, ..., C_n^{n-1}, C_n^n) = lcm(1, 2, ..., n+1)$
15. 若 p 为素数，则 $(x + y + ... + w)^p \equiv x^p + y^p + ... + w^p(mod\ p)$
16. 卡特兰数：1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012
    $h(0) = h(1) = 1, h(n) = \frac{(4n-2)h(n-1)}{n+1} = \frac{C_{2n}^n}{n+1} = C_{2n}^n - C_{2n}^{n-1}$
17. 伯努利数：$B_n = -\frac{1}{n+1}\sum_{i=0}^{n-1} C_{n+1}^i B_i$

$$\sum_{i=1}^{n} i^k = \frac{1}{k+1}\sum_{i=1}^{k+1} C_{k+1}^i B_{k+1-i}(n+1)^i$$

18. FFT 常用素数

$r\ 2^k+1$	r	k	g
3	1	1	2
5	1	2	2
17	1	4	3
97	3	5	5
193	3	6	5
257	1	8	3
7681	15	9	17
12289	3	12	11
40961	5	13	3
65537	1	16	3
786433	3	18	10
5767169	11	19	3
7340033	7	20	3
23068673	11	21	3
104857601	25	22	3
167772161	5	25	3
469762049	7	26	3
998244353	119	23	3
1004535809	479	21	3
2013265921	15	27	31
2281701377	17	27	3
3221225473	3	30	5
75161927681	35	31	3
77309411329	9	33	7
206158430209	3	36	22
2061584302081	15	37	7
2748779069441	5	39	3
6597069766657	3	41	5
39582418599937	9	42	5
79164837199873	9	43	5
263882790666241	15	44	7
1231453023109121	35	45	3
1337006139375617	19	46	3
3799912185593857	27	47	5
4222124650659841	15	48	19
7881299347898369	7	50	6
31525197391593473	7	52	3
180143985094819841	5	55	6
1945555039024054273	27	56	5
4179340454199820289	29	57	3

3.22 数论公式

1. 小于 n 的 i，j，gcd(i,j)=1 的 i,j 的对数与欧拉函数的关系 $\sum_{i=1}^{n}\sum_{j=1}^{n}[gcd(i,j)=1]=2\sum_{i=1}^{n}\sum_{j=1}^{i}[gcd(i,j)=1]-\sum_{i=1}^{n}[gcd(i,i)=1]=\left(2\sum_{i=1}^{n}\varphi(i)\right)-1$

2. 小于 n 的 i,j, 且 gcd(i,j)=1 时,所有 i*j 的和与欧拉函数的关系 $\sum_{i=1}^{n} i \sum_{j=1}^{n}[gcd(i,j)=1]\cdot j = 2\sum_{i=1}^{n} i \sum_{j=1}^{i}[gcd(i,j)=1]\cdot j - \sum_{i=1}^{n}[gcd(i,i)=1]\cdot i$
$= \left(2\sum_{i=1}^{n} i\frac{i\cdot\varphi(i)+[i=1]}{2}\right) - 1 = \left(\sum_{i=1}^{n} i^2\cdot\varphi(i)+[i=1]\right) - 1$

3. 约数，倍数之间重要的变换 $\sum_{k=1}^{n}\sum_{d|k} d\cdot k = \sum_{k=1}^{n}\sum_{d=1}^{\frac{n}{k}} d\cdot k\cdot d = \sum_{d=1}^{n}\sum_{k|d} d\cdot\frac{d}{k} = \sum_{d=1}^{n}\sum_{k=1}^{\frac{n}{d}} d\cdot k\cdot d = \sum_{d=1}^{n}\sum_{d|k} d\cdot k = \sum_{k=1}^{n}\sum_{k|d} d\cdot\frac{d}{k}$

4. $C_m^n = C_{m-1}^{n-1} + C_{m-1}^n$

5. $C_n^m = \frac{n!}{m!(n-m)!}$

6. $C_n^m = C_n^{n-m} = C_{n-1}^m + C_{n-1}^{m-1}$（杨辉三角）

7. $C_n^0 + C_n^1 + C_n^2 + ... + C_n^n = \sum_{i=0}^{n} C_n^i = 2^n$

8. $C_n^0 + C_n^2 + C_n^4 + ... = C_n^1 + C_n^3 + C_n^5 + ... = 2^{n-1}$

9. $C_n^m + C_{n+1}^m + C_{n+2}^m + ... + C_{n+m}^m = \sum_{i=0}^{m} C_{n+i}^m = C_{n+m+1}^{m+1}$

10. $kC_n^k = nC_{n-1}^{k-1}$，$\frac{C_n^k}{k+1} = \frac{C_{n+1}^{k+1}}{n+1}$

11. $\sum_{k=0}^{n} k * C_m^k = \sum_{k=0}^{n-1} m * C_{m-1}^k$

12. $C_m^n \% p = C_{m/p}^{n/p} * C_{m\%p}^{n\%p} \% p$（Lucas 定理）

13. $C_{m+n}^k = \sum_{i=0}^{k} C_m^i C_n^{k-i}$（Vandermonde 恒等式）

14. $\frac{n!}{\prod_{i=1}^{k} a_i}$（有重复元素的全排列公式，$a_i$ 为第 i 种元素的个数，k 为元素种类数）

第四章　数学

4.1　C(n,m) mod p (n 很大 p 可以很大)

```
LL C(const LL &n, const LL &m, const int &pr) {
    LL ans = 1;
    for (int i = 1; i <= m; i++) {
        LL a = (n - m + i) % pr;
        LL b = i % pr;
        ans = (ans * (a * Inv(b, pr)) % pr) % pr;
    }
    return ans;
}
```

4.2　Lucas 定理

```
//C(n, m) mod p(n 很大 p 较小(不知道能不能为非素数)
LL Lucas(LL n, LL m, const int &pr) {
    if (m == 0) return 1;
    return C(n % pr, m % pr, pr) * Lucas(n / pr, m / pr, pr) % pr;
}
```

4.3　计算从 C(n, 0) 到 C(n, p) 的值

```
//by Yuhao Du
int p;
std::vector<int> gao(int n) {
    std::vector<int> ret(p+1,0);
    if (n==0) {
        ret[0]=1;
    } else if (n%2==0) {
        std::vector<int> c = gao(n/2);
        for(int i = 0; i <= p+1; i++) {
            for(int j = 0; j <= p+1; j++) {
                if (i+j<=p) ret[i+j]+=c[i]*c[j];
            }
        }
    } else {
        std::vector<int> c = gao(n-1);
        for(int i = 0; i <= p+1; i++) {
            for(int j = 0; j <= 2; j++) {
```

```
                if (i+j<=p) ret[i+j]+=c[i];
            }
        }
    }
    return ret;
}
```

4.4 计算第一类斯特林数

```
int seq[60][maxn << 1] , ptr = 0;
long long B[maxn << 1] , C[maxn << 1];

int DFS( int l , int r ){
    if( l == r ){
        int id = ptr ++ ;
        seq[id][1] = l ;
        seq[id][0] = 1 ;
        return id;
    } else {
        int mid = l + r >> 1;
        int lid = DFS( l , mid );
        int rid = DFS( mid + 1 , r );
        ptr -= 2;
        int newid = ptr ++ ;
        int len = 1;
        while( len <= r - l + 1 ) len <<= 1;
        for(int i = 0 ; i < len ; ++ i) B[i] = seq[lid][i] , C[i] = seq[rid][i] , seq[lid][i]
            = seq[rid][i] = 0;
        ntt( B , len , 1 );
        ntt( C , len , 1 );
        for(int i = 0 ; i < len ; ++ i) B[i] = B[i] * C[i] % Mod;
        ntt( B , len , -1 );
        for(int i = 0 ; i < len ; ++ i) seq[newid][i] = B[i];
        return newid;
    }
}

//int id = DFS( 0 , N - 1 );
//for(int i = N ; i >= 0 ; -- i) {
//  printf( "f[%d] is %d \n" , N - i , seq[id][i] );
//}
```

4.5 自适应辛普森

```
double F(double x) {
    //Simpson公式用到的函数
}
double simpson(double a, double b) {//三点Simpson法，这里要求F是一个全局函数
    double c = a + (b - a) / 2;
```

```
    return (F(a) + 4 * F(c) + F(b))*(b - a) / 6;
}
double asr(double a, double b, double eps, double A) {//自适应Simpson公式（递归过程）。已知整个区间[a,b]上的三点Simpson值A
    double c = a + (b - a) / 2;
    double L = simpson(a, c), R = simpson(c, b);
    if (fabs(L + R - A) <= 15 * eps)return L + R + (L + R - A) / 15.0;
    return asr(a, c, eps / 2, L) + asr(c, b, eps / 2, R);
}
double asr(double a, double b, double eps) {//自适应Simpson公式（主过程）
    return asr(a, b, eps, simpson(a, b));
}
```

4.6 博弈论

```
Nim Game
    最经典最基础的博弈.
    n堆石子,双方轮流从任意一堆石子中取出至少一个,不能取的人输.
    对于一堆x个石子的情况,容易用归纳法得到SG(x)=x.
    所以所有石子个数的异或和为0是必败态,否则为必胜态.

Bash Game
    每人最多一次只能取m个石子,其他规则同Nim Game.
    依旧数学归纳…SG(x)=xmod(m+1).

NimK Game
    每人一次可以从最多K堆石子中取出任意多个,其他规则同Nim Game.
    结论:在二进制下各位上各堆石子的数字之和均为(K+1)的倍数的话则为必败态,否则为必胜态.
    这个证明要回到原始的方法上去.
    补:这个游戏还可以推广,即一个由n个子游戏组成的游戏,每次可以在最多K个子游戏中进行操作.
    然后只要把结论中各堆石子的个数改为各个子游戏的SG值即可,证明也还是一样的.

Anti-Nim Game
    似乎又叫做Misère Nim.
    不能取的一方获胜,其他规则同Nim Game.
    关于所谓的"Anti-SG游戏"及"SJ定理"贾志鹏的论文上有详细说明,不过似乎遇到并不多.
    结论是一个状态是必胜态当且仅当满足以下条件之一:
    SG值不为0且至少有一堆石子数大于1;
    SG值为0且不存在石子数大于1的石子堆.

Staircase Nim
    每人一次可以从第一堆石子中取走若干个,或者从其他石子堆的一堆中取出若干个放到左边一堆里(没有石子的石子堆不会消失),其他规则同Nim Game.
    这个游戏的结论比较神奇:
    当且仅当奇数编号堆的石子数异或和为0时为必败态.
    简单的理解是从偶数编号堆中取石子对手又可以放回到奇数编号堆中,而且不会让对手不能移动.比较意识流,然而可以归纳证明.

Wythoff Game
    有两堆石子,双方轮流从某一堆取走若干石子或者从两堆中取走相同数目的石子,不能取的人输.
    容易推理得出对任意自然数k,都存在唯一的一个必败态使得两堆石子数差为k,设其为Pk=(ak,bk),表示石子数分别为ak,bk(ak<=bk).
```

```
    那么ak为在Pk0(k0<k)中未出现过的最小自然数,bk=ak+k.
    数学班的说,用Betty定理以及显然的单调性就可以推出神奇的结论:
    ak=floor(k*5√+12),bk=floor(k*5√+32).

Take & Break
    有n堆石子,双方轮流取出一堆石子,然后新增两堆规模更小的石子堆(可以没有石子),无法操作者输.
    这个游戏似乎只能暴力SG,知道一下就好.

树上删边游戏
    给出一个有n个结点的树，有一个点作为树的根节点,双方轮流从树中删去一条边边，之后不与根节点相
        连的部分将被移走,无法操作者输.
    结论是叶子结点的SG值为0,其他结点SG值为其每个儿子结点SG值加1后的异或和,证明也并不复杂.

翻硬币游戏
    n枚硬币排成一排，有的正面朝上，有的反面朝上。
    游戏者根据某些约束翻硬币（如：每次只能翻一或两枚，或者每次只能翻连续的几枚），但他所翻动的
        硬币中，最右边的必须是从正面翻到反面。
    谁不能翻谁输。

    需要先开动脑筋把游戏转化为其他的取石子游戏之类的,然后用如下定理解决:
    局面的 SG 值等于局面中每个正面朝上的棋子单一存在时的 SG 值的异或和。

无向图删边游戏
    一个无向连通图，有一个点作为图的根。
    游戏者轮流从图中删去边， 删去一条边后，不与根节点相连的部分将被移走。
    谁无路可走谁输。

    对于这个模型，有一个著名的定理——Fusion Principle:
    我们可以对无向图做如下改动：将图中的任意一个偶环缩成一个新点，任意一个奇环缩成一个新点加一
        个新边；所有连到原先环上的边全部改为与新点相连。 这样的改动不会影响图的 SG 值。
```

4.7 异或线性基

```cpp
//Author: Menci
struct LinearBasis {
    long long a[MAXL + 1];

    LinearBasis() {
        std::fill(a, a + MAXL + 1, 0);
    }

    LinearBasis(long long *x, int n) {
        build(x, n);
    }

    void insert(long long t) {
        for (int j = MAXL; j >= 0; j--) {
            if (!t) return;
            if (!(t & (1ll << j))) continue;

            if (a[j]) {t ^= a[j];
            } else {
```

```
                for (int k = 0; k < j; k++) {
                    if (t & (1ll << k)) {
                        t ^= a[k];
                    }
                }
                for (int k = j + 1; k <= MAXL; k++) {
                    if (a[k] & (1ll << j)) {
                        a[k] ^= t;
                    }
                }
                a[j] = t;
                break;
            }
        }
    }

    // 数组 x 表示集合 S，下标范围 [1...n]
    void build(long long *x, int n) {
        std::fill(a, a + MAXL + 1, 0);
        for (int i = 1; i <= n; i++) {
            insert(x[i]);
        }
    }

    long long queryMax() {
        long long res = 0;
        for (int i = 0; i <= MAXL; i++) {
            res ^= a[i];
        }
        return res;
    }

    void mergeFrom(const LinearBasis &other) {
        for (int i = 0; i <= MAXL; i++) {
            insert(other.a[i]);
        }
    }

    static LinearBasis merge(const LinearBasis &a, const LinearBasis &b) {
        LinearBasis res = a;
        for (int i = 0; i <= MAXL; i++) res.insert(b.a[i]);
        return res;
    }
};
```

4.8 java 大数开方

```
import java.math.BigInteger;

public class Main {
    static BigInteger n,mod;
    public static BigInteger Sqrt(BigInteger c) {
```

```
        if(c.compareTo(BigInteger.ONE)<=0)
            return c;
        BigInteger temp=null,x;
        x=c.shiftRight((c.bitLength()+1)/2);
        while(true) {
            temp=x;
            x=x.add(c.divide(x)).shiftRight(1);
            if(temp.equals(x)||x.add(BigInteger.ONE).equals(temp)) break;
        }
        return x;
    }
    public static boolean judge(BigInteger c) {
        BigInteger x=Sqrt(c);
        if(x.multiply(x).equals(c)) {
            return true;
        } else {
            return false;
        }
    }
}
```

4.9 单纯形法

```
// 单纯形解线性规划 by zimpha
// 给出m个这样的约束条件: sum(A[i]*X[i])<=B
// 求出X的解, 在满足X[i]>=0的情况下, sum(C[i]*X[i])达到最大
#include <cstdio>
#include <cstring>
#include <algorithm>
#define fo(i,a,b) for(int i=a;i<=b;i++)
using namespace std;
typedef long double db;
const int N=25;
db a[N][N], eps=1e-9;
int id[N*2],n,m,t,x;
double ans[N*2];
bool pd;
db abs(db x) {return x<0?-x:x;}
void pivot(int l,int e) {
    swap(id[n+l],id[e]);
    db t=a[l][e];a[l][e]=1;
    fo(i,0,n) a[l][i]/=t;
    fo(i,0,m)
        if (i!=l&&abs(a[i][e])>eps) {
            db t=a[i][e];a[i][e]=0;
            fo(j,0,n) a[i][j]-=t*a[l][j];
        }
}
void prepare() {
    while (1) {
        int l=0,e=0;
        fo(i,1,m) if (a[i][0]<-eps&&(!l||(rand()&1))) l=i;
```

```
        if (!l) break;
        fo(i,1,n) if (a[l][i]<-eps&&(!e||(rand()&1))) e=i;
        if (!e) {pd=1;return;}
        pivot(l,e);
    }
}
void solve() {
    while (1) {
        int l=0,e=0;db mn=1e18;
        fo(i,1,n) if (a[0][i]>eps) {e=i;break;}
        if (!e) break;
        fo(i,1,m)
            if (a[i][e]>eps&&a[i][0]/a[i][e]<mn) {
                mn=a[i][0]/a[i][e];
                l=i;
            }
        if (!l) {pd=1;return;}
        pivot(l,e);
    }
}
int main() {
    srand(233);
    scanf("%d%d%d",&n,&m,&t);
    fo(i,1,n) scanf("%d",&x),a[0][i]=x;
    fo(i,1,m) {
        fo(j,1,n) scanf("%d",&x),a[i][j]=x;
        scanf("%d",&x);
        a[i][0]=x;
    }
    fo(i,1,n+m) id[i]=i;
    prepare();
    if (pd) { //不存在满足所有约束的解
        printf("Infeasible\n");
        return 0;
    }
    pd=0;
    solve();
    if (pd) { //对于任意的 M，都存在一组解使得目标函数的值大于 M
        printf("Unbounded\n");
        return 0;
    }
    printf("%.15lf\n",-(double)a[0][0]);
    if (t) {
        fo(i,1,m) ans[id[i+n]]=a[i][0];
        fo(i,1,n) printf("%.15lf ",ans[i]);
    }
}
```

4.10 容斥

```
for(int i=0;i<fac.size();i++){ //容斥求 [0,m)内，a1,a2...ak的倍数的和，每个数只记一次
    if(vis[i]==num[i]) continue;// vis数组为 fac[i]这个数要用几次，这里vis[i]=1
```

```
    int n=(m-1)/fac[i];
    ans+=(1+n)*n/2*fac[i]*(vis[i]-num[i]);// num[i]数组为 fac[i]这个数已经用了几次，多了就要减去多用的次数
    n=vis[i]-num[i];          //用于更新已经用的次数
    for(int j=i;j<fac.size();j++){
        if(fac[j]%fac[i]==0){
            num[j]+=n; //在此题中 将所有fac[i]的倍数 更新已使用次数
        }
    }
}
```

4.11 多项式乘法/平方/取模

```
namespace fft {
    typedef int type;
    typedef double db;
    struct cp {
        db x, y;

        cp() { x = y = 0; }

        cp(db x, db y) : x(x), y(y) {}
    };
    inline cp operator+(cp a, cp b) { return cp(a.x + b.x, a.y + b.y); }
    inline cp operator-(cp a, cp b) { return cp(a.x - b.x, a.y - b.y); }
    inline cp operator*(cp a, cp b) { return cp(a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x); }
    inline cp conj(cp a) { return cp(a.x, -a.y); }

    type base = 1;
    vector<cp> roots = {{0, 0},
                        {1, 0}};
    vector<type> rev = {0, 1};
    const db PI = acosl(-1.0);
    void ensure_base(type nbase) {
        if (nbase <= base) {
            return;
        }
        rev.resize(static_cast<unsigned long>(1 << nbase));
        for (type i = 0; i < (1 << nbase); i++) {
            rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (nbase - 1));
        }
        roots.resize(static_cast<unsigned long>(1 << nbase));
        while (base < nbase) {
            db angle = 2 * PI / (1 << (base + 1));
            for (type i = 1 << (base - 1); i < (1 << base); i++) {
                roots[i << 1] = roots[i];
                db angle_i = angle * (2 * i + 1 - (1 << base));
                roots[(i << 1) + 1] = cp(cos(angle_i), sin(angle_i));
            }
            base++;
        }
```

```
    }
    void fft(vector<cp> &a, type n = -1) {
        if (n == -1) {
            n = a.size();
        }
        assert((n & (n - 1)) == 0);
        type zeros = __builtin_ctz(n);
        ensure_base(zeros);
        type shift = base - zeros;
        for (type i = 0; i < n; i++) {
            if (i < (rev[i] >> shift)) {
                swap(a[i], a[rev[i] >> shift]);
            }
        }
        for (type k = 1; k < n; k <<= 1) {
            for (type i = 0; i < n; i += 2 * k) {
                for (type j = 0; j < k; j++) {
                    cp z = a[i + j + k] * roots[j + k];
                    a[i + j + k] = a[i + j] - z;
                    a[i + j] = a[i + j] + z;
                }
            }
        }
    }
    vector<cp> fa, fb;
    vector<type> multiply(vector<type> &a, vector<type> &b) {
        type need = a.size() + b.size() - 1;
        type nbase = 0;
        while ((1 << nbase) < need) nbase++;
        ensure_base(nbase);
        type sz = 1 << nbase;
        if (sz > (type) fa.size())
            fa.resize(static_cast<unsigned long>(sz));
        for (type i = 0; i < sz; i++) {
            type x = (i < (type) a.size() ? a[i] : 0);
            type y = (i < (type) b.size() ? b[i] : 0);
            fa[i] = cp(x, y);
        }
        fft(fa, sz);
        cp r(0, -0.25 / sz);
        for (type i = 0; i <= (sz >> 1); i++) {
            type j = (sz - i) & (sz - 1);
            cp z = (fa[j] * fa[j] - conj(fa[i] * fa[i])) * r;
            if (i != j) {
                fa[j] = (fa[i] * fa[i] - conj(fa[j] * fa[j])) * r;
            }
            fa[i] = z;
        }
        fft(fa, sz);
        vector<type> res(static_cast<unsigned long>(need));
        for (type i = 0; i < need; i++) {
            res[i] = fa[i].x + 0.5;
        }
        return res;
```

```
    }
    vector<type> multiply_mod(vector<type> &a, vector<type> &b, type m, type eq = 0) {
        type need = a.size() + b.size() - 1;
        type nbase = 0;
        while ((1 << nbase) < need) nbase++;
        ensure_base(nbase);
        type sz = 1 << nbase;
        if (sz > (type) fa.size()) {
            fa.resize(static_cast<unsigned long>(sz));
        }
        for (type i = 0; i < (type) a.size(); i++) {
            type x = (a[i] % m + m) % m;
            fa[i] = cp(x & ((1 << 15) - 1), x >> 15);
        }
        fill(fa.begin() + a.size(), fa.begin() + sz, cp{0, 0});
        fft(fa, sz);
        if (sz > (type) fb.size()) {
            fb.resize(static_cast<unsigned long>(sz));
        }
        if (eq) {
            copy(fa.begin(), fa.begin() + sz, fb.begin());
        } else {
            for (type i = 0; i < (type) b.size(); i++) {
                type x = (b[i] % m + m) % m;
                fb[i] = cp(x & ((1 << 15) - 1), x >> 15);
            }
            fill(fb.begin() + b.size(), fb.begin() + sz, cp{0, 0});
            fft(fb, sz);
        }
        db ratio = 0.25 / sz;
        cp r2(0, -1);
        cp r3(ratio, 0);
        cp r4(0, -ratio);
        cp r5(0, 1);
        for (type i = 0; i <= (sz >> 1); i++) {
            type j = (sz - i) & (sz - 1);
            cp a1 = (fa[i] + conj(fa[j]));
            cp a2 = (fa[i] - conj(fa[j])) * r2;
            cp b1 = (fb[i] + conj(fb[j])) * r3;
            cp b2 = (fb[i] - conj(fb[j])) * r4;
            if (i != j) {
                cp c1 = (fa[j] + conj(fa[i]));
                cp c2 = (fa[j] - conj(fa[i])) * r2;
                cp d1 = (fb[j] + conj(fb[i])) * r3;
                cp d2 = (fb[j] - conj(fb[i])) * r4;
                fa[i] = c1 * d1 + c2 * d2 * r5;
                fb[i] = c1 * d2 + c2 * d1;
            }
            fa[j] = a1 * b1 + a2 * b2 * r5;
            fb[j] = a1 * b2 + a2 * b1;
        }
        fft(fa, sz);
        fft(fb, sz);
        vector<type> res(static_cast<unsigned long>(need));
```

```
147         for (type i = 0; i < need; i++) {
148             long long aa = fa[i].x + 0.5;
149             long long bb = fb[i].x + 0.5;
150             long long cc = fa[i].y + 0.5;
151             res[i] = (aa + ((bb % m) << 15) + ((cc % m) << 30)) % m;
152         }
153         return res;
154     }
155     vector<type> square(vector<type> &a) {
156         return multiply(a, a);
157     }
158     vector<type> square_mod(vector<type> &a, type m) {
159         return multiply_mod(a, a, m, 1);
160     }
161     vector<type> kiss_me(vector<type>&b, long long k, type mod) {
162         vector<type> a = b;
163         vector<type> res(1, 1);
164         for (; k; k >>= 1, a = square_mod(a, mod)) {
165             if (k & 1) {
166                 res = multiply_mod(res, a, mod);
167             }
168         }
169         return res;
170     }
171     pair<vector<type>, vector<type> > mul2(vector<type>&b, long long k) {
172         return make_pair(kiss_me(b, k, (type)1e9 + 7), kiss_me(b, k, (type)1e9 + 9));
173     }
174     vector<vector<type> > muln(vector<type>&b, long long k, vector<int> mod_list) {
175         vector< vector<type> > res(mod_list.size());
176         for (int i = 0; i < mod_list.size(); ++i) {
177             res[i] = kiss_me(b, k, mod_list[i]);
178         }
179         return res;
180     }
181 };
```

4.12 快速傅里叶变换

```
1  const double PI = acos(-1.0);
2  //复数结构体
3  struct Complex {
4      double x, y; //实部和虚部 x+yi
5      Complex(double _x = 0.0, double _y = 0.0) { x = _x, y = _y; }
6      Complex operator-(const Complex& b) const { return Complex(x - b.x, y - b.y); }
7      Complex operator+(const Complex& b) const { return Complex(x + b.x, y + b.y); }
8      Complex operator*(const Complex& b) const { return Complex(x * b.x - y * b.y, x * b.y + y
           * b.x); }
9  };
10 /*
11 * 进行FFT和IFFT前的反转变换。
12 * 位置i和 （i二进制反转后位置） 互换
13 * len必须取2的幂
```

```
*/
void change(Complex y[], int len) {
    for (int i = 1, j = len / 2; i < len - 1; i++) {
        if (i < j) std::swap(y[i], y[j]);
        //交换互为小标反转的元素，i<j保证交换一次
        //i做正常的+1，j左反转类型的+1,始终保持i和j是反转的
        int k = len / 2;
        while (j >= k) j -= k, k /= 2;
        if (j < k) j += k;
    }
}

/*
* 做FFT
* len必须为2^k形式，
* on==1时是DFT，on==-1时是IDFT
*/
void fft(Complex y[], int len, int on) {
    change(y, len);
    for (int h = 2; h <= len; h <<= 1) {
        Complex wn(cos(-on * 2 * PI / h), sin(-on * 2 * PI / h));
        for (int j = 0; j < len; j += h) {
            Complex w(1, 0);
            for (int k = j; k < j + h / 2; k++) {
                Complex u = y[k];
                Complex t = w * y[k + h / 2];
                y[k] = u + t, y[k + h / 2] = u - t;
                w = w * wn;
            }
        }
    }
    if (on == -1) for (int i = 0; i < len; i++) y[i].x /= len;
}
```

4.13 快速数论变换

```
// ——
// 模数P为费马素数，G为P的原根。
// $G^{\frac{P-1}{n}}$具有和$w_n=e^{\frac{2i\pi}{n}}$相似的性质。
// 具体的P和G可参考1.11
// ——

const int mod = 119 << 23 | 1;
const int G = 3;
int wn[20];

void getwn() { //  千万不要忘记
    for (int i = 0; i < 20; i++) wn[i] = Pow(G, (mod - 1) / (1 << i), mod);
}

void change(int y[], int len) {
    for (int i = 1, j = len / 2; i < len - 1; i++) {
```

```
        if (i < j) swap(y[i], y[j]);
        int k = len / 2;
        while (j >= k) j -= k, k /= 2;
        if (j < k) j += k;
    }
}

void ntt(int y[], int len, int on) {
    change(y, len);
    for (int h = 2, id = 1; h <= len; h <<= 1, id++) {
        for (int j = 0; j < len; j += h) {
            int w = 1;
            for (int k = j; k < j + h / 2; k++) {
                int u = y[k] % mod;
                int t = 1LL * w * (y[k + h / 2] % mod) % mod;
                y[k] = (u + t) % mod, y[k + h / 2] = ((u - t) % mod + mod) % mod;
                w = 1LL * w * wn[id] % mod;
            }
        }
    } if (on == -1) {
        //  原本的除法要用逆元
        int inv = Pow(len, mod - 2, mod);
        for (int i = 1; i < len / 2; i++) swap(y[i], y[len - i]);
        for (int i = 0; i < len; i++) y[i] = 1LL * y[i] * inv % mod;
    }
}
```

4.14 快速沃尔什变换

```
void fwt(int f[], int m) {
    int n = __builtin_ctz(m);
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            if (j & (1 << i)) {
                int l = f[j ^ (1 << i)], r = f[j];
                f[j ^ (1 << i)] = l + r, f[j] = l - r;
                // or: f[j] += f[j ^ (1 << i)];
                // and: f[j ^ (1 << i)] += f[j];
            }
}

void ifwt(int f[], int m) {
    int n = __builtin_ctz(m);
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            if (j & (1 << i)) {
                int l = f[j ^ (1 << i)], r = f[j];
                f[j ^ (1 << i)] = (l + r) / 2, f[j] = (l - r) / 2;
                // 如果有取模需要使用逆元
                // or: f[j] -= f[j ^ (1 << i)];
                // and: f[j ^ (1 << i)] -= f[j];
            }
```

```
}
```

4.15 分治 fft

```
//dp[i] = sigma(a[j] * dp[i-j]) (j < i);
const int maxn = "Edit";
int dp[maxn], a[maxn];
Complex x[maxn<<2], y[maxn<<2];
void solve(int L, int R){
    if(L == R) return ;
    int mid = (L + R) >> 1;
    solve(L, mid);
    int len = 1, len1 = R - L + 1;
    while(len <= len1) len <<= 1;
    for(int i = 0; i < len1; ++i) x[i] = Complex(a[i], 0);
    for(int i = len1; i <= len; ++i) x[i] = Complex(0, 0);
    for(int i = L; i <= mid; ++i)
        y[i-L] = Complex(dp[i], 0);
    for(int i = mid - L + 1; i <= len; ++i) y[i] = Complex(0, 0);
    fft(x, len, 1);
    fft(y, len, 1);
    for(int i = 0; i < len; ++i) x[i] = x[i] * y[i];
    fft(x, len, -1);
    for(int i = mid + 1; i <= R; ++i){
        dp[i] += x[i-L].x + 0.5;
    }
    solve(mid + 1, R);
}
```

4.16 polya 项链染色

```
/*
 *  c种颜色的珠子，组成长为s的项链，项链没有方向和起始位置
 */
int gcd(int a, int b) {
    return b ? gcd(b, a % b) : a;
}

int main(int argc, const char * argv[]) {
    int c, s;
    while (std::cin >> c >> s) {
        int k;
        long long p[64];
        p[0] = 1;                   // power of c
        for (k = 0; k < s; k++) {
            p[k + 1] = p[k] * c;
        }
        // reflection part
        long long count = s & 1 ? s * p[s / 2 + 1] : (s / 2) * (p[s / 2] + p[s / 2 + 1]);
```

```
        // rotation part
        for (k = 1 ; k <= s ; k++) {
            count += p[gcd(k, s)];
            count /= 2 * s;
        }
        std::cout << count << '\n';
    }
    return 0;
}
```

4.17 染色多项式

```
完全图: t(t-1)(t-2)...(t-(n-1))
有n个顶点的树: t(t-1)^(n-1)
环: (t-1)^n + (-1)^n(t-1)
加边减边: P(G)=P(G+e)+P(G•e)
```

4.18 错位排列递推公式

```
d[1] = 0;
d[2] = 1;
for (int i = 3; i < maxn; i++) {
    d[i] = (i-1)*(d[i-1]+d[i-2]);
}
```

4.19 BBP 公式求 pi 十六进制的第 k 位

```
// BBP算法 询问十六进制下圆周率的第n位
// 时间复杂度 O(nlogn)

using ll = long long;

ll remain( ll m, ll n, ll k, ll extra) {
    ll temp1=1,temp2=1;
    if (n==0) return extra%k;
    if (n==1) return (m*extra)%k;
    while(n>1) {
        temp1=m;
        temp1*=temp1;
        if(temp1>=k)temp1%=k;
        if(n%2==1)temp2=m*temp2;
        temp2%=k;
        m=temp1;
        n=n/2;
    }
    temp1=(temp1*temp2)%k;
    return (temp1*extra)%k;
```

```
}

ll remain_nex( ll m, ll n, ll k) {
    ll temp1 = 1, temp2 = 1;
    if (n == 0) return 1;
    if (n == 1) return m%k;
    while (n>1) {
        temp1 = m;
        temp1 *= temp1;
        if (temp1 >= k) temp1%=k;
        if (n % 2 == 1) temp2=m*temp2;
        temp2 %= k;
        m = temp1;
        n = n / 2;
    }
    return (temp1*temp2)%k;
}

char compute_n(int j) {
     ll m;
    long double sum=0,temp=1.0,temp1;
    int i;
    j--;
    temp1=1.0;
    for (i=0;i<=j;i++) sum=sum+remain(16,j-i,8*i+1,4)/(long double)(8.0*i+1);
    for (i=0;i<=j;i++) sum=sum-remain(16,j-i,8*i+4,2)/(long double)(8.0*i+4);
    for (i=0;i<=j;i++) sum=sum-remain_nex(16,j-i,8*i+5)/(long double)(8.0*i+5);
    for (i=0;i<=j;i++) sum=sum-remain_nex(16,j-i,8*i+6)/(long double)(8.0*i+6);
    temp=1.0;
    for (;temp>0.000001;i++) {
        temp=temp/16.0;sum=sum+(4.0/(8*i+1)-2.0/(8*i+4)-1.0/(8*i+5)-1.0/(8*i+6))*temp;
    }
    for (;sum<0;) sum=sum+16;
    m=sum;
    sum=sum-m;
    sum=sum*16;
    m=sum;
    return (char)(m<10 ? m+48: m+55);
}
```

第五章　图论

5.1　前向星

```cpp
const int maxn = 10005;   //点的最大个数

int head[maxn], cnt=0;//head用来表示以i为起点的第一条边存储的位置，cnt读入边的计数器

struct Edge {
    int next; //同一起点的上一条边的储存位置
    int to; //第i条边的终点
    int w; //第i条边权重
};

Edge edge[maxn];

void addedge(int u,int v,int w) {
    edge[cnt].w = w;
    edge[cnt].to = v;
    edge[cnt].next = head[u];
    head[u] = cnt++;
}

void traverse() {
    for(int i=0; i<=n; i++) {
        for(int j=head[i]; j!=-1; j=edge[j].next) {
            std::cout << i << " " << head[i].to << " " << head[i].w << '\n';
        }
    }
}
```

5.2　并查集

```cpp
int fa[N];

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

int find(int u) {
    return fa[u] == u ? fa[u] : fa[u] = find(fa[u]);
}
```

```

void unin(int u, int v) {
    fa[find(v)] = find(u);
}
```

5.3 可撤销并查集（按秩合并）

```cpp
#include <iostream>
#include <stack>
#include <utility>

class UFS {
    private:
        int *fa, *rank;
        std::stack <std::pair <int*, int> > stk ;
    public:
        UFS() {}
        UFS(int n) {
            fa = new int[(const int)n + 1];
            rank = new int[(const int)n + 1];
            memset (rank, 0, n+1);
            for (int i = 1; i <= n; ++i) {
                fa [i] = i;
            }
        }
        inline int find(int x) {
            while (x ^ fa[x]) {
                x = fa[x];
            }
            return x ;
        }
        inline int Join (int x, int y) {
            x = find(x), y = find(y);
            if (x == y) {
                return 0;
            }
            if (rank[x] <= rank[y]) {
                stk.push(std::make_pair (fa + x, fa[x]));
                fa[x] = y;
                if (rank[x] == rank[y]) {
                    stk.push(std::make_pair (rank + y, rank[y]));
                    ++rank[y];
                    return 2;
                }
                return 1 ;
            }
            stk.push(std::make_pair(fa + y, fa [y]));
            return fa[y] = x, 1;
        }
        inline void Undo ( )  {
            *stk.top( ).first = stk.top( ).second ;
            stk.pop( ) ;
```

```
        }
}T;
```

5.4 Kruskal 最小生成树

```
#include <vector>
#include <algorithm>

#define maxm 1000
#define maxn 1000

class Kruskal {
    struct UdEdge {
        int u, v, w;
        UdEdge(){}
        UdEdge(int u,int v,int w):u(u), v(v), w(w){}
    };
    int N, M;
    UdEdge pool[maxm];
    UdEdge *E[maxm];
    int P[maxn];
    int Find(int x){
        if(P[x] == x)
            return x;
        return P[x] = Find(P[x]);
    }
    public:
    static bool cmp(const UdEdge *a, const UdEdge *b) {
        return a->w < b->w;
    }
    void Clear(int n) {
        N = n;
        M = 0;
    }
    void AddEdge(int u, int v, int w) {
        pool[M] = UdEdge(u, v, w);
        E[M] = &pool[M];
        ++M;
    }
    int Run() {
        int i, ans=0;
        for(i = 1; i <= N; ++i)
            P[i] = i;
        std::sort(E, E+M, cmp);
        for(i = 0; i < M; ++i) {
            UdEdge *e = E[i];
            int x = Find(e->u);
            int y = Find(e->v);
            if(x != y) {
                P[y] = x;
                ans += e->w;
            }
```

```
        }
        return ans;
    }
};
```

5.5 Prim 最小生成树

```
int d[maxn][maxn];
int lowc[maxn];
int vis[maxn];

int prim(int n) {
    int ans = 0;
    memset(vis, 0, sizeof(vis));
    for (int i = 2; i <= n; i++)
        lowc[i] = d[1][i];
    vis[1] = 1;
    for (int i = 1; i < n; i++) {
        int minc = INF;
        int p = -1;
        for (int j = 1; j <= n; j++) {
            if (!vis[j] && minc > lowc[j]) {
                minc = lowc[j];
                p = j;
            }
        }
        vis[p] = 1;
        ans += minc;
        for (int j = 1; j <= n; j++) {
            if (!vis[j] && lowc[j] > d[p][j])
                lowc[j] = d[p][j];
        }
    }
    return ans;
}
```

5.6 SPFA 最短路

```
#include <queue>
#include <cstring>
#include <vector>
#define maxn 10007
#define INF 0x7FFFFFFF
using namespace std;
struct Edge{
    int v,w;
    Edge(int v,int w):v(v),w(w){}
};
int d[maxn];
```

```
bool inq[maxn];
vector<Edge> G[maxn];
void SPFA(int s){
    queue<int> q;
    memset(inq,0,sizeof(inq));
    for(int i=0;i<maxn;++i)
        d[i]=INF;
    d[s]=0;
    inq[s]=1;
    q.push(s);
    int u;
    while(!q.empty()){
        u=q.front();
        q.pop();
        inq[u]=0;
        for(vector<Edge>::iterator e=G[u].begin();e!=G[u].end();++e) {
            if(d[e->v]>d[u]+e->w){
                d[e->v]=d[u]+e->w;
                if(!inq[e->v]){
                    q.push(e->v);
                    inq[e->v]=1;
                }
            }
        }
    }
}
```

5.7 dijkstra 最短路

```
#include <vector>
#include <queue>
#define INF 0x7FFFFFFF
#define maxn 1000
using namespace std;
class Dijkstra{
private:
    struct HeapNode{
        int u;
        int d;
        HeapNode(int u, int d) :u(u), d(d){}
        bool operator < (const HeapNode &b) const{
            return d > b.d;
        }
    };
    struct Edge{
        int v;
        int w;
        Edge(int v, int w) :v(v), w(w){}
    };
    vector<Edge>G[maxn];
    bool vis[maxn];
public:
```

```
    int d[maxn];
    void clear(int n){
        int i;
        for(i=0;i<n;++i)
            G[i].clear();
        for(i=0;i<n;++i)
            d[i] = INF;
        memset(vis, 0, sizeof(vis));
    }
    void AddEdge(int u, int v, int w){
        G[u].push_back(Edge(v, w));
    }
    void Run(int s){
        int u;
        priority_queue<HeapNode> q;
        d[s] = 0;
        q.push(HeapNode(s, 0));
        while (!q.empty()){
            u = q.top().u;
            q.pop();
            if (!vis[u]){
                vis[u] = 1;
                for (vector<Edge>::iterator e = G[u].begin(); e != G[u].end(); ++e)
                    if (d[e->v] > d[u] + e->w){
                        d[e->v] = d[u] + e->w;
                        q.push(HeapNode(e->v, d[e->v]));
                    }
            }
        }
    }
};
```

5.8 Floyd 任意两点间最短路

```
//#define inf maxn*maxw+10
for(int i = 0; i < n; i++) {
    for(int j = 0; j < n; j++) {
        d[i][j] = inf;
    }
}
d[0][0] = 0;
for(int k = 0; k < n; k++) {
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            d[i][j] = std::min(d[i][j], d[i][k] + d[k][j]);
        }
    }
}
```

5.9 2-SAT 问题

```
class TwoSAT{
    private:
        const static int maxm=maxn*2;

        int S[maxm],c;
        vector<int> G[maxm];

        bool DFS(int u){
            if(vis[u^1])
                return false;
            if(vis[u])
                return true;
            vis[u]=1;
            S[c++]=u;
            for(auto &v:G[u])
                if(!DFS(v))
                    return false;
            return true;
        }

    public:
        int N;
        bool vis[maxm];

        void Clear(){
            for(int i=2;i<(N+1)*2;++i)
                G[i].clear();
            memset(vis,0,sizeof(bool)*(N+1)*2);
        }

        void AddClause(int x,int xv,int y,int yv){
            x=x*2+xv;
            y=y*2+yv;
            G[x].push_back(y);
            G[y].push_back(x);
        }

        bool Solve(){
            for(int i=2;i<(N+1)*2;i+=2)
                if(!vis[i]&&!vis[i+1]){
                    c=0;
                    if(!DFS(i)){
                        while(c>0)
                            vis[S[--c]]=0;
                        if(!DFS(i+1))
                            return false;
                    }
                }
            return true;
        }
    };
```

5.10 tarjan 强连通分量

```cpp
//written by kuangbin
const int maxn = "Edit";
const int maxm = "Edit";

struct node {
    int to, next;
} edge[maxm];

int head[maxn], tot;
int low[maxn], dfn[maxn], stack[maxn], belong[maxn];
int cur, top, scc;
bool instack[maxn];
int num[maxn];

int in[maxn], out[maxn];

void init() {
    tot = 0;
    std::fill(head, head+maxn, -1);
    std::fill(in, in+maxn, 0);
    std::fill(out, out+maxn, 0);
}

void addedge(int u, int v) {
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
}

void tarjan(int u) {
    int v;
    low[u] = dfn[u] = ++cur;
    stack[top++] = u;
    instack[u] = 1;
    for (int i = head[u]; i != -1; i = edge[i].next) {
        v = edge[i].to;
        if (!dfn[v]) {
            tarjan(v);
            if (low[u] > low[v]) low[u] = low[v];
        } else if (instack[v] && low[u] > dfn[v]) {
            low[u] = dfn[v];
        }
    }
    if (low[u] == dfn[u]) {
        scc++;
        do {
            v = stack[--top];
            instack[v] = 0;
            belong[v] = scc;
```

```
            num[scc]++;
        } while (v != u);
    }
}

void solve(int n) {
    std::fill(dfn, dfn+maxn, 0);
    std::fill(instack, instack+maxn, 0);
    std::fill(num, num+maxn, 0);
    cur = scc = top = 0;
    for (int i = 1; i <= n; i++) {
        if (!dfn[i]) {
            tarjan(i);
        }
    }
}

void in_out(int n) {
    for (int u = 1; u <= n; u++) {
        for (int i = head[u]; i != -1; i = edge[i].next) {
            if (belong[u] != belong[edge[i].to]) {
                in[belong[edge[i].to]]++;
                out[belong[u]]++;
            }
        }
    }
}
```

5.11 Kosaraju 强连通分量

```
#include <vector>
#include <algorithm>

const int maxn = "Edit";

class Kosaraju {
private:
    std::vector<int> s[maxn],g[maxn];
    bool vis[maxn]={0};
public:
    int st[maxn], top=0, contract[maxn]={0};
    int n, m;
    void dfs(int x){
        vis[x]=1;
        for(int i=0;i<(int)s[x].size();++i){
            if(!vis[s[x][i]])dfs(s[x][i]);
        }
        st[top++]=x;
    }
    void dfs2(int x,int k){
        if(contract[x])return;
        contract[x]=k;/*x屬於第k個contract*/
```

```
        for(int i=0;i<(int)g[x].size();++i){
            dfs2(g[x][i],k);
        }
    }
    void addedge(int a, int b) {
        s[a].push_back(b);
        g[b].push_back(a);
    }
    void kosaraju() {
        for(int i=0;i<n;++i){
            if(!vis[i]) {
                dfs(i);
            }
        }
        for(int i=top-1,t=0;i>=0;--i){
            if(!contract[st[i]]) {
                dfs2(st[i],++t);
            }
        }
    }
};
```

5.12 点双联通分量

```
//Author: CookiC
#include <stack>
#include <vector>
#define maxn 1000
using namespace std;

class BCC{
private:
    int clk, cnt;
    int pre[maxn];
    stack<int> s;

    int DFS(int u,int f){
        int lowu = pre[u] = ++clk;
        int child = 0;
        s.push(u);
        for (auto it = G[u].begin(); it != G[u].end(); ++it){
            int v = *it;
            if (!pre[v]){
                s.push(u);
                ++child;
                int lowv = DFS(v, u);
                if (lowu > lowv)
                    lowu = lowv;
                if (lowv >= pre[u]){
                    iscut[u] = 1;
                    ++cnt;
                    int x;
```

```
                    do{
                        x = s.top();
                        s.pop();
                        if (num[x] != cnt)
                            num[x] = cnt;
                    }while (x != u);
                }
            }
            else if (pre[v] < pre[u] && v != f){
                if (lowu > pre[v])
                    lowu = pre[v];
            }
        }
        if (f < 0 && child == 1)
            iscut[u] = 0;
        return lowu;
    }
public:
    vector<int> G[maxn];
    bool iscut[maxn];
    int num[maxn];

    void Clear(int n){
        for (int i = 0; i < n; ++i)
            G[i].clear();
    }

    void AddEdge(int u,int v){
        G[u].push_back(v);
        G[v].push_back(u);
    }

    void Find(){
        int i;
        memset(pre, 0, sizeof(pre));
        memset(iscut, 0, sizeof(iscut));
        memset(num,0,sizeof(num));
        clk = cnt = 0;
        for (i = 0; i < n; ++i)
            if (!pre[i]){
                while(!s.empty())
                    s.pop();
                DFS(i,-1);
            }
    }
};
```

5.13 边双联通分量

```
//Author: XieNaoban
//在求桥的基础上修改
#include <algorithm>
```

```cpp
#include <cstring>
#include <vector>
#include <cmath>
#include <set>

class CutEdge {
private:
    int N;
    int clk, pre[Maxn];

    int DFS(int u, int f) {
        int lowu = pre[u] = ++clk;
        for (auto e = G[u].begin(); e != G[u].end(); ++e) {
            int v = *e;
            if (!pre[v]) {
                int lowv = DFS(v, u);
                lowu = min(lowu, lowv);
                if (lowv > pre[u]) {
                    Cut[u].insert(v);
                    Cut[v].insert(u);
                }
            }
            else if (pre[u] > pre[v]) {
                int cnt = 0; //重复边的处理
                for (const auto &e : G[u]) if (e == v) ++cnt;
                if (cnt > 1 || v != f) {
                    lowu = min(lowu, pre[v]);
                }
            }
        }
        return lowu;
    }

    void DFS2(int u, int id) {
        ID[u] = id;
        for (const auto &v : G[u]) if (!ID[v]) {
            if (Cut[u].count(v)) {
                ++Num;
                G2[id].push_back(Num);
                G2[Num].push_back(id);
                DFS2(v, Num);
            }
            else DFS2(v, id);
        }
    }

public:
    vector<int> G[Maxn];
    set<int> Cut[Maxn];

    vector<int> G2[Maxn]; //缩点后的图（以ID为结点）
    int ID[Maxn]; //每个点所在的子图
    int Num; //ID个数

```

```
    void Clear(int n) {
        N = n;
        memset(ID, 0, sizeof(ID));
        memset(pre, 0, sizeof(pre));
        for (int i = 1; i <= N; ++i) {
            G[i].clear();
            G2[i].clear();
            Cut[i].clear();
        }
        clk = 0;
        Num = 1;
    }

    void AddEdge(int u, int v) {
        G[u].push_back(v);
        G[v].push_back(u);
    }

    void Find() {
        for (int i = 1; i <= N; ++i)
            if (!pre[i])
                DFS(i, -1);
    }

    //求边双联通部分
    int BCC() { //要求先运行Find
        DFS2(1, Num);
        return Num;
    }
};
```

5.14 求桥

```
class bcc_bridges {
    public:
        struct edge {
            int y;
            edge * next;
        };
        edge e[M], *li[N];
        int etop;
        void init() {
            memset(li, 0, sizeof(li));
            etop = 0;
        }
        inline void add_edge(int u, int v) {
            e[etop].y = v;
            e[etop].next = li[u];
            li[u] = &e[etop++];
        }
        std::vector<std::pair<int, int>> briges;
        int dfn[N],low[N];
```

```
        int clk;
        void dfs(int u, int fa) {
            dfn[u]=low[u]=++clk;
            int v;
            for (edge * t = li[u]; t; t = t->next) {
                v = t->y;
                if (!dfn[v]) {
                    dfs(v,u);
                    low[u]=std::min(low[u],low[v]);
                    if(low[v] > dfn[u])
                        briges.emplace_back(u, v); // u <-> v is a bridge
                }
                else if(dfn[v] < dfn[u] && v != fa)
                    low[u]=std::min(low[u],dfn[v]);
            }
        }
        void find_bridge(int n) {
            clk = 0;
            std::fill(dfn + 1, dfn + n + 1, 0);
            std::fill(low, low + n + 1, 0);
            for (int i = 1; i <= n; ++i) {
                if (!dfn[i])
                    dfs(i, 0);
            }
        }
    };
```

5.15 欧拉回路

```
const int maxn = 100;

int n;
int step;
int path[maxn];

void find_path_u(int now, int mat[][maxn]) {
    for (int i=n-1; i>=0; i--) {
        while (mat[now][i]) {
            mat[now][i]--, mat[i][now]--;
            find_path_u(i, mat);
        }
    }
    path[step++] = now;
}

void find_path_d(int now, int mat[][maxn]) {
    for (int i=n-1; i>=0; i--) {
        while (mat[now][i]) {
            mat[now][i]--;
            find_path_d(i, mat);
        }
    }
```

```
    path[step++] = now;
}


int euler_circuit(int start, int mat[][maxn]) {
    step = 0;
    find_path_u(start, mat);
    // find_path_d(start, mat);
    return step;
}

int main() {

}
```

5.16 k 短路

```
#include <cstdio>
#include <cstring>
#include <queue>
#include <vector>
#include <algorithm>
using namespace std;

const int maxn = 10000 + 5;
const int INF = 0x3f3f3f3f;
int s, t, k;

bool vis[maxn];
int dist[maxn];

struct Node {
    int v, c;
    Node (int _v = 0, int _c = 0) : v(_v), c(_c) {}
    bool operator < (const Node &rhs) const {
        return c + dist[v] > rhs.c + dist[rhs.v];
    }
};

struct Edge {
    int v, cost;
    Edge (int _v = 0, int _cost = 0) : v(_v), cost(_cost) {}
};

vector<Edge>E[maxn], revE[maxn];

void Dijkstra(int n, int s) {
    memset(vis, false, sizeof(vis));
    for (int i = 1; i <= n; i++) dist[i] = INF;
    priority_queue<Node>que;
    dist[s] = 0;
    que.push(Node(s, 0));
```

```
    while (!que.empty()) {
        Node tep = que.top(); que.pop();
        int u = tep.v;
        if (vis[u]) continue;
        vis[u] = true;
        for (int i = 0; i < (int)E[u].size(); i++) {
            int v = E[u][i].v;
            int cost = E[u][i].cost;
            if (!vis[v] && dist[v] > dist[u] + cost) {
                dist[v] = dist[u] + cost;
                que.push(Node(v, dist[v]));
            }
        }
    }
}

int astar(int s) {
    priority_queue<Node> que;
    que.push(Node(s, 0)); k--;
    while (!que.empty()) {
        Node pre = que.top(); que.pop();
        int u = pre.v;
        if (u == t) {
            if (k) k--;
            else return pre.c;
        }
        for (int i = 0; i < (int)revE[u].size(); i++) {
            int v = revE[u][i].v;
            int c = revE[u][i].cost;
            que.push(Node(v, pre.c + c));
        }
    }
    return -1;
}

void addedge(int u, int v, int w) {
    revE[u].push_back(Edge(v, w));
    E[v].push_back(Edge(u, w));
}

int main() {
    int n, m, u, v, w;
    while (scanf("%d%d", &n, &m) != EOF) {
        for (int i = 0; i <= n; i++) {
            E[i].clear();
            revE[i].clear();
        }
        int aaa;
        scanf("%d%d%d%d", &s, &t, &k, &aaa);
        for (int i = 0; i < m; i++) {
            scanf("%d%d%d", &u, &v, &w);
            addedge(u, v, w);
        }
        Dijkstra(n, t);
```

```
90          if (dist[s] == INF) {
91              printf("No Solution\n");
92              continue;
93          }
94          if (s == t) k++;
95          ans = astar(s);
96      }
97      return 0;
98  }
```

5.17 最小环

```
1   int min=INT_MAX;
2
3   for(k=1;k<=n;k++) {
4       for(i=1;i<=n;i++) {
5           for(j=1;j<=n;++j) {
6               if(dist[i][j]!=INF&&map[j][k]!=INF&&map[k][i]!=INF&&dist[i][j]+dist[j][k]+map[k][i
                    ]<mindis) {
7                   mindis=min(mindis,dist[i][j]+map[j][k]+map[k][i]);
8               }
9           }
10      }
11      for(i=1;i<=n;i++) {
12          for(j=1;j<=n;j++) {
13              if(dist[i][k]!=INF&&dist[k][j]!=INF&&dist[i][k]+dist[k][j]<dist[i][j]) {
14                  dist[i][j]=dist[i][k]+dist[k][j];
15                  pre[i][j]=pre[k][j];
16              }
17          }
18      }
19  }
```

5.18 最小树形图

```
1   #include <cstdio>
2   #include <cmath>
3   #define type int
4
5   type c[mm], in[mn], w[mn], ans;
6   int s[mm], t[mm], id[mn], pre[mn], q[mn], vis[mn];
7
8   type Directed_MST(int root,int NV,int NE) {
9       type ret=0, sum=0, tmp;
10      int i, j, u, v, r;
11      bool huan=1;
12      for (i=0;i<=NV;++i) in[i]=0, id[i]=i, pre[i]=-1;
13      while (huan) {
14          for(i=0;i<=NV;++i)
```

```
            if(pre[j=id[i]]>=0) {
                if(pre[i]<0)in[i]+=w[j],id[i]=id[j];
                else in[i]+=w[i],ret+=w[i];
            }
        for(i=0;i<=NV;++i)pre[i]=-1,vis[i]=0;
        for(i=0;i<NE;++i)
            if((u=id[s[i]])!=(v=id[t[i]])&&(w[v]>(tmp=c[i]-in[t[i]])||pre[v]<0))
                pre[v]=u,w[v]=tmp;
        for(i=1;i<=NV;++i)
            if(i!=root&&id[i]==i&&pre[i]<0)return -1;
        for(pre[root]=-1,sum=i=0;i<=NV;++i)
            if(pre[i]>=0)sum+=w[i];
        for(i=huan=0;i<=NV;++i)
            if(!vis[i]) {
                r=0,j=i;
                while(j>=0&&vis[j]>=0) {
                    if(vis[j]>0) {
                        while(q[--r]!=j)id[q[r]]=j,vis[q[r]]=-1;
                        huan=1,vis[j]=-1;
                    }
                    else vis[q[r++]=j]=1,j=pre[j];
                }
                while(r--)vis[q[r]]=pre[q[r]]=-1;
            }
    }
    return ret+sum;
}

int main() {
    int n,m,e,T,cas=0;
    scanf("%d",&T);
    while(T--) {
        scanf("%d%d",&n,&m),--n;
        e=0;
        while(m--)scanf("%d%d%d",&s[e],&t[e],&c[e]),e+=(s[e]!=t[e]);
        ans=Directed_MST(0,n,e);
        if(ans<0)printf("Case␣#%d:␣Possums!\n",++cas);
        else printf("Case␣#%d:␣%d\n",++cas,ans);
    }
    return 0;
}
```

5.19 次小生成树 (Prim)

```
// 0-indexed
bool vis[maxn];
int d[maxn][maxn];
int lowc[maxn];
int pre[maxn];
int Max[maxn][maxn];    // Max[i][j]表示i到j的路径上的最大边权
bool used[maxn][maxn];
int Prim(int n) {
```

```
    int ans = 0;
    memset(vis, false, sizeof(vis));
    memset(Max, 0, sizeof(Max));
    memset(used, false, sizeof(used));
    vis[0] = true;
    pre[0] = -1;
    for (int i = 1; i < n; i++) {
        lowc[i] = d[0][i];
        pre[i] = 0;
    }
    lowc[0] = 0;
    for (int i = 1; i < n; i++) {
        int minc = INF;
        int p = -1;
        for (int j = 0; j < n; j++)
            if (!vis[j] && minc > lowc[j]) {
                minc = lowc[j];
                p = j;
            }
        if (minc == INF)return -1;
        ans += minc;
        vis[p] = true;
        used[p][pre[p]] = used[pre[p]][p] = true;
        for (int j = 0; j < n; j++) {
            if (vis[j]) Max[j][p] = Max[p][j] = max(Max[j][pre[p]], lowc[p]);
            if (!vis[j] && lowc[j] > d[p][j]) {
                lowc[j] = d[p][j];
                pre[j] = p;
            }
        }
    }
    return ans;
}
int SMST(int n, int ans) {
    int Min = INF;
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            if (d[i][j] != INF && !used[i][j])
                Min = min(Min, ans + d[i][j] - Max[i][j]);
    if (Min == INF) return -1;
    return Min;
}
```

5.20 次小生成树 (Kruskal)

```
//1-indexed
struct edge {
    int s, t, w;    //从s到t 权值w
    bool vis;
    edge() {}
    edge(int s, int t, int w) :s(s), t(t), w(w) {}
    bool operator < (const edge e) const {
```

```
        return w < e.w;
    }
}e[maxm];

int pre[maxn];
int Max[maxn][maxn];    // Max[i][j]表示从i到j路径上的最大边权

int find(int x) {
    int r = x, i = x, j;
    while (pre[r] != r)
        r = pre[r];
    while (i != r) {    // 状态压缩
        j = pre[i];
        pre[i] = r;
        i = j;
    }
    return r;
}

int kruskal(int n, int m) { // n为边数 m为点数
    int lef = m - 1, ans = 0;
    memset(Max, 0, sizeof(Max));
    vector<int>v[maxn];
    for (int i = 1; i <= m; i++) {
        pre[i] = i;
        v[i].push_back(i);
    }
    sort(e + 1, e + n + 1);
    for (int i = 1; i <= n; i++) {
        int fs = find(e[i].s), ft = find(e[i].t), len1, len2;
        if (fs != ft) {
            pre[fs] = ft;
            ans += e[i].w;
            lef--; e[i].vis = true;
            len1 = v[fs].size(), len2 = v[ft].size();
            for (int j = 0; j < len1; j++)
                for (int k = 0; k < len2; k++)
                    Max[v[fs][j]][v[ft][k]] = Max[v[ft][k]][v[fs][j]] = e[i].w;
            int tmp[maxn];
            for (int j = 0; j < len1; j++)
                tmp[j] = v[fs][j];
            for (int j = 0; j < len2; j++)
                v[fs].push_back(v[ft][j]);
            for (int j = 0; j < len1; j++)
                v[ft].push_back(tmp[j]);
        }
        if (!lef)break;
    }
    if (lef) ans = -1;  // 图不连通
    return ans;
}

int SMST(int n, int ans) {  // n为边数,ans为最小生成树权值
    int ret = INF;
```

```
    for (int i = 1; i <= n; i++)
        if (!e[i].vis)
            ret = min(ret, ans + e[i].w - Max[e[i].s][e[i].t]);
    if (ret == INF) return -1;
    return ret;
}
```

5.21 最小生成树计数

```
// 无向图，求生成树个数 Determinant算法
ll A[maxn][maxn], B[maxn][maxn];
ll determinant(int n) {
    ll res = 1;
    for (int i = 1; i <= n; i++) {
        if (!B[i][i]) {
            bool flag = false;
            for (int j = i + 1; j <= n; j++) {
                if (B[j][i]) {
                    flag = true;
                    for (int k = i; k<n; k++)
                        swap(B[i][k], B[j][k]);
                    res = -res;
                    break;
                }
            }
            if (!flag) return 0;
        }
        for (int j = i + 1; j <= n; j++) {
            while (B[j][i]) {
                ll t = B[i][i] / B[j][i];
                for (int k = i; k <= n; k++) {
                    B[i][k] = B[i][k] - t * B[j][k];
                    swap(B[i][k], B[j][k]);
                }
                res = -res;
            }
        }
        res *= B[i][i];
    }
    return res;
}
int main()
{
    int n, m, k;
    while (~scanf("%d%d%d", &n, &m, &k)) {
        memset(A, 0, sizeof(A));
        memset(B, 0, sizeof(B));
        for (int i = 1; i <= m; i++) {
            int a, b;
            scanf("%d%d", &a, &b);
            A[a][b] = A[b][a] = 1;
        }
```

```
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                if (i != j && !A[i][j]) {
                    B[i][i]++;
                    B[i][j] = -1;
                }
            }
        }
        n--;
        ll ans = determinant(n);
        printf("%lld\n", ans);
    }
}
```

5.22 最小树形图计数

```
// 有向图最小生成树计数
struct node {
    int a, b, cost;
}edge[maxm];

int n, m, o;
ll ans, mod;
int pre[maxn], ka[maxn];
ll G[maxn][maxn], B[maxn][maxn];
bitset<maxn> vis;
vector<int> v[maxn];

bool cmp(node a, node b) { return a.cost < b.cost; }
int find(int x) { return pre[x] == x ? pre[x] : pre[x] = find(pre[x]); }

ll det(ll a[][maxn], int n) { //Matrix-Tree定理求Kirchhoff矩阵
    for (int i = 0; i<n; i++)
        for (int j = 0; j<n; j++) a[i][j] %= mod;
    ll ret = 1;
    for (int i = 1; i<n; i++) {
        for (int j = i + 1; j<n; j++)
            while (a[j][i]) {
                ll t = a[i][i] / a[j][i];
                for (int k = i; k<n; k++) a[i][k] = (a[i][k] - a[j][k] * t) % mod;
                for (int k = i; k<n; k++) swap(a[i][k], a[j][k]);
                ret = -ret;
            }
        if (a[i][i] == 0) return 0;
        ret = ret * a[i][i] % mod;
    }
    return (ret + mod) % mod;
}

void Matrix_Tree() {
    for (int i = 1; i <= n; i++) { //根据访问标记找出连通分量
        if (vis[i]) {
```

```
            v[find(i)].push_back(i);
            vis[i] = 0;
        }
    }
    for (int i = 1; i <= n; i++) {
        if (v[i].size() > 1) { //枚举连通分量
            memset(B, 0, sizeof(B));
            int len = v[i].size();
            for (int a = 0; a < len; a++) {
                for (int b = a + 1; b < len; b++) {
                    int la = v[i][a], lb = v[i][b];
                    B[b][a] -= G[la][lb];
                    B[a][b] = B[b][a];
                    B[a][a] += G[la][lb];
                    B[b][b] += G[la][lb];
                } //构造矩阵
            }
            ll ret = det(B, len) % mod;
            ans = ans * ret % mod;
            for (int j = 0; j < len; j++)
                pre[v[i][j]] = i;
        }
    }
    for (int i = 1; i <= n; i++) { //连通图缩点+初始化
        pre[i] = find(i);
        ka[i] = pre[i];
        v[i].clear();
    }
}

int main()
{
    while (scanf("%d%d%lld", &n, &m, &mod), n || m || mod) {
        for (int i = 1; i <= m; i++)
            scanf("%d%d%d", &edge[i].a, &edge[i].b, &edge[i].cost);
        sort(edge + 1, edge + m + 1, cmp);
        for (int i = 1; i <= n; i++)
            v[i].clear();
        for (int i = 1; i <= n; i++)
            pre[i] = ka[i] = i;
        vis.reset();
        memset(G, 0, sizeof(G));
        ans = 1;
        o = edge[1].cost;
        for (int i = 1; i <= m; i++) {
            int pa = find(edge[i].a), pb = find(edge[i].b);
            if (pa != pb) {
                vis[pa] = 1;
                vis[pb] = 1;
                ka[find(pa)] = find(pb);
                G[pa][pb]++;
                G[pb][pa]++;
            }
            if (i == m || edge[i + 1].cost != o) { //所有相同的边并成一组
```

```
                Matrix_Tree();
                o = edge[i + 1].cost;
            }
        }
        bool done = true;
        for (int i = 2; i <= n; i++) {
            if (ka[i] != ka[i - 1]) {
                done = false;
                break;
            }
        }
        if (!done) printf("0\n");
        else {
            ans %= mod;
            printf("%lld\n", ans);
        }
    }
    return 0;
}
```

5.23 Dinic 最大流

```
#include <queue>
#include <vector>
#include <cstring>
#include <algorithm>

const int maxn = "Edit";
const int inf = 0x7FFFFFFF;

struct Edge {
    int c, f;
    unsigned v, flip;
    Edge(unsigned v, int c, int f, unsigned flip) : v(v), c(c), f(f), flip(flip) {}
};

/*
*b:BFS使用 ，
*a:可改进量 ， 不会出现负数可改进量。
*p[v]:u到v的反向边，即v到u的边。*cur[u]:i开始搜索的位置 ，此位置前所有路已满载。*s:源点。
*t:汇点 。
*/

class Dinic {
private:
    bool b[maxn];
    int a[maxn];
    unsigned p[maxn], cur[maxn], d[maxn];
    std::vector<Edge> G[maxn];
public:
    unsigned s, t;
    void Init(unsigned n) {
```

```
        for(int i=0; i<=n; ++i)
            G[i].clear();
    }
    void AddEdge(unsigned u, unsigned v, int c) {
        G[u].push_back(Edge(v, c, 0, G[v].size()));
        G[v].push_back(Edge(u, 0, 0, G[u].size()-1)); //使用无向图时将0改为c即可
    }
    bool BFS() {
        unsigned u, v;
        std::queue<unsigned> q;
        memset(b, 0, sizeof(b));
        q.push(s);
        d[s] = 0;
        b[s] = 1;
        while (!q.empty()) {
            u = q.front();
            q.pop();
            for (auto it = G[u].begin(); it != G[u].end(); ++it) {
                Edge &e = *it;
                if(!b[e.v] && e.c > e.f){
                    b[e.v] = 1;
                    d[e.v] = d[u] + 1;
                    q.push(e.v);
                }
            }
        }
        return b[t];
    }
    int DFS(unsigned u, int a){
        if(u==t || a==0)
            return a;
        int flow = 0, f;
        for (unsigned &i = cur[u]; i<G[u].size(); ++i){
            Edge &e = G[u][i];
            if (d[u]+1 == d[e.v] && (f = DFS(e.v, std::min(a, e.c - e.f))) > 0) {
                a -= f;
                e.f += f;
                G[e.v][e.flip].f -= f;
                flow += f;
                if (!a) break;
            }
        }
        return flow;
    }
    int MaxFlow(unsigned s, unsigned t){
        int flow = 0;
        this->s = s;
        this->t = t;
        while (BFS()) {
            memset(cur, 0, sizeof(cur));
            flow += DFS(s, inf);
        }
        return flow;
    }
```

```
};
```

5.24 ISAP 最大流

```
const int maxn = "Edit";
struct ISAP {
    int n, m, s, t;       //结点数，边数（包括反向弧），源点编号和汇点编号
    vector<Edge> edges;  //边表。edges[e]和edges[e^1]互为反向弧
    vector<int> G[maxn]; //邻接表，G[i][j]表示结点i的第j条边在e数组中的序号
    bool vis[maxn];       //BFS使用
    int d[maxn];          //起点到i的距离
    int cur[maxn];        //当前弧下标
    int p[maxn];          //可增广路上的一条弧
    int num[maxn];        //距离标号计数
    void init(int n) {
        this->n = n;
        for (int i = 0; i < n; i++) G[i].clear();
        edges.clear();
    }
    void AddEdge(int from, int to, int cap) {
        edges.pb(Edge(from, to, cap, 0));
        edges.pb(Edge(to, from, 0, 0));
        int m = edges.size();
        G[from].pb(m - 2);
        G[to].pb(m - 1);
    }
    int Augumemt() {
        int x = t, a = INF;
        while (x != s) {
            Edge& e = edges[p[x]];
            a = min(a, e.cap - e.flow);
            x = edges[p[x]].from;
        }
        x = t;
        while (x != s) {
            edges[p[x]].flow += a;
            edges[p[x] ^ 1].flow -= a;
            x = edges[p[x]].from;
        }
        return a;
    }
    void BFS() {
        clr(vis, 0);
        clr(d, 0);
        queue<int> q;
        q.push(t);
        d[t] = 0;
        vis[t] = 1;
        while (!q.empty()) {
            int x = q.front();
            q.pop();
            int len = G[x].size();
```

```
            for (int i = 0; i < len; i++) {
                Edge& e = edges[G[x][i]];
                if (!vis[e.from] && e.cap > e.flow) {
                    vis[e.from] = 1;
                    d[e.from] = d[x] + 1;
                    q.push(e.from);
                }
            }
        }
    }
    int Maxflow(int s, int t) {
        this->s = s;
        this->t = t;
        int flow = 0;
        BFS();
        clr(num, 0);
        for (int i = 0; i < n; i++)
            if (d[i] < INF) num[d[i]]++;
        int x = s;
        clr(cur, 0);
        while (d[s] < n) {
            if (x == t) {
                flow += Augumemt();
                x = s;
            }
            int ok = 0;
            for (int i = cur[x]; i < G[x].size(); i++) {
                Edge& e = edges[G[x][i]];
                if (e.cap > e.flow && d[x] == d[e.to] + 1) {
                    ok = 1;
                    p[e.to] = G[x][i];
                    cur[x] = i;
                    x = e.to;
                    break;
                }
            }
            if (!ok) {//Retreat
                int m = n - 1;
                for (int i = 0; i < G[x].size(); i++) {
                    Edge& e = edges[G[x][i]];
                    if (e.cap > e.flow) m = min(m, d[e.to]);
                }
                if (--num[d[x]] == 0) break; //gap优化
                num[d[x] = m + 1]++;
                cur[x] = 0;
                if (x != s) x = edges[p[x]].from;
            }
        }
        return flow;
    }
};
```

5.25 最小费用最大流

```cpp
#include <iostream>
#include <vector>
#include <queue>

const int MAXE = 1000;
const int MAXN = 1000;
const int INF = 1000000;

using ii = std::pair<int, int>;

struct edge {
    int u, v, cost, cap, flow;
} E[MAXE], * pred[MAXN];

std::vector<edge *> g[MAXN];
int N, M, EE, dist[MAXN], phi[MAXN];

inline edge * opp(edge * e) {
    return E + ((e - E) ^ 1);
}

void inti() {
    for (int i = 0; i <= N; i++) {
        g[i].clear();
    }
    EE = 0;
}

void add_edge(int u, int v, int cost, int cap) {
    E[EE] = { u, v, cost, cap, 0 };
    g[u].emplace_back(E + (EE++));
    E[EE] = { v, u, -cost, 0, 0 };
    g[v].emplace_back(E + (EE++));
}

bool dijkstra(int S, int T) {
    std::fill(dist, dist + N, INF);
    std::fill(pred, pred + N, nullptr);
    std::priority_queue<ii, std::vector<ii>, std::greater<ii>> pq;
    dist[S] = 0;
    for (pq.emplace(dist[S], S); !pq.empty(); ) {
        int u = pq.top().second;
        pq.pop();
        for (auto e : g[u]) {
            if (e->cap - e->flow > 0 && dist[e->v] > dist[e->u] + e->cost + phi[e->u] - phi[e->v]) {
                dist[e->v] = dist[e->u] + e->cost + phi[e->u] - phi[e->v];
                pred[e->v] = e;
                pq.emplace(dist[e->v], e->v);
            }
        }
    }
```

```
    for (int i = 0; i < N; i++) {
        phi[i] = std::min(INF, phi[i] + dist[i]);
    }
    return dist[T] != INF;
}

std::pair<int, int> mincost_maxflow(int S, int T) {
    int mincost = 0, maxflow = 0;
    std::fill(phi, phi + N, 0);
    while (dijkstra(S, T)) {
        int flow = INF;
        for (edge * e = pred[T]; e; e = pred[e->u])
            flow = std::min(flow, e->cap - e->flow);
        for (edge * e = pred[T]; e; e = pred[e->u]) {
            mincost += e->cost * flow;
            e->flow += flow;
            opp(e)->flow -= flow;
        }
        maxflow += flow;
    }
    return std::make_pair(mincost, maxflow);
}
```

5.26 ZKW 费用流

```
const int inf = ~0U>>1;
const int N = "Edit";

typedef struct seg{
    int to,op,cost,nxt,f;
}seg;

seg v[N*40];

int ans =0,tot,dis[N],base[N],vis[N],ttf = 0;

int S,T; int cur[N];

void inti() {
    memset(base,0,sizeof(base));
    memset(dis,0,sizeof(dis));
    tot = 0; ans = 0; ttf = 0;
    memset(vis,0,sizeof(vis));
}

int aug(int u,int flow){
    if (u == T){
        ans += flow * dis[S];
        ttf += flow;
        return flow;
    }
    vis[u] = 1;
```

```
    int now = 0;
    for (int i = base[u];i;i = v[i].nxt){
        int x = v[i].to;
        if (vis[x] || v[i].f <= 0 || dis[u] != dis[x] + v[i].cost)
            continue;
        int tmp = aug(x,std::min(flow - now,v[i].f));
        v[i].f -= tmp; v[v[i].op].f += tmp;
        now += tmp;
        if (now == flow) return flow;
    }
    return now;
}


int modlabel() {
    int del = inf;
    for (int i = S; i <= T; i++) {
        if (vis[i]) for (int j = base[i];j;j = v[j].nxt) {
            if (v[j].f){
                int x = v[j].to;
                if (!vis[x]) del = std::min(del,dis[x] + v[j].cost - dis[i]);
            }
        }
    }
    if (del == inf) {
        return 0;
    }
    for (int i = S;i <= T;i++) {
        if (vis[i]) {
            vis[i] = 0,dis[i] += del,cur[i] = base[i];
        }
    }
    return 1;
}


int zkw() {
    for (int i = S;i <= T;i++) cur[i] = base[i];
    int fl, t = 0;
    do {
        t = 0;
        while((t = aug(S,inf))) memset(vis,0,sizeof(vis));
    } while(modlabel());
    return ans;
}

void add(int x, int y, int c, int f){
    v[++tot].to = y; v[tot].op = tot + 1;
    v[tot].f = f; v[tot].cost = c;
    v[tot].nxt = base[x]; base[x] = tot;
    v[++tot].to = x; v[tot].op = tot - 1;
    v[tot].f = 0; v[tot].cost = -c;
    v[tot].nxt = base[y]; base[y] = tot;
}
```

5.27 上下界网络流

```
/*
        首先建立一个源S和一个汇T，一般称为附加源和附加汇。
        对于图中的每条弧<u,v>，假设它容量上界为c，下界b，那么把这条边拆为三条只有上界的弧。
        一条为<S,v>，容量为b；
    一条为<u,T>，容量为b；
    一条为<u,v>，容量为c-b。
        其中前两条弧一般称为附加弧。
        然后对这张图跑最大流，以S为源，以T为汇，如果所有的附加弧都满流，则原图有可行流;否则就是无解。
        这时，每条非附加弧的流量加上它的容量下界，就是原图中这条弧应该有的流量。

        对于原图中的每条弧，我们把c-b称为它的自由流量，意思就是只要它流满了下界，这些流多少都没问题。
    既然如此，对于每条弧<u,v>，我们强制给v提供b单位的流量，并且强制从u那里拿走b单位的流量，这一步对应着两条附加弧。
    如果这一系列强制操作能完成的话，也就是有一组可行流了。
    注意：这张图的最大流只是对应着原图的一组可行流，而不是原图的最大或最小流。
*/
using namespace std;
const int oo = (1LL << 31) - 1;
const int LEN = 1e5 + 5;
struct node {
        int x, y, l, r;
} a[LEN];
namespace ISAP {
        int flow, tot, n, m, src, tar, qh, qt, cnt, ans;
        struct edge {
                int vet, next, len;
        } E[LEN * 2];
        int dis[LEN], gap[LEN], head[LEN], cur[LEN], q[LEN], vis[LEN], IN[LEN];
        void add(int u, int v, int c) {
                E[++tot] = (edge){v, head[u], c};
                head[u] = tot;
        }
        void join(int u, int v, int c) {
                add(u, v, c);
                add(v, u, 0);
        }
        void bfs(int s) {
                qh = qt = 0;
                q[++qt] = s;
                dis[s] = 0;
                vis[s] = 1;
                while (qh < qt) {
                        int u = q[++qh];
                        gap[dis[u]]++;
                        for (int e = head[u]; e != -1; e = E[e].next) {
                                int v = E[e].vet;
                                if (E[e ^ 1].len && !vis[v]) {
```

```
                                        dis[v] = dis[u] + 1;
                                        vis[v] = 1;
                                        q[++qt] = v;
                                }
                        }
                }
        }
        int isap(int u, int aug) {
                if (u == tar) return aug;
                int flow = 0;
                for (int e = head[u]; e != -1; e = E[e].next) {
                        int v = E[e].vet;
                        if (E[e].len && dis[v] == dis[u] - 1) {
                                int tmp = isap(v, min(aug - flow, E[e].len));
                                E[e].len -= tmp;
                                E[e ^ 1].len += tmp;
                                flow += tmp;
                                head[u] = e;
                                if (flow == aug || dis[src] == cnt) return flow;
                        }
                }
                if (!--gap[dis[u]++]) dis[src] = cnt;
                ++gap[dis[u]];
                head[u] = cur[u];
                return flow;
        }
        void init() {
                tot = -1, gap[0] = 0;
                for (int i = 1; i <= cnt; i++) {
                        dis[i] = gap[i] = vis[i] = IN[i] = 0;
                        head[i] = -1;
                }
        }
        int maxflow(int s, int t) {
                src = s, tar = t;
                int res = 0;
                for (int i = 1; i <= cnt; i++) cur[i] = head[i];
                bfs(tar);
                while (dis[src] < cnt) res += isap(src, oo);
                return res;
        }
}
using namespace ISAP;
int main() {
        scanf("%d␣%d", &n, &m);
        cnt = n;
        src = ++cnt, tar = ++cnt;
        init();
        for (int i = 1; i <= m; i++) {
                int x, y, l, r;
                scanf("%d␣%d␣%d␣%d", &x, &y, &l, &r);
                a[i] = (node){x, y, l, r};
                join(x, y, r - l);
                IN[y] += l, IN[x] -= l;
```

```
101         }
102         for (int i = 1; i <= n; i++) {
103                 if (IN[i] < 0) join(i, tar, -IN[i]);
104                 else {
105                         join(src, i, IN[i]);
106                         flow += IN[i];
107                 }
108         }
109         int ans = maxflow(src, tar);
110         if (flow == ans) {
111                 puts("YES");
112                 for (int i = 1; i <= m; i++) printf("%d\n", a[i].l + E[i * 2 - 1].len);
113         } else puts("NO");
114         return 0;
115 }
116
117 /*
118         先来看有源汇可行流
119     建模方法:
120     建立弧<t,s>, 容量下界为0, 上界为oo。
121     然后对这个新图（实际上只是比原图多了一条边）按照无源汇可行流的方法建模,
122         如果所有附加弧满流, 则存在可行流。
123     求原图中每条边对应的实际流量的方法, 同无源汇可行流, 只是忽略掉弧<t,s>就好。
124     而且这时候弧<t,s>的流量就是原图的总流量。
125     理解方法:
126     有源汇相比无源汇的不同就在于, 源和汇是不满足流量平衡的, 那么连接<t,s>之后,
127         源和汇也满足了流量平衡, 就可以直接按照无源汇的方式建模。
128     注意: 这张图的最大流只是对应着原图的一组可行流, 而不是原图的最大或最小流。
129
130         有源汇最大流
131     建模方法:
132     首先按照有源汇可行流的方法建模, 如果不存在可行流, 更别提什么最大流了。
133     如果存在可行流, 那么在运行过有源汇可行流的图上（就是已经存在流量的那张图, 流量不要清零）,
134         跑一遍从s到t的最大流（这里的s和t是原图的源和汇, 不是附加源和附加汇）, 就是原图的最大
                流。
135     理解方法:
136     为什么要在那个已经有了流量的图上跑最大流? 因为那张图保证了每条弧的容量下界, 在这张图上跑最
        大流,
137         实际上就是在容量下界全部满足的前提下尽量多得获得“自由流量”。
138     注意, 在这张已经存在流量的图上, 弧<t,s>也是存在流量的, 千万不要忽略这条弧。
139         因为它的相反弧<s,t>的流量为<t,s>的流量的相反数, 且<s,t>的容量为0, 所以这部分的流量也是
                会被算上的。
140 */
141 using namespace std;
142 typedef long long ll;
143 const int LEN = 1e5 + 5;
144 const int oo = (1LL << 31) - 1;
145 namespace DINIC {
146         int tot, n, m, src, tar, qh, qt, cnt, s, t, S, T;
147         int ans, flow;
148         struct edge {
149                 int vet, next, len;
150         } E[LEN * 2];
151         int dis[LEN], gap[LEN], head[LEN], cur[LEN], q[LEN], vis[LEN], IN[LEN];
```

```
152         void add(int u, int v, int c) {
153                 E[++tot] = (edge){v, head[u], c};
154                 head[u] = tot;
155         }
156         void join(int u, int v, int c) {
157                 add(u, v, c);
158                 add(v, u, 0);
159         }
160         void init() {
161                 tot = -1;
162                 for (int i = 1; i <= cnt; i++) head[i] = -1;
163         }
164         bool bfs() {
165                 for (int i = 1; i <= cnt; i++) dis[i] = 0;
166                 qh = qt = 0;
167                 q[++qt] = src;
168                 dis[src] = 1;
169                 while (qh < qt) {
170                         int u = q[++qh];
171                         for (int e = head[u]; e != -1; e = E[e].next) {
172                                 int v = E[e].vet;
173                                 if (E[e].len && !dis[v]) {
174                                         dis[v] = dis[u] + 1;
175                                         if (v == tar) return 1;
176                                         q[++qt] = v;
177                                 }
178                         }
179                 }
180                 return dis[tar];
181         }
182         int dfs(int u, int aug) {
183                 if (u == tar || !aug) return aug;
184                 int tmp = 0;
185                 for (int &e = cur[u]; e != -1; e = E[e].next) {
186                         int v = E[e].vet;
187                         if (dis[v] == dis[u] + 1) {
188                                 if (tmp = dfs(v, min(aug, E[e].len))) {
189                                         E[e].len -= tmp;
190                                         E[e ^ 1].len += tmp;
191                                         return tmp;
192                                 }
193                         }
194                 }
195                 return 0;
196         }
197         int maxflow(int s, int t) {
198                 src = s, tar = t;
199                 int res = 0, flow = 0;
200                 while (bfs()) {
201                         for (int i = 1; i <= cnt; i++) cur[i] = head[i];
202                         while (flow = dfs(src, oo)) res += flow;
203                 }
204                 return res;
205         }
```

```
}
using namespace DINIC;
int main() {
        scanf("%d␣%d␣%d␣%d", &n, &m, &s, &t);
        cnt = n;
        S = ++cnt, T = ++cnt;
        init();
        for (int i = 1; i <= m; i++) {
                int x, y, l, r;
                scanf("%d␣%d␣%d␣%d", &x, &y, &l, &r);
                join(x, y, r - l);
                IN[y] += l, IN[x] -= l;
        }
        for (int i = 1; i <= n; i++) {
                if (IN[i] < 0) join(i, T, -IN[i]);
                else if (IN[i] > 0) {
                        flow += IN[i];
                        join(S, i, IN[i]);
                }
        }
        join(t, s, oo);
        ans = maxflow(S, T);
        if (ans != flow) puts("please␣go␣home␣to␣sleep");
        else {
                ans = maxflow(s, t);
                printf("%lld\n", ans);
        }
        return 0;
}

/*
        先来看有源汇可行流
    建模方法:
    建立弧<t,s>, 容量下界为0, 上界为oo。
    然后对这个新图（实际上只是比原图多了一条边）按照无源汇可行流的方法建模,
        如果所有附加弧满流, 则存在可行流。
    求原图中每条边对应的实际流量的方法, 同无源汇可行流, 只是忽略掉弧<t,s>就好。
    而且这时候弧<t,s>的流量就是原图的总流量。
    理解方法:
    有源汇相比无源汇的不同就在于, 源和汇是不满足流量平衡的, 那么连接<t,s>之后,
        源和汇也满足了流量平衡, 就可以直接按照无源汇的方式建模。
    注意: 这张图的最大流只是对应着原图的一组可行流, 而不是原图的最大或最小流。

        有源汇最小流
    有源汇最小流的常见建模方法比较多, 我就只说我常用的一种。
    建模方法:
    首先按照有源汇可行流的方法建模, 但是不要建立<t,s>这条弧。
    然后在这个图上, 跑从附加源ss到附加汇tt的最大流。
    这时候再添加弧<t,s>, 下界为0, 上界oo。
    在现在的这张图上, 从ss到tt的最大流, 就是原图的最小流。
    理解方法:
    我们前面提到过, 有源汇可行流的流量只是对应一组可行流, 并不是最大或者最小流。
    并且在跑完有源汇可行流之后, 弧<t,s>的流量就是原图的流量。
```

```
259     从这个角度入手，我们想让弧<t,s>的流量尽量小，就要尽量多的消耗掉那些“本来不需要经过<t,s>”的流量。
260     于是我们在添加<t,s>之前，跑一遍从ss到tt的最大流，就能尽量多的消耗那些流量啦QwQ。
261 */
262 using namespace std;
263 typedef long long ll;
264 const int LEN = 2e5 + 5;
265 const int oo = (1LL << 31) - 1;
266 namespace DINIC {
267         int tot, n, m, src, tar, qh, qt, cnt, s, t, S, T, ans, flow;
268         struct edge {
269                 int vet, next, len;
270         } E[LEN * 2];
271         int dis[LEN], gap[LEN], head[LEN], cur[LEN], q[LEN], vis[LEN], IN[LEN];
272         void add(int u, int v, int c) {
273                 E[++tot] = (edge){v, head[u], c};
274                 head[u] = tot;
275         }
276         void join(int u, int v, int c) {
277                 add(u, v, c);
278                 add(v, u, 0);
279         }
280         void init() {
281                 tot = -1;
282                 for (int i = 1; i <= cnt; i++) head[i] = -1;
283         }
284         bool bfs() {
285                 for (int i = 1; i <= cnt; i++) dis[i] = 0;
286                 qh = qt = 0;
287                 q[++qt] = src;
288                 dis[src] = 1;
289                 while (qh < qt) {
290                         int u = q[++qh];
291                         for (int e = head[u]; e != -1; e = E[e].next) {
292                                 int v = E[e].vet;
293                                 if (E[e].len && !dis[v]) {
294                                         dis[v] = dis[u] + 1;
295                                         if (v == tar) return 1;
296                                         q[++qt] = v;
297                                 }
298                         }
299                 }
300                 return dis[tar];
301         }
302         int dfs(int u, int aug) {
303                 if (u == tar || !aug) return aug;
304                 int tmp = 0;
305                 for (int &e = cur[u]; e != -1; e = E[e].next) {
306                         int v = E[e].vet;
307                         if (dis[v] == dis[u] + 1) {
308                                 if (tmp = dfs(v, min(aug, E[e].len))) {
309                                         E[e].len -= tmp;
310                                         E[e ^ 1].len += tmp;
311                                         return tmp;
```

```
                                }
                        }
                }
                return 0;
        }
        int maxflow(int s, int t) {
                src = s, tar = t;
                int res = 0, flow = 0;
                while (bfs()) {
                        for (int i = 1; i <= cnt; i++) cur[i] = head[i];
                        while (flow = dfs(src, oo)) res += flow;
                }
                return res;
        }
}
using namespace DINIC;
int main() {
        scanf("%d %d %d %d", &n, &m, &s, &t);
        cnt = n;
        S = ++cnt, T = ++cnt;
        init();
        for (int i = 1; i <= m; i++) {
                int x, y, l, r;
                scanf("%d %d %d %d", &x, &y, &l, &r);
                join(x, y, r - l);
                IN[y] += l, IN[x] -= l;
        }
        for (int i = 1; i <= n; i++) {
                if (IN[i] < 0) join(i, T, -IN[i]);
                else if (IN[i] > 0) {
                        flow += IN[i];
                        join(S, i, IN[i]);
                }
        }
        ans = maxflow(S, T);
        flow -= ans;
        join(t, s, oo);
        ans = maxflow(S, T);
        if (ans != flow) puts("please go home to sleep");
        else printf("%d\n", ans);
        return 0;
}
```

5.28 图匹配理论

```
1 二分图匹配:
2 点覆盖、最小点覆盖
3 点覆盖集即一个点集，使得所有边至少有一个端点在集合里。或者说是“点” 覆盖了所有“边”。。极小
    点覆盖(minimal vertex covering)：本身为点覆盖，其真子集都不是。最小点覆盖(minimum vertex
    covering)：点最少的点覆盖。点覆盖数(vertex covering number)：最小点覆盖的点数。
4
5 边覆盖、极小边覆盖
```

边覆盖集即一个边集，使得所有点都与集合里的边邻接。或者说是“边”覆盖了所有“点”。极小边覆盖(minimal edge covering)：本身是边覆盖，其真子集都不是。最小边覆盖(minimum edge covering)：边最少的边覆盖。边覆盖数(edge covering number)：最小边覆盖的边数。

独立集、极大独立集

独立集即一个点集，集合中任两个结点不相邻，则称V为独立集。或者说是导出的子图是零图（没有边）的点集。极大独立集(maximal independent set)：本身为独立集，再加入任何点都不是。最大独立集(maximum independent set)：点最多的独立集。独立数(independent number)：最大独立集的点。

团

团即一个点集，集合中任两个结点相邻。或者说是导出的子图是完全图的点集。极大团(maximal clique)：本身为团，再加入任何点都不是。最大团(maximum clique)：点最多的团。团数(clique number)：最大团的点数。

边独立集、极大边独立集

边独立集即一个边集，满足边集中的任两边不邻接。极大边独立集(maximal edge independent set)：本身为边独立集，再加入任何边都不是。最大边独立集(maximum edge independent set)：边最多的边独立集。边独立数(edge independent number)：最大边独立集的边数。

边独立集又称匹配(matching)，相应的有极大匹配(maximal matching)，最大匹配(maximum matching)，匹配数(matching number)。

支配集、极小支配集

支配集即一个点集，使得所有其他点至少有一个相邻点在集合里。或者说是一部分的“点”支配了所有“点”。极小支配集(minimal dominating set)：本身为支配集，其真子集都不是。最小支配集(minimum dominating set)：点最少的支配集。支配数(dominating number)：最小支配集的点数。

边支配集、极小边支配集

边支配集即一个边集，使得所有边至少有一条邻接边在集合里。或者说是一部分的“边”支配了所有“边”。极小边支配集(minimal edge dominating set)：本身是边支配集，其真子集都不是。最小边支配集(minimum edge dominating set)：边最少的边支配集。边支配数(edge dominating number)：最小边支配集的边数。

最小路径覆盖

最小路径覆盖(path covering)：是“路径”覆盖“点”，即用尽量少的不相交简单路径覆盖有向无环图G的所有顶点，即每个顶点严格属于一条路径。路径的长度可能为0(单个点)。

最小路径覆盖数=G的点数－最小路径覆盖中的边数。应该使得最小路径覆盖中的边数尽量多，但是又不能让两条边在同一个顶点相交。拆点：将每一个顶点i拆成两个顶点Xi和Yi。然后根据原图中边的信息，从X部往Y部引边。所有边的方向都是由X部到Y部。因此，所转化出的二分图的最大匹配数则是原图G中最小路径覆盖上的边数。因此由最小路径覆盖数=原图G的顶点数－二分图的最大匹配数便可以得解。

匹配

匹配(matching)是一个边集，满足边集中的边两两不邻接。匹配又称边独立集(edge independent set)。

在匹配中的点称为匹配点(matched vertex)或饱和点；反之，称为未匹配点(unmatched vertex)或未饱和点。

交错轨(alternating path)是图的一条简单路径，满足任意相邻的两条边，一条在匹配内，一条不在匹配内。

增广轨(augmenting path)：是一个始点与终点都为未匹配点的交错轨。

最大匹配(maximum matching)是具有最多边的匹配。

匹配数(matching number)是最大匹配的大小。

完美匹配(perfect matching)是匹配了所有点的匹配。

完备匹配(complete matching)是匹配了二分图较小集合（二分图X，Y中小的那个）的所有点的匹配。

增广轨定理：一个匹配是最大匹配当且仅当没有增广轨。

所有匹配算法都是基于增广轨定理：一个匹配是最大匹配当且仅当没有增广轨。这个定理适用于任意图。

```
40
41 二分图的性质
42 二分图中，点覆盖数是匹配数。
43 (1) 二分图的最大匹配数等于最小覆盖数，即求最少的点使得每条边都至少和其中的一个点相关联，很显然
       直接取最大匹配的一段节点即可。
44 (2) 二分图的独立数等于顶点数减去最大匹配数，很显然的把最大匹配两端的点都从顶点集中去掉这个时候
       剩余的点是独立集，这是|V|-2*|M|，同时必然可以从每条匹配边的两端取一个点加入独立集并且保持
       其独立集性质。
45 (3) DAG的最小路径覆盖，将每个点拆点后作最大匹配，结果为n-m，求具体路径的时候顺着匹配边走就可
       以，匹配边i->j',j->k',k->l'....构成一条有向路径。
46 (4) 最大匹配数=左边匹配点+右边未匹配点。因为在最大匹配集中的任意一条边，如果他的左边没标记，右
       边被标记了，那么我们就可找到一条新的增广路，所以每一条边都至少被一个点覆盖。
47 (5) 最小边覆盖=图中点的个数-最大匹配数=最大独立集。
48
49 定理1: 最小覆盖数 = 最大匹配数
50 定理2: 最大独立集S 与 最小覆盖集T 互补
51
52 有向无环图最小不相交路径覆盖
53 定义: 用最少的不相交路径覆盖所有顶点。
54 定理: 把原图中的每个点V拆成Vx和Vy，如果有一条有向边A->B，那么就加边Ax-By。这样就得到了一个二分
       图，最小路径覆盖=原图的节点数-新图最大匹配。
55
56 有向无环图最小可相交路径覆盖
57 定义: 用最小的可相交路径覆盖所有顶点。
58 算法: 先用floyd求出原图的传递闭包，即如果a到b有路，那么就加边a->b。然后就转化成了最小不相交路
       径覆盖问题。
59
60 Kuhn-Munkers算法的几种变形应用
61 1.Kuhn-Munkers算法是求最大权完备匹配，如果要求最小权完备匹配怎么办? 方法很简单，只需将所有的边
       权值取其相反数，求最大权完备匹配，匹配的值再取相反数即可。
62 2.Kuhn-Munkers算法的运行要求是必须存在一个完备匹配，如果求一个最大权匹配(不一定完备)该如何办?
       依然很简单，把不存在的边权值赋为0。
63 3.Kuhn-Munkers算法求得的最大权匹配是边权值和最大，如果我想要边权之积最大，又怎样转化? 还是不难
       办到，每条边权取自然对数，然后求最大和权匹配，求得的结果a再算出e^a就是最大积匹配。
```

5.29 二分图最大匹配匈牙利算法

```
1  int n, m;
2  int g[maxn][maxn];  //0-labeled
3  int linker[maxn];
4  bool used[maxn];
5
6  bool dfs(int u) {
7      int v;
8      for(v = 0; v < m; v++) {
9          if(g[u][v] && !used[v]) {
10             used[v] = true;
11             if(linker[v] == -1 || dfs(linker[v])) {
12                 linker[v] = u;
13                 return true;
14             }
```

```
        }
    }
    return false;
}

int hungary() {
    int res = 0;
    int u;
    memset(linker, -1, sizeof(linker));
    for(u = 0; u < n; u++) {
        memset(used, 0, sizeof(used));
        if(dfs(u)) {
            res++;
        }
    }
    return res;
}
```

5.30 二分图最大权匹配 KM 算法

```
const int maxn = "Edit";
const int inf = 2e9;

int n, cost[maxn][maxn];
int lx[maxn], ly[maxn], match[maxn], slack[maxn], prev[maxn];
bool vy[maxn];

void augment(int root) {
    std::fill(vy + 1, vy + n + 1, false);
    std::fill(slack + 1, slack + n + 1, inf);
    int py;
    match[py = 0] = root;
    do {
        vy[py] = true;
        int x = match[py], delta = inf, yy;
        for (int y = 1; y <= n; y++) {
            if (!vy[y]) {
                if (lx[x] + ly[y] - cost[x][y] < slack[y]) {
                    slack[y] = lx[x] + ly[y] - cost[x][y];
                    prev[y] = py;
                }
                if (slack[y] < delta) {
                    delta = slack[y];
                    yy = y;
                }
            }
        }
        for (int y = 0; y <= n; y++) {
            if (vy[y]) {
                lx[match[y]] -= delta;
                ly[y] += delta;
            } else {
```

```
                slack[y] -= delta;
            }
        }
        py = yy;
    } while (match[py] != -1);
    do {
        int pre = prev[py];
        match[py] = match[pre];
        py = pre;
    } while (py);
}

int KM() {
    for (int i = 1; i <= n; i++) {
        lx[i] = ly[i] = 0;
        match[i] = -1;
        for (int j = 1; j <= n; j++) {
            lx[i] = std::max(lx[i], cost[i][j]);
        }
    }
    int answer = 0;
    for (int root = 1; root <= n; root++) {
        augment(root);
    }
    for (int i = 1; i <= n; i++) {
        answer += lx[i];
        answer += ly[i];
        //printf("%d %d\n", match[i], i);
    }
    return answer;
}
```

第六章　数据结构

6.1　树状数组

```
void add(int i, int x) {
    for(;i <= n; i += i & -i)
        tree[i] += x;
}

int sum(int i) {
    int ret = 0;
    for(; i; i -= i & -i) ret += tree[i];
    return ret;
}
```

6.2　差分数组

```
//Author:CookiC
/*
*a为原数组
*C为差分数组
*/
int a[]={0, 1, 1, 1, 1, 1, 1};
int N, C[maxn];

int Sum(unsigned n) {
    int sum = 0;
    while(n>0){
        sum += C[n];
        n -= lowbit(n);
    }
    return sum;
}

void Add(unsigned n, int d) {
    while(n<=N){
        C[n]+=d;
        n+=lowbit(n);
    }
}

void Add(int L,int R, int d) {
    Add(L,d);
```

```
    Add(R+1,-d);
}

void Init() {
    memset(C, 0, sizeof(C));
    Add(1, a[1]);
    for(int i=2; i<=N; ++i)
        Add(i, a[i]-a[i-1]);
}

void Update() {
    for(int i=1; i<=N; ++i)
        a[i] = Sum(i);
}
```

6.3 二维树状数组

```
int N;
int c[maxn][maxn];

inline int lowbit(int t) {
    return t&(-t);
}

void update(int x, int y, int v) {
    for (int i=x; i<=N; i+=lowbit(i)) {
        for (int j=y; j<=N; j+=lowbit(j)) {
            c[i][j]+=v;
        }
    }
}

int query(int x, int y) {
    int s = 0;
    for (int i=x; i>0; i-=lowbit(i)) {
        for (int j=y; j>0; j-=lowbit(j)) {
            s += c[i][j];
        }
    }
    return s;
}

int sum(int x, int y, int xx, int yy) {
    x--, y--;
    return query(xx, yy) - query(xx, y) - query(x, yy) + query(x, y);
}
```

6.4 树状数组求逆序对

```cpp
//树状数组求逆序对
const int maxn = "Edit";

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

bool cmp(std::pair<int, int> no1, std::pair<int, int> no2) {
    return no1.first < no2.first;
}

int d[maxn], p[maxn], n;
std::pair<int, int> start[maxn];

void add(int x) {
    while (x <= n) {
        d[x]++;
        x += lowbit(x);
    }
}

long long sum(int x) {
    long long sum = 0;
    while (x) {
        sum += d[x];
        x -= lowbit(x);
    }
    return sum;
}

int main(int argc, char *argv[]) {
    long long ans;
    std::cin>>n;
    memset(d,0,sizeof(d));
    ans=0;
    for (int i=1;i<=n;i++) {
        std::cin >> start[i].first;
        start[i].second = i;
    }
    std::sort(start+1, start+n+1, cmp);
    int id = 1;
    p[start[1].second]=1;
    for (int i = 2; i <= n; ++i) {
        if (start[i].first == start[i-1].first) {
            p[start[i].second] = id;
        } else {
            p[start[i].second] = ++id;
        }
    }
    for (int i=1;i<=n;i++) {
       add(p[i]);
       ans += i - sum(p[i]);
    }
    std::cout << ans << std::endl;
```

```
    return 0;
}
```

6.5 堆

```
const int N = 1000;

template <class T>
class Heap {
    private:
        T h[N];
        int len;
    public:
        Heap() {
            len = 0;
        }
        inline void push(const T& x) {
            h[++len] = x;
            std::push_heap(h+1, h+1+len, std::greater<T>());
        }
        inline T pop() {
            std::pop_heap(h+1, h+1+len, std::greater<T>());
            return h[len--];
        }
        inline T& top() {
            return h[1];
        }
        inline bool empty() {
            return len == 0;
        }
};
```

6.6 RMQ

```
//A为原始数组，d[i][j]表示从i开始，长度为(1<<j)的区间最小值

int A[maxn];
int d[maxn][30];

void init(int A[], int len) {
    for (int i = 0; i < len; i++)d[i][0] = A[i];
    for (int j = 1; (1 << j) <= len; j++) {
        for (int i = 0; i + (1 << j) - 1 < len; i++) {
            d[i][j] = min(d[i][j - 1], d[i + (1 << (j - 1))][j - 1]);
        }
    }
}

int query(int l, int r) {
```

```
    int p = 0;
    while ((1 << (p + 1)) <= r - l + 1)p++;
    return min(d[l][p], d[r - (1 << p) + 1][p]);
}
```

6.7 RMQ

```
//author: wavator
#include <algorithm>
#include <vector>

template <class T>
struct RMQ {
    std::vector<std::vector<T> > rmq;
    // vector<T> rmq[20]; or T[100002][20] if need speed
    //T kInf = numeric_limits<T>::max(); // if need return a value when the interval fake
    void init(const std::vector<T>& a) { // 0 base
        int n = (int)a.size(), base = 1, depth = 1;
        while (base < n)
            base <<= 1, ++depth;
        rmq.assign((unsigned)depth, a);
        for (int i = 0; i < depth - 1; ++i)
            for (int j = 0; j < n; ++j) {
                rmq[i + 1][j] = std::min(rmq[i][j], rmq[i][std::min(n - 1, j + (1 << i))]);
            }
    }
    T q(int l, int r) { // [l, r)
        if(l>r)return 0x3f3f3f3f;
        int dep = 31 - __builtin_clz(r - l); // log(b - a)
        return min(rmq[dep][l], rmq[dep][r - (1 << dep)]);
    }
};
```

6.8 线段树

```
//A为原始数组，sum记录区间和，Add为懒惰标记

int A[maxn], sum[maxn << 2], Add[maxn << 2];

void pushup(int rt) {
    sum[rt] = sum[rt << 1] + sum[rt << 1 | 1];
}

void pushdown(int rt, int l, int r) {
    if (Add[rt]) {
        int mid = (l + r) >> 1;
        Add[rt << 1] += Add[rt];
        Add[rt << 1 | 1] += Add[rt];
        sum[rt << 1] += (mid - l + 1)*Add[rt];
```

```
15          sum[rt << 1 | 1] += (r - mid)*Add[rt];
16          Add[rt] = 0;
17      }
18  }
19
20  void build(int l, int r, int rt) {
21      if (l == r) {
22          sum[rt] = A[l];
23          return;
24      }
25      int mid = (l + r) >> 1;
26      build(l, mid, rt << 1);
27      build(mid + 1, r, rt << 1 | 1);
28      pushup(rt);
29  }
30
31  //区间加值
32  void update(int L, int R, int val, int l, int r, int rt) {
33      if (L <= l && R >= r) {
34          Add[rt] += val;
35          sum[rt] += (r - l + 1)*val;
36          return;
37      }
38      pushdown(rt, l, r);
39      int mid = (l + r) >> 1;
40      if (L <= mid)update(L, R, val, l, mid, rt << 1);
41      if (R > mid)update(L, R, val, mid + 1, r, rt << 1 | 1);
42      pushup(rt);
43  }
44
45  //点修改
46  void update(int index, int val, int l, int r, int rt) {
47      if (l == r) {
48          sum[rt] = val;
49          return;
50      }
51      int mid = (l + r) >> 1;
52      if (index <= mid)update(index, val, l, mid, rt << 1);
53      else update(index, val, mid + 1, r, rt << 1 | 1);
54      pushup(rt);
55  }
56
57  //区间查询
58  int query(int L, int R, int l, int r, int rt) {
59      if (L <= l && R >= r) {
60          return sum[rt];
61      }
62      pushdown(rt, l, r);
63      int mid = (l + r) >> 1;
64      int ret = 0;
65      if (L <= mid)ret += query(L, R, l, mid, rt << 1);
66      if (R > mid)ret += query(L, R, mid + 1, r, rt << 1 | 1);
67      return ret;
68  }
```

6.9 ZKW 线段树

```cpp
const int maxn = 50009;
using ll = long long;

ll T[maxn*4];
int M,n;
void build() {
    for(M=1;M<=n+1;M<<=1);
    for(int i=1;i<=n;i++)
        std::cin >> T[i+M];
    for(int i=M-1;i;i--)
        T[i]=T[i<<1]+T[i<<1|1];
}

void update(int x,int val) {
    T[x+=M]=val;    //修改
//  T[x+=M]+=val;   //加值
    for(x>>=1;x>=1;x>>=1) {
        T[x]=T[x<<1]+T[x<<1|1];
    }
}

ll query(int l,int r) {
    l=l+M-1,r=r+M+1;
    ll ans=0;
    for(;l^r^1;l>>=1,r>>=1) {
        if(~l&1) ans+=T[l^1];
        if(r&1)  ans+=T[r^1];
    }
    return ans;
}
```

6.10 吉司机线段树

```cpp
//使用方法
//Build(1, 1, n) 建树
//读入ql,  qr,  qt 调用函数 XXX(1, 1, n)
using ll = long long;

const int N = "Edit";
const int M = N<<2;

int mx[M], sx[M], cx[M], mn[M], sn[M], cn[M];
ll sum[M];
int ta[M];

inline void update(int x){
    int l = x<<1, r = x<<1|1;
```

```
    sum[x] = sum[l] + sum[r];
    if (mx[l] == mx[r]) {
        mx[x] = mx[l], cx[x] = cx[l] + cx[r], sx[x] = std::max(sx[l], sx[r]);
    } else { // r>l
        if (mx[l] > mx[r]) std::swap(l,r);
        mx[x] = mx[r];
        cx[x] = cx[r];
        sx[x] = std::max(sx[r], mx[l]);
    }
    if (mn[l] == mn[r]) {
        mn[x] = mn[l], cn[x] = cn[l] + cn[r], sn[x] = std::min(sn[l], sn[r]);
    } else { // r<l
        if (mn[l] < mn[r]) std::swap(l,r);
        mn[x] = mn[r];
        cn[x] = cn[r];
        sn[x] = std::min(sn[r], mn[l]);
    }
}

//建树
inline void Build(int x, int l, int r){
    if (l == r) {
        int a;
        std::cin >> a;
        sum[x] = mx[x] = mn[x] = a; cx[x] = cn[x] = 1;
        sx[x] = -(1<<30); sn[x]=1<<30;  ta[x]=0;
        return;
    }
    int mid=(l+r)>>1;
    Build(x<<1,l,mid);
    Build(x<<1|1,mid+1,r);
    update(x);
}

inline void _add(int x, int l, int r, int t) {
    sum[x] += (ll)(r-l+1)*t;
    mn[x]+=t; sn[x]+=t; mx[x]+=t; sx[x]+=t;
    ta[x]+=t;
}

inline void _min(int x,int l,int r,int t){
    sum[x] -= (ll)cx[x]*(mx[x]-t);
    mx[x]=t; mn[x]=std::min(mn[x],t);
    if (mn[x] == mx[x]) {
        sum[x] = (ll)(r-l+1)*t; cx[x] = cn[x] = r-l+1; sx[x] = -(1<<30); sn[x] = 1<<30;
    } else {
        sn[x]=std::min(sn[x],t);
    }
}

inline void _max(int x,int l,int r,int t){
    sum[x] += (ll)cn[x]*(t-mn[x]);
    mn[x] = t; mx[x] = std::max(mx[x], t);
    if (mn[x] == mx[x]) {
```

```
69          sum[x]=(ll)(r-l+1)*t; cx[x] = cn[x] = r-l+1; sx[x] = -(1<<30); sn[x] = 1<<30;
70      } else {
71          sx[x] = std::max(sx[x], t);
72      }
73  }
74
75  inline void push(int x, int l, int r){
76      int mid = (l+r)>>1;
77      if (ta[x]) {
78          _add(x<<1, l, mid, ta[x]);
79          _add(x<<1|1, mid+1, r, ta[x]);
80          ta[x] = 0;
81      }
82      if (mx[x<<1]>mx[x] && sx[x<<1]<mx[x]) _min(x<<1, l, mid, mx[x]);
83      if (mx[x<<1|1]>mx[x] && sx[x<<1|1]<mx[x]) _min(x<<1|1, mid+1, r, mx[x]);
84      if (mn[x<<1]<mn[x] && sn[x<<1]>mn[x]) _max(x<<1, l, mid, mn[x]);
85      if (mn[x<<1|1]<mn[x] && sn[x<<1|1]>mn[x]) _max(x<<1|1, mid+1, r, mn[x]);
86  }
87
88  int ql, qr, qt;
89  int n;
90
91  //把一个区间[L,R] 里小于x 的数变成x
92  inline void Mmax(int x, int l, int r){
93      if (mn[x] >= qt) return;
94      if (ql<=l && r<=qr && qt<sn[x]){
95          _max(x, l, r, qt); return;
96      }
97      push(x, l, r); int mid = (l+r)>>1;
98      if (ql<=mid) Mmax(x<<1, l, mid);
99      if (qr>mid) Mmax(x<<1|1, mid+1, r);
100     update(x);
101 }
102
103 //把一个区间[L,R] 里大于x 的数变成x
104 inline void Mmin(int x,int l,int r) {
105     if (mx[x]<=qt) return;
106     if (ql<=l && r<=qr && qt>sx[x]) {
107         _min(x,l,r,qt); return;
108     }
109     push(x,l,r); int mid=(l+r)>>1;
110     if (ql<=mid) Mmin(x<<1, l, mid);
111     if (qr>mid) Mmin(x<<1|1, mid+1, r);
112     update(x);
113 }
114
115 //区间加值
116 inline void Add(int x, int l, int r) {
117     if (ql<=l && r<=qr) {
118         _add(x, l, r, qt); return;
119     }
120     push(x, l, r); int mid=(l+r)>>1;
121     if (ql<=mid) Add(x<<1, l, mid);
122     if (qr>mid) Add(x<<1|1, mid+1, r);
```

```
    update(x);
}

//区间最大值
inline int Max(int x, int l, int r) {
    if (ql<=l && r<=qr) return mx[x];
    push(x, l, r);
    int ret=-(1<<30); int mid=(l+r)>>1;
    if (ql<=mid) ret=std::max(ret, Max(x<<1, l, mid));
    if (qr>mid) ret=std::max(ret, Max(x<<1|1, mid+1, r));
    return ret;
}

//区间最小值
inline int Min(int x, int l, int r) {
    if (ql<=l && r<=qr)  return mn[x];
    push(x, l, r) ;
    int ret=1<<30; int mid=(l+r) >>1;
    if (ql<=mid) ret=std::min(ret, Min(x<<1, l, mid) );
    if (qr>mid)  ret=std::min(ret, Min(x<<1|1, mid+1, r) );
    return ret;
}

//区间求和
inline ll Sum(int x, int l, int r) {
    if (ql<=l && r<=qr)  return sum[x];
    push(x, l, r) ;
    ll ret=0; int mid=(l+r) >>1;
    if (ql<=mid) ret+=Sum(x<<1, l, mid) ;
    if (qr>mid)  ret+=Sum(x<<1|1, mid+1, r) ;
    return ret;
}
```

6.11 扫描线

```
// 矩形面积并（交） 求并FLAG=0，求交FLAG=1
struct Line {
    double l, r, h;
    int d;
    Line() {}
    Line(double l, double r, double h, int d) : l(l), r(r), h(h), d(d) {}
    bool operator < (const Line L) const {
        return h < L.h;
    }
}line[maxn << 1];

int FLAG;   // 求矩形面积并 FLAG = 0，求矩形面积交 FLAG = 1
int Cover[maxn << 3];
double A[maxn << 1];
double Sum[maxn << 3];
double X1[maxn << 1], X2[maxn << 1], Y1[maxn << 1], Y2[maxn << 1];

```

```
18 void pushdown(int rt, int l, int r) {
19     int mid = (l + r) >> 1;
20     if (Cover[rt] != -1) {
21         Cover[rt << 1] = Cover[rt << 1 | 1] = Cover[rt];
22         Sum[rt << 1] = (Cover[rt] > FLAG ? (A[mid + 1] - A[l]) : 0);
23         Sum[rt << 1 | 1] = (Cover[rt] > FLAG ? (A[r + 1] - A[mid + 1]) : 0);
24     }
25 }
26
27 void pushup(int rt, int l, int r) {
28     if (Cover[rt << 1] == -1 || Cover[rt << 1 | 1] == -1) Cover[rt] = -1;
29     else if (Cover[rt << 1] != Cover[rt << 1 | 1]) Cover[rt] = -1;
30     else Cover[rt] = Cover[rt << 1];
31     Sum[rt] = Sum[rt << 1] + Sum[rt << 1 | 1];
32 }
33
34 void build(int l, int r, int rt) {
35     if (l == r) {
36         Cover[rt] = 0;
37         Sum[rt] = 0;
38         return;
39     }
40     int mid = (l + r) >> 1;
41     build(l, mid, rt << 1);
42     build(mid + 1, r, rt << 1 | 1);
43     pushup(rt, l, r);
44 }
45
46 void update(int L, int R, int v, int l, int r, int rt) {
47     if (L <= l && r <= R) {
48         if (Cover[rt] != -1) {
49             Cover[rt] += v;
50             Sum[rt] = (Cover[rt] > FLAG ? (A[r + 1] - A[l]) : 0);
51             return;
52         }
53     }
54     pushdown(rt, l, r);
55     int mid = (l + r) >> 1;
56     if (L <= mid) update(L, R, v, l, mid, rt << 1);
57     if (mid < R) update(L, R, v, mid + 1, r, rt << 1 | 1);
58     pushup(rt, l, r);
59 }
60
61 int find(double key, int n, double d[]) {
62     int l = 1, r = n;
63     while (r >= l) {
64         int mid = (r + l) >> 1;
65         if (d[mid] == key) return mid;
66         else if (d[mid] > key) r = mid - 1;
67         else l = mid + 1;
68     }
69     return -1;
70 }
71
```

```
72  int init(int n) {
73      int N = 0;
74      for (int i = 1; i <= n; i++) {
75          A[++N] = X1[i];
76          line[N] = Line(X1[i], X2[i], Y1[i], 1);
77          A[++N] = X2[i];
78          line[N] = Line(X1[i], X2[i], Y2[i], -1);
79      }
80      sort(A + 1, A + N + 1);
81      sort(line + 1, line + N + 1);
82      int k = 1;
83      for (int i = 2; i <= N; i++)
84          if (A[i] != A[i - 1])
85              A[++k] = A[i];
86      build(1, k - 1, 1);
87      return k;
88  }
89
90  double query(int n, int k) {
91      double ret = 0;
92      for (int i = 1; i < n; i++) {
93          int l = find(line[i].l, k, A);
94          int r = find(line[i].r, k, A) - 1;
95          if (l <= r) update(l, r, line[i].d, 1, k - 1, 1);
96          ret += Sum[1] * (line[i + 1].h - line[i].h);
97      }
98      return ret;
99  }
100 /*
101 int main()
102 {
103     int n, T;
104     scanf("%d", &T);
105     while (T--) {
106         scanf("%d", &n);
107         for (int i = 1; i <= n; i++)
108             scanf("%lf%lf%lf%lf", &X1[i], &Y1[i], &X2[i], &Y2[i]);
109         int k = init(n);
110         double ans = query(n << 1, k);
111         printf("%.2lf\n", ans);
112     }
113 }
114 */
115
116 //================================================================================================
117
118 // 矩形周长并
119 int Sum[maxn << 3], cnt[maxn << 3], vNum[maxn << 3];
120 bool lbd[maxn << 3], rbd[maxn << 3];
121 double X1[maxn << 1], X2[maxn << 1], Y1[maxn << 1], Y2[maxn << 1];
122 double A[maxn << 1];
123
124 struct Line {
125     double l, r, h;
```

```
126     int label;
127     Line() {}
128     Line(double l, double r, double h, int label) :l(l), r(r), h(h), label(label) {}
129     bool operator < (const Line L) const {
130         return h < L.h;
131     }
132 }line[maxn << 1];
133 
134 void pushup(int l, int r, int rt) {
135     if (cnt[rt]) {
136         lbd[rt] = rbd[rt] = true;
137         Sum[rt] = A[r + 1] - A[l];
138         vNum[rt] = 2;
139     }
140     else if (l == r) Sum[rt] = vNum[rt] = lbd[rt] = rbd[rt] = 0;
141     else {
142         lbd[rt] = lbd[rt << 1];
143         rbd[rt] = rbd[rt << 1 | 1];
144         Sum[rt] = Sum[rt << 1] + Sum[rt << 1 | 1];
145         vNum[rt] = vNum[rt << 1] + vNum[rt << 1 | 1];
146         if (rbd[rt << 1] && lbd[rt << 1 | 1]) vNum[rt] -= 2;
147     }
148 }
149 
150 void update(int L, int R, int v, int l, int r, int rt) {
151     if (L <= l && r <= R) {
152         cnt[rt] += v;
153         pushup(l, r, rt);
154         return;
155     }
156     int mid = (l + r) >> 1;
157     if (L <= mid) update(L, R, v, l, mid, rt << 1);
158     if (R > mid) update(L, R, v, mid + 1, r, rt << 1 | 1);
159     pushup(l, r, rt);
160 }
161 
162 int find(double key, int n, double d[]) {
163     int l = 1, r = n;
164     while (r >= l) {
165         int mid = (r + l) >> 1;
166         if (d[mid] == key) return mid;
167         else if (d[mid] > key) r = mid - 1;
168         else l = mid + 1;
169     }
170     return -1;
171 }
172 
173 int init(int n) {
174     for (int i = 1; i <= n; i++) {
175         A[i] = X1[i]; A[i + n] = X2[i];
176         line[i].l = X1[i]; line[i].r = X2[i];
177         line[i].h = Y1[i]; line[i].label = 1;
178         line[i + n].l = X1[i]; line[i + n].r = X2[i];
179         line[i + n].h = Y2[i]; line[i + n].label = -1;
```

```
    }
    n <<= 1;
    int k = 1;
    sort(A + 1, A + n + 1);
    sort(line + 1, line + n + 1);
    for (int i = 2; i <= n; i++)
        if (A[i] != A[i - 1])
            A[++k] = A[i];
    return k;
}

double query(int n, int k) {
    double ret = 0, lst = 0;
    for (int i = 1; i <= n; i++) {
        if (line[i].l < line[i].r) {
            int l = find(line[i].l, k, A);
            int r = find(line[i].r, k, A);
            update(l, r - 1, line[i].label, 1, k - 1, 1);
        }
        ret += vNum[1] * (line[i + 1].h - line[i].h);
        ret += abs(Sum[1] - lst);
        lst = Sum[1];
    }
    return ret;
}
/*
int main()
{
    int n;
    while (~scanf("%d", &n)) {
        for (int i = 1; i <= n; i++)
            scanf("%lf%lf%lf%lf", &X1[i], &Y1[i], &X2[i], &Y2[i]);
        int k = init(n);
        double ans = query(n << 1, k);
        printf("%lf\n", ans);
    }
    return 0;
}
*/
```

6.12 固定大小矩形最大点覆盖

```
// 扫描线 求 矩形最大点覆盖
struct Line {
    ll x, y1, y2, k;    // k 为矩形权值
    bool operator < (const Line nod) const {
        return x < nod.x || (x == nod.x && k < nod.k);
    }
}line[maxn];
struct segTree {
    ll ma, l, r, lazy;
}tree[maxn << 2];
```

```cpp
ll yy[maxn];
int cnt, ycnt;
void pushup(int rt) {
    tree[rt].ma = max(tree[rt << 1].ma, tree[rt << 1 | 1].ma) + tree[rt].lazy;
}
void build(int l, int r, int rt) {
    tree[rt].ma = tree[rt].lazy = 0;
    tree[rt].l = yy[l], tree[rt].r = yy[r];
    if (r - l == 1) return;
    int mid = (l + r) >> 1;
    build(l, mid, rt << 1);
    build(mid, r, rt << 1 | 1);
    pushup(rt);
}
void update(ll L, ll R, ll w, int rt) {
    if (tree[rt].l >= L && tree[rt].r <= R) {
        tree[rt].lazy += w;
        tree[rt].ma += w;
        return;
    }
    if (L < tree[rt << 1].r)
        update(L, min(R, tree[rt << 1].r), w, rt << 1);
    if (R > tree[rt << 1 | 1].l)
        update(max(tree[rt << 1 | 1].l, L), R, w, rt << 1 | 1);
    pushup(rt);
}
int main()
{
    ll n, W, H, x, y, w, ma;
    while (~scanf("%lld%lld%lld", &n, &W, &H)) {
        cnt = 0; ycnt = 1; ma = 0;
        for (int i = 1; i <= n; i++) {
            scanf("%lld%lld%lld", &x, &y, &w);
            line[cnt].x = x; line[cnt].y1 = y; line[cnt].y2 = y + H;
            line[cnt++].k = w;
            line[cnt].x = x + W; line[cnt].y1 = y; line[cnt].y2 = y + H;
            line[cnt++].k = -w;
            yy[ycnt++] = y;
            yy[ycnt++] = y + H;
        }
        sort(yy + 1, yy + ycnt);
        ycnt = unique(yy + 1, yy + ycnt) - (yy + 1);
        sort(line, line + cnt);
        build(1, ycnt, 1);
        for (int i = 0; i < cnt; i++) {
            update(line[i].y1, line[i].y2, line[i].k, 1);
            if (line[i].k > 0) ma = max(ma, tree[1].ma);
        }
        printf("%lld\n", ma);
    }
    return 0;
}
```

6.13 二维线段树 (单点更新区间最值)

```
// 二维线段树单点更新+区间最值 树套树实现
int n;
int y2rt[maxn], x2rt[maxn];

struct Nodey {
    int l, r;
    int Max, Min;
};

struct Nodex {
    int l, r;
    Nodey nodey[maxn << 2];

    void build(int l, int r, int rt) {
        nodey[rt].l = l;
        nodey[rt].r = r;
        nodey[rt].Max = -inf;
        nodey[rt].Min = inf;
        if (l == r) {
            y2rt[l] = rt;
            return;
        }
        int mid = (l + r) >> 1;
        build(l, mid, rt << 1);
        build(mid + 1, r, rt << 1 | 1);
    }

    int queryMin(int rt, int L, int R) {
        if (nodey[rt].l == L && nodey[rt].r == R)
            return nodey[rt].Min;
        int mid = (nodey[rt].l + nodey[rt].r) >> 1;
        if (R <= mid) return queryMin(rt << 1, L, R);
        else if (L > mid) return queryMin(rt << 1 | 1, L, R);
        else return min(queryMin(rt << 1, L, mid), queryMin(rt << 1 | 1, mid + 1, R));
    }

    int queryMax(int rt, int L, int R) {
        if (nodey[rt].l == L && nodey[rt].r == R)
            return nodey[rt].Max;
        int mid = (nodey[rt].l + nodey[rt].r) >> 1;
        if (R <= mid) return queryMax(rt << 1, L, R);
        else if (L > mid) return queryMax((rt << 1) | 1, L, R);
        else return max(queryMax(rt << 1, L, mid), queryMax((rt << 1) | 1, mid + 1, R));
    }
}nodex[maxn << 2];

void build(int l, int r, int rt) {
    nodex[rt].l = l;
    nodex[rt].r = r;
    nodex[rt].build(1, n, 1);
    if (l == r) {
        x2rt[l] = rt;
```

```
        return;
    }
    int mid = (l + r) >> 1;
    build(l, mid, rt << 1);
    build(mid + 1, r, rt << 1 | 1);
}

// 点修改
void update(int x, int y, int val) {
    int rtx = x2rt[x];
    int rty = y2rt[y];
    nodex[rtx].nodey[rty].Min = nodex[rtx].nodey[rty].Max = val;
    for (int i = rtx; i; i >>= 1) {
        for (int j = rty; j; j >>= 1) {
            if (i == rtx && j == rty)continue;
            if (j == rty) {
                nodex[i].nodey[j].Min = min(nodex[i << 1].nodey[j].Min, nodex[(i << 1) | 1].
                    nodey[j].Min);
                nodex[i].nodey[j].Max = max(nodex[i << 1].nodey[j].Max, nodex[(i << 1) | 1].
                    nodey[j].Max);
            }
            else {
                nodex[i].nodey[j].Min = min(nodex[i].nodey[j << 1].Min, nodex[i].nodey[(j <<
                    1) | 1].Min);
                nodex[i].nodey[j].Max = max(nodex[i].nodey[j << 1].Max, nodex[i].nodey[(j <<
                    1) | 1].Max);
            }
        }
    }
}

int queryMin(int rt, int x1, int x2, int y1, int y2) {
    if (nodex[rt].l == x1 && nodex[rt].r == x2)
        return nodex[rt].queryMin(1, y1, y2);
    int mid = (nodex[rt].l + nodex[rt].r) >> 1;
    if (x2 <= mid)return queryMin(rt << 1, x1, x2, y1, y2);
    else if (x1 > mid)return queryMin(rt << 1 | 1, x1, x2, y1, y2);
    else return min(queryMin(rt << 1, x1, mid, y1, y2), queryMin(rt << 1 | 1, mid + 1, x2, y1,
        y2));
}

int queryMax(int rt, int x1, int x2, int y1, int y2) {
    if (nodex[rt].l == x1 && nodex[rt].r == x2)
        return nodex[rt].queryMax(1, y1, y2);
    int mid = (nodex[rt].l + nodex[rt].r) >> 1;
    if (x2 <= mid)return queryMax(rt << 1, x1, x2, y1, y2);
    else if (x1 > mid)return queryMax(rt << 1 | 1, x1, x2, y1, y2);
    else return max(queryMax(rt << 1, x1, mid, y1, y2), queryMax(rt << 1 | 1, mid + 1, x2, y1,
        y2));
}
```

6.14 二维线段树 (区间加值单点查询)

```cpp
// 二维线段树区间加值+单点查询 树套树实现
int n;
int x2rt[maxn], y2rt[maxn];

struct Nodey {
    int l, r;
    int val;
};

struct Nodex {
    int l, r;
    Nodey nodey[maxn << 2];

    void build(int l, int r, int rt) {
        nodey[rt].l = l;
        nodey[rt].r = r;
        nodey[rt].val = 0;
        if (l == r) {
            y2rt[l] = rt;
            return;
        }
        int mid = (l + r) >> 1;
        build(l, mid, rt << 1);
        build(mid + 1, r, rt << 1 | 1);
    }

    void addVal(int rt, int L, int R, int val) {
        if (nodey[rt].l == L && nodey[rt].r == R) {
            nodey[rt].val += val;
            return;
        }
        int mid = (nodey[rt].l + nodey[rt].r) >> 1;
        if (R <= mid) addVal(rt << 1, L, R, val);
        else if (L > mid) addVal(rt << 1 | 1, L, R, val);
        else {
            addVal(rt << 1, L, mid, val);
            addVal(rt << 1 | 1, mid + 1, R, val);
        }
    }
}nodex[maxn << 2];

void build(int l, int r, int rt) {
    nodex[rt].l = l;
    nodex[rt].r = r;
    nodex[rt].build(1, n, 1);
    if (l == r) {
        x2rt[l] = rt;
        return;
    }
    int mid = (l + r) >> 1;
    build(l, mid, rt << 1);
    build(mid + 1, r, rt << 1 | 1);
```

```
}

void addVal(int rt, int x1, int x2, int y1, int y2, int val) {
    if (nodex[rt].l == x1 && nodex[rt].r == x2) {
        nodex[rt].addVal(1, y1, y2, val);
        return;
    }
    int mid = (nodex[rt].l + nodex[rt].r) >> 1;
    if (x2 <= mid) addVal(rt << 1, x1, x2, y1, y2, val);
    else if (x1 > mid) addVal(rt << 1 | 1, x1, x2, y1, y2, val);
    else {
        addVal(rt << 1, x1, mid, y1, y2, val);
        addVal(rt << 1 | 1, mid + 1, x2, y1, y2, val);
    }
}

int getVal(int x, int y) {
    int ret = 0;
    for (int i = x2rt[x]; i; i >>= 1)
        for (int j = y2rt[y]; j; j >>= 1)
            ret += nodex[i].nodey[j].val;
    return ret;
}
```

6.15 主席树

```
// 主席树 支持查询[l,r]区间第k大，以及区间内不重复数字个数
// M = maxn * 30;
int n, q, m, tot;   // n为数组大小，m为离散化后数组大小
int A[maxn], T[maxn];   // A为原数组，T为离散化数组
int tree[M], lson[M], rson[M], Cnt[M];  // Cnt[i]表示节点i的子树包含数字的总数

void Init_hash() {
    for (int i = 1; i <= n; i++) T[i] = A[i];
    sort(T + 1, T + n + 1);
    m = unique(T + 1, T + n + 1) - T - 1;
}

inline int Hash(int x) { return lower_bound(T + 1, T + m + 1, x) - T; }

int build(int l, int r) {
    int root = tot++;
    Cnt[root] = 0;
    if (l != r) {
        int mid = (l + r) >> 1;
        lson[root] = build(l, mid);
        rson[root] = build(mid + 1, r);
    }
    return root;
}

int update(int root, int pos, int val) {
```

```
    int newroot = tot++, tmp = newroot;
    Cnt[newroot] = Cnt[root] + val;
    int l = 1, r = m;
    while (l < r) {
        int mid = (l + r) >> 1;
        if (pos <= mid) {
            lson[newroot] = tot++; rson[newroot] = rson[root];
            newroot = lson[newroot]; root = lson[root];
            r = mid;
        }
        else {
            rson[newroot] = tot++; lson[newroot] = lson[root];
            newroot = rson[newroot]; root = rson[root];
            l = mid + 1;
        }
        Cnt[newroot] = Cnt[root] + val;
    }
    return tmp;
}

void init() {   // 查询l~r第k大
    Init_hash();
    tree[0] = build(1, m);
    for (int i = 1; i <= n; i++) {
        int pos = Hash(A[i]);
        tree[i] = update(tree[i - 1], pos, 1);
    }
}

int query(int lrt, int rrt, int k) {    // 查询l~r第k大: T[query(tree[l - 1], tree[r], k)]
    int l = 1, r = m;
    while (l < r) {
        int mid = (l + r) >> 1;
        if (Cnt[lson[rrt]] - Cnt[lson[lrt]] >= k) {
            r = mid;
            lrt = lson[lrt];
            rrt = lson[rrt];
        }
        else {
            l = mid + 1;
            k -= Cnt[lson[rrt]] - Cnt[lson[lrt]];
            lrt = rson[lrt];
            rrt = rson[rrt];
        }
    }
    return l;
}

void init() {   // 查询l~r内不重复数字个数
    tree[0] = build(1, n);
    map<int, int>mp;
    for (int i = 1; i <= n; i++) {
        if (mp.find(A[i]) == mp.end())
            tree[i] = update(tree[i - 1], i, 1);
```

```
        else {
            int tmp = update(tree[i - 1], mp[A[i]], -1);
            tree[i] = update(tmp, i, 1);
        }
        mp[A[i]] = i;
    }
}

int query(int root, int pos) {  // 查询l~r内不重复数字个数: query(tree[r], l)
    int ret = 0;
    int l = 1, r = n;
    while (pos > l) {
        int mid = (l + r) >> 1;
        if (pos <= mid) {
            ret += Cnt[rson[root]];
            root = lson[root];
            r = mid;
        }
        else {
            root = rson[root];
            l = mid + 1;
        }
    }
    return ret + Cnt[root];
}
```

6.16 主席树动态 k 大

```
// 主席树求[l,r]第k大, 可单点修改 使用树状数组套主席树在线操作, 树状数组维护改变量
// M = maxn * 40;
int n, q, m, tot;
int A[maxn], T[maxn];
int tree[maxn], lson[M], rson[M], Cnt[M];
int Ntree[maxn], use[maxn]; // Ntree[i]表示动态第i棵树的树根, use[i]表示第i个树根是谁在使用

struct Query {
    int kind;
    int l, r, k;
}query[10005];

void Init_hash(int k) {
    sort(T, T + k);
    m = unique(T, T + k) - T;
}

int Hash(int x) { return lower_bound(T, T + m, x) - T; }

int build(int l, int r) {
    int root = tot++;
    Cnt[root] = 0;
    if (l != r) {
        int mid = (l + r) >> 1;
```

```
25          lson[root] = build(l, mid);
26          rson[root] = build(mid + 1, r);
27      }
28      return root;
29  }
30
31  int update(int root, int pos, int val) {
32      int newroot = tot++, tmp = newroot;
33      int l = 0, r = m - 1;
34      Cnt[newroot] = Cnt[root] + val;
35      while (l < r) {
36          int mid = (l + r) >> 1;
37          if (pos <= mid) {
38              lson[newroot] = tot++; rson[newroot] = rson[root];
39              newroot = lson[newroot]; root = lson[root];
40              r = mid;
41          }
42          else {
43              rson[newroot] = tot++; lson[newroot] = lson[root];
44              newroot = rson[newroot]; root = rson[root];
45              l = mid + 1;
46          }
47          Cnt[newroot] = Cnt[root] + val;
48      }
49      return tmp;
50  }
51
52  inline int lowbit(int x) { return x & (-x); }
53
54  int sum(int x) {
55      int ret = 0;
56      while (x > 0) {
57          ret += Cnt[lson[use[x]]];
58          x -= lowbit(x);
59      }
60      return ret;
61  }
62
63  void Modify(int x, int pos, int val) {
64      while (x <= n) {
65          Ntree[x] = update(Ntree[x], pos, val);
66          x += lowbit(x);
67      }
68  }
69
70  int Query(int left, int right, int k) {
71      int lrt = tree[left - 1];
72      int rrt = tree[right];
73      int l = 0, r = m - 1;
74      for (int i = left - 1; i; i -= lowbit(i)) use[i] = Ntree[i];
75      for (int i = right; i; i -= lowbit(i)) use[i] = Ntree[i];
76      while (l < r) {
77          int mid = (l + r) >> 1;
78          // sum(right) - sum(left - 1)为改变量, Cnt[lson[rrt]] - Cnt[lson[lrt]]为基础差值
```

```
79          int tmp = sum(right) - sum(left - 1) + Cnt[lson[rrt]] - Cnt[lson[lrt]];
80          if (tmp >= k) {
81              r = mid;
82              for (int i = left - 1; i; i -= lowbit(i))
83                  use[i] = lson[use[i]];
84              for (int i = right; i; i -= lowbit(i))
85                  use[i] = lson[use[i]];
86              lrt = lson[lrt];
87              rrt = lson[rrt];
88          }
89          else {
90              l = mid + 1;
91              k -= tmp;
92              for (int i = left - 1; i; i -= lowbit(i))
93                  use[i] = rson[use[i]];
94              for (int i = right; i; i -= lowbit(i))
95                  use[i] = rson[use[i]];
96              lrt = rson[lrt];
97              rrt = rson[rrt];
98          }
99      }
100     return l;
101 }
102
103 int main()
104 {
105     int Tcase;
106     char op[10];
107     scanf("%d", &Tcase);
108     while (Tcase--) {
109         scanf("%d%d", &n, &q);
110         tot = 0; m = 0;
111         for (int i = 1; i <= n; i++) {
112             scanf("%d", &A[i]);
113             T[m++] = A[i];
114         }
115         for (int i = 0; i < q; i++) {
116             scanf("%s", op);
117             if (op[0] == 'Q') {
118                 query[i].kind = 0;
119                 scanf("%d%d%d", &query[i].l, &query[i].r, &query[i].k);
120             }
121             else {
122                 query[i].kind = 1;
123                 scanf("%d%d", &query[i].l, &query[i].r);
124                 T[m++] = query[i].r;
125             }
126         }
127         Init_hash(m);
128         tree[0] = build(0, m - 1);
129         for (int i = 1; i <= n; i++)
130             tree[i] = update(tree[i - 1], Hash(A[i]), 1);
131         for (int i = 1; i <= n; i++) Ntree[i] = tree[0];
132         for (int i = 0; i < q; i++) {
```

```
            if (query[i].kind == 0)
                printf("%d\n", T[Query(query[i].l, query[i].r, query[i].k)]);
            else {
                Modify(query[i].l, Hash(A[query[i].l]), -1);
                Modify(query[i].l, Hash(query[i].r), 1);
                A[query[i].l] = query[i].r;
            }
        }
    }
    return 0;
}
```

6.17 Treap 树

```
typedef int value;

enum { LEFT, RIGHT };
struct node {
    int size, priority;
    value x, subtree;
    node *child[2];
    node(const value &x): size(1), x(x), subtree(x) {
        priority = rand();
        child[0] = child[1] = nullptr;
    }
};

inline int size(const node *a) { return a == nullptr ? 0 : a->size; }

inline void update(node *a) {
    if (a == nullptr) return;
    a->size = size(a->child[0]) + size(a->child[1]) + 1;
    a->subtree = a->x;
    if (a->child[LEFT] != nullptr) a->subtree = a->child[LEFT]->subtree + a->subtree;
    if (a->child[RIGHT] != nullptr) a->subtree = a->subtree + a->child[RIGHT]->subtree;
}

node *rotate(node *a, bool d) {
    node *b = a->child[d];
    a->child[d] = b->child[!d];
    b->child[!d] = a;
    update(a); update(b);
    return b;
}

node *insert(node *a, int index, const value &x) {
    if (a == nullptr && index == 0) return new node(x);
    int middle = size(a->child[LEFT]);
    bool dir = index > middle;
    if (!dir) a->child[LEFT]  = insert(a->child[LEFT], index, x);
    else      a->child[RIGHT] = insert(a->child[RIGHT], index - middle - 1, x);
    update(a);
```

```
39      if (a->priority > a->child[dir]->priority) a = rotate(a, dir);
40      return a;
41  }
42
43  node *erase(node *a, int index) {
44      assert(a != nullptr);
45      int middle = size(a->child[LEFT]);
46      if (index == middle) {
47          if (a->child[LEFT] == nullptr && a->child[RIGHT] == nullptr) {
48              delete a;
49              return nullptr;
50          } else if (a->child[LEFT] == nullptr) a = rotate(a, RIGHT);
51          else if (a->child[RIGHT] == nullptr) a = rotate(a, LEFT);
52          else a = rotate(a, a->child[LEFT]->priority < a->child[RIGHT]->priority);
53          a = erase(a, index);
54      } else {
55          bool dir = index > middle;
56          if (!dir) a->child[LEFT] = erase(a->child[LEFT], index);
57          else      a->child[RIGHT] = erase(a->child[RIGHT], index - middle - 1);
58      }
59      update(a);
60      return a;
61  }
62
63  void modify(node *a, int index, const value &x) {
64      assert(a != nullptr);
65      int middle = size(a->child[LEFT]);
66      if (index == middle) a->x = x;
67      else {
68          bool dir = index > middle;
69          if (!dir) modify(a->child[LEFT], index, x);
70          else      modify(a->child[RIGHT], index - middle - 1, x);
71      }
72      update(a);
73  }
74
75  value query(node *a, int l, int r) {
76      assert(a != nullptr);
77      if (l <= 0 && size(a) - 1 <= r) return a->subtree;
78      int middle = size(a->child[LEFT]);
79      if (r < middle) return query(a->child[LEFT], l, r);
80      if (middle < l) return query(a->child[RIGHT], l - middle - 1, r - middle - 1);
81      value res = a->x;
82      if (l < middle && a->child[LEFT] != nullptr)
83          res = query(a->child[LEFT], l, r) + res;
84      if (middle < r && a->child[RIGHT] != nullptr)
85          res = res + query(a->child[RIGHT], l - middle - 1, r - middle - 1);
86      return res;
87  }
```

6.18 函数式 Treap

```cpp
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>
#include <cstdlib>
#include <ctime>
using namespace std;
const int MAXN=100001;
static void read(int &n) {
    char c='+';int x=0;bool flag=0;
    while(c<'0'||c>'9'){c=getchar();if(c=='-')flag=1;}
    while(c>='0'&&c<='9'){x=(x<<1)+(x<<3)+(c-48);c=getchar();}
    flag==1?n=-x:n=x;
}
int ch[MAXN][3];// 0左孩子 1右孩子
int val[MAXN];// 每一个点的权值
int pri[MAXN];// 随机生成的附件权值
int siz[MAXN];// 以i为节点的树的节点数量
int sz;// 总结点的数量
void update(int x) {
    siz[x]=1+siz[ch[x][0]]+siz[ch[x][1]];
}
int new_node(int v) {
    siz[++sz]=1;// 新开辟一个节点
    val[sz]=v;
    pri[sz]=rand();
    return sz;
}
int merge(int x,int y) {// 合并
    if(!x||!y)    return x+y;// x和y中必定有一个是0
    if(pri[x]<pri[y])// 把x加到左边的树上
    {
        ch[x][1]=merge(ch[x][1],y);// 不懂的看GIF图
        update(x);
        return x;
    }
    else
    {
        ch[y][0]=merge(x,ch[y][0]);
        update(y);
        return y;
    }
}
void split(int now,int k,int &x,int &y) {
    if(!now) x=y=0;// 到达叶子节点
    else {
        if(val[now]<=k)// 分离右子树
            x=now,split(ch[now][1],k,ch[now][1],y);
        else
            y=now,split(ch[now][0],k,x,ch[now][0]);
        update(now);
    }
}
```

```
int kth(int now,int k) {// 查询排名
    while(1) {
        if(k<=siz[ch[now][0]])
            now=ch[now][0];// 在左子树中，且数量小于左子树的大小，迭代寻找
        else if(k==siz[ch[now][0]]+1)
            return now;// 找到了
        else
            k-=siz[ch[now][0]]+1,now=ch[now][1];// 去右子树找
    }
}
int main() {
    srand((unsigned)time(NULL));
    int n;
    read(n);
    int root=0,x,y,z;
    for(int i=1;i<=n;i++) {
        int how,a;
        read(how);read(a);
        if(how==1) {// 插入
            split(root,a,x,y);
            root=merge(merge(x,new_node(a)),y);
        }
        else if(how==2) {//删除x
        {
            split(root,a,x,z);
            split(x,a-1,x,y);
            y=merge(ch[y][0],ch[y][1]);
            root=merge(merge(x,y),z);
        }
        else if(how==3) {//查询x的排名
            split(root,a-1,x,y);
            printf("%d\n",siz[x]+1);
            root=merge(x,y);
        }
        else if(how==4) {// 查询排名为x的数
            printf("%d\n",val[kth(root,a)]);
        }
        else if(how==5) {// 求x的前驱
            split(root,a-1,x,y);
            printf("%d\n",val[kth(x,siz[x])]);
            root=merge(x,y);
        }
        else if(how==6) {// 求x的后继
            split(root,a,x,y);
            printf("%d\n",val[kth(y,1)]);
            root=merge(x,y);
        }
    }
    return 0;
}
```

6.19 Splay 树

```cpp
// splay tree. HDU 3726: 插入、删除、合并

const int MAXN = 20010;
struct Node;
Node* null;
struct Node {
    Node *ch[2], *fa;//指向儿子和父亲结点
    int size, key;
    Node() {
        ch[0] = ch[1] = fa = null;
    }
    inline void setc(Node* p, int d) {
        ch[d] = p;
        p->fa = this;
    }
    inline bool d() {
        return fa->ch[1] == this;
    }
    void push_up() {
        size = ch[0]->size + ch[1]->size + 1;
    }
    void clear() {
        size = 1;
        ch[0] = ch[1] = fa = null;
    }
    inline bool isroot() {
        return fa == null || this != fa->ch[0] && this != fa->ch[1];
    }
};

inline void rotate(Node* x) {
    Node *f = x->fa, *ff = x->fa->fa;
    int c = x->d(), cc = f->d();
    f->setc(x->ch[!c], c);
    x->setc(f, !c);
    if (ff->ch[cc] == f)ff->setc(x, cc);
    else x->fa = ff;
    f->push_up();
}

inline void splay(Node* &root, Node* x, Node* goal) {
    while (x->fa != goal) {
        if (x->fa->fa == goal)rotate(x);
        else {
            bool f = x->fa->d();
            x->d() == f ? rotate(x->fa) : rotate(x);
            rotate(x);
        }
    }
    x->push_up();
    if (goal == null)root = x;
}
```

```
53
54  //找到 r 子树里面的第 k 个
55  Node* get_kth(Node* r, int k) {
56      Node* x = r;
57      while (x->ch[0]->size + 1 != k) {
58          if (k < x->ch[0]->size + 1)x = x->ch[0];
59          else {
60              k -= x->ch[0]->size + 1;
61              x = x->ch[1];
62          }
63      }
64      return x;
65  }
66
67
68  void erase(Node* &root, Node* x) {
69      splay(root, x, null);
70      Node* t = root;
71      if (t->ch[1] != null) {
72          root = t->ch[1];
73          splay(root, get_kth(root, 1), null);
74          root->setc(t->ch[0], 0);
75      }
76      else {
77          root = root->ch[0];
78      }
79      root->fa = null;
80      if (root != null)root->push_up();
81  }
82
83  void insert(Node* &root, Node* x) {
84      if (root == null) {
85          root = x;
86          return;
87      }
88      Node* now = root;
89      Node* pre = root->fa;
90      while (now != null) {
91          pre = now;
92          now = now->ch[x->key >= now->key];
93      }
94      x->clear();
95      pre->setc(x, x->key >= pre->key);
96      splay(root, x, null);
97  }
98
99  void merge(Node* &A, Node* B) {
100     if (A->size <= B->size)swap(A, B);
101     queue<Node*>Q;
102     Q.push(B);
103     while (!Q.empty()) {
104         Node* fr = Q.front();
105         Q.pop();
106         if (fr->ch[0] != null)Q.push(fr->ch[0]);
```

```
107         if (fr->ch[1] != null)Q.push(fr->ch[1]);
108         fr->clear();
109         insert(A, fr);
110     }
111 }
112 
113 Node pool[MAXN], *tail;
114 
115 struct Edge {
116     int u, v;
117 }edge[60010];
118 int a[MAXN];
119 bool del[60010];
120 struct QUERY {
121     char op[10];
122     int u, v;
123 }query[500010];
124 int y[500010];
125 
126 Node* node[MAXN];
127 Node* root[MAXN];
128 int F[MAXN];
129 int find(int x) {
130     if (F[x] == -1)return x;
131     return F[x] = find(F[x]);
132 }
133 
134 void debug(Node *root) {
135     if (root == null)return;
136     debug(root->ch[0]);
137     printf("size: %d, key = %d\n", root->size, root->key);
138     debug(root->ch[1]);
139 }
140 
141 int main()
142 {
143     int n, m;
144     int iCase = 0;
145     while (scanf("%d%d", &n, &m) == 2) {
146         if (n == 0 && m == 0)break;
147         iCase++;
148         memset(F, -1, sizeof(F));
149         tail = pool;
150         null = tail++;
151         null->size = 0; null->ch[0] = null->ch[1] = null->fa = null;
152         null->key = 0;
153         for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
154         for (int i = 0; i < m; i++) {
155             scanf("%d%d", &edge[i].u, &edge[i].v);
156             del[i] = false;
157         }
158         int Q = 0;
159         while (1) {
160             scanf("%s", &query[Q].op);
```

```
            if (query[Q].op[0] == 'E')break;
            if (query[Q].op[0] == 'D') {
                scanf("%d", &query[Q].u);
                query[Q].u--;
                del[query[Q].u] = true;
            }
            else if (query[Q].op[0] == 'Q') {
                scanf("%d%d", &query[Q].u, &query[Q].v);
            }
            else {
                scanf("%d%d", &query[Q].u, &query[Q].v);
                y[Q] = a[query[Q].u];
                a[query[Q].u] = query[Q].v;
            }
            Q++;
        }
        for (int i = 1; i <= n; i++) {
            node[i] = tail++;
            node[i]->clear();
            node[i]->key = a[i];
            root[i] = node[i];
        }
        for (int i = 0; i < m; i++)
            if (!del[i]) {
                int u = edge[i].u;
                int v = edge[i].v;
                int t1 = find(u);
                int t2 = find(v);
                if (t1 == t2)continue;
                F[t2] = t1;
                merge(root[t1], root[t2]);
            }
        vector<int>ans;
        for (int i = Q - 1; i >= 0; i--) {
            if (query[i].op[0] == 'D') {
                int u = edge[query[i].u].u;
                int v = edge[query[i].u].v;
                int t1 = find(u);
                int t2 = find(v);
                if (t1 == t2)continue;
                F[t2] = t1;
                merge(root[t1], root[t2]);
            }
            else if (query[i].op[0] == 'Q') {
                int u = query[i].u;
                int k = query[i].v;
                u = find(u);
                if (k <= 0 || k > root[u]->size) {
                    ans.push_back(0);
                }
                else {
                    k = root[u]->size - k + 1;
                    Node* p = get_kth(root[u], k);
                    ans.push_back(p->key);
```

```
                }
            }
            else {
                int u = query[i].u;
                int t1 = find(u);
                Node* p = node[u];
                erase(root[t1], p);
                p->clear();
                p->key = y[i];
                a[u] = y[i];
                insert(root[t1], p);
            }
        }
        double ret = 0;
        int sz = ans.size();
        for (int i = 0; i < sz; i++)ret += ans[i];
        if (sz)ret /= sz;
        printf("Case %d: %.6lf\n", iCase, ret);
    }
    return 0;
}
```

6.20 Splay 树

```
// splay tree: 仅伸展操作
#include<cstdio>
#include<iostream>
#include<algorithm>
#include<cstring>
#include<queue>
using namespace std;

const int maxn = 100005;
struct Node;
Node* null;
struct Node {
    Node *ch[2], *fa;
    int size, rev, key;
    Node() { ch[0] = ch[1] = fa = null; rev = 0; }
    inline void push_up() {
        if (this == null)return;
        size = ch[0]->size + ch[1]->size + 1;
    }
    inline void setc(Node* p, int d) {
        ch[d] = p;
        p->fa = this;
    }
    inline bool d() {
        return fa->ch[1] == this;
    }
    void clear() {
        size = 1;
```

```
29          ch[0] = ch[1] = fa = null;
30          rev = 0;
31      }
32      void Update_Rev() {
33          if (this == null)return;
34          swap(ch[0], ch[1]);
35          rev ^= 1;
36      }
37      inline void push_down() {
38          if (this == null)return;
39          if (rev) {
40              ch[0]->Update_Rev();
41              ch[1]->Update_Rev();
42              rev = 0;
43          }
44      }
45      inline bool isroot() {
46          return fa == null || this != fa->ch[0] && this != fa->ch[1];
47      }
48  };
49  Node pool[maxn], *tail;
50  Node *node[maxn], *root;
51
52  inline void rotate(Node* x) {
53      Node *f = x->fa, *ff = x->fa->fa;
54      f->push_down();
55      x->push_down();
56      int c = x->d(), cc = f->d();
57      f->setc(x->ch[!c], c);
58      x->setc(f, !c);
59      if (ff->ch[cc] == f)ff->setc(x, cc);
60      else x->fa = ff;
61      f->push_up();
62  }
63
64  inline void splay(Node* &root, Node* x, Node* goal) {
65      while (x->fa != goal) {
66          if (x->fa->fa == goal) rotate(x);
67          else {
68              x->fa->fa->push_down();
69              x->fa->push_down();
70              x->push_down();
71              bool f = x->fa->d();
72              x->d() == f ? rotate(x->fa) : rotate(x);
73              rotate(x);
74          }
75      }
76      x->push_up();
77      if (goal == null)root = x;
78  }
79
80  Node* get_kth(Node* r, int k) {
81      Node* x = r;
82      x->push_down();
```

```
    while (x->ch[0]->size + 1 != k) {
        if (k < x->ch[0]->size + 1)x = x->ch[0];
        else {
            k -= x->ch[0]->size + 1;
            x = x->ch[1];
        }
        x->push_down();
    }
    return x;
}

Node* get_next(Node* p) {
    p->push_down();
    p = p->ch[1];
    p->push_down();
    while (p->ch[0] != null) {
        p = p->ch[0];
        p->push_down();
    }
    return p;
}

void build(Node* &x, int l, int r, Node* fa) {
    if (l > r)return;
    int mid = (l + r) >> 1;
    x = tail++;
    x->clear();
    x->fa = fa;
    node[mid] = x;
    build(x->ch[0], l, mid - 1, x);
    build(x->ch[1], mid + 1, r, x);
    x->push_up();
}

void init(int n) {
    tail = pool;
    null = tail++;
    null->fa = null->ch[0] = null->ch[1] = null;
    null->size = 0; null->rev = 0;
    Node *p = tail++;
    p->clear();
    root = p;
    p = tail++;
    p->clear();
    root->setc(p, 1);
    build(root->ch[1]->ch[0], 1, n, root->ch[1]);
    root->ch[1]->push_up();
    root->push_up();
}

int a[maxn], b[maxn];
bool cmp(int i, int j) { return a[i] < a[j] || (a[i] == a[j] && i < j); }

int main() {
```

```
    int n;
    while (scanf("%d", &n), n) {
        for (int i = 1; i <= n; i++) {
            scanf("%d", &a[i]);
            b[i] = i;
        }
        init(n);
        sort(b + 1, b + n + 1, cmp);
        for (int i = 1; i <= n; i++) {
            splay(root, node[b[i]], null);
            int sz = root->ch[0]->size;
            printf("%d", root->ch[0]->size);
            if (i == n) printf("\n");
            else printf("␣");
            splay(root, get_kth(root, i), null);
            splay(root, get_kth(root, sz + 2), root);
            root->ch[1]->ch[0]->Update_Rev();
        }
    }
    return 0;
}
```

6.21 Splay 树

```
typedef int value;

enum { LEFT, RIGHT };
struct node {
    node * child[2], * parent;
    value v, subtree;
    int size;
} pool[MAXN], * pool_next = pool;

node * allocate(const value & v) {
    node * x = pool_next++;
    x->parent = x->child[LEFT] = x->child[RIGHT] = nullptr;
    x->subtree = x->v = v;
    x->size = 1;
    return x;
}

struct tree {
    node * root;
    tree(): root(allocate(0)) {}

    bool child_dir(const node * x, const node * y) { return (x->child[LEFT] == y) ? LEFT :
        RIGHT; }
    bool is_child(const node * x, const node * y) { return x->child[LEFT] == y || x->child[
        RIGHT] == y; }

    void update(node * x) {
        x->size = 1;
```

```
        x->subtree = x->v;
        FOR (d, 2) if (x->child[d] != nullptr) {
            x->size += x->child[d]->size;
            if (d == LEFT) x->subtree = x->child[LEFT]->subtree + x->subtree;
            else x->subtree = x->subtree + x->child[RIGHT]->subtree;
        }
    }

    void set_child(node * x, bool dir, node * y) {
        if ((x->child[dir] = y) != nullptr) y->parent = x;
        update(x);
    }

    node * rotate(node * x, bool dir) {
        node * parent = x->parent, * y = x->child[dir];
        set_child(x, dir, y->child[!dir]);
        set_child(y, !dir, x);
        set_child(parent, child_dir(parent, x), y);
        return y;
    }

    node * splay(node * x) {
        node * old_p = nullptr;
        while (x->parent != nullptr) {
            node * p = x->parent;
            x = rotate(p, child_dir(p, x));
            if (old_p != nullptr && is_child(p, old_p)) rotate(p, child_dir(p, old_p));
            old_p = p;
        }
        return x;
    }

    node * insert(int order, const value & v) { // order is 0-indexed
        bool dir = LEFT;
        node * parent = root, * x = parent->child[LEFT];
        while (x != nullptr) {
            int left_size = (x->child[LEFT] == nullptr) ? 0 : x->child[LEFT]->size;
            parent = x;
            if (order <= left_size) x = x->child[dir = LEFT];
            else {
                order -= left_size + 1;
                x = x->child[dir = RIGHT];
            }
        }
        set_child(parent, dir, x = allocate(v));
        return splay(x);
    }

    node * find(int order) {
        node * x = root->child[LEFT];
        while (true) {
            int left_size = (x->child[LEFT] == nullptr) ? 0 : x->child[LEFT]->size;
            if (order < left_size) x = x->child[LEFT];
            else if (order == left_size) break;
```

```
            else {
                order -= left_size + 1;
                x = x->child[RIGHT];
            }
        }
        return splay(x);
    }

    void erase(const int& order) {
        node * x = find(order);
        if (x->child[LEFT] == nullptr) set_child(root, LEFT, x->child[RIGHT]);
        else if (x->child[RIGHT] == nullptr) set_child(root, LEFT, x->child[LEFT]);
        else {
            node * y = x->child[RIGHT];
            while (y->child[LEFT] != nullptr) y = y->child[LEFT];
            y = splay(y);
            set_child(y, LEFT, x->child[LEFT]);
            set_child(root, LEFT, y);
        }
    }

    value query(int e) { // e is the prefix length desired.
        node * x = root->child[LEFT];
        if (e <= 0) return 0;
        if (e >= x->size) return x->subtree;
        x = find(e - 1);
        if (x->child[LEFT] != nullptr) return x->child[LEFT]->subtree * x->v;
        else return x->v;
    }
};
```

6.22 点分治

```
const int maxn = "Edit";

struct Edge {
    int to, nxt, dis;
} g[maxn];
int head[maxn], cnt, f[maxn], dd[maxn], size[maxn], d[maxn];
int n, k, rt, ans, con, len;
bool vis[maxn];

void add(int u, int v, int dis) {
    g[++ cnt] = (Edge){v, head[u], dis};
    head[u] = cnt;
}

void add_edge(int u, int v, int dis) {
    add(u, v, dis);
    add(v, u, dis);
}

```

```
20 void clr(){
21     for(int i = 1; i <= n; i ++) {
22         vis[i] = f[i] = size[i] = head[i] = dd[i] = 0;
23     }
24     cnt = rt = 0, f[0] = 1e9, con = n, len = ans = 0;
25 }
26 
27 void getrt(int u, int fafa){
28     size[u] = 1;
29     f[u] = 0;
30     for(int i = head[u]; i; i = g[i].nxt){
31         int v = g[i].to; if(v == fafa || vis[v]) continue;
32         getrt(v, u);
33         size[u] += size[v];
34         f[u] = std::max(f[u], size[v]);
35     }
36     f[u] = std::max(f[u], con - size[u]);
37     if(f[u] < f[rt]) {
38         rt = u;
39     }
40 }
41 
42 void getdis(int u, int fafa){
43     size[u] = 1;
44     dd[++ len] = d[u];
45     for(int i = head[u]; i; i = g[i].nxt){
46         int v = g[i].to; if(v == fafa || vis[v]) continue;
47         d[v] = d[u] + g[i].dis; getdis(v, u);
48         size[u] += size[v];
49     }
50 }
51 
52 int cal(int u, int w){
53     len = 0; d[u] = w; getdis(u, 0);
54     std::sort(dd + 1, dd + len + 1);
55     int l = 1, r = len, sum = 0;
56     while(l < r){
57         if(dd[l] + dd[r] <= k) sum += r - l, l ++;
58         else r --;
59     }
60     return sum;
61 }
62 
63 void solve(int u){
64     vis[u] = 1; ans += cal(u, 0);
65     for(int i = head[u]; i; i = g[i].nxt){
66         int v = g[i].to; if(vis[v]) continue;
67         ans -= cal(v, g[i].dis);
68         rt = 0; con = size[v];
69         getrt(v, 0);
70         solve(rt);
71     }
72 }
```

6.23 树上启发式合并

```cpp
// 树上启发式合并: dsu on tree
int n, x, y, Son, Max;
int sz[maxn], son[maxn];
ll sum, ans[maxn];
vector<int> v[maxn];

void getson(int u, int fa) {
    sz[u] = 1;
    for (int i = 0; i < v[u].size(); i++) {
        int to = v[u][i];
        if (to != fa) {
            getson(to, u);
            sz[u] += sz[to];
            if (sz[to] > sz[son[u]])
                son[u] = to;
        }
    }
}

void add(int u, int fa, int val) {
    // 更新节点数据
    // cnt[attr[u]] += val;
    for (int i = 0; i < v[u].size(); i++) {
        int to = v[u][i];
        if (to != fa && to != Son)
            add(to, u, val);
    }
}

void dfs(int u, int fa, int k) {
    for (int i = 0; i < v[u].size(); i++) {
        int to = v[u][i];
        if (to != fa && to != son[u])
            dfs(to, u, 0);
    }
    if (son[u]) dfs(son[u], u, 1), Son = son[u];
    add(u, fa, 1); Son = 0;
    // 此处统计u节点处答案
    // ans[u] = sum;
    if (!k) add(u, fa, -1), Max = sum = 0;
}

// getson(1, 0);
// dfs(1, 0, 0);
```

6.24 0-1trie 区间异或最大值

```cpp

// written by calabash_boy
// 01Trie求区间异或和的最大值
```

```
4
5  #include <cstdio>
6  #include <cstring>
7  #include <algorithm>
8  using namespace std;
9  const int MAX = 1e6+100;
10 int bas[35];
11 const int INF = 2147483645;
12
13 struct Trie {
14     int nxt[MAX<<2][2];  int l[MAX<<2];
15     int cnt;  int ansl,ansr,ansv;
16     void init() {
17         cnt =0;
18         memset(nxt[0],0,sizeof (nxt[0]));
19         memset(l,0x3f3f3f3f,sizeof(l));
20         ansv = 0;
21     }
22     int create() {
23         cnt++;
24         memset(nxt[cnt],0,sizeof (nxt[cnt]));
25         return cnt;
26     }
27     void insert(int id,int x) {
28         int y = 0;
29         for (int i=30;i>=0;i--) {
30             int t = x&bas[i];
31             t>>=i;
32             if (!nxt[y][t]) {
33                 nxt[y][t] = create();
34             }
35             y = nxt[y][t];
36         }
37         l[y] = min(l[y],id);
38     }
39     void query(int id,int x) {
40         int y=0;  int res =0;
41         for (int i=30;i>=0;i--) {
42             int t = x&bas[i];
43             t>>=i;
44             if (nxt[y][!t]) {
45                 y =nxt[y][!t];
46                 res+=bas[i];
47             } else{
48                 y = nxt[y][t];
49             }
50         }
51         if (res==ansv) {
52             if (l[y]<ansl) {
53                 ansl = l[y];  ansr = id;
54             }
55         } else if (res>ansv) {
56             ansv = res;
57             ansl = l[y];
```

```
            ansr = id;
        }
    }
    void print(int id) {
        printf("Case␣#%d:\n%d␣%d\n",id,ansl+1,ansr);
    }
}trie;

void init() {
    bas[0] = 1;
    for (int i=1;i<=30;i++) {
        bas[i] = bas[i-1]<<1;
    }
}
int main() {
    init();
    int n, Cas;
    scanf("%d",&Cas);
    for (int i=1;i<=Cas;i++) {
        trie.init();  trie.insert(0,0);
        scanf("%d",&n);
        int sum=0;
        for (int j=1;j<=n;j++) {
            int ai;
            scanf("%d",&ai);  sum^=ai;
            trie.query(j,sum);  trie.insert(j,sum);
        }
        trie.print(i);
    }
    return 0;
}
```

6.25 0-1trie 子树异或最大值

```
// 可持久化01Trie+DFS序 子树上的点抑或最大值:
// written by calabash_boy

#include <iostream>
#include <cstdio>
using namespace std;
const int MAX = 1e5+100;
int bas[35];  int nxt[MAX<<5][2];
int root[MAX];  int sum[MAX<<5];
int n,q;  vector<int>E[MAX];
int st[MAX],en[MAX],rk[MAX];
int a[MAX];  int cnt;  int tot;
void sheet(){
    bas[0]=1;
    for (int i=1;i<=30;i++){
        bas[i] = bas[i-1]<<1;
    }
}
```

```
void init(){
    for (int i=0;i<=n;i++){  E[i].clear(); }
    cnt =tot=0;
    memset(nxt[0],0,sizeof nxt[0]);
}
void input(){
    for (int i=1;i<=n;i++){  scanf("%d",a+i);  }
    for (int u=2;u<=n;u++){
        int v;  scanf("%d",&v);
        E[u].push_back(v);  E[v].push_back(u);
    }
}
void dfs(int node ,int  father ){
    st[node] = ++tot;  rk[tot] = node;
    for (int des:E[node]){
        if(des==father){  continue; }
        dfs(des,node);
    }
    en[node] = tot;
}
int create(){
    cnt++;
    memset(nxt[cnt],0,sizeof nxt[cnt]);
    return cnt;
}
int insert(int rt,int val){
    int y = ++cnt;  int x = rt;  int res = y;
    for (int i=30;i>=0;i--){
        sum[y] = sum[x]+1;
        nxt[y][0] = nxt[x][0];  nxt[y][1] = nxt[x][1];
        int t = val&bas[i];
        t>>=i;
        nxt[y][t] = create();
        y = nxt[y][t];  x = nxt[x][t];
    }
    sum[y] = sum[x]+1;
    return res;
}
int query(int l,int r,int val){
    int res =0; int x = l; int y = r;
    for (int i=30;i>=0;i--){
        int t = val&bas[i];
        t>>=i;
        if (sum[nxt[y][!t]]-sum[nxt[x][!t]]){
            y = nxt[y][!t];  x = nxt[x][!t];
            res+=bas[i];
        }else{
            y = nxt[y][t];  x = nxt[x][t];
        }
    }
    return res;
}
void solve(){
    dfs(1,0);
```

```
    for (int i=1;i<=n;i++){
        root[i] = insert(root[i-1],a[rk[i]]);
    }
    while (q--){
        int nod,x;
        scanf("%d%d",&nod,&x);
        printf("%d\n",query(root[st[nod]-1],root[en[nod]],x));
    }
}
int main(){
    sheet();
    while (scanf("%d%d",&n,&q)!=EOF){
        init();
        input();
        solve();
    }
    return 0;
}
```

6.26 莫队算法

```
//Author:marszed
/*
*离线区间处理问题。
*从区间[l,r]得到区间[l+1,r+1] [l-1,r-1]信息的转移复杂度为O(1)。
*siz为块大小。
*cnt为位于第几个块。
*modify()函数为转移函数。
*/

#include <iostream>
#include <algorithm>
#include <cmath>

const int maxn = 2e5 + 10;

int n, siz, q;
int a[maxn];

struct Node {
    int id, l, r, val, cnt;

    int operator< (const Node& b) {
        return cnt == b.cnt ? r < b.r : cnt < b.cnt;
    }
} nod[maxn];

void modify(int i, int flag) {

}

void mo() {
```

```
    std::cin >> n >> q;
    siz = sqrt(n);
    for (int i = 1; i <= n; i++) {
        std::cin >> a[i];
    }
    for (int i = 1; i <= q; i++) {
        std::cin >> nod[i].l >> nod[i].r;
        nod[i].id = i;
        nod[i].cnt = nod[i].l / siz;
    }
    std::sort(nod + 1, nod + q + 1);
    int l = 0, r = 0;
    for (int i = 1; i <= q; i++) {
        while (l < nod[i].l - 1)    modify(++l, 1);
        while (l >= nod[i].l)       modify(l--, 1);
        while (r < nod[i].r)        modify(++r, 1);
        while (r > nod[i].r)        modify(r--, 1);
        ans[nod[i].id] = Ans;
    }
}

int main() {}
```

6.27 最近公共祖先 (在线)

```
// 时间复杂度 O(nlogn+q)
// By CSL

const int maxn = "Edit";
std::vector<int> G[maxn], sp;
int dep[maxn], dfn[maxn];

std::pair<int, int> dp[21][maxn << 1];

void init(int n) {
    for (int i = 0; i < n; i++) G[i].clear();
    sp.clear();
}

void dfs(int u, int fa) {
    dep[u] = dep[fa] + 1;
    dfn[u] = sp.size();
    sp.push_back(u);
    for (auto& v : G[u]) {
        if (v == fa) continue;
        dfs(v, u);
        sp.push_back(u);
    }
}

void initrmq() {
    int n = sp.size();
```

```
    for (int i = 0; i < n; i++) dp[0][i] = {dfn[sp[i]], sp[i]};
    for (int i = 1; (1 << i) <= n; i++)
        for (int j = 0; j + (1 << i) - 1 < n; j++)
            dp[i][j] = std::min(dp[i - 1][j], dp[i - 1][j + (1 << (i - 1))]);
}

int lca(int u, int v) {
    int l = dfn[u], r = dfn[v];
    if (l > r) std::swap(l, r);
    int k = 31 - __builtin_clz(r - l + 1);
    return std::min(dp[k][l], dp[k][r - (1 << k) + 1]).second;
}
```

6.28 最近公共祖先 (离线)

```
// 时间复杂度 O(n+q)
// By CSL

#include <iostream>
#include <algorithm>
#include <vector>

const int maxn = "Edit";
int par[maxn];                                  //并查集
int ans[maxn];                                  //存储答案
std::vector<int> G[maxn];                       //邻接表
std::vector<std::pair<int, int>> query[maxn];   //存储查询信息
bool vis[maxn];                                 //是否被遍历

inline void init(int n) {
    for (int i = 1; i <= n; i++) {
        G[i].clear(), query[i].clear();
        par[i] = i, vis[i] = 0;
    }
}

int find(int u) {
    return par[u] == u ? par[u] : par[u] = find(par[u]);
}

void unite(int u, int v) {
    par[find(v)] = find(u);
}

inline void add_edge(int u, int v) {
    G[u].push_back(v);
}

inline void add_query(int id, int u, int v) {
    query[u].push_back(std::make_pair(v, id));
    query[v].push_back(std::make_pair(u, id));
}
```

```
38
39 void tarjan(int u) {
40     vis[u] = 1;
41     for (auto& v : G[u]) {
42         if (vis[v]) continue;
43         tarjan(v);
44         unite(u, v);
45     }
46     for (auto& q : query[u]) {
47         int &v = q.first, &id = q.second;
48         if (!vis[v]) continue;
49         ans[id] = find(v);
50     }
51 }
```

6.29 最近公共祖先

```
1  // LCA ST算法
2  int n, top, root;
3  int a[maxn << 1], d[maxn], st[maxn];
4  int f[maxn << 1][18], loc[maxn << 1][18];
5  vector<int> v[maxn];
6
7  int log2(int x) {
8      int k = 0;
9      while (x > 1) {
10         x /= 2;
11         k++;
12     }
13     return k;
14 }
15
16 void dfs(int u, int dep) {
17     d[u] = dep;
18     a[++top] = u;
19     for (int i = 0; i<=v[u].size(); i++) {
20         int to = v[u][i];
21         dfs(to, dep + 1);
22         a[++top] = u;
23     }
24 }
25
26 void init() {
27     int s = log2(top);
28     for (int i = 1; i <= top; i++) {
29         f[i][0] = d[a[i]];
30         loc[i][0] = a[i];
31     }
32     for (int j = 1; j <= s; j++) {
33         int k = top - (1 << j) + 1;
34         for (int i = 1; i <= k; i++) {
35             int x = i + (1 << (j - 1));
```

```
            if (f[i][j - 1] <= f[x][j - 1]) {
                f[i][j] = f[i][j - 1];
                loc[i][j] = loc[i][j - 1];
            }
            else {
                f[i][j] = f[x][j - 1];
                loc[i][j] = loc[x][j - 1];
            }
        }
    }
}

int query(int x, int y) {
    x = st[x], y = st[y];
    if (x > y) swap(x, y);
    int i = log2(y - x);
    int k = y - (1 << i) + 1;
    return f[x][i] < f[k][i] ? loc[x][i] : loc[k][i];
}

//================================================================================

// LCA Tarjan算法
int n, root, cnt;
int pre[maxn], ans[maxn];
vector<int> v[maxn], s[maxn], num[maxn];

int find(int x) { return pre[x] == x ? x : pre[x] = find(pre[x]); }

void dfs(int u) {
    pre[u] = u;
    for (int i = 0; i < v[u].size(); i++) {
        int to = v[u][i];
        dfs(to);
        pre[find(pre[to])] = find(pre[u]);
    }
    for (int i = 0; i < s[u].size(); i++) {
        int to = s[u][i];
        if (pre[to] != to)
            ans[num[u][i]] = find(pre[to]);
    }
}

/*
for (int i = 1; i <= q; i++) {
    scanf("%d%d", &x, &y);
    if (x == y) ans[i] = x;
    s[x].push_back(y);
    s[y].push_back(x);
    num[x].push_back(i);
    num[y].push_back(i);
}
dfs(root);
*/
```

```
90
91  //================================================================================
92
93  // LCA 倍增算法
94  int n, ma, root;
95  int d[maxn], f[maxn][20];
96  vector<int> v[maxn];
97  inline void dfs(int u, int dep, int fa) {
98      d[u] = dep;
99      f[u][0] = fa;
100     ma = max(ma, dep);
101     for (int i = 0; i < v[u].size(); i++)
102         if (v[u][i] != fa) dfs(v[u][i], dep + 1, u);
103 }
104 inline int log2(int x) {
105     int k = 0;
106     while (x > 1) {
107         x >>= 1;
108         k++;
109     }
110     return k;
111 }
112 inline void init() {
113     dfs(root, 0, 0);
114     int s = log2(ma);
115     for (int j = 1; j <= s; j++)
116         for (int i = 1; i <= n; i++)
117             f[i][j] = f[f[i][j - 1]][j - 1];
118 }
119 // 求x与y的LCA
120 inline int query(int x, int y) {
121     if (d[x] < d[y]) swap(x, y);
122     int s = log2(d[x] - d[y]);
123     while (d[x] > d[y]) {
124         if (d[x] - (1 << s) >= d[y])
125             x = f[x][s];
126         s--;
127     }
128     s = log2(d[x]);
129     while (s > -1) {
130         if (f[x][s] != f[y][s]) {
131             x = f[x][s];
132             y = f[y][s];
133         }
134         s--;
135     }
136     return x == y ? x : f[x][0];
137 }
138 // 判断a与p是否在同一树边上（p在a上方）
139 inline bool check(int a, int p) {
140     if (d[a] < d[p]) return false;
141     if (d[a] == d[p]) return a == p;
142     int s = log2(d[a] - d[p]);
143     while (d[a] > d[p]) {
```

```
144         if (d[a] - (1 << s) >= d[p])
145             a = f[a][s];
146         s--;
147     }
148     return a == p;
149 }
150 // 求一条树边上x到y的距离
151 inline int getlen(int x, int y) {
152     int ret = 0;
153     if (d[x] < d[y]) swap(x, y);
154     int s = log2(d[x] - d[y]);
155     while (d[x] > d[y]) {
156         if (d[x] - (1 << s) >= d[y]) {
157             ret += (1 << s);
158             x = f[x][s];
159         }
160         s--;
161     }
162     return ret;
163 }
```

6.30 树链剖分

```
1  // 树链剖分 点权
2  /**
3  * top[v] 表示v所在的重链的顶端节点
4  * fa[v] 表示v的父节点
5  * deep[v] 表示v的深度(根的深度为1)
6  * snum[v] 表示以v为根的子树的节点数
7  * p[v] 表示v所在(线段树中)的位置
8  * fp[v] 与p[v]相反, 表示对应位置的节点
9  * son[v] 表示v的重儿子
10 * Edge 存树边
11 **/
12
13 struct Edge {
14     int to, next;
15 }edge[maxn << 1];
16
17 int pos, n, m, tot; // n 为节点数
18 int head[maxn], top[maxn], fa[maxn], deep[maxn], num[maxn], p[maxn], fp[maxn], son[maxn];
19
20 void init() {
21     tot = 0;
22     pos = 1;
23     memset(head, -1, sizeof(head));
24     memset(son, -1, sizeof(son));
25     for (int i = 0; i <= n; i++)
26         v[i].clear();
27 }
28
29 void addedge(int u, int v) {
```

```
30      edge[tot].to = v;
31      edge[tot].next = head[u];
32      head[u] = tot++;
33  }
34
35  void dfs1(int u, int pre, int d) {
36      deep[u] = d;
37      fa[u] = pre;
38      num[u] = 1;
39      for (int i = head[u]; i != -1; i = edge[i].next) {
40          int to = edge[i].to;
41          if (to != pre) {
42              dfs1(to, u, d + 1);
43              num[u] += num[to];
44              if (son[u] == -1 || num[to] > num[son[u]])
45                  son[u] = to;
46          }
47      }
48  }
49
50  void dfs2(int u, int sp) {
51      top[u] = sp;
52      p[u] = pos++;
53      fp[p[u]] = u;
54      if (son[u] == -1) return;
55      dfs2(son[u], sp);
56      for (int i = head[u]; i != -1; i = edge[i].next) {
57          int to = edge[i].to;
58          if (to != son[u] && to != fa[u])
59              dfs2(to, to);
60      }
61  }
62  /*
63  // 使用范例
64  int getsum(int a, int b) {
65      int f1 = top[a], f2 = top[b];
66      int ret = 0;
67      while (f1 != f2) {
68          if (deep[f1] < deep[f2]) {
69              swap(f1, f2);
70              swap(a, b);
71          }
72          ret += query(p[f1], p[a], 1, n - 1, 1);
73          a = fa[f1]; f1 = top[a];
74      }
75      if (a == b) return ret;
76      if (deep[a] > deep[b]) swap(a, b);
77      return ret + query(p[son[a]], p[b], 1, n - 1, 1);
78  }
79  */
```

第七章　字符串

7.1　KMP

```cpp
//Author:CookiC
//返回下标最大的匹配串
#include <cstring>

void getFail(char *P, int *f) {
    int i, j;
    f[0] = 0;
    f[1] = 0;
    for(i=1; P[i]; ++i) {
        j = f[i];
        while(j && P[i]!=P[j]) {
            j = f[j];
        }
        f[i+1] = P[i]==P[j]? j+1: 0;
    }
}

int kmp(char *T, char *P) {
    int ans = -1;
    int n = strlen(T), m = strlen(P);
    int f[maxn];
    getFail(P, f);
    int j = 0;
    for(int i=0; i<n; ++i){
        while(j && P[j]!=T[i])
        j = f[j];
        if(P[j]==T[i]) {
            ++j;
        }
        if(j==m) {
            j = f[j];
            ans = i-m+1;
        }
    }
    return ans;
}
```

7.2　扩展 KMP

```
const int maxn = "Edit";
int ans, nexr[maxn], ex[maxn];
void getnexr(char s[]) {
    int i = 0, j, po, len = strlen(s);
    nexr[0] = len;
    while (s[i] == s[i+1] && i + 1 < len) i++;
    nexr[1] = i;
    po = 1;
    for (i = 2; i < len; i++) {
        if (nexr[i-po] + i < nexr[po] + po) {
            nexr[i] = nexr[i - po];
        } else {
            j = nexr[po] + po - i;
            if (j < 0) j = 0;
            while (i + j < len && s[j] == s[i+j]) j++;
            nexr[i] = j;
            po = i;
         }
    }
}

void exkmp(char s1[], char s2[]) {
    int i = 0, j, po = 0, len = strlen(s1), l2 = strlen(s2);
    while (s1[i] == s2[i] && i < l2 && i < len) i++;
    ex[0] = i;
    for (i = 1; i < len; i++) {
        if (nexr[i - po] + i < ex[po] + po) {
            ex[i] = nexr[i-po];
        } else {
            j = ex[po] + po - i;
            if (j < 0) j = 0;
            while (i + 1 < len && s1[j+i] == s2[j]) j++;
            ex[i] = j;
            po = i;
         }
    }
}
```

7.3 TRIE

```
int tree[maxn][26];
int sum[maxn];
int tot;
void Insert(char *str) {
    int len = strlen(str);
    int root = 0;
    for (int i = 0; i < len; i++) {
        int id = str[i] - 'a';
        if (!tree[root][id]) tree[root][id] = ++tot;
        sum[tree[root][id]]++;
        root = tree[root][id];
    }
```

```
}

int Find(char *str) {
    int len = strlen(str);
    int root = 0;
    for (int i = 0; i < len; i++) {
        int id = str[i] - 'a';
        if (!tree[root][id]) return 0;
        root = tree[root][id];
    }
    return sum[root];
}
```

7.4 AC 自动机

```
#include <cstdio>
#include <iostream>
#include <algorithm>
#include <cstring>
#include <queue>
using namespace std;

struct Trie {
    int next[500010][26], fail[500010], end[500010];
    int root, L;
    int newnode() {
        for (int i = 0; i < 26; i++)
            next[L][i] = -1;
        end[L++] = 0;
        return L - 1;
    }
    void init() {
        L = 0;
        root = newnode();
    }
    void insert(char buf[]) {
        int len = strlen(buf);
        int now = root;
        for (int i = 0; i < len; i++) {
            if (next[now][buf[i] - 'a'] == -1)
                next[now][buf[i] - 'a'] = newnode();
            now = next[now][buf[i] - 'a'];
        }
        end[now]++;
    }
    void build() {
        queue<int>Q;
        fail[root] = root;
        for (int i = 0; i < 26; i++)
            if (next[root][i] == -1)
                next[root][i] = root;
            else {
```

```
                fail[next[root][i]] = root;
                Q.push(next[root][i]);
            }
            while (!Q.empty()) {
                int now = Q.front();
                Q.pop();
                for (int i = 0; i < 26; i++)
                    if (next[now][i] == -1)
                        next[now][i] = next[fail[now]][i];
                    else
                    {
                        fail[next[now][i]] = next[fail[now]][i];
                        Q.push(next[now][i]);
                    }
            }
    }
    // 查询buf字符串包含的模板串
    int query(char buf[]) {
        int len = strlen(buf);
        int now = root;
        int res = 0;
        for (int i = 0; i < len; i++) {
            now = next[now][buf[i] - 'a'];
            int temp = now;
            while (temp != root) {
                res += end[temp];
                end[temp] = 0;
                temp = fail[temp];
            }
        }
        return res;
    }
};
char buf[1000010];
```

7.5 后缀数组 (倍增)

```
//author: Menci
#include <algorithm>
#include <string>
#include <iostream>

const int maxn = 1000;

char s[maxn];
int n, ht[maxn], rk[maxn], sa[maxn];

inline void suffixArray() {
    static int set[maxn + 1], a[maxn + 1];
    std::copy(s, s + n, set + 1);
    std::sort(set + 1, set + n + 1);
    int *end = std::unique(set + 1, set + n + 1);
```

```
16      for (int i = 1; i <= n; i++) a[i] = std::lower_bound(set + 1, end, s[i]) - set;
17
18      static int fir[maxn + 1], sec[maxn + 1], tmp[maxn + 1], buc[maxn + 1];
19      for (int i = 1; i <= n; i++) buc[a[i]]++;
20      for (int i = 1; i <= n; i++) buc[i] += buc[i - 1];
21      for (int i = 1; i <= n; i++) rk[i] = buc[a[i] - 1] + 1;
22
23      for (int t = 1; t <= n; t *= 2) {
24          for (int i = 1; i <= n; i++) fir[i] = rk[i];
25          for (int i = 1; i <= n; i++) sec[i] = i + t > n ? 0 : rk[i + t];
26
27          std::fill(buc, buc + n + 1, 0);
28          for (int i = 1; i <= n; i++) buc[sec[i]]++;
29          for (int i = 1; i <= n; i++) buc[i] += buc[i - 1];
30          for (int i = 1; i <= n; i++) tmp[n - --buc[sec[i]]] = i;
31
32          std::fill(buc, buc + n + 1, 0);
33          for (int i = 1; i <= n; i++) buc[fir[i]]++;
34          for (int i = 1; i <= n; i++) buc[i] += buc[i - 1];
35          for (int j = 1, i; j <= n; j++) i = tmp[j], sa[buc[fir[i]]--] = i;
36
37          bool unique = true;
38          for (int j = 1, i, last = 0; j <= n; j++) {
39              i = sa[j];
40              if (!last) rk[i] = 1;
41              else if (fir[i] == fir[last] && sec[i] == sec[last]) rk[i] = rk[last], unique =
                    false;
42              else rk[i] = rk[last] + 1;
43
44              last = i;
45          }
46
47          if (unique) break;
48      }
49
50      for (int i = 1, k = 0; i <= n; i++) {
51          if (rk[i] == 1) k = 0;
52          else {
53              if (k > 0) k--;
54              int j = sa[rk[i] - 1];
55              while (i + k <= n && j + k <= n && a[i + k] == a[j + k]) k++;
56          }
57          ht[rk[i]] = k;
58      }
59  }
60
61  int main() {
62      std::cin >> n >> s;
63      suffixArray();
64      for (int i = 1; i <= n; i++) {
65          std::cout << sa[i] << "␣";
66      }
67  }
```

7.6 后缀数组 (sais)

```cpp
namespace SA {
    int sa[N], rk[N], ht[N], s[N<<1], t[N<<1], p[N], cnt[N], cur[N];
    #define pushS(x) sa[cur[s[x]]--] = x
    #define pushL(x) sa[cur[s[x]]++] = x
    #define inducedSort(v) std::fill_n(sa, n, -1); std::fill_n(cnt, m, 0);        \
        for (int i = 0; i < n; i++) cnt[s[i]]++;                                  \
        for (int i = 1; i < m; i++) cnt[i] += cnt[i-1];                           \
        for (int i = 0; i < m; i++) cur[i] = cnt[i]-1;                            \
        for (int i = n1-1; ~i; i--) pushS(v[i]);                                  \
        for (int i = 1; i < m; i++) cur[i] = cnt[i-1];                            \
        for (int i = 0; i < n; i++) if (sa[i] > 0 &&  t[sa[i]-1]) pushL(sa[i]-1); \
        for (int i = 0; i < m; i++) cur[i] = cnt[i]-1;                            \
        for (int i = n-1;  ~i; i--) if (sa[i] > 0 && !t[sa[i]-1]) pushS(sa[i]-1)
    void sais(int n, int m, int *s, int *t, int *p) {
        int n1 = t[n-1] = 0, ch = rk[0] = -1, *s1 = s+n;
        for (int i = n-2; ~i; i--) t[i] = s[i] == s[i+1] ? t[i+1] : s[i] > s[i+1];
        for (int i = 1; i < n; i++) rk[i] = t[i-1] && !t[i] ? (p[n1] = i, n1++) : -1;
        inducedSort(p);
        for (int i = 0, x, y; i < n; i++) if (~(x = rk[sa[i]])) {
            if (ch < 1 || p[x+1] - p[x] != p[y+1] - p[y]) ch++;
            else for (int j = p[x], k = p[y]; j <= p[x+1]; j++, k++)
                if ((s[j]<<1|t[j]) != (s[k]<<1|t[k])) {ch++; break;}
            s1[y = x] = ch;
        }
        if (ch+1 < n1) sais(n1, ch+1, s1, t+n, p+n1);
        else for (int i = 0; i < n1; i++) sa[s1[i]] = i;
        for (int i = 0; i < n1; i++) s1[i] = p[sa[i]];
        inducedSort(s1);
    }
    template<typename T>
    int mapCharToInt(int n, const T *str) {
        int m = *max_element(str, str+n);
        std::fill_n(rk, m+1, 0);
        for (int i = 0; i < n; i++) rk[str[i]] = 1;
        for (int i = 0; i < m; i++) rk[i+1] += rk[i];
        for (int i = 0; i < n; i++) s[i] = rk[str[i]] - 1;
        return rk[m];
    }
    // Ensure that str[n] is the unique lexicographically smallest character in str.
    template<typename T>
    void suffixArray(int n, const T *str) {
        int m = mapCharToInt(++n, str);
        sais(n, m, s, t, p);
        for (int i = 0; i < n; i++) rk[sa[i]] = i;
        for (int i = 0, h = ht[0] = 0; i < n-1; i++) {
            int j = sa[rk[i]-1];
            while (i+h < n && j+h < n && s[i+h] == s[j+h]) h++;
            if (ht[rk[i]] = h) h--;
        }
    }
};
```

7.7 后缀自动机

```cpp
//Author:CookiC
#include<cstring>
#define MAXN 10000

struct State{
    State *f,*c[26];
    int len;
};

State *root,*last,*cur;
State StatePool[MAXN];

State* NewState(int len){
    cur->len=len;
    cur->f=0;
    memset(cur->c,0,sizeof(cur->c));
    return cur++;
}

void Init(){
    cur=StatePool;
    last=StatePool;
    root=NewState(0);
}

void Extend(int w){
    State *p = last;
    State *np = NewState(p->len+1);
    while(p&&!p->c[w]) {
        p->c[w] = np;
        p = p->f;
    }
    if(!p) {
        np->f=root;
    } else {
        State *q=p->c[w];
        if(p->len+1==q->len) {
            np->f=q;
        } else {
            State *nq = NewState(p->len+1);
            memcpy(nq->c, q->c, sizeof(q->c));
            nq->f = q->f;
            q->f = nq;
            np->f = nq;
            while(p&&p->c[w]==q) {
                p->c[w]=nq;
                p=p->f;
            }
        }
```

```
    }
    last=np;
}

bool Find(char *s,int len) {
    int i;
    State *p=root;
    for(i=0;i<len;++i) {
        if(p->c[s[i]-'a']) {
            p=p->c[s[i]-'a'];
        } else {
            return false;
        }
    }
    return true;
}
```

7.8 最长回文子串

```
const int maxn=2000005;
int f[maxn];
std::string a, s;
int manacher() {
    int n=0, res=0, maxr=0, pos=0;
    for (int i=0; a[i]; i++) {
        s[++n] = '#', s[++n] = a[i];
        s[++n] = '#';
    }
    for (int i=1; i<=n; i++) {
        f[i] = (i<maxr? std::min(f[pos*2-i], maxr-i+1): 1);
        while (i-f[i]>0 && i+f[i]<=n && s[i-f[i]]==s[i+f[i]]) {
            f[i]++;
        }
        if (i+f[i]-1 > maxr) {
            maxr=i+f[i]-1;
            pos=i;
        }
        res = std::max(res,f[i]-1);
    }
    return res;
}
```

7.9 回文树

```
#include <iostream>
#include <vector>
#define rep(i, a, b) for (int i=a; i<=b; i++)
#define drep(i, a, b) for (int i=a; i>=b; i--)

```

```cpp
const int inf = 1e9;
typedef long long ll;
const int maxn=300010;

char s[maxn];
int n;
struct Ptree {
    int last;
    struct Node {
        int cnt, lenn, fail, son[27];
        Node(int lenn, int fail):lenn(lenn), fail(fail), cnt(0){
            memset(son, 0, sizeof(son));
        };
    };
    std::vector<Node> st;
    inline int newnode(int lenn, int fail=0) {
        st.emplace_back(lenn, fail);
        return st.size()-1;
    }
    inline int getfail(int x, int n) {
        while (s[n-st[x].lenn-1] != s[n]) x=st[x].fail;
        return x;
    }
    inline void extend(int c, int i) {
        int cur=getfail(last, i);
        if (!st[cur].son[c]) {
            int nw=newnode(st[cur].lenn+2, st[getfail(st[cur].fail, i)].son[c]);
            st[cur].son[c]=nw;
        }
        st[ last=st[cur].son[c] ].cnt++;
    }
    void init() {
        scanf("%s", s+1);
        n=strlen(s+1);
        s[0]=0;
        newnode(0, 1), newnode(-1);
        last=0;
        rep(i, 1, n) extend(s[i]-'a', i);
    }
    ll count() {
        drep(i, st.size()-1, 0) st[st[i].fail].cnt += st[i].cnt;
        ll ans = 0;
        rep(i, 2, st.size()-1) ans = std::max(ans, 1LL*st[i].lenn*st[i].cnt);
        return ans;
    }
};
```

7.10 字符串哈希算法

```cpp
// RS Hash Function
unsigned int RSHash(char *str) {
    unsigned int b = 378551;
```

```
    unsigned int a = 63689;
    unsigned int hash = 0;
    while (*str) {
        hash = hash * a + (*str++);
        a *= b;
    }
    return (hash & 0x7FFFFFFF);
}

// JS Hash Function
unsigned int JSHash(char *str) {
    unsigned int hash = 1315423911;
    while (*str) {
        hash ^= ((hash << 5) + (*str++) + (hash >> 2));
    }
    return (hash & 0x7FFFFFFF);
}

// P. J. Weinberger Hash Function
unsigned int PJWHash(char *str) {
    unsigned int BitsInUnignedInt = (unsigned int)(sizeof(unsigned int) * 8);
    unsigned int ThreeQuarters    = (unsigned int)((BitsInUnignedInt  * 3) / 4);
    unsigned int OneEighth        = (unsigned int)(BitsInUnignedInt / 8);
    unsigned int HighBits         = (unsigned int)(0xFFFFFFFF) << (BitsInUnignedInt -
        OneEighth);
    unsigned int hash             = 0;
    unsigned int test             = 0;
    while (*str) {
        hash = (hash << OneEighth) + (*str++);
        if ((test = hash & HighBits) != 0) {
            hash = ((hash ^ (test >> ThreeQuarters)) & (~HighBits));
        }
    }
    return (hash & 0x7FFFFFFF);
}

// ELF Hash Function
unsigned int ELFHash(char *str) {
    unsigned int hash = 0;
    unsigned int x    = 0;
    while (*str) {
        hash = (hash << 4) + (*str++);
        if ((x = hash & 0xF0000000L) != 0) {
            hash ^= (x >> 24);
            hash &= ~x;
        }
    }
    return (hash & 0x7FFFFFFF);
}

// BKDR Hash Function
unsigned int BKDRHash(char *str) {
    unsigned int seed = 131; // 31 131 1313 13131 131313 etc..
    unsigned int hash = 0;
```

```
    while (*str) {
        hash = hash * seed + (*str++);
    }
    return (hash & 0x7FFFFFFF);
}

// SDBM Hash Function
unsigned int SDBMHash(char *str) {
    unsigned int hash = 0;
    while (*str) {
        hash = (*str++) + (hash << 6) + (hash << 16) - hash;
    }
    return (hash & 0x7FFFFFFF);
}

// DJB Hash Function
unsigned int DJBHash(char *str) {
    unsigned int hash = 5381;
    while (*str) {
        hash += (hash << 5) + (*str++);
    }
    return (hash & 0x7FFFFFFF);
}

// AP Hash Function
unsigned int APHash(char *str) {
    unsigned int hash = 0;
    int i;
    for (i=0; *str; i++) {
        if ((i & 1) == 0) {
            hash ^= ((hash << 7) ^ (*str++) ^ (hash >> 3));
        } else {
            hash ^= (~((hash << 11) ^ (*str++) ^ (hash >> 5)));
        }
    }
    return (hash & 0x7FFFFFFF);
}
```

7.11 字符串哈希表

```
typedef unsigned long long ull;
const ull base = 163;
char s[maxn];
ull hash[maxn];

void init() {
    p[0] = 1;
    hash[0] = 0;
    int n = strlen(s + 1);
   for(int i = 1; i <=100000; i ++)p[i] =p[i-1] * base;
   for(int i = 1; i <= n; i ++)hash[i] = hash[i - 1] * base + (s[i] - 'a');
}
```

```
13
14 ull get(int l, int r, ull g[]) {
15     return g[r] - g[l - 1] * p[r - l + 1];
16 }
17
18 struct HASHMAP {
19     int size;
20     int head[maxh], next[maxn], f[maxn];    // maxh 为hash链表最大长度
21     ull state[maxn];
22     void init() {
23         size = 0;
24         memset(head, -1, sizeof(head));
25     }
26     int insert(ull val, int id) {
27         int h = val % maxh;
28         for (int i = head[h]; i != -1; i = next[i])
29             if (val == state[i]) return f[i];
30         f[size] = id;
31         state[size] = val;
32         next[size] = head[h];
33         head[h] = size;
34         return f[size++];
35     }
36 };
```

第八章　几何

8.1　平面几何公式

```
三角形:
    1. 半周长 P=(a+b+c)/2
    2. 面积 S=aHa/2=absin(C)/2=sqrt(P(P-a)(P-b)(P-c))
    3. 中线 Ma=sqrt(2(b^2+c^2)-a^2)/2=sqrt(b^2+c^2+2bccos(A))/2
    4. 角平分线 Ta=sqrt(bc((b+c)^2-a^2))/(b+c)=2bccos(A/2)/(b+c)
    5. 高线 Ha=bsin(C)=csin(B)=sqrt(b^2-((a^2+b^2-c^2)/(2a))^2)
    6. 内切圆半径 r=S/P=asin(B/2)sin(C/2)/sin((B+C)/2)
                                =4Rsin(A/2)sin(B/2)sin(C/2)=sqrt((P-a)(P-b)(P-c)/P)
                                =Ptan(A/2)tan(B/2)tan(C/2)
    7. 外接圆半径 R=abc/(4S)=a/(2sin(A))=b/(2sin(B))=c/(2sin(C))


    四边形:
    D1,D2为对角线,M对角线中点连线,A为对角线夹角
    1. a^2+b^2+c^2+d^2=D1^2+D2^2+4M^2
    2. S=D1D2sin(A)/2
    (以下对圆的内接四边形)
    3. ac+bd=D1D2
    4. S=sqrt((P-a)(P-b)(P-c)(P-d)),P为半周长


    正n边形:
    R为外接圆半径,r为内切圆半径
    1. 中心角 A=2PI/n
    2. 内角 C=(n-2)PI/n
    3. 边长 a=2sqrt(R^2-r^2)=2Rsin(A/2)=2rtan(A/2)
    4. 面积 S=nar/2=nr^2tan(A/2)=nR^2sin(A)/2=na^2/(4tan(A/2))


    圆:
    1. 弧长 l=rA
    2. 弦长 a=2sqrt(2hr-h^2)=2rsin(A/2)
    3. 弓形高 h=r-sqrt(r^2-a^2/4)=r(1-cos(A/2))=atan(A/4)/2
    4. 扇形面积 S1=rl/2=r^2A/2
    5. 弓形面积 S2=(rl-a(r-h))/2=r^2(A-sin(A))/2


    棱柱:
    1. 体积 V=Ah,A为底面积,h为高
    2. 侧面积 S=lp,l为棱长,p为直截面周长
    3. 全面积 T=S+2A

```

```
43
44      棱锥:
45      1. 体积 V=Ah/3,A为底面积,h为高
46      (以下对正棱锥)
47      2. 侧面积 S=lp/2,l为斜高,p为底面周长
48      3. 全面积 T=S+A
49
50
51      棱台:
52      1. 体积 V=(A1+A2+sqrt(A1A2))h/3,A1.A2为上下底面积,h为高
53      (以下为正棱台)
54      2. 侧面积 S=(p1+p2)l/2,p1.p2为上下底面周长,l为斜高
55      3. 全面积 T=S+A1+A2
56
57
58      圆柱:
59      1. 侧面积 S=2PIrh
60      2. 全面积 T=2PIr(h+r)
61      3. 体积 V=PIr^2h
62
63
64      圆锥:
65      1. 母线 l=sqrt(h^2+r^2)
66      2. 侧面积 S=PIrl
67      3. 全面积 T=PIr(l+r)
68      4. 体积 V=PIr^2h/3
69
70
71      圆台:
72      1. 母线 l=sqrt(h^2+(r1-r2)^2)
73      2. 侧面积 S=PI(r1+r2)l
74      3. 全面积 T=PIr1(l+r1)+PIr2(l+r2)
75      4. 体积 V=PI(r1^2+r2^2+r1r2)h/3
76
77
78      球:
79      1. 全面积 T=4PIr^2
80      2. 体积 V=4PIr^3/3
81
82
83      球台:
84      1. 侧面积 S=2PIrh
85      2. 全面积 T=PI(2rh+r1^2+r2^2)
86      3. 体积 V=PIh(3(r1^2+r2^2)+h^2)/6
87
88
89      球扇形:
90      1. 全面积 T=PIr(2h+r0),h为球冠高,r0为球冠底面半径
91      2. 体积 V=2PIr^2h/3
```

8.2 求凸包

```cpp
/*
 *  Graham 求凸包 O(N * logN)
 *  CALL: nr = graham(pnt, int n, res); res[]为凸包点集;
 */
struct point {
    double x, y;
};

bool mult(point sp, point ep, point op) {
    return (sp.x - op.x) * (ep.y - op.y) >= (ep.x - op.x) * (sp.y - op.y);
}

inline bool operator < (const point &l, const point &r) {
    return l.y < r.y || (l.y == r.y && l.x < r.x);
}

int graham(point pnt[], int n, point res[]) {
    int i, len, top = 1;
    sort(pnt, pnt + n);
    if (n == 0) {
        return 0;
    }
    res[0] = pnt[0];
    if (n == 1) {
        return 1;
    }
    res[1] = pnt[1];
    if (n == 2) {
        return 2;
    }
    res[2] = pnt[2];
    for (i = 2; i < n; i++) {
        while (top && mult(pnt[i], res[top], res[top - 1])) {
            top--;
        }
        res[++top] = pnt[i];
    }
    len = top;
    res[++top] = pnt[n - 2];
    for (i = n - 3; i >= 0; i--) {
        while (top != len && mult(pnt[i], res[top], res[top - 1])) {
            top--;
        }
        res[++top] = pnt[i];
    }
    return top; //  返回凸包中点的个数
}
```

8.3 四点共面

```cpp
struct  point {
    double x, y, z;
```

```
    point  operator - (point &o) {
        point  ans;
        ans.x = this->x - o.x;
        ans.y = this->y - o.y;
        ans.z = this->z - o.z;
        return ans;
    }
};

double  dot_product(const point &a, const point &b) {
    return a.x * b.x + a.y * b.y + a.z * b.z;
}

point cross_product(const point &a, const point &b) {
    point  ans;
    ans.x = a.y * b.z - a.z * b.y;
    ans.y = a.z * b.x - a.x * b.z;
    ans.z = a.x * b.y - a.y * b.x;
    return ans;
}

int main() {
    point p[4];
    int T;
    for (scanf("%d", &T); T--;) {
        for (int i = 0; i < 4; ++i) {
            scanf("%lf%lf%lf", &p[i].x, &p[i].y, &p[i].z);
        }
        puts(dot_product(p[3] - p[0], cross_product(p[2] - p[0], p[1] - p[0])) == 0.0 ? "Yes\n
            " : "No\n");
    }
    return 0;
}
```

8.4 多边形重心

```
/*
 *  求多边形重心
 *  INIT: pnt[]已按顺时针(或逆时针)排好序; | CALL: res = bcenter(pnt, n);
 */
struct point {
    double x, y;
};

point bcenter(point pnt[], int n) {
    point p, s;
    double tp, area = 0, tpx = 0, tpy = 0;
    p.x = pnt[0].x;
    p.y = pnt[0].y;
    for (int i = 1; i <= n; ++i) {   //  point:0 ~ n - 1
        s.x = pnt[(i == n) ? 0 : i].x;
        s.y = pnt[(i == n) ? 0 : i].y;
```

```
        tp = (p.x * s.y - s.x * p.y);
        area += tp / 2;
        tpx += (p.x + s.x) * tp;
        tpy += (p.y + s.y) * tp;
        p.x = s.x;
        p.y = s.y;
    }
    s.x = tpx / (6 * area);
    s.y = tpy / (6 * area);
    return s;
}
```

8.5 旋转卡壳

```
struct Point {
    int x, y;
    Point(int _x = 0, int _y = 0) {
        x = _x;
        y = _y;
    }
    Point operator - (const Point &b)const {
        return Point(x - b.x, y - b.y);
    }
    int operator ^(const Point &b)const {
        return x * b.y - y * b.x;
    }
    int operator *(const Point &b)const {
        return x * b.x + y * b.y;
    }
    void input() {
        scanf("%d%d", &x, &y);
        return ;
    }
};

//  距离的平方
int dist2(Point a, Point b) {
    return (a - b) * (a - b);
}

//  二维凸包,int
const int MAXN = 50010;
Point list[MAXN];
int Stack[MAXN], top;
bool _cmp(Point p1, Point p2) {
    int tmp = (p1 - list[0]) ^ (p2 - list[0]);
    if (tmp > 0) {
        return true;
    }
    else if (tmp == 0 && dist2(p1, list[0]) <= dist2(p2, list[0])) {
        return true;
    } else {
```

```
39          return false;
40      }
41  }
42
43  void Graham(int n) {
44      Point p0;
45      int k = 0;
46      p0 = list[0];
47      for (int i = 1; i < n; i++) {
48          if (p0.y > list[i].y || (p0.y == list[i].y && p0.x > list[i].x)) {
49              p0 = list[i];
50              k = i;
51          }
52      }
53      swap(list[k], list[0]);
54      sort(list + 1, list + n, _cmp);
55      if (n == 1) {
56          top = 1;
57          Stack[0] = 0;
58          return ;
59      }
60      if (n == 2) {
61          top = 2;
62          Stack[0] = 0;
63          Stack[1] = 1;
64          return ;
65      }
66      Stack[0] = 0;
67      Stack[1] = 1;
68      top = 2;
69      for (int i = 2; i < n; i++) {
70          while (top > 1 && ((list[Stack[top - 1]] - list[Stack[top - 2]]) ^ (list[i] - list[
                Stack[top - 2]])) <= 0) {
71              top--;
72          }
73          Stack[top++] = i;
74      }
75      return ;
76  }
77
78  //  旋转卡壳,求两点间距离平方的最大值
79  int rotating_calipers(Point p[],int n) {
80      int ans = 0;
81      Point v;
82      int cur = 1;
83      for (int i = 0; i < n; i++) {
84          v = p[i] - p[(i + 1) % n];
85          while ((v ^ (p[(cur + 1) % n] - p[cur])) < 0) {
86              cur = (cur + 1) % n;
87          }
88          ans = max(ans, max(dist2(p[i], p[cur]), dist2(p[(i + 1) % n], p[(cur + 1) % n])));
89      }
90      return ans;
91  }
```

```

Point p[MAXN];

int main() {
    int n;
    while (scanf("%d", &n) == 1) {
        for (int i = 0; i < n; i++) {
            list[i].input();
        }
        Graham(n);
        for (int i = 0; i < top; i++) {
            p[i] = list[Stack[i]];
        }
        printf("%d\n", rotating_calipers(p, top));
    }
    return 0;
}
```

8.6 模拟退火

```
//模拟退火
//平面上找一个点 使得 sigma(1..N)dist(a, i)*Wi 最小

#include <cstdlib>
#include <cstdio>
#include <cstring>
#include <cmath>

#define INF (1e17)
#define EPS (1e-3)
#define PI  (acos(-1.0))
#define FIRE(x) (x *= 0.99)
using namespace std;
const int MAXN = 10000 + 10;
int N;
double total = INF;
struct Point {
    double x, y, w;
    Point (double _x, double _y) : x(_x), y(_y) {}
    Point (void) {}
    void Read(void) {
        scanf("%lf%lf%lf", &x, &y, &w);
    }
    void operator += (Point t) {
        x += t.x; y += t.y;
    }
    void operator /= (int N) {
        x /= N, y /= N;
    }
};
Point now, ans, point[MAXN];
inline double Dist(Point a, Point b) {
```

```
    return sqrt((a.x - b.x) * (a.x - b.x) +
                (a.y - b.y) * (a.y - b.y));
}
inline double Statistic(Point p) {
    double res = 0.0;
    for (int i = 0; i < N; i++) res += Dist(p, point[i]) * point[i].w;
    if (res < total) total = res, ans = p;
    return res;
}
inline double Rand(void) {
    return (rand() % 1000 + 1) / 1000.0;
}
int main(void) {
    srand(10086);
    scanf("%d", &N);
    register int i;
    for (i = 0; i < N; i++) point[i].Read(), now += point[i];
    now /= N;
    double T = 100000.0, alpha, sub;
    while (T > EPS) {
        alpha = 2.0 * PI * Rand();
        Point tmp(now.x + T * cos(alpha), now.y + T * sin(alpha));
        sub = Statistic(now) - Statistic(tmp);
        if(sub >= 0 || exp(sub / T) >= Rand()) now = tmp;
        FIRE(T);
    }
    T = 0.001;
    for (i = 1; i <= 1000; ++i) {
        alpha = 2.0 * PI * Rand();
        Point tmp(ans.x + T * cos(alpha) * Rand(), ans.y + T * sin(alpha) * Rand());
        Statistic(tmp);
    }
    printf("%.3lf␣%.3lf\n", ans.x, ans.y);
    return 0;
}
```

8.7 半平面交

```
#include <cstdio>
#include <algorithm>
#include <queue>
#include <cmath>
using namespace std;
const double eps = 1e-8;
struct Point{
    double x,y;
    Point(double xx=0.0,double yy=0.0) :x(xx) ,y(yy) {}
    Point operator - (const Point &b) const {
        return Point(x-b.x,y-b.y) ;
    }
    Point operator +(const Point &b) const {
        return Point(x+b.x,y+b.y) ;
```

```
15      }
16      Point operator /(const double &b) const {
17          return Point(x/b,y/b) ;
18      }
19      Point operator *(const double &b) const {
20          return Point(x*b,y*b) ;
21      }
22      double operator ^(const Point &b) const {
23          return x*b.y-y*b.x;
24      }
25  };
26  typedef Point myvec;
27  double cross(myvec a,myvec b) {
28      return a^b;
29  }
30  struct Line{
31      Point p;
32      myvec v;
33      double ang;
34      Line() {}
35      Line( Point pp,myvec vv) :p(pp) ,v(vv) {}
36      bool operator < (const  Line &l) const {
37          return ang < l.ang;
38      }
39
40  };
41  //点p在有向直线L的左边（线上不算）
42  bool on_left( Line l,Point p) {
43      return cross(l.v,p-l.p) >0;
44  }
45  //直线交点 假设交点唯一存在
46  Point get_inter_section(Line a,Line b) {
47      myvec u = a.p - b. p;
48      double t = cross(b.v,u) /cross(a.v,b.v) ;
49      return a.p+a.v*t;
50
51  }
52  int half_plane_inter_section(Line *L,int n,Point *poly) {
53      sort(L,L+n) ;//级角排序
54      int fir,lst;//双向队列的第一个元素和最后一个元素的下标
55      Point *p = new Point[n];//p[i] 为q[i]和q[i+1]的交点
56      Line *q = new Line[n];//双端队列
57      q[ fir = lst = 0 ] = L[0];//双端队列初始化为只有一个半平面的L[0]
58      for( int i =1; i <n ; ++i)
59      {
60          while( fir < lst && !on_left(L[i],p[lst-1])  )
61              lst--;
62          while( fir<lst && !on_left(L[i],p[fir])  )
63              fir++;
64          q[++lst] = L[i];
65          if( fabs( cross(q[lst].v,q[lst-1].v)  )  < eps ) { //两向量平行且同向 取内侧一个
66              lst--;
67              if( on_left(q[lst],L[i].p)  )
68                  q[lst] = L[i];
```

```
69          }
70          if( fir < lst )
71              p[lst-1] = get_inter_section(q[lst-1],q[lst]) ;
72      }
73      while( fir< lst && !on_left(q[fir],p[lst-1]) )
74          lst--;//删除无用的平面
75      if(lst - fir <=1 )
76          return 0;//空集
77      p[lst] = get_inter_section(q[lst],q[fir]) ;//计算首尾两个半平面的交点
78      //从 deque 复制到输出中
79      int m = 0 ;
80       for( int i = fir;i<=lst;++i)
81          poly[m++] = p[i];
82       return m;
83  }
```

8.8 计算几何

```
1   //上交计算几何算法
2   /***************************************
3   * COMPUTATIONAL GEOMETRY ROUTINES
4   * WRITTEN BY : LIU Yu (C) 2003
5   ***************************************/
6   //      叉乘
7   //      两个点的距离
8   //      点到直线距离
9   //      返回直线 Ax + By + C =0  的系数
10  //      线段
11  //      圆
12  //      两个圆的公共面积
13  //      矩形
14  //      根据下标返回多边形的边
15  //      两个矩形的公共面积
16  //      多边形  ,逆时针或顺时针给出x,y
17  //      多边形顶点
18  //      多边形的边
19  //      多边形的周长
20  //      判断点是否在线段上
21  //      判断两条线断是否相交, 端点重合算相交
22  //      判断两条线断是否平行
23  //      判断两条直线断是否相交
24  //      直线相交的交点
25  //      判断是否简单多边形
26  //      求多边形面积
27  //      判断是否在多边形上
28  //      判断是否在多边形内部
29  //      点阵的凸包, 返回一个多边形
30  //    最近点对的距离
31  #include <cmath>
32  #include <cstdio>
33  #include <memory>
34  #include <algorithm>
```

```cpp
#include <iostream>
using namespace std;
typedef double TYPE;
#define Abs(x) (((x)>0)?(x):(-(x)))
#define Sgn(x) (((x)<0)?(-1):(1))
#define Max(a,b) (((a)>(b))?(a):(b))
#define Min(a,b) (((a)<(b))?(a):(b))
#define Epsilon 1e-10
#define Infinity 1e+10
#define Pi 3.14159265358979323846
TYPE Deg2Rad(TYPE deg) {
    return (deg * Pi / 180.0);
}
TYPE Rad2Deg(TYPE rad) {
    return (rad * 180.0 / Pi);
}
TYPE Sin(TYPE deg) {
    return sin(Deg2Rad(deg));
}
TYPE Cos(TYPE deg) {
    return cos(Deg2Rad(deg));
}
TYPE ArcSin(TYPE val) {
    return Rad2Deg(asin(val));
}
TYPE ArcCos(TYPE val) {
    return Rad2Deg(acos(val));
}
TYPE Sqrt(TYPE val) {
    return sqrt(val);
}
struct POINT {
    TYPE x;
    TYPE y;
    TYPE z;
    POINT() : x(0), y(0), z(0) {};
    POINT(TYPE _x_, TYPE _y_, TYPE _z_ = 0) : x(_x_), y(_y_), z(_z_) {};
};
// cross product of (o->a) and (o->b)// 叉乘
TYPE Cross(const POINT & a, const POINT & b, const POINT & o) {
    return (a.x - o.x) * (b.y - o.y) - (b.x - o.x) * (a.y - o.y);
}
// planar points' distance//  两个点的距离
TYPE Distance(const POINT & a, const POINT & b) {
    return Sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y) + (a.z - b.z) * (a.z
                - b.z));
}
struct LINE {
    POINT a;
    POINT b;
    LINE() {};
    LINE(POINT _a_, POINT _b_) : a(_a_), b(_b_) {}};
//点到直线距离
double PointToLine(POINT p0 ,POINT p1 ,POINT p2 ,POINT &cp) {
```

```
    double d = Distance(p1 ,p2);
    double s = Cross(p1 ,p2 ,p0) / d;
    cp.x = p0.x + s*( p2.y-p1.y) / d;
    cp.y = p0.y - s*( p2.x-p1.x) / d;
    return Abs(s);
}
// 返回直线 Ax + By + C =0  的系数
void Coefficient(const LINE & L, TYPE & A, TYPE & B, TYPE & C) {
    A = L.b.y - L.a.y;
    B = L.a.x - L.b.x;
    C = L.b.x * L.a.y - L.a.x * L.b.y;
}
void Coefficient(const POINT & p,const TYPE a,TYPE & A,TYPE & B,TYPE & C) {
    A = Cos(a);    // 线段
    B = Sin(a);
    C = - (p.y * B + p.x * A);
}
struct SEG {
    POINT a;
    POINT b;
    SEG() {};
    SEG(POINT  _a_, POINT _b_):a(_a_),b(_b_) {};
};
// 圆
struct CIRCLE {
    TYPE x;
    TYPE y;
    TYPE r;
    CIRCLE() {}
    CIRCLE(TYPE _x_, TYPE _y_, TYPE _r_) : x(_x_), y(_y_), r(_r_) {}};
POINT Center(const CIRCLE & circle) {
    return POINT(circle.x, circle.y);
}
TYPE Area(const CIRCLE & circle) {
    return Pi * circle.r * circle.r;    //两个圆的公共面积
}
TYPE CommonArea(const  CIRCLE & A, const CIRCLE & B) {
    TYPE area = 0.0;
    const CIRCLE & M = (A.r > B.r) ? A : B;
    const CIRCLE & N = (A.r > B.r) ? B : A;
    TYPE D = Distance(Center(M), Center(N));
    if ((D < M.r + N.r) && (D > M.r - N.r)) {
        TYPE cosM = (M.r * M.r + D * D - N.r * N.r) / (2.0 * M.r * D);
        TYPE cosN = (N.r * N.r + D * D - M.r * M.r) / (2.0 * N.r * D);
        TYPE alpha = 2.0 * ArcCos(cosM);
        TYPE beta  = 2.0 * ArcCos(cosN);
        TYPE TM = 0.5*M.r*M.r*Sin(alpha);
        TYPE TN=0.5 * N.r * N.r * Sin(beta);
        TYPE FM = (alpha / 360.0) * Area(M);
        TYPE FN = (beta / 360.0) * Area(N);
        area = FM + FN - TM - TN;
    } else if (D <=  M.r - N.r) {
        area = Area(N);
    }
```

```
    return area;
}
//    矩形
//    矩形的线段
//        2
//      ———————————— b
//      |              |
//    3 |              |  1
//    a ————————————
//          0
struct RECT {
    POINT a; // 左下点  POINT b;  // 右上点
    RECT() {};
    RECT(const POINT & _a_, const POINT & _b_) {
        a = _a_;
        b = _b_;
    }
};
//根据下标返回多边形的边
SEG Edge(const RECT & rect, int idx) {
    SEG edge;
    while (idx < 0) idx +=  4;
    switch (idx % 4) {
    case 0:
        edge.a = rect.a;
        edge.b = POINT(rect.b.x, rect.a.y);
        break;
    case 1:
        edge.a = POINT(rect.b.x, rect.a.y);
        edge.b = rect.b;
        break;
    case 2:
        edge.a = rect.b;
        edge.b = POINT(rect.a.x, rect.b.y);
        break;
    case 3:
        edge.a = POINT(rect.a.x, rect.b.y);
        edge.b = rect.a;
        break;
    default:
        break;
    }
    return edge;
}
TYPE Area(const RECT & rect) {
    return (rect.b.x - rect.a.x) * (rect.b.y - rect.a.y);
}
//  两个矩形的公共面积
TYPE CommonArea(const  RECT & A, const RECT & B) {
    TYPE area = 0.0;
    POINT LL(Max(A.a.x, B.a.x), Max(A.a.y, B.a.y));
    POINT UR(Min(A.b.x,  B.b.x), Min(A.b.y, B.b.y));
    if ((LL.x <=  UR.x) && (LL.y <= UR.y)) {
        area = Area(RECT(LL, UR));
```

```
197     }
198     return area;
199 }// 多边形  ,逆时针或顺时针给出x,y
200 struct POLY {
201     int n; //n个点  TYPE * x;  //x,y为点的指针，首尾必须重合  TYPE * y;
202     POLY() : n(0), x(NULL), y(NULL) {};
203     POLY(int _n_, const TYPE * _x_, const TYPE * _y_) {
204         n = _n_;
205         x = new TYPE[n + 1];
206         memcpy(x,  _x_, n*sizeof(TYPE));
207         x[n] = _x_[0];
208         y = new TYPE[n + 1];
209         memcpy(y, _y_, n*sizeof(TYPE));
210         y[n] = _y_[0];
211     }
212 };//多边形顶点
213 POINT Vertex(const POLY & poly, int idx) {
214     idx %= poly.n;    //多边形的边
215     return POINT(poly.x[idx],  poly.y[idx]);
216 }
217 SEG Edge(const POLY & poly, int idx) {
218     idx %= poly.n;
219     return SEG(POINT(poly.x[idx],  poly.y[idx]),
220                POINT(poly.x[idx  + 1], poly.y[idx + 1]));
221 } //多边形的周长
222 TYPE Perimeter(const POLY & poly) {
223     TYPE p = 0.0;
224     for (int i = 0; i < poly.n; i++)
225         p = p + Distance(Vertex(poly, i), Vertex(poly, i + 1));
226     return p;
227 }
228 bool IsEqual(TYPE a, TYPE b) {
229     return (Abs(a - b) < Epsilon);
230 }
231 bool IsEqual(const POINT & a, const POINT & b) {
232     return (IsEqual(a.x, b.x) && IsEqual(a.y, b.y));
233 }
234 bool IsEqual(const LINE & A, const LINE & B) {
235     TYPE A1, B1, C1;
236     TYPE A2, B2, C2;
237     Coefficient(A, A1, B1, C1);
238     Coefficient(B, A2, B2, C2);
239     return IsEqual(A1 * B2, A2 * B1) &&
240            IsEqual(A1 * C2, A2 * C1) &&  IsEqual(B1 * C2, B2 * C1);
241 }
242 // 判断点是否在线段上
243 bool IsOnSeg(const SEG & seg, const POINT & p) {
244     return (IsEqual(p, seg.a) || IsEqual(p, seg.b)) ||
245            (((p.x - seg.a.x) * (p.x - seg.b.x) < 0 || (p.y - seg.a.y) * (p.y - seg.b.y) < 0)
                &&
246             (IsEqual(Cross(seg.b, p, seg.a), 0)));
247 }
248 //判断两条线断是否相交，端点重合算相交
249 bool IsIntersect(const SEG & u, const SEG & v) {
```

```
250     return (Cross(v.a, u.b, u.a) * Cross(u.b, v.b, u.a) >=  0) &&
251            (Cross(u.a, v.b, v.a) * Cross(v.b, u.b, v.a) >= 0) &&(Max(u.a.x, u.b.x)>=Min(v.
252                   a.x, v.b.x))&&(Max(v.a.x, v.b.x)>= Min(u.a.x,u.b.x))&&(Max(u.a.y, u.b.y)>=
                       Min(
253                              v.a.y, v.b.y))&&(Max(v.a.y, v.b.y)>=Min(u.a.y,  u.b.y));
254 }
255 //判断两条线断是否平行
256 bool IsParallel(const LINE & A, const LINE & B) {
257     TYPE A1, B1, C1;
258     TYPE A2, B2, C2;
259     Coefficient(A, A1, B1, C1);
260     Coefficient(B, A2, B2, C2);
261     return (A1*B2==  A2*B1) &&((A1 * C2 != A2 * C1) || (B1 * C2 != B2 * C1));
262 }
263 //判断两条直线断是否相交
264 bool IsIntersect(const LINE & A, const LINE & B) {
265     return !IsParallel(A, B);    //直线相交的交点
266 }
267
268 POINT Intersection(const LINE & A, const LINE & B) {
269     TYPE A1, B1, C1;
270     TYPE A2, B2, C2;
271     Coefficient(A, A1, B1, C1);
272     Coefficient(B, A2, B2, C2);
273     POINT I(0, 0);
274     I.x = - (B2 * C1 - B1 * C2) / (A1 * B2 - A2 * B1);
275     I.y =   (A2 * C1 - A1 * C2) / (A1 * B2 - A2 * B1);
276     return I;
277 }
278
279 bool IsInCircle(const CIRCLE & circle, const RECT & rect) {
280     return (circle.x - circle.r >=  rect.a.x) &&
281            (circle.x + circle.r <= rect.b.x) &&(circle.y - circle.r >=  rect.a.y) &&
282            (circle.y + circle.r <= rect.b.y);
283 }
284 //判断是否简单多边形
285 bool IsSimple(const POLY & poly) {
286     if (poly.n < 3) return false;
287     SEG L1, L2;
288     for (int i = 0; i < poly.n - 1; i++) {
289         L1 = Edge(poly, i);
290         for (int j = i + 1; j < poly.n; j++) {
291             L2 = Edge(poly, j);
292             if (j == i+1) {
293                 if  (IsOnSeg(L1, L2.b)||IsOnSeg(L2, L1.a))  return false;
294             } else if (j == poly.n - i - 1) {
295                 if (IsOnSeg(L1, L2.a) || IsOnSeg(L2, L1.b))  return false;
296             } else {
297                 if (IsIntersect(L1, L2))  return false;    // for i
298             }
299         } // for j
300     }
301     return true;
302 }
```

```
303 //求多边形面积
304 TYPE Area(const POLY & poly) {
305     if (poly.n < 3) return TYPE(0);
306     double s = poly.y[0] * (poly.x[poly.n - 1] - poly.x[1]);
307     for (int i = 1; i < poly.n; i++) {
308         s += poly.y[i] * (poly.x[i - 1] - poly.x[(i + 1) % poly.n]);
309     }
310     return s/2;
311 }
312 //判断是否在多边形上
313 bool IsOnPoly(const POLY & poly, const POINT & p) {
314     for (int i = 0; i < poly.n; i++) {
315         if (IsOnSeg(Edge(poly,  i), p))   return true;
316     }
317     return false;
318 }
319 //判断是否在多边形内部
320 bool IsInPoly(const POLY & poly, const POINT & p) {
321     SEG L(p, POINT(Infinity, p.y));
322     int count = 0;
323     for (int i = 0; i < poly.n; i++) {
324         SEG S = Edge(poly, i);
325         if (IsOnSeg(S, p)) {
326             return false; //如果想让 在poly上则返回 true,则改为true
327         }
328         if (!IsEqual(S.a.y, S.b.y)) {
329             POINT & q = (S.a.y > S.b.y)?(S.a):(S.b);
330             if (IsOnSeg(L, q)) {
331                 ++count;
332             }
333             else if(!IsOnSeg(L,S.a)&&!IsOnSeg(L,S.b)&&IsIntersect(S,L)) {
334                 ++count;
335             }
336         }
337     }
338     return (count % 2 != 0);
339 }
340 // 点阵的凸包，返回一个多边形
341 POLY ConvexHull(const POINT * set, int n) {
342     POINT * points = new POINT[n];
343     memcpy(points, set, n * sizeof(POINT));
344     TYPE * X = new TYPE[n];
345     TYPE * Y = new TYPE[n];
346     int i, j, k = 0, top = 2;
347     for(i = 1; i < n; i++) {
348         if((points[i].y<points[k].y)||((points[i].y==points[k].y)&&
349                                        (points[i].x<points[k].x))) {
350             k  = i;
351         }
352     }
353     std::swap(points[0], points[k]);
354     for (i = 1; i < n - 1; i++) {
355         k = i;
356         for (j = i + 1; j < n; j++) {
```

```
            if ((Cross(points[j], points[k], points[0]) >0)||((Cross(points[j], points[k],
                    points[0]) == 0) && (Distance(points[0], points[j])<Distance(points[0],
                        points[k]
                                                                                    )))) {
                k = j;
            }
        }
        std::swap(points[i], points[k]);
    }
    X[0] = points[0].x;
    Y[0] = points[0].y;
    X[1] = points[1].x;
    Y[1] = points[1].y;
    X[2] = points[2].x;
    Y[2] = points[2].y;
    for (i = 3; i < n; i++) {
        while(Cross(points[i],POINT(X[top],Y[top]),POINT(X[top
                    -1],Y[top-1]))>=0) {
            top--;
        }
        ++top;
        X[top] = points[i].x;
        Y[top] = points[i].y;
    }
    delete [] points;
    POLY poly(++top, X, Y);
    delete [] X;
    delete [] Y;
    return poly;
}
//最近点对的距离, Written By PrincessSnow
#define MAXN 100000
POINT pt[MAXN];
bool cmp(POINT n1, POINT n2) {
    return (n1.x<n2.x  || n1.x==n2.x  && n1.y<n2.y);
}
double Get(double dis, int mid, int start, int end) {
    int s=mid,  e=mid,  i, j;
    double t;
    while(s > start && pt[mid].x - pt[s].x <= dis)     s--;
    while(e < end && pt[e].x - pt[mid].x <=  dis)     e++;
    for(i=s; i <= e; i++)
        for(j=i+1;  j <= e && j <=  i+7; j++) {
            t = Distance(pt[i], pt[j]);
            if(t < dis)     dis=t;
        }
    return dis;
}
double ClosestPairDistance(int start, int end) {
    int m = end-start+1, mid, i;
    double t1, t2, dis=-1, t;
    if(m <= 3) {
        for(i=start; i < end; i++) {
            t = Distance(pt[i] , pt[i+1]);
```

```
            if(t < dis || dis ==  -1)     dis = t;
        }
        t = Distance(pt[start] , pt[end]);
        if(t < dis) dis=t;
        return dis;
    }
    if(m%2  == 0)     mid = start + m/2 - 1;
    else               mid = start + m/2;
    if(m%2  == 0) {
        t1 = ClosestPairDistance(start, mid);
        t2 = ClosestPairDistance(mid+1, end);
    } else {
        t1 =ClosestPairDistance(start,mid);
        t2=ClosestPairDistance(mid+1,end);
    }
    if(t1 < t2)     dis = t1;
    else          dis = t2;
    dis = Get(dis, mid, start, end);
    return dis;
}


//1.  球面上两点最短距离
// 计算圆心角lat 表示纬度, -90 <= w <= 90, lng 表示经度
// 返回两点所在大圆劣弧对应圆心角,  0 <= angle <=  pi
double angle(double lng1, double lat1, double lng2, double lat2) {
    double dlng = fabs(lng1 - lng2) * pi / 180;
    while(dlng >= pi+pi)      dlng -= pi+pi;
    if(dlng > pi)     dlng = pi + pi - dlng;
    lat1 *= pi / 180,     lat2 *= pi / 180;
    return acos( cos(lat1)*cos(lat2)*cos(dlng) + sin(lat1)*sin(lat2) );
}
// 计算距离, r为球半径
double line_dist(double  r, double lng1, double lat1, double lng2, double lat2) {
    double dlng = fabs(lng1 - lng2) * pi / 180;
    while(dlng  >= pi+pi)      dlng -= pi+pi;
    if(dlng > pi)     dlng = pi + pi - dlng;
    lat1 *= pi / 180,     lat2 *= pi / 180;
    return r*sqrt(2-2*( cos(lat1)*cos(lat2)*cos(dlng)+ sin(lat1)*sin(lat2)) );
}
// 计算球面距离, r为球半径
double sphere_dist(double r, double lng1,double lat1,double lng2, double lat2) {
    return r * angle(lng1, lat1, lng2, lat2);
}


//2.  三点求圆心坐标
double GetRadiusBy3Points(double  x1, double y1,
                          double x2, double y2,  double x3, double y3, double &x, double &y) {
    // 由 ( x - x1 )^2 + ( y - y1 )^2 = ( x - x2 )^2 + ( y - y2 )^2 得
    // 2*( x2 - x1 )*x + 2*( y2 - y1 )*y = x2^2 - x1^2 + y2^2 - y1^2
    // 同理得
    // 2*( x3 - x2 )*x + 2*( y3 - y2 )*y = x3^2 - x2^2 + y3^2 - y2^2
    // 由行列式解方程得 x , y
```

```
464     double a11, a12, a21, a22, b1, b2;
465     double d, d1, d2 ;
466     a11 = 2 * ( x3 - x2 );
467     a12 = 2 * ( y3 - y2 );
468     a21 = 2 * ( x2 - x1 );
469     a22 = 2 * ( y2 - y1 );
470     b1 = x3*x3 - x2*x2 + y3*y3 - y2*y2;
471     b2 = x2*x2 - x1*x1 + y2*y2 - y1*y1;
472     d = a11*a22 - a12*a21;
473     d1 = b1*a22 - a12*b2;
474     d2 = a11*b2 - b1*a21;
475     // x , y 是圆心坐标
476     x = d1 / d;
477     y = d2 / d;
478     return  (x1 - x)*(x1 - x) + (y1 - y)*(y1 - y);
479 }
480 //
481 //
482 //3.  三角形几个重要的点
483 //设三角形的三条边为a,  b, c, 且不妨假设a <=  b <=  c
484 //三角形的面积可以根据海伦公式算得, 如下:
485 //s = sqrt(p * (p - a) * (p - b) * (p - c)), p = (a +  b + c) / 2
486 //1. 费马点(该点到三角形三个顶点的距离之和最小)
487 //有个有趣的结论: 若三角形的三个内角均小于120度,
488 //那么该点连接三个顶点形成的三个角均为120度; 若三角形存在一个内角
489 //大于120度, 则该顶点就是费马点)
490 //计算公式如下:
491 //若有一个内角大于120度(这里假设为角C), 则距离为a  +  b
492 //若三个内角均小于120度, 则距离为
493 //sqrt((a * a +  b * b + c * c + 4 * sqrt(3.0) * s) / 2),其中
494 //2.内心——角平分线的交点
495 //令x =  (a + b - c) / 2, y =  (a - b +  c) / 2, z =  (-a +  b + c) / 2, h =  s / p
496 // 计算公式为 sqrt(x * x + h * h) + sqrt(y * y +  h * h) +  sqrt(z * z +  h * h)
497 //3.重心——中线的交点
498 //ACM算法模板集
499 //  - 46 -
500 //计算公式如下:
501 //2.0 / 3 * (sqrt((2 * (a * a +  b * b) - c * c) / 4)
502 //              +  sqrt((2 * (a * a +  c * c) - b * b) / 4)
503 //              +  sqrt((2 * (b * b + c * c) - a * a) / 4))
504 //4.垂心——垂线的交点
505 //计算公式如下:
506 //3 * (c / 2 / sqrt(1 - cosC * cosC))
```

第九章　类

9.1　点类

```
struct point {
    double x, y;
    point() { };
    point(double x, double y) :x(x), y(y) { }
    point operator - (const point &b) const {
        return point(x - b.x, y - b.y);
    }
    point operator + (const point &b) const {
        return point(x + b.x, y + b.y);
    }
    point operator * (const double k) const {
        return point(k * x, k * y);
    }
    point operator / (const double k) const {
        return point(x / k, y / k);
    }
    double slope() {
        return y / x;
    }
};
```

9.2　分数类

```
struct Fraction {
    long long num;
    long long den;
    Fraction(long long num=0,long long den=1) {
        if(den<0) {
            num=-num;
            den=-den;
        }
        assert(den!=0);
        long long g=gcd(abs(num),den);
        this->num=num/g;
        this->den=den/g;
    }
    Fraction operator +(const Fraction &o)const {
        return Fraction(num*o.den+o.num,den*o.den);
    }
```

```
    Fraction operator -(const Fraction &o)const {
        return Fraction(num*o.den-den*o.num,den*o.den);
    }
    Fraction operator *(const Fraction &o)const {
        return Fraction(num*o.num,den*o.den);
    }
    Fraction operator /(const Fraction &o)const {
        return Fraction(num*o.den,den*o.num);
    }
    bool operator <(const Fraction &o)const {
        return num*o.den< den*o.num;
    }
    bool operator ==(const Fraction &o)const {
        return num*o.den==den*o.num;
    }
};
```

9.3 矩阵

```
#define maxm 10
typedef long long LL;

const LL Mod=1e9+7;
struct Matrix {
    int n, m;
    LL mat[maxm][maxm];
    void clear() {
        memset(mat, 0, sizeof(mat));
    }

    Matrix(int n, int m) :n(n), m(m) {
        //不要设置默认构造函数，让编译器检查初始化遗漏
        clear();
    }

    Matrix operator +(const Matrix &M) const {
        Matrix res(n, m);
        for (LL i = 0; i < n; ++i) for (LL j = 0; j < m; ++j) {
            res.mat[i][j] = (mat[i][j] + M.mat[i][j]) % Mod;
        }
        return res;
    }

    Matrix operator *(const Matrix &M) const {
        if (m != M.n){
            std::cout << "Wrong!" << std::endl;
            return Matrix(-1, -1);
        }
        Matrix res(n, M.m);
        res.clear();
        int i,j,k;
        for (i = 0; i < n; ++i)
```

```
            for (j = 0; j < M.m; ++j)
                for (k = 0; k < m; ++k) {
                    res.mat[i][j] += mat[i][k] * M.mat[k][j]%Mod;
                    res.mat[i][j] %= Mod;
                }
        return res;
    }
    Matrix operator *(const LL &x) const {
        Matrix res(n,m);
        int i,j;
        std::cout << n << '␣' << m << std::endl;
        for (i = 0; i < n; ++i)
            for (j = 0; j < m; ++j)
                res[i][j] = mat[i][j] * x % Mod;
        return res;
    }

    Matrix operator ^(LL b) const { // 矩阵快速幂 ， 取余Mod
        if (n != m)
            return Matrix(-1, -1);
        Matrix a(*this);
        Matrix res(n, n);
        res.clear();
        for (LL i = 0; i < n; ++i)
            res.mat[i][i] = 1;
        for (; b; b >>= 1) {
            if (b & 1) {
                res = a * res;
            }
            a = a * a;
        }
        return res;
    }

    LL* operator [](int i) {
        return mat[i];
    }

    void Print() const {
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < m; ++j)
                std::cout << mat[i][j] << '␣';
            std::cout << '\n';
        }
    }
};
```

9.4 01 矩阵

```
#include <bitset>
#define maxn 1000
struct Matrix01{
```

```
    int n,m;
    std::bitset<maxn> a[maxn];
    void Resize(int x,int y){
        n=x;
        m=y;
    }
    std::bitset<maxn>& operator [] (int n) {
        return a[n];
    }
    void print(){
        for(int i = 0; i < n; ++i)
            std::cout << a[i] << std::endl;
        }
};

Matrix01 operator & (Matrix01 &a,Matrix01 &b){ int i,j,k;
    Matrix01 c;
    c.Resize(a.n,b.m);
    for(i = 0; i < a.n; ++i) {
    c[i].reset();
    for(j = 0; j < b.m; ++j)
        if(a[i][j])
            c[i]|=b[j];
        }
    return c;
}
```

9.5 简单大数

```
struct Date_Analysis {
    int bit[6];
    inline void Clear(){memset(bit,0,sizeof(bit));}
    Date_Analysis(){Clear();}
    inline void Set(int t){Clear(); while(t)bit[++bit[0]]=t%cube, t/=cube; }
    inline int &operator [] (int x){ return bit[x];}
    inline void print() {
        printf("%d",bit[bit[0]]);
        for(int i=bit[0]-1;i>0;i--)printf("%09d",bit[i]);
        printf("\n");
    }
    inline Date_Analysis operator + (Date_Analysis b) {
        Date_Analysis c;c.Clear();
        c[0]=max(bit[0],b[0])+1;
        for(int i=1;i<=c[0];i++)
            c[i]+=b[i]+bit[i],c[i+1]+=c[i]/cube,c[i]%=cube;
        while(!c[c[0]]) c[0]--;
        return c;
    }
    inline void operator += (Date_Analysis b){*this=*this+b;}
    inline void operator = (int x){Set(x);}
};
```

9.6 大数

```cpp
#include <bits/stdc++.h>
using namespace std;
class BigNum {
public:
    static const int maxn = 9999;
    static const int maxsize = 10;
    static const int dlen = 4;
    int a[105];     //可以控制大数的位数
    int len;        //大数长度
    BigNum(){ len = 1;memset(a,0,sizeof(a)); }   //构造函数
    BigNum(const int);        //将一个int类型的变量转化为大数
    BigNum(const char*);      //将一个字符串类型的变量转化为大数
    BigNum(const BigNum &);  //拷贝构造函数
    BigNum &operator=(const BigNum &);   //重载赋值运算符，大数之间进行赋值运算

    BigNum operator+(const BigNum &) const;   //重载加法运算符，两个大数之间的相加运算
    BigNum operator-(const BigNum &) const;   //重载减法运算符，两个大数之间的相减运算
    BigNum operator*(const BigNum &) const;   //重载乘法运算符，两个大数之间的相乘运算
    BigNum operator/(const int   &) const;    //重载除法运算符，大数对一个整数进行相除运算

    BigNum operator^(const int  &) const;    //大数的n次方运算
    int    operator%(const int  &) const;    //大数对一个int类型的变量进行取模运算
    bool   operator>(const BigNum & T)const;   //大数和另一个大数的大小比较
    bool   operator>(const int & t)const;      //大数和一个int类型的变量的大小比较

    void print();       //输出大数

    friend istream&operator >>(istream&in, BigNum &b) {
        char ch[maxsize*4];
        int i = -1;
        in>>ch;
        int l=strlen(ch);
        int count=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[count]=sum;
            count++;
        }
        b.len =count++;
        return in;
    }
    friend ostream& operator<<(ostream& out,  BigNum& b) {  //重载输出运算符
        int i;
        out << b.a[b.len - 1];
        for(i = b.len - 2 ; i >= 0 ; i--) {
```

```
50              out.width(dlen);
51              out.fill('0');
52              out << b.a[i];
53          }
54          return out;
55      }
56
57  };
58
59  BigNum::BigNum(const int b) {     //将一个int类型的变量转化为大数
60      int c,d = b;
61      len = 0;
62      memset(a,0,sizeof(a));
63      while(d > maxn) {
64          c = d - (d / (maxn + 1)) * (maxn + 1);
65          d = d / (maxn + 1);
66          a[len++] = c;
67      }
68      a[len++] = d;
69  }
70
71  BigNum::BigNum(const char*s) {     //将一个字符串类型的变量转化为大数
72      int t,k,index,l,i;
73      memset(a,0,sizeof(a));
74      l=strlen(s);
75      len=l/dlen;
76      if(l%dlen)
77          len++;
78      index=0;
79      for(i=l-1;i>=0;i-=dlen) {
80          t=0;
81          k=i-dlen+1;
82          if(k<0)
83              k=0;
84          for(int j=k;j<=i;j++)
85              t=t*10+s[j]-'0';
86          a[index++]=t;
87      }
88  }
89
90  BigNum::BigNum(const BigNum & T) : len(T.len) { //拷贝构造函数
91      int i;
92      memset(a,0,sizeof(a));
93      for(i = 0 ; i < len ; i++)
94          a[i] = T.a[i];
95  }
96
97  BigNum & BigNum::operator=(const BigNum & n) {  //重载赋值运算符，大数之间进行赋值运算
98      int i;
99      len = n.len;
100     memset(a,0,sizeof(a));
101     for(i = 0 ; i < len ; i++)
102         a[i] = n.a[i];
103     return *this;
```

```
}


BigNum BigNum::operator+(const BigNum & T) const {  //两个大数之间的相加运算
    BigNum t(*this);
    int i,big;      //位数
    big = T.len > len ? T.len : len;
    for(i = 0 ; i < big ; i++) {
        t.a[i] +=T.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 & T) const {   //两个大数之间的相减运算
    int i,j,big;
    bool flag;
    BigNum t1,t2;
    if(*this>T) {
        t1=*this;
        t2=T;
        flag=0;
    } else {
        t1=T;
        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];
```

```
158     return t1;
159 }
160
161 BigNum BigNum::operator*(const BigNum & T) const {  //两个大数之间的相乘运算
162     BigNum ret;
163     int i,j,up;
164     int temp,temp1;
165     for(i = 0 ; i < len ; i++) {
166         up = 0;
167         for(j = 0 ; j < T.len ; j++) {
168             temp = a[i] * T.a[j] + ret.a[i + j] + up;
169             if(temp > maxn) {
170                 temp1 = temp - temp / (maxn + 1) * (maxn + 1);
171                 up = temp / (maxn + 1);
172                 ret.a[i + j] = temp1;
173             } else {
174                 up = 0;
175                 ret.a[i + j] = temp;
176             }
177         }
178         if(up != 0)
179             ret.a[i + j] = up;
180     }
181     ret.len = i + j;
182     while(ret.a[ret.len - 1] == 0 && ret.len > 1)
183         ret.len--;
184     return ret;
185 }
186 BigNum BigNum::operator/(const int & b) const {  //大数对一个整数进行相除运算
187     BigNum ret;
188     int i,down = 0;
189     for(i = len - 1 ; i >= 0 ; i--) {
190         ret.a[i] = (a[i] + down * (maxn + 1)) / b;
191         down = a[i] + down * (maxn + 1) - ret.a[i] * b;
192     }
193     ret.len = len;
194     while(ret.a[ret.len - 1] == 0 && ret.len > 1)
195         ret.len--;
196     return ret;
197 }
198 int BigNum::operator %(const int & b) const {   //大数对一个int类型的变量进行取模运算
199     int i,d=0;
200     for (i = len-1; i>=0; i--) {
201         d = ((d * (maxn+1))% b + a[i])% b;
202     }
203     return d;
204 }
205 BigNum BigNum::operator^(const int & n) const {   //大数的n次方运算
206     BigNum t,ret(1);
207     int i;
208     if(n<0)
209         exit(-1);
210     if(n==0)
211         return 1;
```

```
212     if(n==1)
213         return *this;
214     int m=n;
215     while(m>1) {
216         t=*this;
217         for( i=1;i<<1<=m;i<<=1) {
218             t=t*t;
219         }
220         m-=i;
221         ret=ret*t;
222         if(m==1)
223             ret=ret*(*this);
224     }
225     return ret;
226 }
227
228 bool BigNum::operator>(const BigNum & T) const {  //大数和另一个大数的大小比较
229     int ln;
230     if(len > T.len)
231         return true;
232     else if(len == T.len) {
233         ln = len - 1;
234         while(a[ln] == T.a[ln] && ln >= 0)
235             ln--;
236         return ln >= 0 && a[ln] > T.a[ln];
237     } else
238         return false;
239 }
240
241 bool BigNum::operator >(const int & t) const {  //大数和一个int类型的变量的大小比较
242     BigNum b(t);
243     return *this>b;
244 }
245
246 void BigNum::print() {    //输出大数
247     int i;
248     cout << a[len - 1];
249     for(i = len - 2 ; i >= 0 ; i--) {
250         cout.width(dlen);
251         cout.fill('0');
252         cout << a[i];
253     }
254     cout << endl;
255 }
```

9.7 java 大数

1. valueOf(parament); 将参数转换为制定的类型。如：String s=”12345”;BigInteger c=BigInteger.valueOf(s); 则 c=12345;
2. add(); 大整数相加;
3. subtract(); 相减;

4. multiply(); 相乘;
5. divide(); 相除取整;
6. remainder(); 取余;
7. pow(); a.pow(b)=aƀ;
8. gcd(); 最大公约数;
9. abs(); 绝对值;
10. negate(); 取反数;
11. mod(); a.mod(b)=a%b=a.remainder(b);
12. max(); min();
13. public int comareTo();
14. boolean equals(); 是否相等;
15. BigInteger(String val); 将指定字符串转换为十进制表示形式;
16. BigInteger(String val,int radix); 将指定基数的 BigInteger 的字符串表示形式转换为 BigInteger。

A=BigInteger.ONE 1
B=BigInteger.TEN 10
C=BigInteger.ZERO 0

第十章　技巧与黑科技

10.1　离散化

```cpp
//数组离散化 含重复元素
std::sort(sub_a, sub_a+n);
int size = std::unique(sub_a, sub_a+n) - sub_a;//size为离散化后元素个数
for (i = 0; i < n; i++) {
    a[i] = std::lower_bound(sub_a, sub_a+size, a[i]) - sub_a + 1;//k为b[i]经离散化后对应的值
}

//坐标离散化
int compress(int *x1, int *x2, int w){
    std::vector<int> xs;
    for (int i = 0; i < N; i++) {
      for (int d = -1; d <= 1; d++) {
          int tx1 = x1[i] + d, tx2 = x2[i] + d;
          if (1 <= tx1 && tx1 <= w) xs.push_back(tx1);
          if (1 <= tx2 && tx2 <= w) xs.push_back(tx2);
      }
    }
    std::sort(xs.begin(), xs.end());
    xs.erase(unique(xs.begin(), xs.end()), xs.end());
    for (int i = 0; i < N; i++) {
      x1[i] = find(xs.begin(), xs.end(), x1[i]) - xs.begin();
      x2[i] = find(xs.begin(), xs.end(), x2[i]) - xs.begin();
    }
    return xs.size();
}
```

10.2　快速枚举子集

```cpp
void print_subset(int n, int s) {
    for (int i = 0; i < n; i++) {
        if (s & (1 << i)) {
            std::cout << i << "␣";
        }
        std::cout << '\n';
    }
}
int main(int argc, char *argv[]) {
    int n;
    std::cin >> n;
```

```
    for (int i = 0; i < (1 << n); i++) print_subset(n, i);
}

//当x代表集合 x的子集: for (int i = x; i; i=(i-1)&x) {}
```

10.3 bitset

```
#include <bitset>
b.any()             b中是否存在置为1的二进制位?
b.none()            b中不存在置位1的二进制位吗?
b.count()           b中置为1的二进制位的个数
b.size()            b中二进制位的个数
b[pos]              访问b中在pos处的二进制位
b.test(pos)         b中在pos处的二进制位是否为1?
b.set()             把b中所有二进制位都置为1
b.set(pos)          把b中在pos处的二进制位置为1
b.reset()           把b中所有二进制位置为0
b.reset(pos)        把b中在pos处的二进制位置为0
b.flip()            把b中所有二进制位逐位取反
b.flip(pos)         把b中在pos处的二进制位取反
b.to\_ulong()       用b中同样的二进制位返回一个unsigned long值
os << b             把b中的位集中输出到os流
```

10.4 位运算

```
//去掉最后一位
 x >> 1
//在最后加一个0
x << 1
//在最后加一个1
x << 1 + 1
//把最后一位变成1
 x | 1
//把最后一位变成0
x | 1 - 1
//最后一位取反
 x ^ 1
//把右数第k位变成1
 x | (1 <<(k-1))
//把右数第k位变成0
 x & ~ (1 << (k-1))
//右数第k位取反
x ^ (1 << (k-1))
//取末三位
x & 7
//取末k位
 x & (1 << k-1)
//取右数第k位
 x >>(k-1) & 1
```

```
//把末k位变成1
x |(1 << k-1)
//末k位取反
 x ^ (1 << k-1)
//把右边连续的1变成0
x & (x+1)
//x个1
((1<<x-1)
//二进制里1的数量
(x>>16)+(x&((1<<16)-1))
```

10.5 跳舞链

```
struct DLX{
    const static int maxn=20010;
    #define FF(i,A,s) for(int i = A[s];i != s;i = A[i])
    int L[maxn],R[maxn],U[maxn],D[maxn];
    int size,col[maxn],row[maxn],s[maxn],H[maxn];
    bool vis[70];
    int ans[maxn],cnt;
    void init(int m){
        for(int i=0;i<=m;i++){
            L[i]=i-1;R[i]=i+1;U[i]=D[i]=i;s[i]=0;
        }
        memset(H,-1,sizeof(H));
        L[0]=m;R[m]=0;size=m+1;
    }
    void link(int r,int c){
         U[size]=c;D[size]=D[c];U[D[c]]=size;D[c]=size;
         if(H[r]<0)H[r]=L[size]=R[size]=size;
         else {
             L[size]=H[r];R[size]=R[H[r]];
             L[R[H[r]]]=size;R[H[r]]=size;
         }
         s[c]++;col[size]=c;row[size]=r;size++;
     }
    void del(int c){//精确覆盖
        L[R[c]]=L[c];R[L[c]]=R[c];
        FF(i,D,c)FF(j,R,i)U[D[j]]=U[j],D[U[j]]=D[j],--s[col[j]];
    }
    void add(int c){  //精确覆盖
        R[L[c]]=L[R[c]]=c;
        FF(i,U,c)FF(j,L,i)++s[col[U[D[j]]=D[U[j]]=j]];
    }
    bool dfs(int k){//精确覆盖
        if(!R[0]){
            cnt=k;return 1;
        }
        int c=R[0];FF(i,R,0)if(s[c]>s[i])c=i;
        del(c);
        FF(i,D,c){
            FF(j,R,i)del(col[j]);
```

```
            ans[k]=row[i];if(dfs(k+1))return true;
            FF(j,L,i)add(col[j]);
        }
        add(c);
        return 0;
    }
    void remove(int c){//重复覆盖
        FF(i,D,c)L[R[i]]=L[i],R[L[i]]=R[i];
    }
     void resume(int c){//重复覆盖
         FF(i,U,c)L[R[i]]=R[L[i]]=i;
     }
    int A(){//估价函数
        int res=0;
        memset(vis,0,sizeof(vis));
        FF(i,R,0)if(!vis[i]){
                res++;vis[i]=1;
                FF(j,D,i)FF(k,R,j)vis[col[k]]=1;
            }
        return res;
    }
    void dfs(int now,int &ans){//重复覆盖
        if(R[0]==0)ans=min(ans,now);
        else if(now+A()<ans){
            int temp=INF,c;
            FF(i,R,0)if(temp>s[i])temp=s[i],c=i;
            FF(i,D,c){
                remove(i);FF(j,R,i)remove(j);
                dfs(now+1,ans);
                FF(j,L,i)resume(j);resume(i);
            }
        }
    }
}dlx;
```

10.6 K-D 树

```
#include <queue>
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int N = 55555, K = 5;
const int inf = 0x3f3f3f3f;

#define sqr(x) (x)*(x)
int k,n,idx;   //k为维数,n为点数
struct point {
    int x[K];
    bool operator < (const point &u) const {
     return x[idx]<u.x[idx];
    }
```

```
}po[N];

typedef pair<double,point>tp;
priority_queue<tp>nq;

struct kdTree {
    point pt[N<<2];
    int son[N<<2];

    void build(int l,int r,int rt=1,int dep=0) {
        if(l>r) return;
        son[rt]=r-l;
        son[rt*2]=son[rt*2+1]=-1;
        idx=dep%k;
        int mid=(l+r)/2;
        nth_element(po+l,po+mid,po+r+1);
        pt[rt]=po[mid];
        build(l,mid-1,rt*2,dep+1);
        build(mid+1,r,rt*2+1,dep+1);
    }
    void query(point p,int m,int rt=1,int dep=0) {
        if(son[rt]==-1) return;
        tp nd(0,pt[rt]);
        for(int i=0;i<k;i++) nd.first+=sqr(nd.second.x[i]-p.x[i]);
        int dim=dep%k,x=rt*2,y=rt*2+1,fg=0;
        if(p.x[dim]>=pt[rt].x[dim]) swap(x,y);
        if(~son[x]) query(p,m,x,dep+1);
        if(nq.size()<m) nq.push(nd),fg=1;
        else {
            if(nd.first<nq.top().first) nq.pop(),nq.push(nd);
            if(sqr(p.x[dim]-pt[rt].x[dim])<nq.top().first) fg=1;
        }
        if(~son[y]&&fg) query(p,m,y,dep+1);
    }
}kd;

void print(point &p) {
    for(int j=0;j<k;j++) printf("%d%c",p.x[j],j==k-1?'\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",&po[i].x[j]);
        kd.build(0,n-1);
        int t,m;
        for(scanf("%d",&t);t--;) {
             point ask;
             for(int j=0;j<k;j++) scanf("%d",&ask.x[j]);
             scanf("%d",&m); kd.query(ask,m);
             printf("the closest %d points are:\n", m);
             point pt[20];
             for(int j=0;!nq.empty();j++) pt[j]=nq.top().second,nq.pop();
             for(int j=m-1;j>=0;j--) print(pt[j]);
        }
```

```
    }
    return 0;
}
```

10.7 随机

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

std::vector<int> permutation(100);
for (int i = 0; i < 100; i++) {
    permutation[i] = i+1;
}
std::mt19937_64 mt1(1); //64位
std::mt19937 mt2(2); //32位
shuffle(permutation.begin(), permutation.end(), mt2); // 打乱序列
for (auto it: permutation) {
    std::cout << it << "␣";
}
```

10.8 珂朵莉树（Old Driver Tree）

```
#include <set>
#include <algorithm>

using ll = long long;

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

std::set<node> s;

//分割SET 返回一个pos位置的迭代器
std::set<node>::iterator split(int pos) {
    auto it = s.lower_bound(node(pos));
    if (it != s.end() && it->l == pos) return it;
    --it;
    if (pos > it->r) return s.end();
    int L = it->l, R = it->r;
    ll V = it->v;
    s.erase(it);
    s.insert(node(L, pos - 1, V));
    return s.insert(node(pos, R, V)).first;
```

```
}

//区间加值
void add(int l, int r, ll val=1) {
    split(l);
    auto itr = split(r+1), itl = split(l);
    for (; itl != itr; ++itl) itl->v += val;
}

//区间赋值
void assign(int l, int r, ll val = 0) {
    split(l);
    auto itr = split(r+1), itl = split(l);
    s.erase(itl, itr);
    s.insert(node(l, r, val));
}
```

10.9 CDQ 分治

```
//Author:marsed
/*
*将区间分成左右两部分 递归处理
一层递归计算当前左区间的修改操作对右区间的查询操作的影响
当flag为1代表修改操作 为0代表查询操作
*/
#include <algorithm>
#define mid (l + r)/2

const int maxn = "Edit";

struct Node {
    int id, x1,x2;
    int operator<(const Node &b) {   //按照参数的优先级排序
        return ;
    }
};

Node nod[maxn], tmp[maxn];

void cdq(int l, int r) {
    if (l == r) return;
    cdq(l, mid); cdq(mid + 1, r);
    int p = l, q = mid + 1, cnt = 0;
    while (p <= mid&&q <= r) {
        if (nod[p] < nod[q]) {
            if (nod[p].flag) ;    //左区间里的修改操作会对右区间的查询操作有影响 计算影响
            tmp[cnt++] = nod[p++];
        } else {
            if (!nod[q].flag) ;//计算右区间的查询操作的值
            tmp[cnt++] = nod[q++];
        }
    }
```

```
    while (p <= mid) tmp[cnt++] = nod[p++];
    while (q <= r) {
        if (!nod[q].flag) ;
        tmp[cnt++] = nod[q++];
    }
    for (int i = l; i <= r; i++)
        nod[i] = tmp[i - l];
}

int main()
{
    cdq(1, q);
    return 0;
}
```

10.10 内置位运算函数

```
— Built-in Function: int __builtin_ffs (unsigned int x)
Returns one plus the index of the least significant 1-bit of x, or if x is zero, returns zero.
返回右起第一个‘1’的位置。

— Built-in Function: int __builtin_clz (unsigned int x)
Returns the number of leading 0-bits in x, starting at the most significant bit position. If x
    is 0, the result is undefined.
返回左起第一个‘1’之前0的个数。

— Built-in Function: int __builtin_ctz (unsigned int x)
Returns the number of trailing 0-bits in x, starting at the least significant bit position. If
    x is 0, the result is undefined.
返回右起第一个‘1’之后的0的个数。

— Built-in Function: int __builtin_popcount (unsigned int x)
Returns the number of 1-bits in x.
返回‘1’的个数。

— Built-in Function: int __builtin_parity (unsigned int x)
Returns the parity of x, i.e. the number of 1-bits in x modulo 2.
返回‘1’的个数的奇偶性。

— Built-in Function: int __builtin_ffsl (unsigned long)
Similar to __builtin_ffs, except the argument type is unsigned long.

— Built-in Function: int __builtin_clzl (unsigned long)
Similar to __builtin_clz, except the argument type is unsigned long.

— Built-in Function: int __builtin_ctzl (unsigned long)
Similar to __builtin_ctz, except the argument type is unsigned long.

— Built-in Function: int __builtin_popcountl (unsigned long)
Similar to __builtin_popcount, except the argument type is unsigned long.

— Built-in Function: int __builtin_parityl (unsigned long)
```

```
Similar to __builtin_parity, except the argument type is unsigned long.

— Built-in Function: int __builtin_ffsll (unsigned long long)
Similar to __builtin_ffs, except the argument type is unsigned long long.

— Built-in Function: int __builtin_clzll (unsigned long long)
Similar to __builtin_clz, except the argument type is unsigned long long.

— Built-in Function: int __builtin_ctzll (unsigned long long)
Similar to __builtin_ctz, except the argument type is unsigned long long.

— Built-in Function: int __builtin_popcountll (unsigned long long)
Similar to __builtin_popcount, except the argument type is unsigned long long.

— Built-in Function: int __builtin_parityll (unsigned long long)
Similar to __builtin_parity, except the argument type is unsigned long long.
```

10.11 0-1 分数规划

```cpp
template <size_t N, typename T, typename Z = double>
struct zero_one_plan {
    Z f[N];
    Z solve(T *c, T *s, int n, int k) { // max-> sigma(c[i])/sigma(s[i])
        Z l=0,r=*max_element(c,c+n);
        while(fabs(r-l)>eps){
            Z mid=(l+r)/2.;
            rep(i,0,n)f[i]=1.*c[i]-mid*s[i];
            nth_element(f,f+k,f+n,greater<Z>());
            Z sm=0;
            rep(i,0,k)sm+=f[i];
            if(sm>-eps)l=mid;
            else r=mid;
        }
        return l;
    }
};
```

10.12 BM 线性递推

```cpp
//author: xudyh

namespace linear_seq {
    const int N = 10010;
    typedef long long ll;
    constexpr ll mod = (ll) 1e9 + 7;

    ll pow_mod(ll a, ll b) {
        ll r = 1;
        for (a %= mod; b; b >>= 1, a = a * a % mod) {
```

```
11              if (b & 1)r = r * a % mod;
12          }
13          return r;
14      }
15
16      ll res[N], base[N], _c[N], _md[N];
17      vector<int> Md;
18
19      void mul(ll *a, ll *b, int k) {
20          k <<= 1;
21          for (int i = 0; i < k; ++i) _c[i] = 0;
22          k >>= 1;
23          for (int i = 0; i < k; ++i) {
24              if (a[i]) {
25                  for (int j = 0; j < k; ++j) {
26                      _c[i + j] = (_c[i + j] + a[i] * b[j]) % mod;
27                  }
28              }
29          }
30          for (int i = k + k - 1; i >= k; i--) {
31              if (_c[i]) {
32                  for (const int md: Md) {
33                      _c[i - k + md] = (_c[i - k + md] - _c[i] * _md[md]) % mod;
34                  }
35              }
36          }
37          for (int i = 0; i < k; ++i) {
38              a[i] = _c[i];
39          }
40      }
41
42      int solve(ll n, vector<int> a, vector<int> b) { // a 系数 b 初值 b[n+1]=a[0]*b[n]+...
43  //        printf("SIZE %d\n",SZ(b));
44          ll ans = 0, pnt = 0;
45          int k = (int) a.size();
46          assert(a.size() == b.size());
47          for (int i = 0; i < k; ++i) {
48              _md[k - 1 - i] = -a[i];
49          }
50          _md[k] = 1;
51          Md.clear();
52          for (int i = 0; i < k; ++i) {
53              if (_md[i] != 0) {
54                  Md.push_back(i);
55              }
56          }
57          for (int i = 0; i < k; ++i) {
58              res[i] = base[i] = 0;
59          }
60          res[0] = 1;
61          while ((1ll << pnt) <= n) {
62              pnt++;
63          }
64          for (int p = pnt; p >= 0; p--) {
```

```
65              mul(res, res, k);
66              if ((n >> p) & 1) {
67                  for (int i = k - 1; i >= 0; i--) {
68                      res[i + 1] = res[i];
69                  }
70                  res[0] = 0;
71                  for (const int md: Md) {
72                      res[md] = (res[md] - res[k] * _md[md]) % mod;
73                  }
74              }
75          }
76          for (int i = 0; i < k; ++i) {
77              ans = (ans + res[i] * b[i]) % mod;
78          }
79          if (ans < 0) ans += mod;
80          return ans;
81      }
82
83      vector<int> BM(vector<int> s) {
84          vector<int> C(1, 1), B(1, 1);
85          int L = 0, m = 1, b = 1;
86          for (int n = 0; n < (int) s.size(); ++n) {
87              ll d = 0;
88              for (int i = 0; i <= L; ++i) {
89                  d = (d + (ll) C[i] * s[n - i]) % mod;
90              }
91              if (d == 0) {
92                  ++m;
93              }
94              else if (2 * L <= n) {
95                  vector<int> T = C;
96                  ll c = mod - d * pow_mod(b, mod - 2) % mod;
97                  while (C.size() < B.size() + m) {
98                      C.push_back(0);
99                  }
100                 for (int i = 0; i < (int) B.size(); ++i) {
101                     C[i + m] = (C[i + m] + c * B[i]) % mod;
102                 }
103                 L = n + 1 - L;
104                 B = T;
105                 b = d;
106                 m = 1;
107             } else {
108                 ll c = mod - d * pow_mod(b, mod - 2) % mod;
109                 while (C.size() < B.size() + m) {
110                     C.push_back(0);
111                 }
112                 for (int i = 0; i < (int) B.size(); ++i) {
113                     C[i + m] = (C[i + m] + c * B[i]) % mod;
114                 }
115                 ++m;
116             }
117         }
118         return C;
```

```
119     }
120
121     int gao(vector<int> a, ll n) {
122         vector<int> c = BM(a);
123         c.erase(c.begin());
124         for (int &x:c) {
125             x = (mod - x) % mod;
126         }
127         return solve(n, c, vector<int>(a.begin(), a.begin() + c.size()));
128     }
129 }
```