Nantong University ICPC Team Notebook (2018-19)

thirtiseven wanggann Simon

Oct 7 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

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

1.7 python 输入

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

1.8 int128 输入输出

```cpp
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

```cpp
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

```cpp
//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 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));
    }
}
```

第三章　数学

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 暴力判回文数

```
bool is_palindrome(int bob) {
    int clare = bob, dave = 0;
    while (clare){
        dave = dave * 10 + clare % 10;
        clare /= 10;
    }
    if(bob == dave) {
        return true;
    } else {
        return false;
    }
}
```

3.7 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;
}
```

3.8 最小公倍数

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

3.9 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;
}
```

3.10 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;
}
```

3.11 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 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;
}
```

3.14 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;
}
```

3.15 快速乘法取模

```cpp
//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 ;
}
```

3.16 快速幂取模

```cpp
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;
}
```

3.17 计算从 C(n, 0) 到 C(n, p) 的值

```cpp
//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;
}
```

3.18 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] );
//}
```

3.19 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;
}
```

3.20 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();
    }
    inline void insert(ll k,ll v) {
        ll idx = k % mod;
        for(ll i = head[idx]; ~i; i = nxt[i]) {
            if(key[i] == k) {
                value[i] = std::min(value[i], v);
                return ;
            }
        }
        key[tol] = k;
        value[tol] = v;
        nxt[tol] = head[idx];
        head[idx] = tol++;
    }
    inline ll operator [](const ll &k) const {
        ll idx = k % mod;
        for(ll i=head[idx]; ~i; i=nxt[i]) {
            if(key[i]==k) return value[i];
        }
        return -1;
    }
}mp;
```

```

inline ll fpow(ll a, ll b, ll mod) {
    a %= mod;
    ll ans = 1;
    while (b) {
        if(b&1) ans = ans * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return ans;
}
inline ll exgcd(ll a,ll b,ll &x,ll &y) {
    if (b==0) {
        x=1, y=0;
        return a;
    }
    ll ans = exgcd(b, a%b, y, x);
    y -= a/b*x;
    return ans;
}

inline ll Bsgs(ll a,ll b,ll mod) {
    a %= mod, b %= mod;
    if (b==1) return 0;
    ll m = ceil(sqrt(mod)), inv, y;
    exgcd(fpow(a, m, mod), mod, inv, y);
    inv = (inv % mod + mod) % mod;
    mp.insert(1, 1);
    for(ll i=1, e=1; i<m; i++) {
        e = e * a % mod;
        if(mp[e] == -1) mp.insert(e, i+1);
    }
    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);
}
```

3.21 二分分数树（Stern-Brocot Tree)

```cpp
//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);
}
```

3.22 计算莫比乌斯函数

```cpp
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;
        }
    }
}
```

3.23 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;
}
```

3.24 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。
```

3.25 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;
    }
};
```

3.26 java 大数开方

```java
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;
        }
    }
}
```

3.27 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));
        for (type i = 0; i < need; i++) {
            long long aa = fa[i].x + 0.5;
            long long bb = fb[i].x + 0.5;
            long long cc = fa[i].y + 0.5;
            res[i] = (aa + ((bb % m) << 15) + ((cc % m) << 30)) % m;
        }
        return res;
    }
    vector<type> square(vector<type> &a) {
        return multiply(a, a);
    }
    vector<type> square_mod(vector<type> &a, type m) {
        return multiply_mod(a, a, m, 1);
    }
    vector<type> kiss_me(vector<type>&b, long long k, type mod) {
        vector<type> a = b;
        vector<type> res(1, 1);
        for (; k; k >>= 1, a = square_mod(a, mod)) {
            if (k & 1) {
                res = multiply_mod(res, a, mod);
            }
        }
```

```cpp
        return res;
    }
    pair<vector<type>, vector<type> > mul2(vector<type>&b, long long k) {
        return make_pair(kiss_me(b, k, (type)1e9 + 7), kiss_me(b, k, (type)1e9 + 9));
    }
    vector<vector<type> > muln(vector<type>&b, long long k, vector<int> mod_list) {
        vector< vector<type> > res(mod_list.size());
        for (int i = 0; i < mod_list.size(); ++i) {
            res[i] = kiss_me(b, k, mod_list[i]);
        }
        return res;
    }
};
```

3.28 快速傅里叶变换

```cpp
const double PI = acos(-1.0);
//复数结构体
struct Complex {
    double x, y; //实部和虚部 x+yi
    Complex(double _x = 0.0, double _y = 0.0) { x = _x, y = _y; }
    Complex operator-(const Complex& b) const { return Complex(x - b.x, y - b.y); }
    Complex operator+(const Complex& b) const { return Complex(x + b.x, y + b.y); }
    Complex operator*(const Complex& b) const { return Complex(x * b.x - y * b.y, x * b.y + y
        * b.x); }
};
/*
* 进行FFT和IFFT前的反转变换。
* 位置i和 (i二进制反转后位置) 互换
* len必须取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;
}
```

3.29 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;
    }
}
```

3.30 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];
            }
}
```

3.31 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);
}
```

3.32 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

3.33 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 $\sum_{i=1}^{n} i \sum_{j=1}^{n} [gcd(i,j)=1] \cdot j = 2\sum_{i=1}^{n} i \sum_{j=1}^{i} [gcd(i,j) = 1] \cdot j - \sum_{i=1}^{n} [gcd(i,i)=1] \cdot i$
$= \left(2\sum_{i=1}^{n} i \frac{i \cdot \varphi(i) + [i=1]}{2}\right) - 1 = \left(\sum_{i=1}^{n} i^2 \cdot \varphi(i) + [i=1]\right) - 1$

3.34 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)
```

第四章 图论

4.1 前向星

```
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';
        }
    }
}
```

4.2 并查集

```
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]);
}
```

```cpp

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

4.3 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;
```

4.4 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;
    }
};
```

4.5 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;
}
```

4.6 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;
                }
            }
        }
    }
}
```

4.7 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]));
                    }
            }
        }
    }
};
```

4.8 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]);
        }
    }
}
```

4.9 Dinic 最大流

```cpp
#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;
    }
};
```

4.10 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;
        }
    };
```

4.11 tarjan 强连通分量

```
//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];

```

```cpp
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]]++;
            }
        }
    }
}
```

4.12 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);
            }
        }
    }
};
```

4.13 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];
```

```cpp
    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);
            }
    }
};
```

4.14 边双联通分量

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

4.15 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);
```

```cpp
            std::fill(low, low + n + 1, 0);
            for (int i = 1; i <= n; ++i) {
                if (!dfn[i])
                    dfs(i, 0);
            }
        }
    };
```

4.16 欧拉回路

```cpp
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() {

}
```

4.17 二分图最大匹配匈牙利算法

```
int n, m;
int g[maxn][maxn];  //0-labeled
int linker[maxn];
bool used[maxn];

bool dfs(int u) {
    int v;
    for(v = 0; v < n; v++) {
        if(g[u][v] && !used[v]) {
            used[v] = true;
            if(linker[v] == -1 || dfs(linker[v])) {
                linker[v] = u;
                return true;
            }
        }
    }
    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;
}
```

4.18 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;
}
```

4.19 k 短路

```
#include <cstdio>
#include <cstring>
#include <queue>
```

```cpp
#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);
        if (dist[s] == INF) {
            printf("No␣Solution\n");
            continue;
        }
        if (s == t) k++;
        ans = astar(s);
    }
    return 0;
}
```

4.20 最小环

```
int min=INT_MAX;

for(k=1;k<=n;k++) {
    for(i=1;i<=n;i++) {
        for(j=1;j<=n;++j) {
            if(dist[i][j]!=INF&&map[j][k]!=INF&&map[k][i]!=INF&&dist[i][j]+dist[j][k]+map[k][i
               ]<mindis) {
                mindis=min(mindis,dist[i][j]+map[j][k]+map[k][i]);
            }
        }
    }
    for(i=1;i<=n;i++) {
        for(j=1;j<=n;j++) {
            if(dist[i][k]!=INF&&dist[k][j]!=INF&&dist[i][k]+dist[k][j]<dist[i][j]) {
                dist[i][j]=dist[i][k]+dist[k][j];
                pre[i][j]=pre[k][j];
            }
        }
    }
}
```

4.21 最小树形图

```
#include <cstdio>
#include <cmath>
#define type int

type c[mm], in[mn], w[mn], ans;
int s[mm], t[mm], id[mn], pre[mn], q[mn], vis[mn];

type Directed_MST(int root,int NV,int NE) {
    type ret=0, sum=0, tmp;
    int i, j, u, v, r;
    bool huan=1;
    for (i=0;i<=NV;++i) in[i]=0, id[i]=i, pre[i]=-1;
    while (huan) {
        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;
}
```

4.22 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;
}
```

4.23 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;
}
```

4.24 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);
    }
}
```

4.25 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;
        }
    }
```

```cpp
    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;
}
```

4.26 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);
}
```

4.27 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;
```

```cpp
    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;
}
```

第五章　数据结构

5.1　树状数组

```cpp
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;
}
```

5.2　差分数组

```cpp
//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);
}
```

5.3 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);
}
```

5.4 堆

```cpp
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;
        }
};
```

5.5 RMQ

```cpp
//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]);
}
```

5.6 RMQ

```cpp
//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)]);
    }
};
```

5.7 线段树

```cpp
//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];
        sum[rt << 1 | 1] += (r - mid)*Add[rt];
        Add[rt] = 0;
    }
}

void build(int l, int r, int rt) {
    if (l == r) {
        sum[rt] = A[l];
        return;
    }
    int mid = (l + r) >> 1;
    build(l, mid, rt << 1);
    build(mid + 1, r, rt << 1 | 1);
    pushup(rt);
}

//区间加值
void update(int L, int R, int val, int l, int r, int rt) {
    if (L <= l && R >= r) {
        Add[rt] += val;
        sum[rt] += (r - l + 1)*val;
        return;
    }
    pushdown(rt, l, r);
    int mid = (l + r) >> 1;
    if (L <= mid)update(L, R, val, l, mid, rt << 1);
    if (R > mid)update(L, R, val, mid + 1, r, rt << 1 | 1);
    pushup(rt);
}

//点修改
void update(int index, int val, int l, int r, int rt) {
    if (l == r) {
        sum[rt] = val;
        return;
    }
    int mid = (l + r) >> 1;
    if (index <= mid)update(index, val, l, mid, rt << 1);
    else update(index, val, mid + 1, r, rt << 1 | 1);
    pushup(rt);
}

//区间查询
int query(int L, int R, int l, int r, int rt) {
    if (L <= l && R >= r) {
        return sum[rt];
    }
    pushdown(rt, l, r);
    int mid = (l + r) >> 1;
    int ret = 0;
    if (L <= mid)ret += query(L, R, l, mid, rt << 1);
    if (R > mid)ret += query(L, R, mid + 1, r, rt << 1 | 1);
    return ret;
}
```

5.8 ZKW 线段树

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

```

```cpp
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;
}
```

5.9 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]) {
        sum[x]=(ll)(r-l+1)*t; cx[x] = cn[x] = r-l+1; sx[x] = -(1<<30); sn[x] = 1<<30;
    } else {
        sx[x] = std::max(sx[x], t);
    }
}

inline void push(int x, int l, int r){
    int mid = (l+r)>>1;
    if (ta[x]) {
        _add(x<<1, l, mid, ta[x]);
        _add(x<<1|1, mid+1, r, ta[x]);
        ta[x] = 0;
    }
    if (mx[x<<1]>mx[x] && sx[x<<1]<mx[x]) _min(x<<1, l, mid, mx[x]);
    if (mx[x<<1|1]>mx[x] && sx[x<<1|1]<mx[x]) _min(x<<1|1, mid+1, r, mx[x]);
    if (mn[x<<1]<mn[x] && sn[x<<1]>mn[x]) _max(x<<1, l, mid, mn[x]);
    if (mn[x<<1|1]<mn[x] && sn[x<<1|1]>mn[x]) _max(x<<1|1, mid+1, r, mn[x]);
}

int ql, qr, qt;
int n;

//把一个区间[L,R] 里小于x 的数变成x
inline void Mmax(int x, int l, int r){
    if (mn[x] >= qt) return;
    if (ql<=l && r<=qr && qt<sn[x]){
        _max(x, l, r, qt); return;
    }
    push(x, l, r); int mid = (l+r)>>1;
    if (ql<=mid) Mmax(x<<1, l, mid);
    if (qr>mid) Mmax(x<<1|1, mid+1, r);
    update(x);
}

//把一个区间[L,R] 里大于x 的数变成x
inline void Mmin(int x,int l,int r) {
    if (mx[x]<=qt) return;
    if (ql<=l && r<=qr && qt>sx[x]) {
        _min(x,l,r,qt); return;
    }
    push(x,l,r); int mid=(l+r)>>1;
    if (ql<=mid) Mmin(x<<1, l, mid);
    if (qr>mid) Mmin(x<<1|1, mid+1, r);
    update(x);
}

//区间加值
inline void Add(int x, int l, int r) {
    if (ql<=l && r<=qr) {
        _add(x, l, r, qt); return;
    }
    push(x, l, r); int mid=(l+r)>>1;
    if (ql<=mid) Add(x<<1, l, mid);
    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;
```

```cpp
    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;
}
```

5.10 扫描线

```cpp
// 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];

void pushdown(int rt, int l, int r) {
    int mid = (l + r) >> 1;
    if (Cover[rt] != -1) {
        Cover[rt << 1] = Cover[rt << 1 | 1] = Cover[rt];
        Sum[rt << 1] = (Cover[rt] > FLAG ? (A[mid + 1] - A[l]) : 0);
        Sum[rt << 1 | 1] = (Cover[rt] > FLAG ? (A[r + 1] - A[mid + 1]) : 0);
    }
}
```

```

void pushup(int rt, int l, int r) {
    if (Cover[rt << 1] == -1 || Cover[rt << 1 | 1] == -1) Cover[rt] = -1;
    else if (Cover[rt << 1] != Cover[rt << 1 | 1]) Cover[rt] = -1;
    else Cover[rt] = Cover[rt << 1];
    Sum[rt] = Sum[rt << 1] + Sum[rt << 1 | 1];
}

void build(int l, int r, int rt) {
    if (l == r) {
        Cover[rt] = 0;
        Sum[rt] = 0;
        return;
    }
    int mid = (l + r) >> 1;
    build(l, mid, rt << 1);
    build(mid + 1, r, rt << 1 | 1);
    pushup(rt, l, r);
}

void update(int L, int R, int v, int l, int r, int rt) {
    if (L <= l && r <= R) {
        if (Cover[rt] != -1) {
            Cover[rt] += v;
            Sum[rt] = (Cover[rt] > FLAG ? (A[r + 1] - A[l]) : 0);
            return;
        }
    }
    pushdown(rt, l, r);
    int mid = (l + r) >> 1;
    if (L <= mid) update(L, R, v, l, mid, rt << 1);
    if (mid < R) update(L, R, v, mid + 1, r, rt << 1 | 1);
    pushup(rt, l, r);
}

int find(double key, int n, double d[]) {
    int l = 1, r = n;
    while (r >= l) {
        int mid = (r + l) >> 1;
        if (d[mid] == key) return mid;
        else if (d[mid] > key) r = mid - 1;
        else l = mid + 1;
    }
    return -1;
}

int init(int n) {
    int N = 0;
    for (int i = 1; i <= n; i++) {
        A[++N] = X1[i];
        line[N] = Line(X1[i], X2[i], Y1[i], 1);
        A[++N] = X2[i];
        line[N] = Line(X1[i], X2[i], Y2[i], -1);
    }
    sort(A + 1, A + N + 1);
    sort(line + 1, line + N + 1);
    int k = 1;
    for (int i = 2; i <= N; i++)
        if (A[i] != A[i - 1])
            A[++k] = A[i];
    build(1, k - 1, 1);
    return k;
}

double query(int n, int k) {
    double ret = 0;
    for (int i = 1; i < n; i++) {
        int l = find(line[i].l, k, A);
        int r = find(line[i].r, k, A) - 1;
        if (l <= r) update(l, r, line[i].d, 1, k - 1, 1);
        ret += Sum[1] * (line[i + 1].h - line[i].h);
    }
    return ret;
}
/*
int main()
{
    int n, T;
    scanf("%d", &T);
    while (T--) {
        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("%.2lf\n", ans);
    }
}
*/

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

// 矩形周长并
int Sum[maxn << 3], cnt[maxn << 3], vNum[maxn << 3];
bool lbd[maxn << 3], rbd[maxn << 3];
double X1[maxn << 1], X2[maxn << 1], Y1[maxn << 1], Y2[maxn << 1];
double A[maxn << 1];

struct Line {
    double l, r, h;
    int label;
    Line() {}
    Line(double l, double r, double h, int label) :l(l), r(r), h(h), label(label) {}
    bool operator < (const Line L) const {
        return h < L.h;
    }
}line[maxn << 1];

```

```
void pushup(int l, int r, int rt) {
    if (cnt[rt]) {
        lbd[rt] = rbd[rt] = true;
        Sum[rt] = A[r + 1] - A[l];
        vNum[rt] = 2;
    }
    else if (l == r) Sum[rt] = vNum[rt] = lbd[rt] = rbd[rt] = 0;
    else {
        lbd[rt] = lbd[rt << 1];
        rbd[rt] = rbd[rt << 1 | 1];
        Sum[rt] = Sum[rt << 1] + Sum[rt << 1 | 1];
        vNum[rt] = vNum[rt << 1] + vNum[rt << 1 | 1];
        if (rbd[rt << 1] && lbd[rt << 1 | 1]) vNum[rt] -= 2;
    }
}

void update(int L, int R, int v, int l, int r, int rt) {
    if (L <= l && r <= R) {
        cnt[rt] += v;
        pushup(l, r, rt);
        return;
    }
    int mid = (l + r) >> 1;
    if (L <= mid) update(L, R, v, l, mid, rt << 1);
    if (R > mid) update(L, R, v, mid + 1, r, rt << 1 | 1);
    pushup(l, r, rt);
}

int find(double key, int n, double d[]) {
    int l = 1, r = n;
    while (r >= l) {
        int mid = (r + l) >> 1;
        if (d[mid] == key) return mid;
        else if (d[mid] > key) r = mid - 1;
        else l = mid + 1;
    }
    return -1;
}

int init(int n) {
    for (int i = 1; i <= n; i++) {
        A[i] = X1[i]; A[i + n] = X2[i];
        line[i].l = X1[i]; line[i].r = X2[i];
        line[i].h = Y1[i]; line[i].label = 1;
        line[i + n].l = X1[i]; line[i + n].r = X2[i];
        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;
}
*/
```

5.11 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);
```

```cpp
            }
            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));
}
```

5.12 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;
}
```

5.13 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];
}
```

5.14 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;
        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;
    int l = 0, r = m - 1;
    Cnt[newroot] = Cnt[root] + val;
    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;
}

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

int sum(int x) {
    int ret = 0;
    while (x > 0) {
        ret += Cnt[lson[use[x]]];
        x -= lowbit(x);
    }
    return ret;
}

void Modify(int x, int pos, int val) {
    while (x <= n) {
        Ntree[x] = update(Ntree[x], pos, val);
        x += lowbit(x);
    }
}

int Query(int left, int right, int k) {
    int lrt = tree[left - 1];
    int rrt = tree[right];
    int l = 0, r = m - 1;
    for (int i = left - 1; i; i -= lowbit(i)) use[i] = Ntree[i];
    for (int i = right; i; i -= lowbit(i)) use[i] = Ntree[i];
    while (l < r) {
        int mid = (l + r) >> 1;
        // sum(right) - sum(left - 1)为改变量, Cnt[lson[rrt]] - Cnt[lson[lrt]]为基础差值
        int tmp = sum(right) - sum(left - 1) + Cnt[lson[rrt]] - Cnt[lson[lrt]];
        if (tmp >= k) {
            r = mid;
            for (int i = left - 1; i; i -= lowbit(i))
                use[i] = lson[use[i]];
            for (int i = right; i; i -= lowbit(i))
                use[i] = lson[use[i]];
            lrt = lson[lrt];
            rrt = lson[rrt];
        }
        else {
            l = mid + 1;
            k -= tmp;
            for (int i = left - 1; i; i -= lowbit(i))
                use[i] = rson[use[i]];
            for (int i = right; i; i -= lowbit(i))
                use[i] = rson[use[i]];
            lrt = rson[lrt];
            rrt = rson[rrt];
        }
    }
    return l;
}

int main()
{
    int Tcase;
    char op[10];
    scanf("%d", &Tcase);
    while (Tcase--) {
        scanf("%d%d", &n, &q);
        tot = 0; m = 0;
        for (int i = 1; i <= n; i++) {
            scanf("%d", &A[i]);
            T[m++] = A[i];
        }
        for (int i = 0; i < q; i++) {
            scanf("%s", op);
            if (op[0] == 'Q') {
                query[i].kind = 0;
                scanf("%d%d%d", &query[i].l, &query[i].r, &query[i].k);
            }
            else {
                query[i].kind = 1;
                scanf("%d%d", &query[i].l, &query[i].r);
                T[m++] = query[i].r;
            }
        }
        Init_hash(m);
        tree[0] = build(0, m - 1);
        for (int i = 1; i <= n; i++)
            tree[i] = update(tree[i - 1], Hash(A[i]), 1);
        for (int i = 1; i <= n; i++) Ntree[i] = tree[0];
        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;
}
```

5.15 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);
    if (a->priority > a->child[dir]->priority) a = rotate(a, dir);
    return a;
}

node *erase(node *a, int index) {
    assert(a != nullptr);
    int middle = size(a->child[LEFT]);
    if (index == middle) {
        if (a->child[LEFT] == nullptr && a->child[RIGHT] == nullptr) {
            delete a;
            return nullptr;
        } else if (a->child[LEFT] == nullptr) a = rotate(a, RIGHT);
        else if (a->child[RIGHT] == nullptr) a = rotate(a, LEFT);
        else a = rotate(a, a->child[LEFT]->priority < a->child[RIGHT]->priority);
        a = erase(a, index);
    } else {
        bool dir = index > middle;
        if (!dir) a->child[LEFT] = erase(a->child[LEFT], index);
        else      a->child[RIGHT] = erase(a->child[RIGHT], index - middle - 1);
    }
    update(a);
    return a;
}

void modify(node *a, int index, const value &x) {
    assert(a != nullptr);
    int middle = size(a->child[LEFT]);
    if (index == middle) a->x = x;
    else {
        bool dir = index > middle;
        if (!dir) modify(a->child[LEFT], index, x);
        else      modify(a->child[RIGHT], index - middle - 1, x);
    }
    update(a);
}

value query(node *a, int l, int r) {
    assert(a != nullptr);
    if (l <= 0 && size(a) - 1 <= r) return a->subtree;
    int middle = size(a->child[LEFT]);
    if (r < middle) return query(a->child[LEFT], l, r);
    if (middle < l) return query(a->child[RIGHT], l - middle - 1, r - middle - 1);
    value res = a->x;
    if (l < middle && a->child[LEFT] != nullptr)
        res = query(a->child[LEFT], l, r) + res;
    if (middle < r && a->child[RIGHT] != nullptr)
        res = res + query(a->child[RIGHT], l - middle - 1, r - middle - 1);
    return res;
}
```

5.16 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;
    }
};
```

5.17 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;
```

```cpp
}

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

void clr(){
    for(int i = 1; i <= n; i ++) {
        vis[i] = f[i] = size[i] = head[i] = dd[i] = 0;
    }
    cnt = rt = 0, f[0] = 1e9, con = n, len = ans = 0;
}

void getrt(int u, int fafa){
    size[u] = 1;
    f[u] = 0;
    for(int i = head[u]; i; i = g[i].nxt){
        int v = g[i].to; if(v == fafa || vis[v]) continue;
        getrt(v, u);
        size[u] += size[v];
        f[u] = std::max(f[u], size[v]);
    }
    f[u] = std::max(f[u], con - size[u]);
    if(f[u] < f[rt]) {
        rt = u;
    }
}

void getdis(int u, int fafa){
    size[u] = 1;
    dd[++ len] = d[u];
    for(int i = head[u]; i; i = g[i].nxt){
        int v = g[i].to; if(v == fafa || vis[v]) continue;
        d[v] = d[u] + g[i].dis; getdis(v, u);
        size[u] += size[v];
    }
}

int cal(int u, int w){
    len = 0; d[u] = w; getdis(u, 0);
    std::sort(dd + 1, dd + len + 1);
    int l = 1, r = len, sum = 0;
    while(l < r){
        if(dd[l] + dd[r] <= k) sum += r - l, l ++;
        else r --;
    }
    return sum;
}

void solve(int u){
    vis[u] = 1; ans += cal(u, 0);
    for(int i = head[u]; i; i = g[i].nxt){
        int v = g[i].to; if(vis[v]) continue;
```

```
            ans -= cal(v, g[i].dis);
            rt = 0; con = size[v];
            getrt(v, 0);
            solve(rt);
        }
}
```

5.18 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;
```

```cpp
    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() {}
```

5.19 最近公共祖先 (在线)

```cpp
// 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;
}
```

5.20 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));
}

void tarjan(int u) {
    vis[u] = 1;
    for (auto& v : G[u]) {
        if (vis[v]) continue;
        tarjan(v);
        unite(u, v);
    }
    for (auto& q : query[u]) {
        int &v = q.first, &id = q.second;
        if (!vis[v]) continue;
        ans[id] = find(v);
    }
}
```

5.21 最近公共祖先

```
// LCA ST算法
int n, top, root;
int a[maxn << 1], d[maxn], st[maxn];
int f[maxn << 1][18], loc[maxn << 1][18];
vector<int> v[maxn];

int log2(int x) {
    int k = 0;
    while (x > 1) {
        x /= 2;
        k++;
    }
    return k;
}

void dfs(int u, int dep) {
    d[u] = dep;
    a[++top] = u;
    for (int i = 0; i<=v[u].size(); i++) {
        int to = v[u][i];
        dfs(to, dep + 1);
        a[++top] = u;
    }
}

void init() {
    int s = log2(top);
    for (int i = 1; i <= top; i++) {
        f[i][0] = d[a[i]];
        loc[i][0] = a[i];
    }
    for (int j = 1; j <= s; j++) {
        int k = top - (1 << j) + 1;
        for (int i = 1; i <= k; i++) {
            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);
*/

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

// LCA 倍增算法
int n, m, root;
int d[maxn], f[maxn][20];
vector<int> v[maxn];

inline void dfs(int u, int dep) {
    d[u] = dep;
    m = max(m, dep);
    for (int i = 0; i < v[u].size(); i++)
```

```cpp
        dfs(v[u][i], dep + 1);
}

int log2(int x) {
    int k = 0;
    while (x > 1) {
        x >>= 1;
        k++;
    }
    return k;
}

void init() {
    dfs(root, 0);
    int s = log2(m);
    for (int j = 1; j <= s; j++)
        for (int i = 1; i <= n; i++)
            f[i][j] = f[f[i][j - 1]][j - 1];
}

int query(int x, int y) {
    if (d[x] < d[y]) swap(x, y);
    int s = log2(d[x] - d[y]);
    while (d[x] > d[y]) {
        if (d[x] - (1 << s) >= d[y])
            x = f[x][s];
        s--;
    }
    s = log2(d[x]);
    while (s > -1) {
        if (f[x][s] != f[y][s]) {
            x = f[x][s];
            y = f[y][s];
        }
        s--;
    }
    return x == y ? x : f[x][0];
}
```

5.22 树链剖分

```cpp
// 树链剖分 点权
/**
* top[v] 表示v所在的重链的顶端节点
* fa[v] 表示v的父节点
* deep[v] 表示v的深度(根的深度为1)
* snum[v] 表示以v为根的子树的节点数
* p[v] 表示v所在(线段树中)的位置
* fp[v] 与p[v]相反，表示对应位置的节点
* son[v] 表示v的重儿子
* Edge 存树边
**/
```

```

struct Edge {
    int to, next;
}edge[maxn << 1];

int pos, n, m, tot; // n 为节点数
int head[maxn], top[maxn], fa[maxn], deep[maxn], num[maxn], p[maxn], fp[maxn], son[maxn];

void init() {
    tot = 0;
    pos = 1;
    memset(head, -1, sizeof(head));
    memset(son, -1, sizeof(son));
    for (int i = 0; i <= n; i++)
        v[i].clear();
}

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

void dfs1(int u, int pre, int d) {
    deep[u] = d;
    fa[u] = pre;
    num[u] = 1;
    for (int i = head[u]; i != -1; i = edge[i].next) {
        int to = edge[i].to;
        if (to != pre) {
            dfs1(to, u, d + 1);
            num[u] += num[to];
            if (son[u] == -1 || num[to] > num[son[u]])
                son[u] = to;
        }
    }
}

void dfs2(int u, int sp) {
    top[u] = sp;
    p[u] = pos++;
    fp[p[u]] = u;
    if (son[u] == -1) return;
    dfs2(son[u], sp);
    for (int i = head[u]; i != -1; i = edge[i].next) {
        int to = edge[i].to;
        if (to != son[u] && to != fa[u])
            dfs2(to, to);
    }
}
/*
// 使用范例
int getsum(int a, int b) {
    int f1 = top[a], f2 = top[b];
    int ret = 0;
    while (f1 != f2) {
        if (deep[f1] < deep[f2]) {
            swap(f1, f2);
            swap(a, b);
        }
        ret += query(p[f1], p[a], 1, n - 1, 1);
        a = fa[f1]; f1 = top[a];
    }
    if (a == b) return ret;
    if (deep[a] > deep[b]) swap(a, b);
    return ret + query(p[son[a]], p[b], 1, n - 1, 1);
}
*/
```

第六章　字符串

6.1　KMP

```
//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 = new int[m+1];
    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;
}
```

6.2　TRIE

```cpp
#include <cstring>

const int maxn = 10000*50+10;
const int max_stringlen = 26+2;
int trie[maxn][max_stringlen];
int val[maxn];
int trie_index;

int index_of(const char &c) {
    return c - 'a';
}
void trie_init() {
    trie_index = 0;
    memset(val, 0, sizeof(val));
    memset(trie, 0, sizeof(trie));
}
void trie_insert(char *s, int v) { //要求v!=0
    int len = strlen(s);
    int now = 0;
    for (int i = 0; i < len; ++i) {
        int idx = index_of(s[i]);
        int &tr = trie[now][idx];
        if (!tr) {
            tr = ++trie_index;
        }
        now = tr;
    }
    val[now] += v;
}
```

6.3 后缀数组 (倍增)

```cpp
//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);
    for (int i = 1; i <= n; i++) a[i] = std::lower_bound(set + 1, end, s[i]) - set;

    static int fir[maxn + 1], sec[maxn + 1], tmp[maxn + 1], buc[maxn + 1];
    for (int i = 1; i <= n; i++) buc[a[i]]++;
    for (int i = 1; i <= n; i++) buc[i] += buc[i - 1];
    for (int i = 1; i <= n; i++) rk[i] = buc[a[i] - 1] + 1;

    for (int t = 1; t <= n; t *= 2) {
        for (int i = 1; i <= n; i++) fir[i] = rk[i];
        for (int i = 1; i <= n; i++) sec[i] = i + t > n ? 0 : rk[i + t];

        std::fill(buc, buc + n + 1, 0);
        for (int i = 1; i <= n; i++) buc[sec[i]]++;
        for (int i = 1; i <= n; i++) buc[i] += buc[i - 1];
        for (int i = 1; i <= n; i++) tmp[n - --buc[sec[i]]] = i;

        std::fill(buc, buc + n + 1, 0);
        for (int i = 1; i <= n; i++) buc[fir[i]]++;
        for (int i = 1; i <= n; i++) buc[i] += buc[i - 1];
        for (int j = 1, i; j <= n; j++) i = tmp[j], sa[buc[fir[i]]--] = i;

        bool unique = true;
        for (int j = 1, i, last = 0; j <= n; j++) {
            i = sa[j];
            if (!last) rk[i] = 1;
            else if (fir[i] == fir[last] && sec[i] == sec[last]) rk[i] = rk[last], unique = false;
            else rk[i] = rk[last] + 1;

            last = i;
        }

        if (unique) break;
    }

    for (int i = 1, k = 0; i <= n; i++) {
        if (rk[i] == 1) k = 0;
        else {
            if (k > 0) k--;
            int j = sa[rk[i] - 1];
            while (i + k <= n && j + k <= n && a[i + k] == a[j + k]) k++;
        }
        ht[rk[i]] = k;
    }
}

int main() {
    std::cin >> n >> s;
    suffixArray();
    for (int i = 1; i <= n; i++) {
        std::cout << sa[i] << "␣";
    }
}
```

6.4 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--;
        }
    }
};
```

6.5 后缀自动机

```
//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;
}
```

```cpp

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;
}
```

6.6 最长回文子串

```cpp
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;
}
```

6.7 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;
```

```c
    }
    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);
}
```

```c

// 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);
}
```

6.8 字符串哈希表

```c
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');
}

ull get(int l, int r, ull g[]) {
    return g[r] - g[l - 1] * p[r - l + 1];
}

```

```cpp
struct HASHMAP {
    int size;
    int head[maxh], next[maxn], f[maxn];    // maxh 为hash链表最大长度
    ull state[maxn];
    void init() {
        size = 0;
        memset(head, -1, sizeof(head));
    }
    int insert(ull val, int id) {
        int h = val % maxh;
        for (int i = head[h]; i != -1; i = next[i])
            if (val == state[i]) return f[i];
        f[size] = id;
        state[size] = val;
        next[size] = head[h];
        head[h] = size;
        return f[size++];
    }
};
```

第七章　几何

7.1　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

```

```

棱锥:
1. 体积 V=Ah/3,A为底面积,h为高
(以下对正棱锥)
2. 侧面积 S=lp/2,l为斜高,p为底面周长
3. 全面积 T=S+A


棱台:
1. 体积 V=(A1+A2+sqrt(A1A2))h/3,A1.A2为上下底面积,h为高
(以下为正棱台)
2. 侧面积 S=(p1+p2)l/2,p1.p2为上下底面周长,l为斜高
3. 全面积 T=S+A1+A2


圆柱:
1. 侧面积 S=2PIrh
2. 全面积 T=2PIr(h+r)
3. 体积 V=PIr^2h


圆锥:
1. 母线 l=sqrt(h^2+r^2)
2. 侧面积 S=PIrl
3. 全面积 T=PIr(l+r)
4. 体积 V=PIr^2h/3


圆台:
1. 母线 l=sqrt(h^2+(r1-r2)^2)
2. 侧面积 S=PI(r1+r2)l
3. 全面积 T=PIr1(l+r1)+PIr2(l+r2)
4. 体积 V=PI(r1^2+r2^2+r1r2)h/3


球:
1. 全面积 T=4PIr^2
2. 体积 V=4PIr^3/3


球台:
1. 侧面积 S=2PIrh
2. 全面积 T=PI(2rh+r1^2+r2^2)
3. 体积 V=PIh(3(r1^2+r2^2)+h^2)/6


球扇形:
1. 全面积 T=PIr(2h+r0),h为球冠高,r0为球冠底面半径
2. 体积 V=2PIr^2h/3
```

第八章　类

8.1　点类

```cpp
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;
    }
};
```

8.2　分数类

```cpp
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;
    }
};
```

8.3 矩阵

```cpp
#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)
```

```cpp
            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';
        }
    }
};
```

8.4 01 矩阵

```cpp
#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;
}
```

第九章　黑科技

9.1　快速枚举子集

```
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) {}
```

9.2　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))
```

9.3 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 << "␣";
}
```

9.4 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
```

```cpp
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));
}
```

9.5 CDQ 分治

```cpp
//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;
}
```

9.6 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.
```

9.7 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;
    }
};
```

9.8 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) {
            if (b & 1)r = r * a % mod;
        }
        return r;
    }

    ll res[N], base[N], _c[N], _md[N];
    vector<int> Md;

    void mul(ll *a, ll *b, int k) {
        k <<= 1;
        for (int i = 0; i < k; ++i) _c[i] = 0;
        k >>= 1;
        for (int i = 0; i < k; ++i) {
            if (a[i]) {
                for (int j = 0; j < k; ++j) {
                    _c[i + j] = (_c[i + j] + a[i] * b[j]) % mod;
                }
            }
        }
        for (int i = k + k - 1; i >= k; i--) {
            if (_c[i]) {
                for (const int md: Md) {
                    _c[i - k + md] = (_c[i - k + md] - _c[i] * _md[md]) % mod;
                }
            }
        }
        for (int i = 0; i < k; ++i) {
            a[i] = _c[i];
        }
    }

    int solve(ll n, vector<int> a, vector<int> b) { // a 系数 b 初值 b[n+1]=a[0]*b[n]+...
//          printf("SIZE %d\n",SZ(b));
        ll ans = 0, pnt = 0;
        int k = (int) a.size();
        assert(a.size() == b.size());
        for (int i = 0; i < k; ++i) {
            _md[k - 1 - i] = -a[i];
        }
        _md[k] = 1;
        Md.clear();
        for (int i = 0; i < k; ++i) {
            if (_md[i] != 0) {
                Md.push_back(i);
            }
        }
        for (int i = 0; i < k; ++i) {
            res[i] = base[i] = 0;
        }
        res[0] = 1;
        while ((1ll << pnt) <= n) {
            pnt++;
        }
        for (int p = pnt; p >= 0; p--) {
            mul(res, res, k);
            if ((n >> p) & 1) {
                for (int i = k - 1; i >= 0; i--) {
                    res[i + 1] = res[i];
                }
                res[0] = 0;
                for (const int md: Md) {
                    res[md] = (res[md] - res[k] * _md[md]) % mod;
                }
            }
        }
        for (int i = 0; i < k; ++i) {
            ans = (ans + res[i] * b[i]) % mod;
        }
        if (ans < 0) ans += mod;
        return ans;
    }

    vector<int> BM(vector<int> s) {
        vector<int> C(1, 1), B(1, 1);
        int L = 0, m = 1, b = 1;
        for (int n = 0; n < (int) s.size(); ++n) {
            ll d = 0;
            for (int i = 0; i <= L; ++i) {
                d = (d + (ll) C[i] * s[n - i]) % mod;
            }
            if (d == 0) {
                ++m;
            }
            else if (2 * L <= n) {
                vector<int> T = C;
                ll c = mod - d * pow_mod(b, mod - 2) % mod;
                while (C.size() < B.size() + m) {
                    C.push_back(0);
                }
                for (int i = 0; i < (int) B.size(); ++i) {
                    C[i + m] = (C[i + m] + c * B[i]) % mod;
                }
                L = n + 1 - L;
                B = T;
                b = d;
                m = 1;
            } else {
                ll c = mod - d * pow_mod(b, mod - 2) % mod;
                while (C.size() < B.size() + m) {
                    C.push_back(0);
                }
                for (int i = 0; i < (int) B.size(); ++i) {
                    C[i + m] = (C[i + m] + c * B[i]) % mod;
                }
                ++m;
            }
        }
        return C;
    }

    int gao(vector<int> a, ll n) {
        vector<int> c = BM(a);
        c.erase(c.begin());
        for (int &x:c) {
            x = (mod - x) % mod;
        }
        return solve(n, c, vector<int>(a.begin(), a.begin() + c.size()));
    }
}
```