模板之用了就秃

WUST

So Like Coding? You Baldy

October 10, 2019

Contents

0 Header **1**
0.1 pbds . . . 1
0.1.1 head . . . 1
0.2 FastIO . . . 1
0.2.1 FastScanner . . . 1
0.2.2 FastPowAndAdd . . . 1
0.2.3 PythonInput . . . 2
0.2.4 SpecialInput . . . 2
0.3 header . . . 3
0.3.1 comp . . . 4

1 Math **5**
1.1 素数 . . . 5
1.1.1 Eratosthenes 筛法 . . . 5
1.1.2 Euler 筛 . . . 5
1.1.3 MillerRabin 素性测试 . . . 5
1.1.4 PollardRho 快速因数分解 . . . 6
1.1.5 求素因子 . . . 6
1.2 约数 . . . 6
1.2.1 EulerPhi . . . 6
1.2.2 Sieve . . . 7
1.2.3 gcd . . . 8
1.2.4 解乘法逆元 . . . 8
1.3 同余 . . . 8
1.3.1 扩展欧几里得算法 . . . 8
1.3.2 中国剩余定理 . . . 9
1.3.3 扩展中国剩余定理 . . . 9
1.3.4 BSGS . . . 10
1.3.5 exBSGS . . . 10
1.3.6 逆元 . . . 11
1.3.7 模素数二次同余方程 . . . 12
1.4 矩阵与线性方程组 . . . 12
1.4.1 矩阵快速幂 . . . 12
1.4.2 高斯消元 . . . 13
1.4.3 线性基 . . . 13
1.5 组合数学 . . . 14
1.5.1 Lucas . . . 14
1.5.2 exLucas . . . 15
1.5.3 递推组合数 . . . 16
1.5.4 小模数组合数 . . . 17
1.5.5 大模数组合数 . . . 17
1.6 卷积 . . . 18
1.6.1 FFT . . . 18
1.6.2 NTT . . . 19
1.6.3 原根 . . . 20
1.6.4 FWT . . . 21
1.7 多项式 . . . 22
1.7.1 拉格朗日插值 . . . 22
1.7.2 拉格朗日插值 (连续取值) . . . 22
1.8 Others . . . 23
1.8.1 BM . . . 23
1.8.2 exBM . . . 24
1.8.3 杜教筛 . . . 29
1.8.4 欧拉降幂 . . . 30
1.8.5 公式 . . . 32
1.8.6 博弈 . . . 34

2 Graph Theory **35**
2.1 路径 . . . 35
2.1.1 Dijkstra . . . 35
2.1.2 Euler Path . . . 35
2.1.3 K shortest Path(Astar) . . . 36
2.1.4 K shortest Path(可持久化可并堆) . . . 38
2.2 生成树 . . . 40
2.2.1 Kruskal . . . 40
2.2.2 Prim . . . 41
2.2.3 最小树形图 . . . 41
2.2.4 Matrix Tree . . . 42
2.2.5 Steiner Tree . . . 43
2.3 连通性 . . . 44
2.3.1 割点 . . . 44
2.3.2 桥 . . . 45
2.3.3 强连通分量 . . . 45
2.3.4 点双联通分量 . . . 46
2.3.5 边双联通分量 . . . 47
2.4 二分图匹配 . . . 48
2.4.1 Hungary Algorithm . . . 48
2.4.2 Hopcroft-karp Algorithm . . . 48
2.4.3 二分图多重匹配 . . . 50
2.4.4 二分图最大权匹配 (KM 算法) . . . 51
2.4.5 一般图匹配带花树 . . . 52
2.5 网络流 . . . 54
2.5.1 Dinic . . . 54
2.5.2 ISAP . . . 56
2.5.3 MCMF . . . 57
2.5.4 Trick . . . 59
2.5.5 Stoer Wagner . . . 60
2.6 Others . . . 60
2.6.1 拓扑排序 . . . 60
2.6.2 2-SAT . . . 61
2.6.3 差分约束系统 . . . 62
2.6.4 支配树 . . . 63
2.6.5 Stable Matching Problem . . . 65

3 DataStructrue **67**
3.1 SegmentTreeDS . . . 67
3.1.1 SegmentTree . . . 67
3.1.2 离散化区间 . . . 68
3.1.3 动态区间最大子段和 . . . 69
3.1.4 动态开点权值线段树 . . . 70
3.1.5 扫描线 . . . 70
3.2 HLD . . . 75
3.2.1 HLD . . . 75
3.3 RMQ . . . 78
3.3.1 RMQ . . . 78
3.3.2 RMQbyIndex . . . 79
3.3.3 RMQinNM . . . 79
3.4 MO . . . 80
3.4.1 MO . . . 80
3.4.2 MObyModify . . . 81
3.4.3 分块 . . . 82
3.4.4 弹飞绵羊 . . . 83
3.4.5 树莫队 . . . 84
3.5 VirtualTree . . . 86
3.5.1 VirtualTree . . . 86
3.6 PersistentDS . . . 88

3.6.1 主席树区间 k 大 . . . 88
3.6.2 可持久化数组 . . . 89
3.7 Tree . . . 91
3.7.1 LCA . . . 91
3.7.2 前向星 . . . 92
3.7.3 点分治 . . . 92
3.8 Others . . . 94
3.8.1 BITinNM . . . 94
3.8.2 静态区间 k 大划分树 . . . 95

4 String **97**
4.1 KMP . . . 97
4.1.1 KMP . . . 97
4.1.2 exKMP . . . 97
4.2 Trie . . . 98
4.2.1 Trie . . . 98
4.2.2 Persistence Trie . . . 98
4.2.3 01Trie . . . 99
4.3 Manachar . . . 100
4.3.1 Manacher . . . 100
4.4 Aho-Corasick Automation . . . 101
4.4.1 AC Automation . . . 101
4.5 Suffix Array . . . 102
4.5.1 Suffix Array . . . 102
4.6 PalindromicTree . . . 103
4.6.1 PalindromicTree . . . 103
4.7 Hash . . . 105
4.7.1 hash . . . 105
4.7.2 doubleHash . . . 106
4.7.3 二维 hash . . . 109
4.7.4 树 hash 同构 . . . 109
4.8 Suffix Automation . . . 110
4.8.1 SAM . . . 110
4.9 Others . . . 112
4.9.1 最小表示法 . . . 112

5 dp **113**
5.1 BitDP . . . 113
5.1.1 数位 dp 计和 . . . 113
5.1.2 两个数数位 dp . . . 113
5.2 Subsequence . . . 114
5.2.1 MaxSum . . . 114
5.2.2 LIS . . . 115
5.2.3 LongestCommonIncrease . . . 115
5.2.4 LCS . . . 115
5.3 Others . . . 117
5.3.1 矩阵快速幂 . . . 117

6 Others **119**
6.1 mint 类 . . . 119
6.2 不重叠区间贪心 . . . 119
6.3 BigInt 类 . . . 119
6.4 date . . . 125
6.5 Frac 类 . . . 125
6.6 模拟退火 (最小圆覆盖) . . . 126
6.7 string 类 . . . 127
6.8 前缀异或和 . . . 128
6.9 约瑟夫环第 k 个 . . . 128
6.10 二分 . . . 128

6.11 猛男 IO 挂 129
6.12 贪心结论 130
6.13 builtin 131
6.14 n 以内 k 因子的个数 131

0 Header

0.1 pbds

0.1.1 head

```
#include <bits/extc++.h>
#pragma comment(linker, "/STACK:102400000,102400000")
using namespace __gnu_pbds; // tree, gp_hash_table, trie
using namespace __gnu_cxx; // rope
tree<TYPE, null_type, less<>, rb_tree_tag, tree_order_statistics_node_update> tr;
// 可并堆
#include <ext/pb_ds/priority_queue.hpp>
using namespace __gnu_pbds;
__gnu_pbds::priority_queue<int,greater<int>,pairing_heap_tag> q[maxn];
//q[i].join(q[j]) 将j堆并入i

```

0.2 FastIO

0.2.1 FastScanner

```
// 适用于正负整数
template <class T>
inline bool scan(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;
}

template <class T>
inline void out(T x) {
    if (x > 9) out(x / 10);
    putchar(x % 10 + '0');
}

inline int read() {
    int x = 0;
    char ch = getchar();
    while (ch > '9' || ch < '0')ch = getchar();
    while (ch >= '0' && ch <= '9') {
        x = x * 10 + ch - '0';
        ch = getchar();
    }
    return x;
}
```

0.2.2 FastPowAndAdd

```
// 精确快速乘
ll qpmul(ll a, ll b) {
    a %= mod; b %= mod;
    ll res = 0;
    while (b > 0) {
        if (b & 1) {
            res = (res + a);
            if (res >= mod) res -= mod;
        }
        a = (a + a);
        if (a >= mod) a -= mod;
        b >>= 1;
    }
    return res;
}

// O(1)快速乘
ll mul2(ll x,ll y,ll p) {
    ll res=(x*y-(ll)((long double)x/p*y+1.0e-8)*p);
    return res<0?res+p:res;
}

//int128
ll ans = ((__int128) a * b) % p;

// 10进制快速幂，直接读入%s,c 预处理字符串len
char c[1000005], len;
ll qp(ll a) {
    len --;
    a %= mod;
    ll s = a;
    ll res = 1;
    while (len >= 0) {
        ll cur = s;
        for (int i = 1; i <= c[len] - '0'; ++i) {
            res = res * s % mod;
        }
        for (int i = 1; i < 10; ++i) {
            cur = cur * s % mod;
        }
        s = cur;
        len --;
    }
    return res;
}
```

0.2.3 PythonInput

```
// python一行读入
a,b = map(int, input().split())

a = []
for i in input().split():
    a.append(int(i))
```

0.2.4 SpecialInput

```
// 代替gets
scanf("%[^\n]%*c", ss)

void out2(int x, int flag = 1) {
    if (x == 0) {
        if (flag) putchar('0');
        return;
    }
    out2(x >> 1, 0);
    putchar('0' + x % 2);
}
```

0.3 header

```
// Editor -> Live Templates
// add template group acm
// add template main
// C++ Declaration

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

template <class T>
inline bool scan(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;
}

const ll mod = 1e9+7;
const int maxn = $MAXN$;

ll qp(ll x, ll n) {
    ll res = 1; x %= mod;
    while (n > 0) {
        if (n & 1) res = res * x % mod;
        x = x * x % mod;
        n >>= 1;
    }
    return res;
}

int main(int argc, char* argv[]) {
    $CONTENT$
    return 0;
}

// C++ Expression debug
freopen("data.in","r",stdin);
freopen("data.out","w",stdout);
clock_t ST = clock();
cerr << "time: " << ((clock()-ST)*1000.0 / CLOCKS_PER_SEC) << "ms" << endl;


// C++ Expression tkase
```

```
int T;
scanf("%d", &T);
for (int kase = 1; kase <= T; ++kase) { $CONTENT$;
}
```

0.3.1 comp

```
// 1 create directory comp
// 2 create directory comp/test
// 3 create duipai.cpp
// add text

#include<bits/stdc++.h>
using namespace std;
int main(){
    int i;
  for (i=1;;i++){
        printf("The result of No. %d Case is:  ",i);
        system("python3 rand.py");
        system("./std < test/data.in > test/std.out");
        system("./my < test/data.in > test/my.out");
        if (system("diff test/std.out test/my.out")){
            printf("Wrong Answer\n");
            return 0;
        }
        else printf("Accepted\n");
    }
    return 0;
}


// 4 create duipai.sh
#!/bin/bash
g++ std.cpp -o std
g++ my.cpp -o my
python3 rand.py
./a.out

// 5 create rand.py
# coding=utf-8
from random import randint, choice, shuffle
# with open("../cmake-build-debug/data.in", "w") as f:
with open("test/data.in", "w") as f:
    n = randint(1, 10)
    m = randint(1, 10)
    f.write(f"{n} {m}")

// 6 terminal: g++ duipai.cpp
// 7 terminal: sudo chmod 777 duipai.sh
// 8 add my.cpp and std.cpp
// 9 ./duipai.sh
```

1 Math

1.1 素数

1.1.1 Eratosthenes 筛法

```
bool vis[(int)1e6+5];
int prim[(int)1e5], tot; // tot = 78499
void init() {
    for (int i = 2; i < 1e6+5; ++i) {
        if (vis[i]) continue;
        prim[++tot] = i;
        for (int j = i + i; j < 1e6+5; j += i) vis[j] = 1;
    }
}
```

1.1.2 Euler 筛

```
const int maxn = 1e6 + 10;
int prime[maxn], v[maxn], n, cnt;   //每个合数只会被它的最小质因子p筛一次

void Euler_Sieve()
{
    for(int i = 2; i <= n; i ++)
    {
        if(!v[i]) v[i] = i, prime[++cnt] = i;
        for(int j = 1; j <= cnt && i * prime[j] <= n; j ++)
        {
            v[i * prime[j]] = prime[j];
            if(i % prime[j] == 0) break;
        }
    }
}
```

1.1.3 MillerRabin 素性测试

```
typedef long long ll;

bool check(ll a, ll n)
{
    if(n == 2 || a >= n) return true;
    if(n == 1 || !(n & 1)) return false;
    ll d = n - 1;
    while(!(d & 1)) d >>= 1;
    ll t = qp(a, d, n);
    while(d != n - 1 && t != 1 && t != n - 1)
    {
        t = mul(t, t, n);
        d <<= 1;
    }
    return t == n - 1 || d & 1;
}

bool Miller_Rabin(ll n)
{
    static vector<ll> t = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};
    if (n <= 1) return false;
    for (ll k: t) if (!check(k, n)) return false;
    return true;
}
```

1.1.4 PollardRho 快速因数分解

```
mt19937 mt(time(0));
ll pollard_rho(ll n, ll c) {
    ll x = uniform_int_distribution<ll>(1, n - 1)(mt), y = x;
    auto f = [&](ll v) { ll t = mul(v, v, n) + c; return t < n ? t : t - n; };
    while (1) {
        x = f(x); y = f(f(y));
        if (x == y) return n;
        ll d = __gcd(abs(x - y), n);
        if (d != 1) return d;
    }
}

ll fac[100], fcnt;
void get_fac(ll n, ll cc = 19260817) {
    if (n == 4) { fac[fcnt++] = 2; fac[fcnt++] = 2; return; }
    if (Miller_Rabin(n)) { fac[fcnt++] = n; return; }
    ll p = n;
    while (p == n) p = pollard_rho(n, --cc);
    get_fac(p); get_fac(n / p);
}
```

1.1.5 求素因子

```
vector<pair<ll, int>> getFactors(ll x)
{
    vector<pair<ll, int>> fact;
    for (int i = 0; prime[i] <= x / prime[i]; i++)
    {
        if (x % prime[i] == 0)
        {
            fact.emplace_back(prime[i], 0);
            while (x % prime[i] == 0) fact.back().second++, x /= prime[i];
        }
    }
    if (x != 1) fact.emplace_back(x, 1);
    return fact;
}
```

1.2 约数

1.2.1 EulerPhi

```
//计算欧拉phi函数，phi(n)且与n互素的正整数个数

int Euler(int n){
    int rea=n;
    for(int i=2; i*i<=n; i++)
        if(n%i==0)//第一次找到的必为素因子
        {
            rea=rea-rea/i;
            do
                n/=i;//把该素因子全部约掉
            while(n%i==0);
        }
    if(n>1)
        rea=rea-rea/n;
    return rea;
} //单点欧拉 O(sqrt(n))

bool boo[50000];
int p[20000];
void prim(){
    memset(boo,0,sizeof(boo));
    boo[0]=boo[1]=1;
    int k=0;
    for(int i=2; i<50000; i++)
    {
        if(!boo[i])
            p[k++]=i;
        for(int j=0; j<k&&i*p[j]<50000; j++)
        {
            boo[i*p[j]]=1;
            if(!(i%p[j]))
                break;
        }
    }
}//筛选法打表
int phi(int n)
{
    int rea=n;
    for(int i=0; p[i]*p[i]<=n; i++)//对于一些不是素数的可不遍历
        if(n%p[i]==0)
        {
            rea=rea-rea/n;
            do
                n/=p[i];
            while(n%p[i]==0);
        }
    if(n>1)
        rea=rea-rea/n;
    return rea;
} //素数+欧拉

int euler[maxn];
void init() {
    int i, j;
    for(i=1; i<maxn; i++)
        euler[i]=i;
    for(i=2; i<maxn; i+=2)
        euler[i]/=2;
    for(i=3; i<maxn; i+=2)
        if(euler[i]==i) {
            for(j=i; j<=maxn; j+=i)
                euler[j]=euler[j]/i*(i-1);
        }
} //递推欧拉表
```

1.2.2 Sieve

```
//用类似筛法的方法计算phi(1),phi(2),...,phi(n)
```

```
int phi[maxn];

void phi_table(int n)
{
    for (int i = 2; i <= n; i++) phi[i] = 0;
    phi[1] = 1;
    for (int i = 2; i <= n; i++) if (!phi[i])
        for (int j = i; j <= n; j += i)
        {
            if (!phi[j]) phi[j] = j;
            phi[j] = phi[j] / i * (i - 1);
        }
}
```

1.2.3 gcd

```
ll gcd(ll a,ll b) {while(b^=a^=b^=a%=b);return a;}

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

1.2.4 解乘法逆元

```
void exgcd(ll a, ll b, ll c, ll d, ll &x, ll &y) {
    ll z = (a + b - 1) / b;
    if (z <= c / d) {
        x = z;
        y = 1;
        return;
    }
    a -= (z - 1) * b; c -= (z - 1) * d;
    exgcd(d, c, b, a, y, x);
    x += (z - 1) * y;
}

int main(int argc, char* argv[]) {
    int T;
    scanf("%d", &T);
    ll p, x;
    for (int kase = 1; kase <= T; ++kase) {
        scanf("%lld%lld", &p, &x);
        ll b, y;
        exgcd(p, x, p, x - 1, b, y);
        printf("%lld/%lld\n", b * x - p * y, b);
    }
    return 0;
}
```

1.3 同余

1.3.1 扩展欧几里得算法

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

1.3.2 中国剩余定理

```
typedef long long ll;

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

ll crt(ll *a, ll *m, int n)
{
    ll M = 1, ans = 0;
    for(int i = 1; i <= n; i ++) M *= m[i];
    for(int i = 1; i <= n; i ++)
    {
        ll x = 0, y = 0;
        ll Mi = M / m[i];
        exgcd(Mi, m[i], x, y);
        ans = (ans + Mi % M * x % M * a[i] % M + M) % M;
    }
    if(ans < 0) ans += M;
    return ans;
}
```

1.3.3 扩展中国剩余定理

```
typedef long long ll;

const int N = 1e5 + 10;

int n;
ll a[N], r[N];

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

ll excrt()
{
    ll M = a[1], R = r[1], x, y, d;
    for(int i = 2; i <= n; i ++)
    {
        d = exgcd(M, a[i], x, y);
        if((R - r[i]) % d) return -1;
        x = (R - r[i]) / d * x % a[i];
        R -= M * x;
        M = M / d * a[i];
        R %= M;
    }
    return (R % M + M) % M;
}
```

1.3.4 BSGS

```
int qp(int a, int n, int mod)
{
    long long ans = 1, base = a;
    while(n)
    {
        if(n & 1) (ans *= base) %= mod;
        (base *= base) %= mod;
        n >>= 1;
    }
    return ans;
}

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

1.3.5 exBSGS

```
unordered_map<int, int> Hash;

int exBSGS(int a, int b, int p)
{
    a %= p, b %= p;
    if(b == 1) return 0;
    if(!b && !a) return 1;
    if(!a) return -1;
    if(!b)
    {
        int ret = 0, d;
        while((d = __gcd(a, p)) != 1)
        {
            ++ ret, p /= d;
            if(p == 1) return ret;
        }
        return -1;
    }
    int ret = 0, A = a, B = b, P = p, C = 1, d;
    while((d = __gcd(A, P)) != 1)
    {
        if(B % d) return -1;
        P /= d, B /= d;
        C = 1ll * C * (A / d) % P;
        ++ ret;
        if(C == B) return ret;
    }
    Hash.clear();
    int f = 1, t = sqrt(P) + 1;
    for(int i = 0; i < t; i ++)
    {
        Hash[1ll * f * B % P] = i;
        f = 1ll * f * A % P;
    }
    int tf = f;
    f = 1ll * f * C % P;
    for(int i = 1; i <= t; i ++)
    {
        if(Hash.find(f) != Hash.end()) return ret + i * t - Hash[f];
        f = 1ll * f * tf % P;
    }
    return -1;
}
```

1.3.6 逆元

```
/*
1.费马小定理
条件:mod为素数
*/
ll inv(ll x){return qp(x,mod-2);}

/*
2.扩展欧几里得
条件:gcd(a,mod)==1
如果gcd(a,mod)!=1 返回-1
*/
ll inv(ll a,ll p)
{
    ll g,x,y;
    g=exgcd(a,p,x,y);
    return g==1?(x+p)%p:-1;
}

/*
3.公式
a/b%mod=c
->a%(b*mod)/b=c
*/

/*
4.逆元打表
p是模
p要求是奇素数
*/
ll inv[MAX];
```

```
void getinv(int n,ll p)
{
    ll i;
    inv[1]=1;
    for(i=2;i<=n;i++) inv[i]=(p-p/i)*inv[p%i]%p;
}

// log逆元
ll dlog(ll g, ll b, ll p) {
    ll m = sqrt(p - 1);
    map<ll, ll> powers;
    for (long j = 0; j < m; j++) powers[qp(g, j, p)] = j;
    long gm = qp(g, -m + 2 * (p - 1), p);
    for (int i = 0; i < m; i++) {
        if (powers[b]) return i * m + powers[b];
        b = b * gm % p;
    }
    return -1;
}
```

1.3.7 模素数二次同余方程

```
// 要求模为素数，输入n，mod，返回 x^2 % mod = n，可解任意一次二元方程

bool Legendre(ll a,ll p) {
    return qp(a,p-1>>1,p)==1;
}

ll modsqr(ll a,ll p) {
    ll x;
    ll i,k,b;
    if(p==2) x=a%p;
    else if(p%4==3) x=qp(a,p+1>>2,p);
    else {
        for(b=1;Legendre(b,p);++b);
        i=p-1>>1;
        k=0;
        do
        {
            i>>=1;
            k>>=1;
            if(!((1LL*qp(a,i,p)*qp(b,k,p)+1)%p)) k+=p-1>>1;
        }while(!(i&1));
        x=1ll*qp(a,i+1>>1,p)*qp(b,k>>1,p)%p;
    }
    return min(x, p - x);
//    if(p-x<x) x=p-x;
//    if(x==p-x) printf("%d\n",x);
//    else printf("%d %d\n",x,p-x);
}
```

1.4 矩阵与线性方程组

1.4.1 矩阵快速幂

```
const int mod = 1e9 + 7;
typedef long long ll;

```

```
int cur;
struct Matrix {ll a[105][105]; };

Matrix mul(Matrix a, Matrix b)
{
    Matrix res;
    memset(res.a, 0, sizeof res.a);
    for(int i = 0; i < cur; i++)
        for(int j = 0; j < cur; j++)
            for(int k = 0; k < cur; k++)
                (res.a[i][j] +=a.a[i][k] * b.a[k][j] % mod) %= mod;
    return res;
}

Matrix pow(Matrix a, ll n)
{
    Matrix ans, base = a;
    for(int i = 0; i < cur; i++) ans.a[i][i] = 1;
    while(n)
    {
        if(n & 1) ans = mul(ans, base);
        base = mul(base, base);
        n >>= 1;
    }
    return ans;
}
```

1.4.2 高斯消元

```
const int N = 20 + 10;

int n;
double b[N], c[N][N];
//c: 系数矩阵, b: 常数;  二者一起构成增广矩阵

void Gaussian_Elimination()
{
    for(int i = 1; i <= n; i ++)
    {
        //找到x[i]的系数不为0的一个方程
        for(int j = i; j <= n; j ++) if(fabs(c[j][i]) > 1e-8)
        {
            for(int k = 1; k <= n; k ++) swap(c[i][k], c[j][k]);
            swap(b[i], b[j]);
        }
        //消去其他方程的x[i]的系数
        for(int j = 1; j <= n; j ++)
        {
            if(i == j) continue;
            double rate = c[j][i] / c[i][i];
            for(int k = i; k <= n; k ++) c[j][k] -= c[i][k] * rate;
            b[j] -= b[i] * rate;
        }
    }
}
```

1.4.3 线性基

```
typedef long long ll;

struct LinearBasis
{
    ll d[64], tot;

    void ins(ll x)  //插入线性基
    {
        for(int i = 63; i >= 0; i --)
        {
            if((x >> i) & 1)
            {
                if(!d[i]) return void(d[i] = x);
                x ^= d[i];
            }
        }
    }

    ll max_xor()    //在一个序列中取若干个数，使其异或和最大
    {
        ll ans = 0;
        for(int i = 63; i >= 0; i --)
            if((ans ^ d[i]) > ans) ans ^= d[i];
        return ans;
    }

    void init()
    {
        for(int i = 0; i < 64; i ++) if(d[i])
            for(int j = 0; j < i; j ++)
                if(d[i] & (1ll << j)) d[i] ^= d[j];
        for(int i = 0; i < 64; i ++) if(d[i]) d[tot ++] = d[i];
    }

    ll k_th(ll k)   //取任意个元素进行异或的第k小个数
    {
        //考虑能异或出0的情况，tot表示线性基中的元素个数
        k -= (n != tot);
        if(k > (1ll << tot)) return -1;
        ll ans = 0;
        for(int i = 0; i < tot; i ++) if(k & (1ll << i)) ans ^= d[i];
        return ans;
    }
};
```

1.5 组合数学

1.5.1 Lucas

```
const int maxn = 1e6 + 10;

ll fac[maxn], inv[maxn], facinv[maxn];

void init()
{
    fac[0] = inv[0] = facinv[0] = 1;
    fac[1] = inv[1] = facinv[1] = 1;
    for(int i = 2; i < maxn; i++)
    {
        fac[i] = fac[i - 1] * i % mod;
        inv[i] = mod - mod / i * inv[mod % i] % mod;
        facinv[i] = facinv[i - 1] * inv[i] % mod;
    }
}

ll C(int n, int k)
{
    if(k > n || k < 0) return 0;
    return fac[n] * facinv[k] % mod * facinv[n - k] % mod;
}

ll lucas(ll n, ll m)
{
    ll res = 1;
    while(n && m)
    {
        res = res * C(n % mod, m % mod) % mod;
        n /= mod;
        m /= mod;
    }
    return res;
}
```

1.5.2 exLucas

```
typedef long long ll;

ll p, n, m;

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

ll qp(ll a, ll n, ll mod)
{
    ll ans = 1, base = a;
    for(; n; n >>= 1, (base *= base) %= mod) if(n & 1) (ans *= base) %= mod;
    return ans;
}

ll fac(ll n, ll a, ll b)
{
    if(!n) return 1;
    ll res = 1;
    for(ll i = 2; i <= b; i ++)
        if(i % a) (res *= i) %= b;
    res = qp(res, n / b, b);
    for(ll i = 2; i <= n % b; i ++)
        if(i % a) (res *= i) %= b;
    return res * fac(n / a, a, b) % b;
}

ll inv(ll n, ll mod)
```

```
{
    ll x, y;
    exgcd(n, mod, x, y);
    return (x += mod) > mod ? x - mod : x;
}

ll CRT(ll b, ll mod) { return b * inv(p / mod, mod) % p * (p / mod) % p; }

ll C(ll n, ll m, ll a, ll b)
{
    ll up = fac(n, a, b), d1 = fac(m, a, b), d2 = fac(n - m, a, b);
    ll k = 0;
    for(ll i = n; i; i /= a) k += i / a;
    for(ll i = m; i; i /= a) k -= i / a;
    for(ll i = n - m; i; i /= a) k -= i / a;
    return up * inv(d1, b) % b * inv(d2, b) % b * qp(a, k, b) % b;
}

ll exlucas(ll n, ll m)
{
    ll res = 0, tmp = p, b;
    int lim = sqrt(p) + 5;
    for(int i = 2; i <= lim; i ++) if(tmp % i == 0)
    {
        b = 1;
        while(tmp % i == 0) tmp /= i, b *= i;
        (res += CRT(C(n, m, i, b), b)) %= p;
    }
    if(tmp > 1) (res += CRT(C(n, m, tmp, tmp), tmp)) %= p;
    return res;
}
```

1.5.3 递推组合数

$0 \leq m \leq n \leq 1000$

```
const int maxn = 1010;
ll C[maxn][maxn];
void init() {
    C[0][0] = 1;
    for (int i = 1; i < maxn; i++)
    {
        C[i][0] = 1;
        for (int j = 1; j <= i; j++) C[i][j] = (C[i - 1][j - 1] + C[i - 1][j]) % mod;
    }
}
```

$0 \leq m \leq n \leq 10^5$, 模 p 为素数

```
const int maxn = 100010;
ll f[maxn];
ll inv[maxn]; // 阶乘的逆元
void CalFact() {
    f[0] = 1;
    for (int i = 1; i < maxn; i++) f[i] = (f[i - 1] * i) % p;
    inv[maxn - 1] = qp(f[maxn - 1], p - 2);
    for (int i = maxn - 2; ~i; i--) inv[i] = inv[i + 1] * (i + 1) % p;
}
ll C(int n, int m) { return f[n] * inv[m] % p * inv[n - m] % p; }
```

1.5.4 小模数组合数

p 小 n,m 大

```

const int NICO = 100000+10;
const int MOD  = 99991;
ll f[NICO];

ll Lucas(ll a,ll k)
{
    ll res = 1;
    while(a && k)
    {
        ll a1 = a % MOD;
        ll b1 = k % MOD;
        if(a1 < b1) return 0;
        res = res*f[a1]*qp(f[b1]*f[a1-b1]%MOD,MOD-2)%MOD;
        a /= MOD;
        k /= MOD;
    }
    return res;
}

void init()
{
    f[0] = 1;
    for(int i=1;i<=MOD;i++)
    {
        f[i] = f[i-1]*i%MOD;
    }
}

int main()
{
    init();
    cout << Lucas(5,2) << endl;
}
```

1.5.5 大模数组合数

n,m 小 p 大

```
map<int, ll> m;

const int MOD = 1e9+7;
void fun(int n, int k) {
    for (int i = 2; i <= sqrt(n * 1.0); i++) {
        while (n % i == 0) {
            n /= i;
            m[i] += k;
        }
    }
    if (n > 1) {
        m[n] += k;
    }
}

```

```
ll C(ll a, ll b) {
    if (a < b || a < 0 || b < 0)
        return 0;
    m.clear();
    ll ret = 1;
    b = min(a - b, b);
    for (int i = 0; i < b; i++) {
        fun(a - i, 1);
    }
    for (int i = b; i >= 1; i--) {
        fun(i, -1);
    }
    for (__typeof(m.begin()) it = m.begin(); it != m.end(); it++) {
        if ((*it).second != 0) {
            ret *= qp((*it).first, (*it).second);
            ret %= MOD;
        }
    }
    return ret;
}

int main(int argc,char *argv[])
{
    ll a, b;
    while (scanf("%lld%lld", &a, &b) != EOF) {
        printf("%lld\n", C(a, b));
    }
    return 0;
}
```

1.6 卷积

1.6.1 FFT

```
const int maxn = 1e7 + 10;
const double Pi = acos(-1.0);

struct complex
{
    double x, y;
    complex (double xx = 0, double yy = 0) { x = xx, y = yy; }
}a[maxn], b[maxn];

complex operator + (complex a, complex b) { return complex(a.x + b.x, a.y + b.y); }
complex operator - (complex a, complex b) { return complex(a.x - b.x, a.y - b.y); }
complex operator * (complex a, complex b) { return complex(a.x * b.x - a.y * b.y, a.x * b.y + a.y *
     b.x); }

int n, m;
int l, r[maxn];
int limit;

void FFT(complex *A, int type)
{
    for(int i = 0; i < limit; i ++)
        if(i < r[i]) swap(A[i], A[r[i]]);
    for(int mid = 1; mid < limit; mid <<= 1)
    {
        complex Wn(cos(Pi / mid), type * sin(Pi / mid));
        for(int R = mid << 1, j = 0; j < limit; j += R)
        {
            complex w(1, 0);
            for(int k = 0; k < mid; k ++, w = w * Wn)
            {
                complex x = A[j + k], y = w * A[j + mid + k];
                A[j + k] = x + y;
                A[j + mid + k] = x - y;
            }
        }
    }
}

void mul()
{
    l = 0, limit = 1;
    while(limit <= n + m) limit <<= 1, l ++;
    for(int i = 0; i < limit; i ++)
        r[i] = (r[i >> 1] >> 1) | ( (i & 1) << (l - 1));
    FFT(a, 1);
    FFT(b, 1);
    for(int i = 0; i <= limit; i ++) a[i] = a[i] * b[i];
    FFT(a, -1);
    for(int i = 0; i <= n + m; i ++)
        printf("%d ", (int)(a[i].x / limit + 0.5));
}
```

1.6.2 NTT

```
const int maxn = 2097152;
const int mod = 998244353;
const int root = 3;
// 998244353 -> 3, 1e9+7 -> 5,

template<long long mod, long long root>
struct NTT {
    vector<long long> omega;

    NTT() {
        omega.resize(maxn + 1);
        long long x = fpow(root, (mod - 1) / maxn);
        omega[0] = 1ll;
        for (int i = 1; i <= maxn; ++i)
            omega[i] = omega[i - 1] * x % mod;
    }

    long long fpow(long long a, long long n) {
        (n += mod - 1) %= mod - 1;
        long long r = 1;
        for (; n; n >>= 1) {
            if (n & 1) (r *= a) %= mod;
            (a *= a) %= mod;
        }
        return r;
    }

    void bitrev(vector<long long> &v, int n) {
        int z = __builtin_ctz(n) - 1;
        for (int i = 0; i < n; ++i) {
            int x = 0;
            for (int j = 0; j <= z; ++j) x ^= (i >> j & 1) << (z - j);
            if (x > i) swap(v[x], v[i]);
        }
    }

    void ntt(vector<long long> &v, int n) {
        bitrev(v, n);
        for (int s = 2; s <= n; s <<= 1) {
            int z = s >> 1;
            for (int i = 0; i < n; i += s) {
                for (int k = 0; k < z; ++k) {
                    long long x = v[i + k + z] * omega[maxn / s * k] % mod;
                    v[i + k + z] = (v[i + k] + mod - x) % mod;
                    (v[i + k] += x) %= mod;
                }
            }
        }
    }

    void intt(vector<long long> &v, int n) {
        ntt(v, n);
        for (int i = 1; i < n / 2; ++i) swap(v[i], v[n - i]);
        long long inv = fpow(n, -1);
        for (int i = 0; i < n; ++i) (v[i] *= inv) %= mod;
    }

    vector<long long> operator()(vector<long long> a, vector<long long> b) {
        int sz = 1;
        while (sz < a.size() + b.size() - 1) sz <<= 1;
        while (a.size() < sz) a.push_back(0);
        while (b.size() < sz) b.push_back(0);
        ntt(a, sz), ntt(b, sz);
        vector<long long> c(sz);
        for (int i = 0; i < sz; ++i) c[i] = a[i] * b[i] % mod;
        intt(c, sz);
        while (c.size() && c.back() == 0) c.pop_back();
        return c;
    }

    vector<long long> operator()(vector<long long> a, int n) {
        int sz = 1;
        while (sz < n * a.size()) sz <<= 1;
        while (a.size() < sz) a.push_back(0);
        ntt(a, sz);
        for (int i = 0; i < sz; ++i) a[i] = fpow(a[i], n);
        intt(a, sz);
        while (a.size() && a.back() == 0) a.pop_back();
        return a;
    }
};

NTT<mod, root> conv;
```

1.6.3 原根

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

```
#define ll long long
#define IL inline
#define RG register
using namespace std;

ll prm[1000],tot,N,root;

ll Power(ll bs,ll js,ll MOD){
    ll S = 1,T = bs;
    while(js){
        if(js&1)S = S*T%MOD;
        T = T*T%MOD;
        js >>= 1;
    } return S;
}

IL ll GetRoot(RG ll n){
    RG ll tmp = n - 1 , tot = 0;
    for(RG ll i = 2; i <= sqrt(tmp); i ++){
        if(tmp%i==0){
            prm[++tot] = i;
            while(tmp%i==0)tmp /= i;
        }
    }
    if(tmp != 1)prm[++tot] = tmp;            //质因数分解
    for(RG ll g = 2; g <= n-1; g ++){
        bool flag = 1;
        for(RG int i = 1; i <= tot; i ++){     //检测是否符合条件
            if(Power(g,(n-1)/prm[i],n) == 1)
                { flag = 0; break; }
        }
        if(flag)return g;
    }return 0;                        //无解
}

int main(){
    cin >> N;
    root = GetRoot(N);
    cout<<root<<endl;
    return 0;
}
```

1.6.4 FWT

```
//$C_k=\sum_{i \oplus j=k} A_i B_j$
//FWT 完后需要先模一遍
template<typename T>
void fwt(ll a[], int n, T f) {
    for (int d = 1; d < n; d *= 2)
        for (int i = 0, t = d * 2; i < n; i += t)
            for(int j = 0; j < d; j ++)
                f(a[i + j], a[i + j + d]);
}

void AND(ll& a, ll& b) { a += b; }
void OR(ll& a, ll& b) { b += a; }
void XOR (ll& a, ll& b) {
    ll x = a, y = b;
    a = (x + y) % mod;
    b = (x - y + mod) % mod;
}
void rAND(ll& a, ll& b) { a -= b; }
void rOR(ll& a, ll& b) { b -= a; }
void rXOR(ll& a, ll& b) {
    static ll INV2 = (mod + 1) / 2;
    ll x = a, y = b;
    a = (x + y) * INV2 % mod;
    b = (x - y + mod) * INV2 % mod;
}

//FWT 子集卷积
a[popcount(x)][x] = A[x]
b[popcount(x)][x] = B[x]
fwt(a[i]) fwt(b[i])
c[i + j][x] += a[i][x] * b[j][x]
rfwt(c[i])
ans[x] = c[popcount(x)][x]
```

1.7 多项式

1.7.1 拉格朗日插值

```
typedef long long ll;

const int mod = 998244353;
const int maxn = 1e5 + 10;

int x[maxn], y[maxn];

int qp(int a, int n)
{
    ll ans = 1, base = a;
    for(; n; (base *= base) %= mod, n >>= 1) if(n & 1) (ans *= base) %= mod;
    return ans;
}

int lagrange(int n, int *x, int *y, int xi)
{
    int ans = 0;
    for(int i = 0; i <= n; i ++)
    {
        int s1 = 1, s2 = 1;
        for(int j = 0; j <= n; j ++) if(i != j)
        {
            s1 = 1ll * s1 * (xi - x[j]) % mod;
            s2 = 1ll * s2 * (x[i] - x[j]) % mod;
        }
        ans = (1ll * ans + 1ll * y[i] * s1 % mod * qp(s2, mod - 2) % mod) % mod;
    }
    return (ans + mod) % mod;
}
```

1.7.2 拉格朗日插值 (连续取值)

```
const int mod = 'edit';
const int maxn = 'edit';
```

```

int x[maxn], y[maxn];
int s1[maxn], s2[maxn], ifac[maxn];

//如果x的取值是连续一段，可以做到O(n)求解
int lagrange(int n, int *x, int *y, int xi)
{
    int ans = 0;
    s1[0] = (xi - x[0]) % mod, s2[n + 1] = 1;
    for(int i = 1; i <= n; i ++) s1[i] = 1ll * s1[i - 1] * (xi - x[i]) % mod;
    for(int i = n; i >= 0; i --) s2[i] = 1ll * s2[i + 1] * (xi - x[i]) % mod;
    ifac[0] = ifac[1] = 1;
    for(int i = 2; i <= n; i ++) ifac[i] = -1ll * mod / i * ifac[mod % i] % mod;
    for(int i = 2; i <= n; i ++) ifac[i] = 1ll * ifac[i] * ifac[i - 1] % mod;
    for(int i = 0; i <= n; i ++)
        (ans += 1ll * y[i] * (i == 0 ? 1 : s1[i - 1]) % mod * s2[i + 1] % mod * ifac[i] % mod * (((n - i) & 1) ? -1 : 1) * ifac[n - i] % mod) %= mod;
    return (ans + mod) % mod;
}
```

1.8 Others

1.8.1 BM

```
//Berlekamp-Massey
typedef vector<int> VI;
namespace linear_seq
{
#define rep(i,a,n) for (int i=a;i<n;i++)
#define SZ(x) ((int)(x).size())
#define pb(x) push_back(x)
    const ll mod=1e9+7;
    ll powmod(ll a,ll b){ll res=1;a%=mod; assert(b>=0); for(;b;b>>=1){if(b&1)res=res*a%mod;a=a*a%mod;}return res;}
    const int N=10010;
    ll res[N],base[N],_c[N],_md[N];
    vector<int> Md;
    void mul(ll *a,ll *b,int k)
    {
        rep(i,0,k+k) _c[i]=0;
        rep(i,0,k) if (a[i]) rep(j,0,k) _c[i+j]=(_c[i+j]+a[i]*b[j])%mod;
        for (int i=k+k-1;i>=k;i--) if (_c[i])
                rep(j,0,SZ(Md)) _c[i-k+Md[j]]=(_c[i-k+Md[j]]-_c[i]*_md[Md[j]])%mod;
        rep(i,0,k) a[i]=_c[i];
    }
    int solve(ll n,VI a,VI b){
        ll ans=0,pnt=0;
        int k=SZ(a);
        assert(SZ(a)==SZ(b));
        rep(i,0,k) _md[k-1-i]=-a[i];_md[k]=1;
        Md.clear();
        rep(i,0,k) if (_md[i]!=0) Md.push_back(i);
        rep(i,0,k) 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;
                rep(j,0,SZ(Md)) res[Md[j]]=(res[Md[j]]-res[k]*_md[Md[j]])%mod;
            }
        }
        rep(i,0,k) ans=(ans+res[i]*b[i])%mod;
        if (ans<0) ans+=mod;
        return ans;
    }
    VI BM(VI s){
        VI C(1,1),B(1,1);
        int L=0,m=1,b=1;
        rep(n,0,SZ(s)){
            ll d=0;
            rep(i,0,L+1) d=(d+(ll)C[i]*s[n-i])%mod;
            if(d==0) ++m;
            else if(2*L<=n){
                VI T=C;
                ll c=mod-d*powmod(b,mod-2)%mod;
                while (SZ(C)<SZ(B)+m) C.pb(0);
                rep(i,0,SZ(B)) 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*powmod(b,mod-2)%mod;
                while (SZ(C)<SZ(B)+m) C.pb(0);
                rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
                ++m;
            }
        }
        return C;
    }
    int gao(VI a,ll n)
    {
        VI c=BM(a);
        c.erase(c.begin());
        rep(i,0,SZ(c)) c[i]=(mod-c[i])%mod;
        return solve(n,c,VI(a.begin(),a.begin()+SZ(c)));
    }
};//linear_seq::gao(VI{},n-1)
```

1.8.2 exBM

```
// given first m items init[0..m-1] and coefficents trans[0..m-1] or
// given first 2 *m items init[0..2m-1], it will compute trans[0..m-1]
// for you. trans[0..m] should be given as that
//      init[m] = sum_{i=0}^{m-1} init[i] * trans[i]
struct LinearRecurrence
{
    using int64 = long long;
    using vec = std::vector<int64>;

    static void extand(vec& a, size_t d, int64 value = 0)
    {
        if (d <= a.size()) return;
        a.resize(d, value);
    }
    static vec BerlekampMassey(const vec& s, int64 mod)
    {
        std::function<int64(int64)> inverse = [&](int64 a) {
            return a == 1 ? 1 : (int64)(mod - mod / a) * inverse(mod % a) % mod;
        };
        vec A = {1}, B = {1};
        int64 b = s[0];
        for (size_t i = 1, m = 1; i < s.size(); ++i, m++)
        {
            int64 d = 0;
            for (size_t j = 0; j < A.size(); ++j)
            {
                d += A[j] * s[i - j] % mod;
            }
            if (!(d %= mod)) continue;
            if (2 * (A.size() - 1) <= i)
            {
                auto temp = A;
                extand(A, B.size() + m);
                int64 coef = d * inverse(b) % mod;
                for (size_t j = 0; j < B.size(); ++j)
                {
                    A[j + m] -= coef * B[j] % mod;
                    if (A[j + m] < 0) A[j + m] += mod;
                }
                B = temp, b = d, m = 0;
            }
            else
            {
                extand(A, B.size() + m);
                int64 coef = d * inverse(b) % mod;
                for (size_t j = 0; j < B.size(); ++j)
                {
                    A[j + m] -= coef * B[j] % mod;
                    if (A[j + m] < 0) A[j + m] += mod;
                }
            }
        }
        return A;
    }
    static void exgcd(int64 a, int64 b, int64& g, int64& x, int64& y)
    {
        if (!b)
            x = 1, y = 0, g = a;
        else
        {
            exgcd(b, a % b, g, y, x);
            y -= x * (a / b);
        }
    }
    static int64 crt(const vec& c, const vec& m)
    {
        int n = c.size();
        int64 M = 1, ans = 0;
        for (int i = 0; i < n; ++i) M *= m[i];
        for (int i = 0; i < n; ++i)
        {
            int64 x, y, g, tm = M / m[i];
            exgcd(tm, m[i], g, x, y);
            ans = (ans + tm * x * c[i] % M) % M;
        }
        return (ans + M) % M;
    }
```

```cpp
    static vec ReedsSloane(const vec& s, int64 mod)
    {
        auto inverse = [](int64 a, int64 m) {
            int64 d, x, y;
            exgcd(a, m, d, x, y);
            return d == 1 ? (x % m + m) % m : -1;
        };
        auto L = [](const vec& a, const vec& b) {
            int da = (a.size() > 1 || (a.size() == 1 && a[0])) ? a.size() - 1 : -1000;
            int db = (b.size() > 1 || (b.size() == 1 && b[0])) ? b.size() - 1 : -1000;
            return std::max(da, db + 1);
        };
        auto prime_power = [&](const vec& s, int64 mod, int64 p, int64 e) {
            // linear feedback shift register mod p^e, p is prime
            std::vector<vec> a(e), b(e), an(e), bn(e), ao(e), bo(e);
            vec t(e), u(e), r(e), to(e, 1), uo(e), pw(e + 1);
            ;
            pw[0] = 1;
            for (int i = pw[0] = 1; i <= e; ++i) pw[i] = pw[i - 1] * p;
            for (int64 i = 0; i < e; ++i)
            {
                a[i] = {pw[i]}, an[i] = {pw[i]};
                b[i] = {0}, bn[i] = {s[0] * pw[i] % mod};
                t[i] = s[0] * pw[i] % mod;
                if (t[i] == 0)
                {
                    t[i] = 1, u[i] = e;
                }
                else
                {
                    for (u[i] = 0; t[i] % p == 0; t[i] /= p, ++u[i])
                        ;
                }
            }
            for (size_t k = 1; k < s.size(); ++k)
            {
                for (int g = 0; g < e; ++g)
                {
                    if (L(an[g], bn[g]) > L(a[g], b[g]))
                    {
                        ao[g] = a[e - 1 - u[g]];
                        bo[g] = b[e - 1 - u[g]];
                        to[g] = t[e - 1 - u[g]];
                        uo[g] = u[e - 1 - u[g]];
                        r[g] = k - 1;
                    }
                }
                a = an, b = bn;
                for (int o = 0; o < e; ++o)
                {
                    int64 d = 0;
                    for (size_t i = 0; i < a[o].size() && i <= k; ++i)
                    {
                        d = (d + a[o][i] * s[k - i]) % mod;
                    }
                    if (d == 0)
                    {
                        t[o] = 1, u[o] = e;
                    }
                    else
                    {
                        for (u[o] = 0, t[o] = d; t[o] % p == 0; t[o] /= p, ++u[o])
                            ;
                        int g = e - 1 - u[o];
                        if (L(a[g], b[g]) == 0)
                        {
                            extand(bn[o], k + 1);
                            bn[o][k] = (bn[o][k] + d) % mod;
                        }
                        else
                        {
                            int64 coef = t[o] * inverse(to[g], mod) % mod * pw[u[o] - uo[g]] % mod;
                            int m = k - r[g];
                            extand(an[o], ao[g].size() + m);
                            extand(bn[o], bo[g].size() + m);
                            for (size_t i = 0; i < ao[g].size(); ++i)
                            {
                                an[o][i + m] -= coef * ao[g][i] % mod;
                                if (an[o][i + m] < 0) an[o][i + m] += mod;
                            }
                            while (an[o].size() && an[o].back() == 0) an[o].pop_back();
                            for (size_t i = 0; i < bo[g].size(); ++i)
                            {
                                bn[o][i + m] -= coef * bo[g][i] % mod;
                                if (bn[o][i + m] < 0) bn[o][i + m] -= mod;
                            }
                            while (bn[o].size() && bn[o].back() == 0) bn[o].pop_back();
                        }
                    }
                }
            }
            return std::make_pair(an[0], bn[0]);
        };

        std::vector<std::tuple<int64, int64, int>> fac;
        for (int64 i = 2; i * i <= mod; ++i)
        {
            if (mod % i == 0)
            {
                int64 cnt = 0, pw = 1;
                while (mod % i == 0) mod /= i, ++cnt, pw *= i;
                fac.emplace_back(pw, i, cnt);
            }
        }
        if (mod > 1) fac.emplace_back(mod, mod, 1);
        std::vector<vec> as;
        size_t n = 0;
        for (auto&& x : fac)
        {
            int64 mod, p, e;
            vec a, b;
            std::tie(mod, p, e) = x;
            auto ss = s;
            for (auto&& x : ss) x %= mod;
            std::tie(a, b) = prime_power(ss, mod, p, e);
            as.emplace_back(a);
            n = std::max(n, a.size());
        }
        vec a(n), c(as.size()), m(as.size());
        for (size_t i = 0; i < n; ++i)
        {
            for (size_t j = 0; j < as.size(); ++j)
            {
                m[j] = std::get<0>(fac[j]);
                c[j] = i < as[j].size() ? as[j][i] : 0;
            }
            a[i] = crt(c, m);
        }
        return a;
    }

    LinearRecurrence(const vec& s, const vec& c, int64 mod) : init(s), trans(c), mod(mod), m(s.size()) {}
    LinearRecurrence(const vec& s, int64 mod, bool is_prime = true) : mod(mod)
    {
        vec A;
        if (is_prime)
            A = BerlekampMassey(s, mod);
        else
            A = ReedsSloane(s, mod);
        if (A.empty()) A = {0};
        m = A.size() - 1;
        trans.resize(m);
        for (int i = 0; i < m; ++i)
        {
            trans[i] = (mod - A[i + 1]) % mod;
        }
        std::reverse(trans.begin(), trans.end());
        init = {s.begin(), s.begin() + m};
    }
    int64 calc(int64 n)
    {
        if (mod == 1) return 0;
        if (n < m) return init[n];
        vec v(m), u(m << 1);
        int msk = !!n;
        for (int64 m = n; m > 1; m >>= 1) msk <<= 1;
        v[0] = 1 % mod;
        for (int x = 0; msk; msk >>= 1, x <<= 1)
        {
            std::fill_n(u.begin(), m * 2, 0);
            x |= !!(n & msk);
            if (x < m)
                u[x] = 1 % mod;
            else
            { // can be optimized by fft/ntt
                for (int i = 0; i < m; ++i)
                {
                    for (int j = 0, t = i + (x & 1); j < m; ++j, ++t)
                    {
                        u[t] = (u[t] + v[i] * v[j]) % mod;
                    }
                }
                for (int i = m * 2 - 1; i >= m; --i)
                {
                    for (int j = 0, t = i - m; j < m; ++j, ++t)
                    {
                        u[t] = (u[t] + trans[j] * u[i]) % mod;
                    }
                }
            }
            v = {u.begin(), u.begin() + m};
        }
        int64 ret = 0;
        for (int i = 0; i < m; ++i)
        {
            ret = (ret + v[i] * init[i]) % mod;
        }
        return ret;
    }

    vec init, trans;
    int64 mod;
    int m;
};
```

1.8.3 杜教筛

```
#include <bits/stdc++.h>
#include <tr1/unordered_map>

using namespace std;
typedef long long ll;

const int N = 5e6;

bool vis[N + 1];
int mu[N + 1], sumu[N + 1], prim[N + 1], cnt;

tr1::unordered_map<int, int> Smu;

void get_mu(int n)
{
    mu[1] = 1;
    for(int i = 2; i <= n; i ++)
    {
        if(!vis[i]) { prim[++ cnt] = i; mu[i] = -1; }
        for(int j = 1; j <= cnt && prim[j] * i <= n; j ++)
        {
            vis[prim[j] * i] = 1;
            if(i % prim[j] == 0) break;
            else mu[i * prim[j]] = -mu[i];
        }
    }
    for(int i = 1; i <= n; i ++) sumu[i] = sumu[i - 1] + mu[i];
}

int phi[N + 1]; ll sumphi[N + 1];
tr1::unordered_map<ll, ll> Sphi;

void get(int n)
{
    phi[1] = mu[1] = 1;
    for(int i = 2; i <= n; i ++)
    {
        if(!vis[i])
        {
            prim[++ cnt] = i;
            mu[i] = -1; phi[i] = i - 1;
        }
        for(int j = 1; j <= cnt && prim[j] * i <= n; j ++)
        {
            vis[i * prim[j]] = 1;
            if(i % prim[j] == 0)
            {
                phi[i * prim[j]] = phi[i] * prim[j];
                break;
            }
            else mu[i * prim[j]] = -mu[i], phi[i * prim[j]] = phi[i] * (prim[j] - 1);
        }

    }
    for(int i = 1; i <= n; i ++) sumu[i] = sumu[i - 1] + mu[i], sumphi[i] = sumphi[i - 1] + phi[i];
}

ll getSum_mu(int x)
{
    if(x <= N) return sumu[x];
    if(Smu[x]) return Smu[x];
    int ans = 1;
    for(int l = 2, r; l >= 0 && l <= x && r < 2147483647; l = r + 1)
    {
        r = x / (x / l);
        ans -= (r - l + 1) * getSum_mu(x / l);
    }
    return Smu[x] = ans;
}

ll getSum_phi(ll x)
{
    if(x <= N) return sumphi[x];
    if(Sphi[x]) return Sphi[x];
    ll ans = x * (x + 1) / 2;
    for(ll l = 2, r; l <= x; l = r + 1)
    {
        r = x / (x / l);
        ans -= (r - l + 1) * getSum_phi(x / l);
    }
    return Sphi[x] = ans;
}
```

1.8.4 欧拉降幂

```
const int maxn = 1e7+50;

int prim[maxn], vis[maxn];
int tot, phi[maxn];
struct node {
    ll res;
    bool v;
};

node qpow(ll A, ll B, ll C) {
    ll re = 1;
    bool flag = true;
    while (B) {
        if (B & 1) {
            if ((re *= A) >= C) flag = 0;
            re = re % C;
        }
        B = B >> 1;
        if (B) {
            if (A >= C) flag = 0;
            A %= C;
            if ((A *= A) >= C) flag = 0;
            A %= C;
        }
    }
    return node{re, flag};
}

void init(int n) {
    phi[1] = 1;
    for (int i = 2; i <= n; i++) {
        if (!vis[i]) {
            prim[++tot] = i;
            phi[i] = i - 1;
        }
        for (int j = 1; j <= tot && prim[j] * i <= n; j++) {
            vis[i * prim[j]] = 1;
            if (i % prim[j] == 0) {
                phi[i * prim[j]] = phi[i] * prim[j];
                break;
            } else phi[i * prim[j]] = phi[i] * (prim[j] - 1);
        }

    }
}


inline ll Euler(ll x) {
    if (x <= maxn) return phi[x];
    return 0;
}

node f(ll a, ll k, ll p) {
    if (p == 1) return node{0, 0};
    if (k == 0) return node{a % p, a < p};
    ll ep = Euler(p);
    node tmp = f(a, k - 1, ep);
    if (__gcd(a, p) == 1)return qpow(a, tmp.res, p);
    if (!tmp.v) {
        tmp.res += ep;
    }
    return qpow(a, tmp.res, p);
}

int main() {
    ll a, k, p;
    init(1e7+2);
    int T;
    scanf("%d", &T);
    for (int kase = 1; kase <= T; ++kase) {
        // k次a次方模p的值
        scanf("%lld%lld%lld", &a, &k, &p);
        if (k == 0) printf("%lld\n", 1 % p);
        else printf("%lld\n", f(a, k - 1, p).res);
    }
    return 0;
}
```

1.8.5 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. 部分错排公式：n + m 个数中 m 个数必须错排求排列数
   - (a) 1 dp[i] = n*dp[i−1]+(i−1)*(dp[i−1]+dp[i−2]);
   - (b) 2 dp[0] = n!;
   - (c) 3 dp[1] = n*n!;
   - (d) dp[m] 为所求解
6. 海伦公式：$S = \sqrt{p(p-a)(p-b)(p-c)}$，其中 $p = \frac{(a+b+c)}{2}$
7. 求 $C(n,k)$ 中素因子 P 的个数：把 n 转化为 P 进制，并记它每个位上的和为 $S1$ 把 $n-k$，k 做同样的处理，得到 $S2$，$S3$ 则答案为：$\frac{S2+S3-S1}{P-1}$
8. 威尔逊定理：$p\ is\ prime\ \Rightarrow (p-1)! \equiv -1 \pmod{p}$
9. 欧拉定理：$\gcd(a,n) = 1 \Rightarrow a^{\varphi(n)} \equiv 1 \pmod{n}$
10. 欧拉定理推广：$\gcd(n,p) = 1 \Rightarrow a^n \equiv a^{n\%\varphi(p)} \pmod{p}$
11. 模的幂公式：$a^n \pmod{m} = \begin{cases} a^n \mod m & n < \varphi(m) \\ a^{n\%\varphi(m)+\varphi(m)} \mod m & n \ge \varphi(m) \end{cases}$
12. 素数定理：对于不大于 n 的素数个数 $\pi(n)$，$\lim\limits_{n\to\infty} \pi(n) = \frac{n}{\ln n}$
13. 位数公式：正整数 x 的位数 $N = \log_{10}(n) + 1$
14. 斯特灵公式 $n! \approx \sqrt{2\pi n}(\frac{n}{e})^n$
15. 设 $a > 1, m, n > 0$, 则 $\gcd(a^m - 1, a^n - 1) = a^{\gcd(m,n)} - 1$
16. 设 $a > b, \gcd(a,b) = 1$, 则 $\gcd(a^m - b^m, a^n - b^n) = a^{\gcd(m,n)} - b^{\gcd(m,n)}$

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

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

17. 求和公式：
   - (a) $\sum k = \frac{n(n+1)}{2}$
   - (b) $\sum 2k - 1 = n^2$
   - (c) $\sum k^2 = \frac{n(n+1)(2n+1)}{6}$
   - (d) $\sum (2k-1)^2 = \frac{n(4n^2-1)}{3}$

(e) $\sum k^3 = (\frac{n(n+1)}{2})^2$

(f) $\sum (2k-1)^3 = n^2(2n^2-1)$

(g) $\sum k^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$

(h) $\sum k^5 = \frac{n^2(n+1)^2(2n^2+2n-1)}{12}$

(i) $\sum k(k+1) = \frac{n(n+1)(n+2)}{3}$

(j) $\sum k(k+1)(k+2) = \frac{n(n+1)(n+2)(n+3)}{4}$

(k) $\sum k(k+1)(k+2)(k+3) = \frac{n(n+1)(n+2)(n+3)(n+4)}{5}$

18. 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 $N = \frac{(m-1)(n-1)}{2}$

19. $(n+1)lcm(C_n^0, C_n^1, ..., C_n^{n-1}, C_n^n) = lcm(1, 2, ..., n+1)$

20. 若 p 为素数，则 $(x+y+...+w)^p \equiv x^p + y^p + ... + w^p \pmod p$

21. 卡特兰数：1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012
    $h(0) = h(1) = 1, h(n) = \frac{(4n-2)h(n-1)}{n+1} = \frac{C_{2n}^n}{n+1} = C_{2n}^n - C_{2n}^{n-1}$

22. 伯努利数：$B_n = -\frac{1}{n+1}\sum_{i=0}^{n-1} C_{n+1}^i B_i$

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

23. 二项式反演：

$$f_n = \sum_{i=0}^{n} (-1)^i \binom{n}{i} g_i \Leftrightarrow g_n = \sum_{i=0}^{n} (-1)^i \binom{n}{i} f_i$$

$$f_n = \sum_{i=0}^{n} \binom{n}{i} g_i \Leftrightarrow g_n = \sum_{i=0}^{n} (-1)^{n-i} \binom{n}{i} f_i$$

24. 莫比乌斯反演:

    (a) 令 $f(d) = \sum_{i=1}^{n}\sum_{j=1}^{m}[gcd(i,j) = d]$

    (b) $F(n) = \sum_{n|d} f(d) = \lfloor \frac{N}{n} \rfloor \lfloor \frac{M}{n} \rfloor$

    (c) 有 $f(n) = \sum_{n|d} \mu(\lfloor \frac{d}{n} \rfloor) F(d)$

    (d) $\phi(n) = \sum_{d|n} d * \mu(n/d)$

25. 2 的 n 次方，在 pow 时可以精确输出最大 $2^1$023，pow(2,1023)

26. 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

1.8.6 博弈

```
// bash 博弈，n个物品，轮流取[1,m]个物品，无法取则失败
// 当且仅当 n = (m + 1) * r 时先手败
// Nim 博弈：每轮从若干堆石子中的一堆取走若干颗。先手必胜条件为石子数量异或和非零
```

2 Graph Theory

2.1 路径

2.1.1 Dijkstra

```
const int maxn = 1e5 + 10;
const int inf = 0x3f3f3f3f;

int head[maxn], dis[maxn], cnt, n;

struct Edge { int nex,to,w; }edge[20*maxn];

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

void dijkstra(int s)
{
    priority_queue<pair<int, int>, vector<pair<int, int> >, greater<pair<int, int> > > que;
    memset(dis, 0x3f, sizeof dis);
    que.push({0, s}); dis[s] = 0;
    while(!que.empty())
    {
        auto f = que.top(); que.pop();
        int u = f.second, d = f.first;
        if(d != dis[u]) continue;
        for(int i = head[u]; ~i; i = edge[i].nex)
        {
            int v = edge[i].to, w = edge[i].w;
            if(dis[u] + w < dis[v])
            {
                dis[v] = dis[u] + w;
                que.push({dis[v], v});
            }
        }
    }
}
```

2.1.2 Euler Path

```
int S[N << 1], top;
Edge edges[N << 1];
set<int> G[N];

void DFS(int u) {
    S[top++] = u;
    for (int eid: G[u])
    {
        int v = edges[eid].get_other(u);
        G[u].erase(eid);
        G[v].erase(eid);
        DFS(v);
        return;
    }
}

void fleury(int start)
{
    int u = start;
    top = 0; path.clear();
    S[top++] = u;
    while (top)
    {
        u = S[--top];
        if (!G[u].empty())
            DFS(u);
        else path.push_back(u);
    }
}
```

2.1.3 K shortest Path(Astar)

```
const int inf = 0x3f3f3f3f;
const int maxn = 1000 + 10;
const int maxm = 100000 + 10;

int n, k, cnt, head[maxn], revhead[maxn], dis[maxn];
bool vis[maxn];

struct node { int v, w, nex; } edge[maxm], revedge[maxm];

void init()
{
    cnt = 0;
    memset(head, 0xff, sizeof head);
    memset(revhead, 0xff, sizeof revhead);
}

void add(int u, int v, int w)
{
    edge[cnt].v = v, revedge[cnt].v = u;
    edge[cnt].w = revedge[cnt].w = w;
    edge[cnt].nex = head[u];
    revedge[cnt].nex = revhead[v];
    head[u] = revhead[v] = cnt;
    cnt++;
}

void spfa(int src)      //建立反向图，求图中所有点到终点的最短路径
{
    for (int i = 1; i <= n; i++) dis[i] = inf;
    memset(vis, false, sizeof vis);
    vis[src] = 0;
    queue<int> que;
    que.push(src);
    dis[src] = 0;
    while (!que.empty())
    {
        int u = que.front();
        que.pop();
        vis[u] = false;
        for (int i = revhead[u]; ~i; i = revedge[i].nex)
        {
            int v = revedge[i].v, w = revedge[i].w;
            if (dis[v] > dis[u] + w)
            {
                dis[v] = dis[u] + w;
                if (!vis[v])
                {
                    que.push(v);
                    vis[v] = true;
                }
            }
        }
    }
}

struct A
{
    int f, g, h;    //f(n),g(n),h(n)函数
    int id;        //当前点的编号
    bool operator<(const A a) const
    {        //定义比较函数
        if (a.f == f) return a.g < g;
        return a.f < f;
    }
};

int Astar(int src, int des)
{
    int cnt = 0;
    priority_queue <A> Q;
    if (src == des) k++;   //如果起点即为终点
    if (dis[src] == inf) return -1;      //如果起点不能到达终点
    A st, now, tmp;
    st.id = src, st.g = 0, st.f = st.g + dis[src];    //定义起始节点
    Q.push(st);
    while (!Q.empty())
    {
        now = Q.top();
        Q.pop();
        if (now.id == des)     //如果当前节点为终点
        {
            cnt++;
            if (cnt == k) return now.g;    //找到第k短路
        }
        for (int i = head[now.id]; ~i; i = edge[i].nex)
        {
            tmp.id = edge[i].v;
            tmp.g = now.g + edge[i].w;     //到该点的实际花费
            tmp.f = tmp.g + dis[tmp.id];   //到最终状态的估计花费
            Q.push(tmp);
        }
    }
    return -1;  //路径总数小于k
}

int main()
{
    int m, s, t, u, v, w;
    while (scanf("%d%d", &n, &m) != EOF)
    {
        init();
        while (m--)
        {
            scanf("%d%d%d", &u, &v, &w);
            add(u, v, w);
        }
        scanf("%d%d%d", &s, &t, &k);
        spfa(t);    //求所有点到终点的最短路
        printf("%d\n", Astar(s, t));
    }
    return 0;
}
```

2.1.4 K shortest Path(可持久化可并堆)

```
#include <bits/stdc++.h>
#include<ext/pb_ds/priority_queue.hpp>

using namespace std;

const int N = '';
const int M = '';
const int logM = 20;
const int inf = 0x3f3f3f3f;

int n, m, k, S, T;

struct Edge{ int nex, to, w; };

struct Graph
{
    int head[N], cnt;
    Edge edge[M];
    void init(int n) { for(int i = 0; i <= n; i ++) head[i] = 0; cnt = 0; }
    void addedge(int u, int v, int val) { edge[++ cnt].nex = head[u], edge[cnt].to = v, edge[cnt].w = val, head[u] = cnt; }
}g, rg;

int dis[N];

void dijkstra()
{
    priority_queue<pair<int, int>, vector<pair<int, int> >, greater<pair<int, int> > > que;
    memset(dis, inf, sizeof dis);
    que.push({0, T}); dis[T] = 0;
    const int *head = rg.head; const Edge *edge = rg.edge;
    while(!que.empty())
    {
        auto f = que.top(); que.pop();
        int u = f.second, d = f.first;
        if(d != dis[u]) continue;
        for(int i = head[u]; i; i = edge[i].nex)
        {
            int v = edge[i].to, w = edge[i].w;
            if(dis[u] + w < dis[v]) { dis[v] = dis[u] + w; que.push({dis[v], v}); }
        }
    }
```

```cpp
}

bool tree_edge[M], vis[N];
int fa[N], st[N], top;

void dfs(int u)
{
    vis[u] = true;
    st[++ top] = u;
    for(int i = rg.head[u]; i; i = rg.edge[i].nex)
    {
        int v = rg.edge[i].to;
        if(!vis[v] && dis[v] == dis[u] + rg.edge[i].w)
        {
            fa[v] = u;
            tree_edge[i] = true;
            dfs(v);
        }
    }
}

namespace LT
{
    int son[M * logM][2];
    int ht[M * logM], val[M * logM], id[M * logM];
    int tot;

    int newnode(int _val, int _id, int _dis = 0)
    {
        int now = ++ tot;
        val[now] = _val, id[now] = _id;
        ht[now] = _dis, son[now][0] = son[now][1] = 0;
        return now;
    }

    int _copy(int ori)
    {
        int now = ++tot;
        val[now] = val[ori], id[now] = id[ori];
        ht[now] = ht[ori], son[now][0] = son[ori][0], son[now][1] = son[ori][1];
        return now;
    }

    int merge(int a, int b)
    {
        if(!a || !b) return a | b;
        if(val[a] > val[b]) swap(a, b);
        int now = _copy(a);
        son[now][1] = merge(son[now][1], b);
        if(ht[son[now][0]] < ht[son[now][1]]) swap(son[now][0], son[now][1]);
        ht[now] = ht[son[now][1]] + 1;
        return now;
    }

    void insert(int &rt, int val, int id) { rt = merge(newnode(val, id), rt); }
}

int rt[M];

```

```cpp
void build_heap()
{
    for(int i = 1; i <= top; i ++)
    {
        int u = st[i];
        rt[u] = rt[fa[u]];
        for(int i = g.head[u]; i; i = g.edge[i].nex)
        {
            int v = g.edge[i].to;
            if(!tree_edge[i] && dis[v] != inf) LT::insert(rt[u], dis[v] - dis[u] + g.edge[i].w, v);
        }
    }
}

int solve(int k)
{
    if(k == 1) return dis[S];
    __gnu_pbds::priority_queue<pair<int, int>, greater<pair<int, int> > > que;
    que.push({dis[S] + LT::val[rt[S]], rt[S]});
    while(!que.empty())
    {
        pair<int, int> f = que.top(); que.pop();
        if((--k) == 1) return f.first;
        int v = f.first, u = f.second;
        int lc = LT::son[u][0], rc = LT::son[u][1], o = LT::id[u];
        if(rt[o]) que.push({v + LT::val[rt[o]], rt[o]});
        if(lc) que.push({v + LT::val[lc] - LT::val[u], lc});
        if(rc) que.push({v + LT::val[rc] - LT::val[u], rc});
    }
    return -1;
}

void init()
{
    g.init(n), rg.init(n);
    memset(rt, 0, sizeof rt);
    memset(tree_edge, 0, sizeof tree_edge);
    top = LT::tot = 0;
}

void getans()
{
    //input S-T
    init();
    dijkstra();
    dfs(T);
    build_heap();
    cout << solve(k);
}
```

2.2 生成树

2.2.1 Kruskal

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

int n, m, pre[maxn];
struct edge {int u, v, w; } es[maxn];
int Find(int x) { return x == pre[x] ? x : pre[x] = Find(pre[x]); }
```

```
bool cmp(const edge &x, const edge &y) { return x.cost < y.cost; }

int kruskal()
{
    sort(es, es + m, cmp);
    int res = 0;
    for(int i = 0; i < m; i ++)
    {
        int fx = Find(es[i].u), fy = Find(es[i].v);
        if(fx != fy) pre[fx] = fy, res += es[i].cost;
    }
    return res;
}
```

2.2.2 Prim

```
const int maxn = 1000 + 10;
const int inf = 0x3f3f3f3f;

int n, mp[maxn][maxn], cost[maxn];
bool vis[maxn];

int prim()
{
    for(int i = 0; i < n; i ++) cost[u] = inf, vis[u] = false;
    int res = 0; cost[0] = 0;
    for(;;)
    {
        int v = -1;
        for(int u = 0; u < n; u ++)
            if(!vis[u] && (v == -1 || cost[u] < cost[v])) v = u;
        if(v == -1) break;
        res += cost[v];
        vis[v] = true;
        for(int u = 0; u < n; u ++) cost[u] = min(cost[u], mp[v][u]);
    }
    return res;
}
```

2.2.3 最小树形图

```
const int INF = 0x3f3f3f3f;
const int maxn = 10000;
const int maxm = 10000;

struct Edge{int u,v,cost; } edge[maxm];

int pre[maxn], id[maxn], vis[maxn], in[maxn];

int zhuliu(int root, int n, int m)
{
    int res=0, u, v;
    for(;;)
    {
        for(int i=0; i<n; i++) in[i] = INF;
        for(int i=0; i<m; i++) if(edge[i].u != edge[i].v && edge[i].cost < in[edge[i].v])
        {
            pre[edge[i].v] = edge[i].u;
            in[edge[i].v] = edge[i].cost;
        }
        for(int i=0; i<n; i++) if(i != root && in[i] ==INF) return -1;
        int tn=0;
        memset(id, 0xff, sizeof id);
        memset(vis, 0xff, sizeof vis);
        in[root] = 0;
        for(int i=0; i<n;i++)
        {
            res += in[i];
            v = i;
            while( vis[v] != i && id[v] == -1 && v!= root) vis[v] = i, v = pre[v];
            if(v != root && id[v] == -1)
            {
                for(int u = pre[v]; u != v; u = pre[u]) id[u] = tn;
                id[v] = tn++;
            }
        }
        if(tn == 0) break;
        for(int i=0; i<n; i++) if(id[i] == -1) id[i] = tn++;
        for(int i=0; i<m; )
        {
            v = edge[i].v;
            edge[i].u = id[edge[i].u];
            edge[i].v = id[edge[i].v];
            if(edge[i].u != edge[i].v) edge[i++].cost -= in[v];
            else swap(edge[i], edge[--m]);
        }
        n = tn;
        root = id[root];
    }
    return res;
}
```

2.2.4 Matrix Tree

```
const int N = 305;
const int mod = 1e9 + 7;

int n, m, a[N][N];

int Gauss(int n) {
    int ans = 1;
    for (int i = 1; i <= n; i++) {
        for (int k = i + 1; k <= n; k++) {
            while (a[k][i]) {
                int d = a[i][i] / a[k][i];
                for (int j = i; j <= n; j++) {
                    a[i][j] = (a[i][j] - 1LL * d * a[k][j] % mod + mod) % mod;
                }
                std::swap(a[i], a[k]);
                ans = - ans;
            }
        }
        ans = 1LL * ans * a[i][i] % mod;
    }
    return (ans % mod + mod) % mod;
}
```

```
int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= m; i++) {
        int u, v;
        scanf("%d%d", &u, &v);
        a[u][v]--, a[v][u]--;
        a[u][u]++, a[v][v]++;
    }
    printf("%d\n", Gauss(n - 1));
    return 0;
}
```

2.2.5 Steiner Tree

```
/*BZOJ:4774
无向图G从1-n进行编号，选择一些边，使对于1<=i<=d,i号点和n-i+1号点连通，最小化选出的所有边权值和。
1. 枚举子树形态 $dp[S][i] = min(dp[s]+dp[S \ xor \  s])$
2. 按照边进行松弛 $dp[S][i] = min(dp[S][j]+w[j][i])$
其中$S$为选取的子集，$s$ 和$S\ xor\ s$为$S$的状态划分。第二类转移方程可以通过跑一次最短路进行松弛。
本题需要再做一次子集dp，因为不成对的点可能不连通。
*/
#include <bits/stdc++.h>

using namespace std;

const int maxn = 1e4 + 10;
const int inf = 0x3f3f3f3f;

int head[maxn], cnt;
struct Edge {int nex, to, w; }edge[maxn<<1];

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

int f[1<<10][maxn], ans[20];
bool in[maxn];

queue<int> que;

void spfa(int S)
{
    while(!que.empty())
    {
        int u = que.front(); que.pop();
        in[u] = false;
        for(int i = head[u]; ~i; i = edge[i].nex)
        {
            int v = edge[i].to;
            if(f[S][v] > f[S][u] + edge[i].w)
            {
                f[S][v] = f[S][u] + edge[i].w;
                if(!in[v]) que.push(v), in[v] = true;
            }
        }
    }
}

int Steiner_Tree(int n, int d)
{
    memset(f, 0x3f, sizeof f);
    for(int i = 1; i <= d; i++)
        f[1 << (i - 1)][i] = f[1 << (d + i - 1)][n - i + 1] = 0;
    int lim = 1<<(d<<1);
    for(int S = 1; S < lim; S++)
    {
        for(int i = 1; i <= n; i++)
        {
            for(int s = (S - 1) & S; s; s = (s - 1) & S)
                f[S][i] = min(f[S][i], f[s][i] + f[S ^ s][i]);
            if(f[S][i] != inf) que.push(i), in[i] = true;
        }
        spfa(S);
    }
    lim = 1<<d;
    memset(ans, 0x3f, sizeof ans);
    for(int S = 1; S < lim; S++)
        for(int i = 1; i <= n; i++)
            ans[S] = min(ans[S], f[S^(S<<d)][i]);
    for(int S = 1; S < lim; S++)
        for(int s = (S - 1) & S; s; s = (s - 1) & S)
            ans[S] = min(ans[S], ans[s] + ans[S ^ s]);
    return ans[lim - 1] == inf ? -1 : ans[lim - 1];
}

int main()
{
    int n, m, d, u, v, w;
    scanf("%d%d%d", &n, &m, &d);
    memset(head, 0xff, sizeof head);
    while(m--)
    {
        scanf("%d%d%d", &u, &v, &w);
        add(u, v, w);
        add(v, u, w);
    }
    printf("%d\n", Steiner_Tree(n, d));
    return 0;
}
```

2.3 连通性

2.3.1 割点

```
const int maxn = 1e4 + 10;

vector<int> edge[maxn];
int n, dfn[maxn], low[maxn], cnt = 0;
bool vis[maxn], cut[maxn];

void Tarjan(int u, int fa)
{
    dfn[u] = low[u] = ++cnt;
    vis[u] = true;
    int children = 0;
    for (int i = 0; i < edge[u].size(); i++)
    {
        int v = edge[u][i];
        if (v != fa && vis[v])
            low[u] = min(low[u], dfn[v]);
        else if (!vis[v])
        {
            Tarjan(v, u);
            children++;
            low[u] = min(low[u], low[v]);
            if (fa == -1 && children > 1) //若u是根节点且子节点数大于1
                cut[u] = true;    //u是割点
            else if (fa != -1 && low[v] >= dfn[u])    //若u不是根节点且v不能访问到u的父节点
                cut[u] = true;    //u是割点
        }
    }
}
```

2.3.2 桥

```
const int maxn = 1e4 + 10;

vector<int> edge[maxn];
int n, dfn[maxn], low[maxn], father[maxn], cnt = 0;
bool bridge[maxn][maxn];

void Tarjan(int u, int fa)
{
    dfn[u] = low[u] = ++cnt;
    for (int i = 0; i < edge[u].size(); i++)
    {
        int v = edge[u][i];
        if (!dfn[v])    //未访问节点v
        {
            Tarjan(v, u);
            low[u] = min(low[u], low[v]);
            if (low[v] > dfn[u]) //节点v到达祖先必须经过(u,v)
                bridge[u][v] = bridge[v][u] = true;    //(u,v)是桥
        }
        else if (fa != v)    //u的父节点不是v，(u,v)不存在重边
            low[u] = min(low[u], dfn[v]);
    }
}
```

2.3.3 强连通分量

```
const int maxn=1000+10;

vector<int> edge[maxn];

int dfn[maxn], low[maxn];
int stack[maxn], index, tot;
int belong[maxn], inde[maxn], outde[maxn], scc;
bool vis[maxn];

void add(int u, int v)
```

```
{
    edge[u].push_back(v);
    edge[v].push_back(u);
}

void Tarjan(int u)
{
    dfn[u] = low[u] = ++tot;
    stack[++index] = u;
    vis[u] = true;
    int v;
    for(int i = 0;i < edge[u].size(); i++)
    {
        v=edge[u][i];
        if(!dfn[v])
        {
            Tarjan(v);
            low[u] = min(low[v], low[u]);
        }
        else if(vis[v]) low[u] = min(low[v], dfn[u]);
    }
    if(dfn[u] == low[u])
    {
        scc++;
        do
        {
            v = stack[index--];
            vis[v] = false;
            belong[v] = scc;
        }while(v != u);
    }
}
```

2.3.4 点双联通分量

```
const int maxn = 10000 + 10;

struct Edge{ int u, v; };
vector<int> G[maxn], bcc[maxn];

int dfn[maxn], low[maxn], bccno[maxn], idx, bcc_cnt, bridge;
bool iscut[maxn];

stack<Edge> st;

void dfs(int u, int pre)
{
    dfn[u] = low[u] = ++idx;
    int child = 0;
    for(auto v : G[u])
    {
        if(v == pre) continue;
        if(!dfn[v])
        {
            child ++;
            st.push({u, v});
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            if(low[v] >= dfn[u])
            {
                iscut[u] = true;
                bcc[++bcc_cnt].clear();
                Edge x;
                do
                {
                    x = st.top(); st.pop();
                    if(bccno[x.u] != bcc_cnt) { bcc[bcc_cnt].push_back(x.u); bccno[x.u] = bcc_cnt; }
                    if(bccno[x.v] != bcc_cnt) { bcc[bcc_cnt].push_back(x.v); bccno[x.v] = bcc_cnt; }
                } while(x.u != u || x.v != v);
            }
            if(low[v] > dfn[u]) ++ bridge;
        }
        else if(dfn[v] < dfn[u])
        {
            st.push({u, v});
            low[u] = min(low[u], dfn[v]);
        }
    }
    if(pre < 0 && child == 1) iscut[u] = 0;
}

void find_bcc(int n)
{
    memset(dfn, 0, sizeof dfn);
    memset(iscut, 0, sizeof iscut);
    memset(bccno, 0, sizeof bccno);
    for(int i = 1; i <= bcc_cnt; i ++) bcc[i].clear();
    idx = bcc_cnt = bridge = 0;
    for(int i = 0; i < n; i ++) if(!dfn[i]) dfs(i, -1);
}
```

2.3.5 边双联通分量

```
const int maxn = 10000 + 10;

int low[maxn], dfn[maxn], head[maxn], cnt, idx;
int cutEdge[maxn << 2];
struct  Edge { int nex, v; }edge[maxn << 2];

void add(int u, int v) { edge[cnt].nex = head[u], edge[cnt].v = v, head[u] = cnt ++; }

void dfs(int u, int pre)
{
    low[u] = dfn[u] = ++idx;
    for(int i = head[u]; ~i; i = edge[i].nex)
    {
        int v = edge[i].v;
        if(v == pre) continue;
        if(!dfn[v])
        {
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            if(low[v] > dfn[u]) cutEdge[i] = cutEdge[i ^ 1] = 1;
        }
        else if(dfn[v] < dfn[u]) low[u] = min(low[u], dfn[v]);
    }
}
```

2.4 二分图匹配

1. 二分图中的最大匹配数 = 最小点覆盖数
2. 最小路径覆盖 = 最小路径覆盖=｜ G ｜ - 最大匹配数
3. 二分图最大独立集 = 顶点数-最小点覆盖
4. 二分图的最大团 = 补图的最大独立集

2.4.1 Hungary Algorithm

```
const int maxn = 150;

int n;
int edge[maxn][maxn];
int linker[maxn];
bool vis[maxn];

bool path(int u)
{
    for (int v = 1; v <= n; v++)
    {
        if (edge[u][v] && !vis[v])
        {
            vis[v] = true;
            if (linker[v] == -1 || path(linker[v]))
            {
                linker[v] = u;
                return true;
            }
        }
    }
    return false;
}

int hungary()
{
    int res = 0;
    memset(linker, 0xff, sizeof(linker));
    for (int i = 1; i <= n; i++)
    {
        memset(vis, false, sizeof(vis));
        res += path(i);
    }
    return res;
}
```

2.4.2 Hopcroft-karp Algorithm

```
//复杂度O(n^0.5*m),注意这个板子的下标是从0开始的

const int MAXN = 3010;//左边节点数量、右边节点数量
const int MAXM = 3010 * 3010;//边的数量
const int INF = 0x3f3f3f3f;

```

```
struct Edge
{
    int v;
    int next;
} edge[MAXM];

int nx, ny;
int cnt;
int dis;

int first[MAXN];
int xlink[MAXN], ylink[MAXN];
/*xlink[i]表示左集合顶点所匹配的右集合顶点序号，ylink[i]表示右集合i顶点匹配到的左集合顶点序号。*/
int dx[MAXN], dy[MAXN];
/*dx[i]表示左集合i顶点的距离编号，dy[i]表示右集合i顶点的距离编号*/
int vis[MAXN]; //寻找增广路的标记数组

void init()
{
    cnt = 0;
    memset(first, -1, sizeof(first));
    memset(xlink, -1, sizeof(xlink));
    memset(ylink, -1, sizeof(ylink));
}

void read_graph(int u, int v)
{
    edge[cnt].v = v;
    edge[cnt].next = first[u], first[u] = cnt++;
}

int bfs()
{
    queue<int> q;
    dis = INF;
    memset(dx, -1, sizeof(dx));
    memset(dy, -1, sizeof(dy));
    for (int i = 0; i < nx; i++)
    {
        if (xlink[i] == -1)
        {
            q.push(i);
            dx[i] = 0;
        }
    }
    while (!q.empty())
    {
        int u = q.front();
        q.pop();
        if (dx[u] > dis) break;
        for (int e = first[u]; e != -1; e = edge[e].next)
        {
            int v = edge[e].v;
            if (dy[v] == -1)
            {
                dy[v] = dx[u] + 1;
                if (ylink[v] == -1) dis = dy[v];
                else
                {
                    dx[ylink[v]] = dy[v] + 1;
                    q.push(ylink[v]);
                }
            }
        }
    }
    return dis != INF;
}

int find(int u)
{
    for (int e = first[u]; e != -1; e = edge[e].next)
    {
        int v = edge[e].v;
        if (!vis[v] && dy[v] == dx[u] + 1)
        {
            vis[v] = 1;
            if (ylink[v] != -1 && dy[v] == dis) continue;
            if (ylink[v] == -1 || find(ylink[v]))
            {
                xlink[u] = v, ylink[v] = u;
                return 1;
            }
        }
    }
    return 0;
}

int MaxMatch()
{
    int ans = 0;
    while (bfs())
    {
        memset(vis, 0, sizeof(vis));
        for (int i = 0; i < nx; i++)
            if (xlink[i] == -1)
                ans += find(i);
    }
    return ans;
}
```

2.4.3 二分图多重匹配

```
const int maxn = 1e2 + 5;//左边最大点数
const int maxm = 1e2 + 5;//右边最大点数
int graph[maxn][maxm], vis[maxm];//图G和增广路访问标记
int match[maxm][maxn];//左边元素与右边元素第n次匹配
int nx, ny, m;//左边点数，右边点数,边数
int vol[maxm];//右边点多重匹配可容纳值
int cnt[maxm];//右边点已匹配值

bool find_path(int u)//找增广路
{
    for (int i = 0; i < ny; i++)//注意，这里节点是从0开始编号，题目有时是从1开始编号
    {
        if (graph[u][i] && !vis[i])//不在增广路
        {
            vis[i] = 1;//放进增广路
            if (cnt[i] < vol[i])//如果当前已匹配数量小于可容纳量，则直接匹配
            {
                match[i][cnt[i]++] = u;
                return true;
            }
            for (int j = 0; j < cnt[i]; j++)
            {
                if (find_path(match[i][j]))//如果先前已匹配右边的点能另外找到增广路，则此点仍可匹配
                {
                    match[i][j] = u;
                    return true;
                }
            }
        }
    }
    return false;
}

int max_match()//计算多重匹配的最大匹配数
{
    int res = 0;
    memset(match, -1, sizeof(match));
    memset(cnt, 0, sizeof(cnt));
    for (int i = 0; i < nx; i++)
    {
        memset(vis, 0, sizeof(vis));
        if (find_path(i)) res++;
    }
    return res;
}

bool all_match()//判断左边的点是否都与右边的点匹配了
{
    memset(cnt, 0, sizeof(cnt));
    for (int i = 0; i < nx; i++)
    {
        memset(vis, 0, sizeof(vis));
        if (!find_path(i)) return false;
    }
    return true;
}
```

2.4.4 二分图最大权匹配 (KM 算法)

```
const int maxn=1000+10;
const int inf=0x3f3f3f3f;

int n;
int lx[maxn],ly[maxn],edge[maxn][maxn];
int match[maxn],delta;
bool vx[maxn],vy[maxn];

bool dfs(int x) //DFS增广，寻找相等子图的完备匹配
{
    vx[x]=true;
    for(int y=1;y<=n;y++)
    {
        if(!vy[y])
        {
            int tmp=lx[x]+ly[y]-edge[x][y];
            if(!tmp)    //edge(x,y)为可行边
            {
                vy[y]=true;
                if(!match[y]||dfs(match[y]))
                {
                    match[y]=x;
                    return true;
                }
            }
            else delta=min(delta,tmp);
        }
    }
    return false;
}

void KM()
{
    for(int i=1;i<=n;i++)   //初始化可行顶标的值
    {
        lx[i]=-inf;
        ly[i]=0;
        for(int j=1;j<=n;j++)
            lx[i]=max(lx[i],edge[i][j]);
    }
    memset(match,0,sizeof(match));
    for(int x=1;x<=n;x++)
    {
        for(;;)
        {
            delta=inf;
            memset(vx,0,sizeof(vx));
            memset(vy,0,sizeof(vy));
            if(dfs(x)) break;
            for(int i=1;i<=n;i++)   //修改顶标
            {
                if(vx[i]) lx[i]-=delta;
                if(vy[i]) ly[i]+=delta;
            }
        }
    }
}
```

2.4.5 一般图匹配带花树

```
//一般图匹配，带花树算法
const int maxn = 1000 + 10;

vector<int> edge[maxn];
queue<int> que;

int n, pre[maxn], type[maxn], link[maxn], nex[maxn], vis[maxn];

void add(int u, int v)
{
    edge[u].push_back(v);
    edge[v].push_back(u);
```

```
}

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

void combine(int x, int lca)    //如果找到奇环，对当前点x和找到的
{
    while (x != lca)
    {
        int u = link[x], v = nex[u];
        if (Find(v) != lca) nex[v] = u;
        if (type[u] == 1) type[u] = 2, que.push(u);
        pre[Find(x)] = Find(u);
        pre[Find(u)] = Find(v);
        x = v;
    }
}

void contrack(int x, int y)
{
    int lca = x;
    memset(vis, 0, sizeof(vis));
    for (int i = x; i; i = nex[link[i]])
    {
        i = Find(i);
        vis[i] = 1;
    }
    for (int i = y; i; i = nex[link[i]])
    {
        i = Find(i);
        if (vis[i])
        {
            lca = i;
            break;
        }
    }
    if (lca != Find(x)) nex[x] = y;
    if (lca != Find(y)) nex[y] = x;
    combine(x, lca);
    combine(y, lca);
}

void bfs(int s)
{
    memset(type, 0, sizeof(type));
    memset(nex, 0, sizeof(nex));
    for (int i = 1; i <= n; i++) pre[i] = i;
    while (!que.empty()) que.pop();
    que.push(s);
    type[s] = 2;
    while (!que.empty())
    {
        int x = que.front();
        que.pop();
        for (int i = 0; i < edge[x].size(); i++)
        {
            int y = edge[x][i];
            if (Find(x) == Find(y) || link[x] == y || type[y] == 1) continue;
            if (type[y] == 2) contrack(x, y);
            else if (link[y])
            {
                nex[y] = x;
                type[y] = 1;
                type[link[y]] = 2;
                que.push(link[y]);
            } else
            {
                nex[y] = x;
                int pos = y, u = nex[pos], v = link[u];
                while (pos)
                {
                    link[pos] = u;
                    link[u] = pos;
                    pos = v;
                    u = nex[pos];
                    v = link[u];
                }
                return;
            }
        }
    }
}

int maxmatch()
{
    for (int i = 1; i <= n; i++) if (!link[i]) bfs(i);
    int ans = 0;
    for (int i = 1; i <= n; i++) if (link[i]) ans++;
    return ans / 2;
}

void init()
{
    for (int i = 1; i <= n; i++) edge[i].clear();
    memset(link, 0, sizeof(link));
}
```

2.5 网络流

2.5.1 Dinic

```
const int MAX_V = 1000 + 10;
const int INF = 0x3f3f3f3f;

//用于表示边的结构体（终点，流量，反向边）
struct edge{int to, cap, rev;};

vector<edge> G[MAX_V];  //图的邻接表表示
int level[MAX_V];   //顶点到源点的距离标号
int iter[MAX_V];    //当前弧

void add(int from, int to, int cap)
{
    G[from].push_back((edge){to, cap, (int)G[to].size()});
    G[to].push_back((edge){from, 0, (int)G[from].size() - 1});
}
```

```

//计算从源点出发的距离标号
void bfs(int s)
{
    memset(level, -1, sizeof(level));
    queue<int> que;
    level[s] = 0;
    que.push(s);
    while(!que.empty())
    {
        int v = que.front(); que.pop();
        for(int i = 0; i < G[v].size(); i++)
        {
            edge &e = G[v][i];
            if(e.cap > 0 && level[e.to] < 0)
            {
                level[e.to] = level[v] + 1;
                que.push(e.to);
            }
        }
    }
}

//通过DFS寻找增广路
int dfs(int v, int t, int f)
{
    if(v == t) return f;
    for(int &i = iter[v]; i<G[v].size(); i++)
    {
        edge &e = G[v][i];
        if(e.cap > 0 && level[v] < level[e.to])
        {
            int d = dfs(e.to, t, min(f, e.cap));
            if(d > 0)
            {
                e.cap -= d;
                G[e.to][e.rev].cap += d;
                return d;
            }
        }
    }
    return 0;
}

//求解从s到t的最大流
int max_flow(int s, int t)
{
    int flow = 0;
    for(;;)
    {
        bfs(s);
        if(level[t] < 0) return flow;
        memset(iter, 0, sizeof(iter));
        int f;
        while((f = dfs(s,t,INF)) > 0) flow += f;
    }
}
```

2.5.2 ISAP

```
struct Edge {
  int from, to, cap, flow;
  Edge(int u, int v, int c, int f) : from(u), to(v), cap(c), flow(f) {}
};

bool operator<(const Edge& a, const Edge& b) {
  return a.from < b.from || (a.from == b.from && a.to < b.to);
}

struct ISAP {
  int n, m, s, t;
  vector<Edge> edges;
  vector<int> G[maxn];
  bool vis[maxn];
  int d[maxn];
  int cur[maxn];
  int p[maxn];
  int num[maxn];

  void AddEdge(int from, int to, int cap) {
    edges.push_back(Edge(from, to, cap, 0));
    edges.push_back(Edge(to, from, 0, 0));
    m = edges.size();
    G[from].push_back(m - 2);
    G[to].push_back(m - 1);
  }

  bool BFS() {
    memset(vis, 0, sizeof(vis));
    queue<int> Q;
    Q.push(t);
    vis[t] = 1;
    d[t] = 0;
    while (!Q.empty()) {
      int x = Q.front();
      Q.pop();
      for (int i = 0; i < G[x].size(); i++) {
        Edge& e = edges[G[x][i] ^ 1];
        if (!vis[e.from] && e.cap > e.flow) {
          vis[e.from] = 1;
          d[e.from] = d[x] + 1;
          Q.push(e.from);
        }
      }
    }
    return vis[s];
  }

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

  int Augment() {
    int x = t, a = INF;
    while (x != s) {
      Edge& e = edges[p[x]];
      a = min(a, e.cap - e.flow);
      x = edges[p[x]].from;
    }
    x = t;
    while (x != s) {
      edges[p[x]].flow += a;
      edges[p[x] ^ 1].flow -= a;
      x = edges[p[x]].from;
    }
    return a;
  }

  int Maxflow(int s, int t) {
    this->s = s;
    this->t = t;
    int flow = 0;
    BFS();
    memset(num, 0, sizeof(num));
    for (int i = 0; i < n; i++) num[d[i]]++;
    int x = s;
    memset(cur, 0, sizeof(cur));
    while (d[s] < n) {
      if (x == t) {
        flow += Augment();
        x = s;
      }
      int ok = 0;
      for (int i = cur[x]; i < G[x].size(); i++) {
        Edge& e = edges[G[x][i]];
        if (e.cap > e.flow && d[x] == d[e.to] + 1) {
          ok = 1;
          p[e.to] = G[x][i];
          cur[x] = i;
          x = e.to;
          break;
        }
      }
      if (!ok) {
        int m = n - 1;
        for (int i = 0; i < G[x].size(); i++) {
          Edge& e = edges[G[x][i]];
          if (e.cap > e.flow) m = min(m, d[e.to]);
        }
        if (--num[d[x]] == 0) break;
        num[d[x] = m + 1]++;
        cur[x] = 0;
        if (x != s) x = edges[p[x]].from;
      }
    }
    return flow;
  }
};
```

2.5.3 MCMF

```
const int maxn = 10000 + 10;
const int inf = 0x3f3f3f3f;
```

```cpp

struct Edge { int from, to, cap, flow, cost; };

struct MCMF
{
    int n, m;
    vector<Edge> edges;
    vector<int> G[maxn];
    bool inq[maxn];
    int dis[maxn], path[maxn], a[maxn];

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

    void addEdge(int from, int to, int cap, int cost)
    {
        edges.push_back(Edge{from, to, cap, 0, cost});
        edges.push_back(Edge{to, from, 0, 0, -cost});
        m = edges.size();
        G[from].push_back(m - 2);
        G[to].push_back(m - 1);
    }

    bool Bellman_Ford(int s, int t, int& flow, int& cost)
    {
        for(int i = 0; i<= n; i++) dis[i] = inf;
        memset(inq, 0, sizeof inq);
        dis[s]=0, inq[s]=true, path[s]=0, a[s]=inf;
        queue<int> Q;
        Q.push(s);
        while(!Q.empty())
        {
            int u = Q.front(); Q.pop();
            inq[u] = false;
            for(int i = 0; i < G[u].size(); i++)
            {
                Edge& e = edges[G[u][i]];
                if(e.cap > e.flow && dis[e.to] > dis[u] + e.cost)
                {
                    dis[e.to] = dis[u] + e.cost;
                    path[e.to] = G[u][i];
                    a[e.to] = min(a[u], e.cap - e.flow);
                    if(!inq[e.to])
                    {
                        Q.push(e.to);
                        inq[e.to] = true;
                    }
                }
            }
        }
        if(dis[t] == inf) return false;     //求最小费用最大流
        //if(1ll * dis[t] * a[t] > 0) return false; 求可行流最小费用，因此当费用增量大于0时不继续增加流量
        flow += a[t];
        cost += dis[t] * a[t];
        for(int u = t; u != s; u = edges[path[u]].from)
        {
            edges[path[u]].flow += a[t];
            edges[path[u] ^ 1].flow -= a[t];
        }
        return true;
    }

    int mincostMaxFlow(int s, int t)
    {
        int flow = 0, cost = 0;
        while(Bellman_Ford(s, t, flow, cost));
        return cost;
    }
};
```

2.5.4 Trick

建模技巧

二分图带权最大独立集。给出一个二分图，每个结点上有一个正权值。要求选出一些点，使得这些点之间没有边相连，且权值和最大。

解：在二分图的基础上添加源点 S 和汇点 T，然后从 S 向所有 X 集合中的点连一条边，所有 Y 集合中的点向 T 连一条边，容量均为该点的权值。X 结点与 Y 结点之间的边的容量均为无穷大。这样，对于图中的任意一个割，将割中的边对应的结点删掉就是一个符合要求的解，权和为所有权减去割的容量。因此，只需要求出最小割，就能求出最大权和。

公平分配问题。把 m 个任务分配给 n 个处理器。其中每个任务有两个候选处理器，可以任选一个分配。要求所有处理器中，任务数最多的那个处理器所分配的任务数尽量少。不同任务的候选处理器集 $\{p_1, p_2\}$ 保证不同。

解：本题有一个比较明显的二分图模型，即 X 结点是任务，Y 结点是处理器。二分答案 x，然后构图，首先从源点 S 出发向所有的任务结点引一条边，容量等于 1，然后从每个任务结点出发引两条边，分别到达它所能分配到的两个处理器结点，容量为 1，最后从每个处理器结点出发引一条边到汇点 T，容量为 x，表示选择该处理器的任务不能超过 x。这样网络中的每个单位流量都是从 S 流到一个任务结点，再到处理器结点，最后到汇点 T。只有当网络中的总流量等于 m 时才意味着所有任务都选择了一个处理器。这样，我们通过 $O(\log m)$ 次最大流便算出了答案。

区间 k 覆盖问题。数轴上有一些带权值的左闭右开区间。选出权和尽量大的一些区间，使得任意一个数最多被 k 个区间覆盖。

解：本题可以用最小费用流解决，构图方法是把每个数作为一个结点，然后对于权值为 w 的区间 $[u, v)$ 加边 $u \to v$，容量为 1，费用为 $-w$。再对所有相邻的点加边 $i \to i+1$，容量为 k，费用为 0。最后，求最左点到最右点的最小费用最大流即可，其中每个流量对应一组互不相交的区间。如果数值范围太大，可以先进行离散化。

最大闭合子图。给定带权图 G（权值可正可负），求一个权和最大的点集，使得起点在该点集中的任意弧，终点也在该点集中。

解：新增附加源 s 和附加汇 t，从 s 向所有正权点引一条边，容量为权值；从所有负权点向汇点引一条边，容量为权值的相反数。求出最小割以后，$S - \{s\}$ 就是最大闭合子图。

最大密度子图。给出一个无向图，找一个点集，使得这些点之间的边数除以点数的值（称为子图的密度）最大。

解：如果两个端点都选了，就必然要选边，这就是一种推导。如果把每个点和每条边都看成新图中的结点，可以把问题转化为最大闭合子图。

无源汇有上下界可行流：附加源 S 和汇 T；对于边 (u, v, min, max)，记 $d[u] - = min, d[v] + = max$，并添加弧 $(u, v, max - min)$；对于流量不平衡的点 u，设多余流量为 W，如果 $W > 0$，添加弧 $S- > u : W$，否则若 $W < 0$，添加弧 $u- > T : -W$，求改造后的网络 $S - T$ 最大流即可，当且仅当所有附加弧满载时原图有可行流。

有源汇有上下界可行流： 建 $t->s$，容量为 inf，然后和无源汇相同。

有源汇有上下界最大/最小流： 与上面相同，跑完可行流 $S->T$ 后去掉边 $t->s$，最大流为加 $s->t$，最小流为 $G[s][t].cap-max_flow(t,s)$。

2.5.5 Stoer Wagner

```cpp
#define INF 100000000
bool vis[maxn], com[maxn];
int mp[maxn][maxn], w[maxn], s, t;

int maxadj(int n, int v) {
    int CUT = 0;
    memset(vis, 0, sizeof vis);
    memset(w, 0, sizeof w);
    for (int i = 0; i < n; ++i) {
        int num = 0, mx = -INF;
        for (int j = 0; j < v; ++j) {
            if (!com[j] && !vis[j] && w[j] > mx) {
                mx = w[j];
                num = j;
            }
        }
        vis[num] = 1;
        s = t;
        t = num;
        CUT = w[t];
        for (int j = 0; j < v; ++j) {
            if (!com[j] && !vis[j]) w[j] += mp[num][j];
        }
    }
    return CUT;
}

int stoer(int v) {
    int mincut = INF;
    int n = v;
    memset(com, 0, sizeof com);
    for (int i = 0; i < v - 1; ++i) {
        int cut;
        s = 0, t = 0;
        cut = maxadj(n, v);
        n --;
        if (cut < mincut) mincut = cut;
        com[t] = 1;
        for (int j = 0; j < v; ++j) {
            if (!com[j]) {
                mp[j][s] += mp[j][t];
                mp[s][j] += mp[t][j];
            }
        }
    }
    return mincut;
}
```

2.6 Others

2.6.1 拓扑排序

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

vector<int> edge[maxn];
int indegree[maxn];

void add(int u, int v)
{
    edge[u].push_back(v);
    indegree[v]++;
}

void Toposort(int n)
{
    queue<int> que;
    for (int i = 1; i <= n; i++)
        if (!indegree[i]) que.push(i);    //将图中没有前驱，即入度为0的点加入队列
    while (!que.empty())
    {
        int u = que.front();
        que.pop();
        indegree[u] = -1;    //从图中删去此顶点
        for (int i = 0; i < edge[u].size(); i++)
        {
            int v = edge[u][i];
            indegree[v]--;    //删去图中以u为尾的弧
            if (!indegree[v]) que.push(v);    //将新增的当前入度为0的点压入队列中
        }
    }
}
```

2.6.2 2-SAT

```cpp
/*2-SAT连边含义: 选A必选B
    点$x_i$表示选，$x_i'$表示不选
    1.必选$x_i$，等价于$x_i=1$: $x_i'→x_i$
    2.必不选$x_i$，等价于$x_i=0$，$x_i→x_i'$
    3.$x_i$与$x_j$中至少选择一个，等价于$x_iORx_j=1$，连边$x_i'→x_j$,$x_j'→x_i$
    4.$x_i$与$x_j$不都选，等价于$x_iANDx_j=0$，连边$x_i→x_j',x_j→x_i'$
    5.$x_i$与$x_j$情况相同，等价于$x_iXORx_j=0$，连边$x_i→x_j$,$x_i'→x_j'$,$x_j→x_i$,$x_j'→x_i'$
    6.$x_i$与$x_j$情况相反，等价于$x_iXORx_j=1$，连边$x_i→x_j'$,$x_i'→x_j$,$x_j→x_i'$,$x_j'→x_i$
*/

const int maxn = 2e6 + 10;

int n, m, a, va, b, vb;
int low[maxn], dfn[maxn], color[maxn], cnt, scc_cnt;
bool instack[maxn];

vector<int> g[maxn];
stack<int> st;

void Tarjan(int u)
{
    low[u] = dfn[u] = ++cnt;
    st.push(u);
    instack[u] = true;
    for(const auto &v : g[u])
    {
        if(!dfn[v]) Tarjan(v), low[u] = min(low[u], low[v]);
        else if(instack[v]) low[u] = min(low[u], dfn[v]);
    }
    if(low[u] == dfn[u])
    {
        ++scc_cnt;
        do {
            color[u] = scc_cnt;
            u = st.top(); st.pop();
            instack[u] = false;
        } while(low[u] != dfn[u]);
    }
}

inline void add(int a, int b) { g[a].push_back(b); }

inline void AND(int a, int b, int c)
{
    if(c == 1) add(a, a + n), add(b, b + n);
    else add(a + n, b), add(b + n, a);
}

inline void OR(int a, int b, int c)
{
    if(c == 0) add(a + n, a), add(b + n, b);
    else add(a, b + n), add(b, a + n);
}

inline void XOR(int a, int b, int c)
{
    if(c == 0) add(a, b), add(a + n, b + n), add(b, a), add(b + n, a + n);
    else add(a, b + n), add(a + n, b), add(b, a + n), add(b + n, a);
}

bool TWO_SAT()
{
    input();
    for(int i = 1; i <= (n << 1); i ++) if(!dfn[i]) Tarjan(i);
    for(int i = 1; i <= n; i ++)
        if(color[i] == color[i + n]) return false;
    for(int i = 1; i <= n; i ++)
        printf("%d ", color[i] > color[i + n]);
    return true;
}
```

2.6.3 差分约束系统

```
//以$x_i-x_j y$为约束条件，建图求最短路后得到的是最大解。所有的解都不大于且尽可能逼近$dis[x0]$
//最短路对应最大解，最长路对应最小解

const int maxn = 1000 + 10;
const int inf = 0x3f3f3f3f;

struct Edge
{
    int nex, to, w;
} edge[10 * maxn];

```

```
int head[maxn], cnt, dis[maxn], n;
bool vis[maxn];

void init()
{
    cnt = 0;
    memset(head, 0xff, sizeof head);
}

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

void spfa(int u)
{
    int u, v, w;
    for (int i = 1; i <= n; i++) dis[i] = inf, vis[i] = false;
    dis[u] = 0;
    queue<int> que;
    que.push(u);
    vis[u] = true;
    while (!que.empty())
    {
        u = que.front();
        que.pop();
        vis[u] = false;
        for (int i = head[u]; ~i; i = edge[i].nex)
        {
            v = edge[i].v, w = edge[i].w;
            if (dis[u] + w < dis[v])
            {
                dis[v] = dis[u] + w;
                if (!vis[v])
                {
                    que.push(v);
                    vis[v] = true;
                }
            }
        }
    }
}
```

2.6.4 支配树

```
const int N = 2e5 + 10;

int n, m;

struct G
{
    vector<int> edge[N];
    inline void add(int u, int v) { edge[u].push_back(v); }
}a, b, c, d;

```

```
int dfn[N], id[N], fa[N], cnt;

void dfs(int u)
{
    dfn[u] = ++ cnt; id[cnt] = u;
    int len = a.edge[u].size();
    for(auto v : a.edge[u]) if(!dfn[v]) { fa[v] = u; dfs(v); }
}

int semi[N], idom[N], belong[N], val[N];

int find(int x)
{
    if(x == belong[x]) return x;
    int tmp = find(belong[x]);
    if(dfn[semi[val[belong[x]]]] < dfn[semi[val[x]]]) val[x] = val[belong[x]];
    return belong[x] = tmp;
}

void tarjan()
{
    for(int i = cnt; i > 1; i --)
    {
        int u = id[i];
        for(auto v : b.edge[u])
        {
            if(!dfn[v]) continue;
            find(v);
            if(dfn[semi[val[v]]] < dfn[semi[u]]) semi[u] = semi[val[v]];
        }
        c.add(semi[u], u);
        belong[u] = fa[u];
        u = fa[u];
        for(auto v : c.edge[u])
        {
            find(v);
            if(semi[val[v]] == u) idom[v] = u;
            else idom[v] = val[v];
        }
    }
    for(int i = 2; i <= cnt; i ++)
    {
        int u = id[i];
        if(idom[u] != semi[u]) idom[u] = idom[idom[u]];
    }
}

int ans[N];

void dfs_ans(int u)
{
    ans[u] = 1;
    for(auto v : d.edge[u]) dfs_ans(v), ans[u] += ans[v];
}

void solve()
{
    int u, v;
    scanf("%d%d", &n, &m);
    while(m --)
    {
        scanf("%d%d", &u, &v);
        a.add(u, v);
        b.add(v, u);
    }
    for(int i = 1; i <= n; i ++) semi[i] = belong[i] = val[i] = i;
    dfs(1);
    tarjan();
    for(int i = 2; i <= n; i ++) d.add(idom[i], i);
    dfs_ans(1);
    for(int i = 1; i <= n; i ++) printf("%d ", ans[i]);
}
```

2.6.5 Stable Matching Problem

```
const int maxn = 1000 + 10;

int pre[maxn][maxn], order[maxn][maxn], nex[maxn];
int hus[maxn], wife[maxn];
queue<int> que;

void engage(int man, int woman)
{
    int m = hus[woman];
    if(m) wife[m] = 0, q.push(m);
    wife[man] = woman;
    hus[woman] = man;
}

int solve()
{
    for(int i = 1; i <= n; i ++)
    {
        for(int j = 1; j <= n; j ++)
            scanf("%d", &pre[i][j]);
        nex[i] = 1;
        wife[i] = 0;
        que.push(i);
    }
    for(int i = 1; i <= n; i ++)
    {
        for(int j = 1; j <= n; j ++)
        {
            int x;
            scanf("%d", &x);
            order[i][x] = j;
        }
        hus[i] = 0;
    }

    while(!que.empty())
    {
        int man = que.front(); que.pop();
        int woman = pre[man][nex[man] ++];
        if(!hus[woman]) engage(man, woman);
        else if(order[woman][man] < order[woman][hus[woman]]) engage(man, woman);
        else que.push(man);
    }
}
```

3 DataStructrue

3.1 SegmentTreeDS

3.1.1 SegmentTree

```
const int maxn = 2e5+5;
// 序列
int a[maxn];

struct SegmentTree {
#define TYPE int
#define USELAZY 0
    TYPE val[maxn << 2];
    int sz;
//    check this type
    vector<int> lazy;

    inline TYPE comb(const TYPE& a, const TYPE& b) {
        TYPE res;
        res = a + b;
        return res;
    }

    int le, re, k;

    inline void build(int rt, int l, int r) {
        if (USELAZY) lazy[rt] = 0;
        if (l == r) {
            val[rt] = a[l];
            return;
        }
        int mid = l + r >> 1;
        build(rt << 1, l, mid);
        build(rt << 1 | 1, mid + 1, r);
        pushup(rt);
    }
    inline void build() {build(1, 1, sz);}

    inline void init(int sz_) {
        sz = sz_;
        lazy.resize(sz_ << 2);
        build();
    }
    inline void pushup(int rt) {val[rt] = comb(val[rt << 1], val[rt << 1 | 1]);}
    inline void deal(int rt, int kt) {
//        todo:
        val[rt] = comb(val[rt], kt);
    }
    inline void pushdown(int rt, int len) {
        if (lazy[rt]) {
            // check the lazy change
            lazy[rt << 1] += lazy[rt];
            lazy[rt << 1 | 1] += lazy[rt];
            deal(rt << 1, lazy[rt]);
            deal(rt << 1 | 1, lazy[rt]);
            lazy[rt] = 0;
        }
    }

    inline void update(int rt, int l, int r) {
        if (le <= l && r <= re) {
            deal(rt, k);
            return;
        }
        if (USELAZY) pushdown(rt, r - l + 1);
        int mid = l + r >> 1;
        if (le <= mid) update(rt << 1, l, mid);
        if (re > mid) update(rt << 1 | 1, mid + 1, r);
        pushup(rt);
    }

    inline TYPE query(int rt, int l, int r) {
        if (le <= l && r <= re) {
            return val[rt];
        }
        if (USELAZY) pushdown(rt, r - l + 1);
        // check the zero type
        TYPE res;
        int mid = l + r >> 1;
        if (le <= mid) res = comb(res, query(rt << 1, l, mid));
        if (re > mid) res = comb(res, query(rt << 1 | 1, mid + 1, r));
        return res;
    }

    // check return type
    inline int query(int l, int r) {
        le = l, re = r;
        return query(1, 1, sz);
    }
    inline void modify(int l, int r, int kt) {
        le = l, re = r, k = kt;
        update(1, 1, sz);
    }

//    inline void pt(int rt, int l, int r) {
//        if (l == r) {
//            printf("%d ", val[l]);
//            return;
//        }
//        pushdown(rt, r - l + 1);
//        int mid = l + r >> 1;
//        if (le <= mid) pt(rt << 1, l, mid);
//        if (re > mid) pt(rt << 1 | 1, mid + 1, r);
//    }

#undef TYPE
};
```

3.1.2 离散化区间

```
// 原题1e5个区间有2e5个端点，离散化出来4e5个区间
// 然后线段树需要4e5*4=16e5的大小
// 注意三个数组要开离散化数量的四倍，如果不需要sz可以不用这个数组。
int val[maxn << 4];
int lpos[maxn << 2], rpos[maxn << 2], tot, sz[maxn << 2];
vector<int> xpos;
```

```
sort(xpos.begin(), xpos.end());
xpos.erase(unique(xpos.begin(), xpos.end()), xpos.end());
tot = 1;
lpos[1] = rpos[1] = xpos[0];
sz[1] = 1;
for (int i = 1; i < xpos.size(); ++i) {
    if (xpos[i] - xpos[i - 1] != 1) {
        lpos[++tot] = xpos[i - 1] + 1;
        rpos[tot] = xpos[i] - 1;
        sz[tot] = rpos[tot] - lpos[tot] + 1;
    }
    ++tot;
    lpos[tot] = rpos[tot] = xpos[i];
    sz[tot] = 1;
}
le = lower_bound(lpos + 1, lpos + 1 + tot, p[i].x) - lpos;
re = upper_bound(rpos + 1, rpos + 1 + tot, p[i].y) - rpos - 1;
```

3.1.3 动态区间最大子段和

```
namespace ST {
    struct node{
        ll ans,ls,rs,sum;
    }xx[maxn << 2];
    inline void pushdown(int x){
        xx[x].sum=xx[x<<1].sum+xx[x<<1|1].sum;
        xx[x].ls=max(xx[x<<1].ls,xx[x<<1].sum+xx[x<<1|1].ls);
        xx[x].rs=max(xx[x<<1|1].rs,xx[x<<1|1].sum+xx[x<<1].rs);
        xx[x].ans=max(xx[x<<1].ans,max(xx[x<<1|1].ans,xx[x<<1].rs+xx[x<<1|1].ls));
        return;
    }
    inline void build(int k,int l,int r){
        if(l==r){
            xx[k].ls=xx[k].rs=xx[k].ans=xx[k].sum=0;
            return;
        }
        int mid=l+r>>1;
        build(k<<1,l,mid),build(k<<1|1,mid+1,r);
        pushdown(k);
        return;
    }
    inline void change(int k,int l,int r,int x,int y,int w){ // 1, 1, n
        if(x<=l&&r<=y){
            xx[k].ls += w;
            xx[k].rs += w;
            xx[k].ans += w;
            xx[k].sum += w;
//            xx[k].ls=xx[k].rs=xx[k].ans=xx[k].sum=w;
            return;
        }
        int mid=l+r>>1;
        if(x<=mid) change(k<<1,l,mid,x,y,w);
        if(mid<y) change(k<<1|1,mid+1,r,x,y,w);
        pushdown(k);
        return;
    }
    inline node query(int k,int l,int r,int x,int y){
        if(x<=l&&r<=y) {
            return xx[k];
        }
        int mid=l+r>>1;
        if(x<=mid&&!(mid<y)) return query(k<<1,l,mid,x,y);
        else if(!(x<=mid)&&mid<y) return query(k<<1|1,mid+1,r,x,y);
        else{
            node st,t1=query(k<<1,l,mid,x,y),t2=query(k<<1|1,mid+1,r,x,y);
            st.sum=t1.sum+t2.sum;
            st.ls=max(t1.ls,t1.sum+t2.ls);
            st.rs=max(t2.rs,t2.sum+t1.rs);
            st.ans=max(t1.ans,max(t2.ans,t1.rs+t2.ls));
            return st;
        }
    }
}
```

3.1.4 动态开点权值线段树

```
int root[100005];
int ls[1800000], rs[1800000], sum[1800000];
int sz = 0;

void insert(int &k, int l, int r, int val){
    if (!k) k = ++sz;
    if (l == r) {
        sum[k] = 1;
        return;
    }
    int mid = (l + r) >> 1;
    if (val <= mid) insert(ls[k], l, mid, val);
    else insert(rs[k], mid + 1, r, val);
    sum[k] = sum[ls[k]] + sum[rs[k]];
}

int query(int k, int l, int r, int rank) {
    if (l == r) return l;
    int mid = (l + r) >> 1;
    if (sum[ls[k]] >= rank) return query(ls[k], l, mid, rank);
    else return query(rs[k], mid + 1, r, rank - sum[ls[k]]);
}
int merge(int x, int y)
{
    if (!x) return y;
    if (!y) return x;
    ls[x] = merge(ls[x], ls[y]);
    rs[x] = merge(rs[x], rs[y]);
    sum[x] = sum[ls[x]] + sum[rs[x]];
    return x;
}
insert(root[i], 1, n, a[i]);
query(root[p], 1, n, x);
```

3.1.5 扫描线

```
// 范用型扫描线，del储存上界+1，add储存下界，先del后add即可
struct node {
    int lpos, rpos, linepos;
    bool operator < (const node& oth) const {
        return linepos < oth.linepos;
    }
};
vector<node> add, del;
int delpos = 0;
int res = 0;
for (int addpos = 0; addpos < add.size(); ++addpos) {
    while (delpos < del.size() && del[delpos].linepos <= add[addpos].linepos) {
        up(del[delpos].lpos, del[delpos].rpos, -1);
        delpos ++;
    }
    up(add[addpos].lpos, add[addpos].rpos, 1);
    res = max(res, val[1]);
}

// 求面积并
#define maxn 222
#define tmp (st<<1)
#define mid ((l+r)>>1)
#define lson l,mid,tmp
#define rson mid+1,r,tmp|1
using namespace std;
int cnt[maxn<<2];
double sum[maxn<<2];
double x[maxn];
struct Seg{
    double h,l,r;
    int s;
    Seg(){}
    Seg(double a,double b,double c,int d):l(a),r(b),h(c),s(d){}
    bool operator<(const Seg &cmp)const{
        return h<cmp.h;
    }
}ss[maxn];
void push_up(int st,int l,int r){
    if(cnt[st])sum[st]=x[r+1]-x[l];
    else if(l==r)sum[st]=0;
    else sum[st]=sum[tmp]+sum[tmp|1];
}
void update(int L,int R,int c,int l,int r,int st){
    if(L<=l&&r<=R){
        cnt[st]+=c;
        push_up(st,l,r);
        return ;
    }
    if(L<=mid)update(L,R,c,lson);
    if(R>mid)update(L,R,c,rson);
    push_up(st,l,r);
}
int main(){
    int n,tot=1,m;
    while(scanf("%d",&n)&&n){
        double a,b,c,d;
        m=0;
        while(n--){
            scanf("%lf%lf%lf%lf",&a,&b,&c,&d);
            x[m]=a;
            ss[m++]=Seg(a,c,b,1);
            x[m]=c;
            ss[m++]=Seg(a,c,d,-1);
        }
        sort(x,x+m);
        sort(ss,ss+m);
        double ans=0;
        for(int i=0;i<m;++i){
            int l=lower_bound(x,x+m,ss[i].l)-x;
            int r=lower_bound(x,x+m,ss[i].r)-x-1;
            update(l,r,ss[i].s,0,m-1,1);
            ans+=sum[1]*(ss[i+1].h-ss[i].h);
        }
        printf("Test case #%dnTotal explored area: %.2lfnn",tot++,ans);
    }
    return 0;
}

// 面积交
#include<bits/stdc++.h>
#define maxn 100005
#define lson l,mid,rt<<1
#define rson mid+1,r,rt<<1|1
#define pb push_back
using namespace std;

double tree[maxn<<2],tree2[maxn<<2];
int lazy[maxn<<2];
vector<double>ve;

struct seg{
    double l,r,h;
    int flag;
    seg(){}
    seg(double _l,double _r,double _h,int _flag){l=_l,r=_r,h=_h,flag=_flag;}
    bool operator<(const seg &b)const{return h<b.h;}
}s[maxn];

void push_up(int l,int r,int rt){
    if(lazy[rt]) tree[rt]=ve[r]-ve[l-1];
    else if(l==r) tree[rt]=0;
    else tree[rt]=tree[rt<<1]+tree[rt<<1|1];
}

void push_up2(int l,int r,int rt){
    if(lazy[rt]>1) tree2[rt]=ve[r]-ve[l-1];
    else if(l==r) tree2[rt]=0;
    else if(lazy[rt]==1)tree2[rt]=tree[rt<<1]+tree[rt<<1|1];
    else tree2[rt]=tree2[rt<<1]+tree2[rt<<1|1];
}

void build(int l,int r,int rt){
    tree[rt]=0,lazy[rt]=0;
    if(l==r) return;
    int mid=l+r>>1;
    build(lson);
    build(rson);
}

void add(int L,int R,int v,int l,int r,int rt){
    if(L<=l&&R>=r){
        lazy[rt]+=v;
        push_up(l,r,rt);
        push_up2(l,r,rt);
        return;
    }
    int mid=l+r>>1;
    if(L<=mid) add(L,R,v,lson);
    if(R>mid) add(L,R,v,rson);
    push_up(l,r,rt);
    push_up2(l,r,rt);
}

int getid(double x){ return lower_bound(ve.begin(),ve.end(),x)-ve.begin()+1;}

int main(){
    int n;
    int Case=1;
    int T;
    scanf("%d",&T);
    while(T--){
        scanf("%d",&n);
        ve.clear();
        int tot=0;
        double x1,y1,x2,y2;
        for(int i=1;i<=n;i++){
            scanf("%lf %lf %lf %lf",&x1,&y1,&x2,&y2);
            ve.pb(x1),ve.pb(x2);
            s[++tot]=seg(x1,x2,y1,1);
            s[++tot]=seg(x1,x2,y2,-1);
        }
        sort(ve.begin(),ve.end());
        ve.erase(unique(ve.begin(),ve.end()),ve.end());
        sort(s+1,s+tot+1);
        int N=ve.size();
        build(1,N,1);
        double ans=0;
        for(int i=1;i<tot;i++){
            int L=getid(s[i].l);
            int R=getid(s[i].r)-1;
            add(L,R,s[i].flag,1,N,1);
            ans+=tree2[1]*(s[i+1].h-s[i].h);
        }
        printf("%.2f\n",ans);
    }
}

// 求周长并
#include<bits/stdc++.h>
#define maxn 100005
#define lson l,mid,rt<<1
#define rson mid+1,r,rt<<1|1
#define pb push_back
using namespace std;

int tree[maxn<<2];
int lazy[maxn<<2];
vector<int>ve[2];
int k;

```

```
struct seg{
    int l,r,h;
    int flag;
    seg(){}
    seg(int _l,int _r,int _h,int _flag){l=_l,r=_r,h=_h,flag=_flag;}
    bool operator<(const seg &b)const{return h<b.h;}
}s[maxn];

void push_up(int l,int r,int rt){
    if(lazy[rt]) tree[rt]=ve[k][r]-ve[k][l-1];
    else if(l==r) tree[rt]=0;
    else tree[rt]=tree[rt<<1]+tree[rt<<1|1];
}

void build(int l,int r,int rt){
    tree[rt]=0,lazy[rt]=0;
    if(l==r) return;
    int mid=l+r>>1;
    build(lson);
    build(rson);
}

void add(int L,int R,int v,int l,int r,int rt){
    if(L<=l&&R>=r){
        lazy[rt]+=v;
        push_up(l,r,rt);
        return;
    }
    int mid=l+r>>1;
    if(L<=mid) add(L,R,v,lson);
    if(R>mid) add(L,R,v,rson);
    push_up(l,r,rt);
}

int getid(int x){return lower_bound(ve[k].begin(),ve[k].end(),x)-ve[k].begin()+1;}

int main(){
    int n;
    while(~scanf("%d",&n)){
        ve[0].clear();
        ve[1].clear();
        int x1,y1,x2,y2;
        for(int i=1;i<=n;i++){
            scanf("%d %d %d %d",&x1,&y1,&x2,&y2);
            ve[0].pb(x1),ve[0].pb(x2);
            ve[1].pb(y1),ve[1].pb(y2);
            s[i]=seg(x1,x2,y1,1);
            s[i+n]=seg(x1,x2,y2,-1);
            s[i+n+n]=seg(y1,y2,x1,1);
            s[i+n+n+n]=seg(y1,y2,x2,-1);
        }
        int ans=0;
        int pos=1;
        for(k=0;k<2;k++){
            sort(ve[k].begin(),ve[k].end());
            ve[k].erase(unique(ve[k].begin(),ve[k].end()),ve[k].end());
            sort(s+pos,s+pos+n+n);
            int N=ve[k].size();
            build(1,N,1);
            int pre=0;
            for(int i=pos;i<pos+n+n;i++){
                int L=getid(s[i].l);
                int R=getid(s[i].r)-1;
                add(L,R,s[i].flag,1,N,1);
                ans+=abs(tree[1]-pre);
                pre=tree[1];
            }
            pos+=n+n;
        }
        printf("%d\n",ans);
    }
}
```

3.2 HLD

3.2.1 HLD

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;
/*
node 计算点权， path 下放后计算边权， edge 根据边的编号计算边权
work 中没有build需手动写
sz[]数组，以x为根的子树节点个数
top[]数组，当前节点的所在链的顶端节点
son[]数组，重儿子
deep[]数组，当前节点的深度
fa[]数组，当前节点的父亲
idx[]数组，树中每个节点剖分后的新编号
rnk[]数组，idx的逆，表示线段上中当前位置表示哪个节点
*/

const int maxn = 1e5+5;

int sz[maxn], top[maxn], son[maxn], deep[maxn], fa[maxn], idx[maxn], rnk[maxn];
int tot;
int n, le, re;
ll k;

struct HLD {
#define type int

    struct edge {
        int a, b;
        type v;

        edge(int _a, int _b, type _v = 0) : a(_a), b(_b), v(_v) {}
    };

    struct node {
        int to;
        type w;

        node() {}

        node(int _to, type _w) : to(_to), w(_w) {}
    };

    vector<int> mp[maxn];
    vector<edge> e;

    void init(int _n) {
        n = _n;
        for (int i = 0; i <= n; i++) mp[i].clear();
        e.clear();
        e.push_back(edge(0, 0));
    }

    void add_edge(int a, int b, type v = 0) {
//        e.push_back(edge(a,b,v));
        mp[a].push_back(b);
        mp[b].push_back(a);
    }

    void dfs1(int x, int pre, int h) {
        int i, to;
        deep[x] = h;
        fa[x] = pre;
        sz[x] = 1;
        for (i = 0; i < (int) (mp[x].size()); i++) {
            to = mp[x][i];
            if (to == pre) continue;
            dfs1(to, x, h + 1);
            sz[x] += sz[to];
            if (son[x] == -1 || sz[to] > sz[son[x]]) son[x] = to;
        }
    }

    void dfs2(int x, int tp) {
        int i, to;
        top[x] = tp;
        idx[x] = ++tot;
        rnk[idx[x]] = x;
        if (son[x] == -1) return;
        dfs2(son[x], tp);
        for (i = 0; i < (int) (mp[x].size()); i++) {
            to = mp[x][i];
            if (to != son[x] && to != fa[x]) dfs2(to, to);
        }
    }

    void work(int _rt = 1) {
        memset(son, -1, sizeof son);
        tot = 0;
        dfs1(_rt, 0, 0);
        dfs2(_rt, _rt);
    }

    int LCA(int x, int y) {
        while (top[x] != top[y]) {
            if (deep[top[x]] < deep[top[y]]) swap(x, y);
            x = fa[top[x]];
        }
        if (deep[x] > deep[y]) swap(x, y);
        return x;
    }

    void modify_node(int x, int y, type val) {
        while (top[x] != top[y]) {
            if (deep[top[x]] < deep[top[y]]) swap(x, y);
            le = idx[top[x]], re = idx[x];
            k = val;
            update(1, 1, n);
            x = fa[top[x]];
        }
        if (deep[x] > deep[y]) swap(x, y);
        le = idx[x], re = idx[y];
        k = val;
        update(1, 1, n);
    }

    type query_node(int x, int y) {
        type res = 0;
        while (top[x] != top[y]) {
            if (deep[top[x]] < deep[top[y]]) swap(x, y);
            le = idx[top[x]], re = idx[x];
            res += query(1, 1, n);
            x = fa[top[x]];
        }
        if (deep[x] > deep[y]) swap(x, y);
        le = idx[x], re = idx[y];
        res += query(1, 1, n);
        return res;
    }

    //path
//    void init_path()
//    {
//        v[idx[rt]]=0;
//        for(int i=1;i<n;i++)
//        {
//            if(deep[e[i].a]<deep[e[i].b]) swap(e[i].a,e[i].b);
//            a[idx[e[i].a]]=e[i].v;
//        }
//        build(n);
//    }
    void modify_edge(int id, type val) {
        if (deep[e[id].a] > deep[e[id].b]) {
            le = idx[e[id].a], re = idx[e[id].a];
            k = val;
            update(1, 1, n);
        } else {
            le = idx[e[id].b], re = idx[e[id].b];
            k = val;
            update(1, 1, n);
        }
    }

    void modify_path(int x, int y, type val) {
        while (top[x] != top[y]) {
            if (deep[top[x]] < deep[top[y]]) swap(x, y);
            le = idx[top[x]], re = idx[x];
            k = val;
            update(1, 1, n);
            x = fa[top[x]];
        }
        if (deep[x] > deep[y]) swap(x, y);
        if (x != y) {
            le = idx[x] + 1, re = idx[y];
            k = val;
            update(1, 1, n);
        }
    }

    type query_path(int x, int y) {
        type res = 0;
        while (top[x] != top[y]) {
            if (deep[top[x]] < deep[top[y]]) swap(x, y);
            le = idx[top[x]], re = idx[x];
            res += query(1, 1, n);
            x = fa[top[x]];
        }
        if (deep[x] > deep[y]) swap(x, y);
        if (x != y) {
            le = idx[x] + 1, re = idx[y];
            res += query(1, 1, n);
        }
        return res;
    }

#undef type
} hld;
```

3.3 RMQ

3.3.1 RMQ

```
int A[maxn];
int maxx[maxn][22];
void RMQ(int n) {
    for (int i = 1; i <= n; i++)
        maxx[i][0] = A[i];
    for (int j = 1; (1 << j) <= n; j++) {
        for (int i = 1; i + (1 << j) - 1 <= n; i++) {
            maxx[i][j] = max(maxx[i][j - 1], maxx[i + (1 << (j - 1))][j - 1]);
        }
    }
}
int query(int l, int r) {
    int k = 0;
    while ((1 << (k + 1)) <= r - l + 1) k++;
    return max(maxx[l][k], maxx[r - (1 << k) + 1][k]);
}

template <typename T, class F = function<T(const T&, const T&)>>
class SparseTable {
 public:
  int n;
  vector<vector<T>> mat;
  F func;

  SparseTable(const vector<T>& a, const F& f) : func(f) {
    n = static_cast<int>(a.size());
    int max_log = 32 - __builtin_clz(n);
    mat.resize(max_log);
    mat[0] = a;
    for (int j = 1; j < max_log; j++) {
      mat[j].resize(n - (1 << j) + 1);
      for (int i = 0; i <= n - (1 << j); i++) {
        mat[j][i] = func(mat[j - 1][i], mat[j - 1][i + (1 << (j - 1))]);
      }
    }
  }

  T get(int from, int to) const {
    assert(0 <= from && from <= to && to <= n - 1);
    int lg = 32 - __builtin_clz(to - from + 1) - 1;
    return func(mat[lg][from], mat[lg][to - (1 << lg) + 1]);
  }
};
//静态区间最大值
SparseTable<int> solve(v, [&](int i, int j) {return max(i, j);});
```

3.3.2 RMQbyIndex

```
//下标RMQ
int v[MAX],maxx[MAX][22],minn[MAX][22];
int pmax(int a,int b){return v[a]>v[b]?a:b;}
int pmin(int a,int b){return v[a]<v[b]?a:b;}
void RMQ(int n) {
    int i,j;
    for(i=1;i<=n;i++) {
        maxx[i][0]=minn[i][0]=i;
    }
    for(j=1;1<<(j-1)<=n;j++) {
        for(i=1;i+(1<<j)-1<=n;i++) {
            int t=1<<(j-1);
            maxx[i][j]=pmax(maxx[i][j-1],maxx[i+t][j-1]);
            minn[i][j]=pmin(minn[i][j-1],minn[i+t][j-1]);
        }
    }
}
int query(int l,int r) {
    int j=(int)(log10(r-l+1)/log10(2))+1;
    int i=r-(1<<(j-1))+1;
    return pmax(maxx[l][j-1],maxx[i][j-1]);
//  return pmin(minn[l][j-1],minn[i][j-1]);
}
```

3.3.3 RMQinNM

```
//二维RMQ
int v[302][302];
int maxx[302][302][9][9],minn[302][302][9][9];
void RMQ(int n,int m)
{
    int i,j,ii,jj;
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=m;j++)
        {
            maxx[i][j][0][0]=minn[i][j][0][0]=v[i][j];
        }
    }
    for(ii=0;(1<<ii)<=n;ii++)
    {
        for(jj=0;(1<<jj)<=m;jj++)
        {
            if(ii+jj)
            {
                for(i=1;i+(1<<ii)-1<=n;i++)
                {
                    for(j=1;j+(1<<jj)-1<=m;j++)
                    {
                        if(ii)
                        {
                            minn[i][j][ii][jj]=min(minn[i][j][ii-1][jj],minn[i+(1<<(ii-1))][j][ii-1][jj]);
                            maxx[i][j][ii][jj]=max(maxx[i][j][ii-1][jj],maxx[i+(1<<(ii-1))][j][ii-1][jj]);
                        }
                        else
                        {
                            minn[i][j][ii][jj]=min(minn[i][j][ii][jj-1],minn[i][j+(1<<(jj-1))][ii][jj-1]);
                            maxx[i][j][ii][jj]=max(maxx[i][j][ii][jj-1],maxx[i][j+(1<<(jj-1))][ii][jj-1]);
                        }
                    }
                }
            }
        }
    }
}
int query(int x1,int y1,int x2,int y2)
{
    int k1=0;
    while((1<<(k1+1))<=x2-x1+1) k1++;
    int k2=0;
    while((1<<(k2+1))<=y2-y1+1) k2++;
    x2=x2-(1<<k1)+1;
    y2=y2-(1<<k2)+1;
    return max(max(maxx[x1][y1][k1][k2],maxx[x1][y2][k1][k2]),max(maxx[x2][y1][k1][k2],maxx[x2][y2][k1][k2]))
//  return min(min(minn[x1][y1][k1][k2],minn[x1][y2][k1][k2]),min(minn[x2][y1][k1][k2],minn[x2][y2][k1][k2]));
}
```

3.4 MO

3.4.1 MO

```
// const int maxn = 50005;

struct MO {
    int l, r, id;
}q[maxn];

int n, m, col[maxn], block, belong[maxn];
int vis[maxn * 10];
ll res[maxn], ans;
bool cmp(const MO& a, const MO& b) { return belong[a.l] == belong[b.l] ? a.r < b.r : a.l < b.l; }
```

```
void add(int x) {
    vis[x] ++;
    ans += 1ll * x * (vis[x] * vis[x] - (vis[x] - 1) * (vis[x] - 1));
}

void del(int x) {
    vis[x] --;
    ans -= 1ll * x * ((vis[x] + 1) * (vis[x] + 1) - vis[x] * vis[x]);
}

int main() {
    scanf("%d%d", &n, &m);
    block = sqrt(n);
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &col[i]);
        belong[i] = i / block + 1;
    }
    for (int i = 1; i <= m; ++i) {
        scanf("%d%d", &q[i].l, &q[i].r);
        q[i].id = i;
    }
    sort(q + 1, q + 1 + m, cmp);
    int l = 1, r = 0;
    for (int i = 1; i <= m; ++i) {
        while(r < q[i].r) add(col[++r]);
        while(r > q[i].r) del(col[r--]);
        while(l < q[i].l) del(col[l++]);
        while(l > q[i].l) add(col[--l]);
        res[q[i].id] = ans;
    }
    for (int i = 1; i <= m; ++i) printf("%lld\n", res[i]);
    return 0;
}
```

3.4.2 MObyModify

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;
const int maxn = 50005;

struct MO {
    int l, r, id, oppre;
}q[maxn];

int n, m, col[maxn], block, belong[maxn], colpre[maxn];
int changepos[maxn], changepre[maxn], changenow[maxn];
int vis[maxn * 20];
int ans;
int res[maxn];
bool cmp(const MO& a, const MO& b) {
    if (belong[a.l] != belong[b.l]) return a.l < b.l;
    if (belong[a.r] != belong[b.r]) return a.r < b.r;
    return a.oppre < b.oppre;
}
void add(int x) {}

void del(int x) {}
```

```cpp

void unmodify(int pos, int now) {
    if (q[pos].l <= changepos[now] && changepos[now] <= q[pos].r) {
        del(changenow[now]);
        add(changepre[now]);
    }
    col[changepos[now]] = changepre[now];
}

void modify(int pos, int now) {
    if (q[pos].l <= changepos[now] && changepos[now] <= q[pos].r) {
        del(changepre[now]);
        add(changenow[now]);
    }
    col[changepos[now]] = changenow[now];
}

int main() {
    scanf("%d%d", &n, &m);
    block = pow(n, 0.66666);
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &col[i]);
        colpre[i] = col[i];
        belong[i] = i / block + 1;
    }
    char s[2];
    int t = 0, t2 = 0;
    for (int i = 1; i <= m; ++i) {
        scanf("%s", s);
        if (s[0] == 'Q') {
            ++t;
            scanf("%d%d", &q[t].l, &q[t].r);
            q[t].oppre = t2;
            q[t].id = t;
        } else {
            ++t2;
            scanf("%d%d", &changepos[t2], &changenow[t2]);
            changepre[t2] = colpre[changepos[t2]];
            colpre[changepos[t2]] = changenow[t2];
        }
    }
    sort(q + 1, q + 1 + t, cmp);
    int l = 1, r = 0, now = 0;
    for (int i = 1; i <= t; ++i) {
        while(r < q[i].r) add(col[++r]);
        while(r > q[i].r) del(col[r--]);
        while(l < q[i].l) del(col[l++]);
        while(l > q[i].l) add(col[--l]);
        while (now < q[i].oppre) modify(i, ++now);
        while (now > q[i].oppre) unmodify(i, now--);
        res[q[i].id] = ans;
    }
    for (int i = 1; i <= t; ++i) printf("%d\n", res[i]);
    return 0;
}
```

3.4.3 分块

```cpp
// 非预处理数组版
```

```
inline int belong(int x) { return (x - 1) / block + 1; }
inline int lpos(int x) { return 1 + (x - 1) * block; }
inline int rpos(int x) { return min(n, x * block); }
int sz = (n - 1) / block + 1;

// 预处理版，maxn大于1e6已经不可能处理了
const int maxb = 1005;
int n, m;
int belong[maxn], lpos[maxb], rpos[maxb];
int val[maxn], lazy[maxb];
int block;

scanf("%d", &n);
block = sqrt(n);
for (int i = 1; i <= n; ++i) {
    scanf("%d", &val[i]);
    belong[i] = (i - 1) / block + 1;
}
int sz = (n - 1) / block + 1;
for (int i = 1; i <= sz; ++i) {
    lpos[i] = 1 + (i - 1) * block;
    rpos[i] = i * block;
}
rpos[sz] = n;
```

3.4.4 弹飞绵羊

```
int n, m;
int belong[maxn], lpos[maxn], rpos[maxn];
int val[maxn], nxt[maxn], k[maxn], lst[maxn];
int block;

void update(int pos) {
    int llim = lpos[belong[pos]], rlim = rpos[belong[pos]];
    for (int i = pos; i >= llim; --i) {
        if (val[i] + i > rlim) {
            k[i] = 1;
            nxt[i] = val[i] + i;
            if (val[i] + i > n) lst[i] = i;
            else lst[i] = lst[nxt[i]];
        } else {
            k[i] = 1 + k[val[i] + i];
            nxt[i] = nxt[val[i] + i];
            lst[i] = lst[val[i] + i];
        }
    }
}

void init() {
    for (int i = n; i >= 1; --i) {
        int rlim = rpos[belong[i]];
        if (val[i] + i > rlim) {
            k[i] = 1;
            nxt[i] = val[i] + i;
            if (val[i] + i > n) lst[i] = i;
            else lst[i] = lst[nxt[i]];
        } else {
            k[i] = 1 + k[val[i] + i];
            nxt[i] = nxt[val[i] + i];
            lst[i] = lst[val[i] + i];
        }
    }
}

int query(int pos) {
    int res = 0;
    while (pos <= n) {
        res += k[pos];
        if (nxt[pos] > n) printf("%d ", lst[pos]);
        pos = nxt[pos];
    }
    return res;
}

int main(int argc, char* argv[]) {
    scanf("%d%d", &n, &m);
    block = sqrt(n) * 1.6 + 1;
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &val[i]);
        belong[i] = (i - 1) / block + 1;
    }
    int sz = (n - 1) / block + 1;
    for (int i = 1; i <= sz; ++i) {
        lpos[i] = 1 + (i - 1) * block;
        rpos[i] = i * block;
    }
    rpos[sz] = n;
    init();
    while (m--) {
        int op;
        scanf("%d", &op);
        if (op == 1) {
            int pos;
            scanf("%d", &pos);
            printf("%d\n", query(pos));
        } else {
            int pos, kl;
            scanf("%d%d", &pos, &kl);
            val[pos] = kl;
            update(pos);
        }
    }
    return 0;
}
```

3.4.5 树莫队

```
// rnk保存欧拉序
int sz[maxn], top[maxn], son[maxn], deep[maxn], fa[maxn], idx[maxn], ed[maxn], rnk[maxn*2];
int tot, n, m;
vector<int> edge[maxn];
int val[maxn];
vector<int> xpos;

inline void dfs1(int u, int pre, int h) {
    deep[u] = h;
    fa[u] = pre;
    sz[u] = 1;
    for (auto to : edge[u]) {
        if (to == pre) continue;
        dfs1(to, u, h + 1);
        sz[u] += sz[to];
        if (son[u] == 0 || sz[to] > sz[son[u]]) son[u] = to;
    }
}

inline void dfs2(int u, int tp) {
    top[u] = tp;
    idx[u] = ++tot, rnk[tot] = u;
    if (son[u] == 0) {
        ed[u] = ++tot, rnk[tot] = u;
        return;
    }
    dfs2(son[u], tp);
    for (auto to : edge[u]) {
        if (to != son[u] && to != fa[u]) dfs2(to, to);
    }
    ed[u] = ++tot, rnk[tot] = u;
}

inline int LCA(int x, int y) {
    while (top[x] != top[y]) {
        if (deep[top[x]] < deep[top[y]]) swap(x, y);
        x = fa[top[x]];
    }
    if (deep[x] > deep[y]) swap(x, y);
    return x;
}

int belong[maxn*2], block;
int res[maxn], ans;
// 每个点是否访问（欧拉序去重）
int vis[maxn];
// 标记数组
int pre[maxn];
struct MO {
    int l, r, id, lca;
    bool operator < (const MO& oth) const {
        return belong[l] == belong[oth.l] ? r < oth.r : belong[l] < belong[oth.l];
    }
}q[maxm];

inline void add(int x) {
    pre[x] ++;
    if (pre[x] == 1) ans ++;
}

inline void del(int x) {
    pre[x] --;
    if (pre[x] == 0) ans --;
}

inline void deal(int x) {
    vis[x] ? del(val[x]) : add(val[x]);
    vis[x] = !vis[x];
```

```
}

int main(int argc, char* argv[]) {
    scanf("%d%d", &n, &m);
    block = sqrt(n);
    xpos.resize(n + 1);
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &val[i]);
        xpos[i] = val[i];
    }
    sort(xpos.begin(), xpos.end());
    xpos.erase(unique(xpos.begin(), xpos.end()), xpos.end());
    for (int i = 1; i <= n; ++i) val[i] = lower_bound(xpos.begin(), xpos.end(), val[i]) - xpos.begin();
    // 欧拉序长度为n两倍所以分块要分两倍大小
    for (int i = 1; i <= n * 2; ++i) {
        belong[i] = (i - 1) / block + 1;
    }
    for (int i = 1, u, v; i < n; ++i) {
        scanf("%d%d", &u, &v);
        edge[u].push_back(v);
        edge[v].push_back(u);
    }
    // 树剖预处理lca
    dfs1(1, 0, 0);
    dfs2(1, 1);
    for (int i = 1, x, y; i <= m; ++i) {
        scanf("%d%d", &x, &y);
        if (idx[x] > idx[y]) swap(x, y);
        int _lca = LCA(x, y);
        q[i].id = i;
        if (_lca == x) q[i].l = idx[x], q[i].r = idx[y], q[i].lca = 0;
        else q[i].l = ed[x], q[i].r = idx[y], q[i].lca = _lca;
//         cerr << q[i].l << " " << q[i].r << " " << q[i].id << " " << q[i].lca << endl;
    }
    sort(q + 1, q + 1 + m);
    int l = 1, r = 0;
    for (int i = 1; i <= m; ++i) {
        while(r < q[i].r) deal(rnk[++r]);
        while(r > q[i].r) deal(rnk[r--]);
        while(l < q[i].l) deal(rnk[l++]);
        while(l > q[i].l) deal(rnk[--l]);
        if (q[i].lca) deal(q[i].lca);
        res[q[i].id] = ans;
        if (q[i].lca) deal(q[i].lca);
    }
    for (int i = 1; i <= m; ++i) {
        printf("%d\n", res[i]);
    }
    return 0;
}
```

3.5 VirtualTree

3.5.1 VirtualTree

```
const int pow2 = 19;
const int maxn = 1 << pow2;
vector<int> adj0[maxn], adj1[maxn];
```

```
int st[maxn << 1][pow2 + 1], dep[maxn], euler[maxn], euler_clock;
// fa0 是原树的父节点
// fa1 是虚树的父节点
// len 是虚树每个节点的权重，每个节点代表原树的几个节点，也是虚树到它父节点的链的长度
int stk[maxn], fa0[maxn], fa1[maxn], len[maxn];
ll val[maxn];

void link0(int u, int v) { adj0[u].emplace_back(v); adj0[v].emplace_back(u); }
void link1(int u, int v) { adj1[u].emplace_back(v); adj1[v].emplace_back(u); }
void dfs0(int u, int p) {
    fa0[u] = p;
    dep[u] = dep[p] + 1;
    st[++euler_clock][0] = u;
    euler[u] = euler_clock;
    for (const auto& v : adj0[u]) if (v != p) {
            dfs0(v, u);
            st[++euler_clock][0] = u;
        }
}
inline bool cmp(int u, int v) {return dep[u] < dep[v];}
inline int upper(int u, int v) {return cmp(u, v) ? u : v;}
void lca_init() {
    for (int i = 0; i != 31 - __builtin_clz(euler_clock); ++i)
        for (int j = 1; j + (1 << (i + 1)) <= euler_clock; ++j)
            st[j][i + 1] = upper(st[j][i], st[j + (1 << i)][i]);
}
inline int lca(int u, int v) {
    if (u == v) return u;
    u = euler[u];
    v = euler[v];
    if (u > v) swap(u, v);
    int temp = 31 - __builtin_clz(++v - u);
    return upper(st[u][temp], st[v - (1 << temp)][temp]);
}
void build(vector<int>& key) {
    sort(key.begin(), key.end(), [&] (int u, int v) { return euler[u] < euler[v]; });
    key.resize(unique(key.begin(), key.end()) - key.begin());
    int top = 0;
    for (const auto& u : key) {
        if (!top) {
            stk[++top] = u;
            continue;
        }
        int p = lca(u, stk[top]);
        while (euler[p] < euler[stk[top]]) {
            if (euler[p] >= euler[stk[top - 1]]) {
                link1(p, stk[top]);
                if (stk[--top] != p) stk[++top] = p;
                break;
            }
            link1(stk[top - 1], stk[top]);
            --top;
        }
        stk[++top] = u;
    }
    while (top > 1) {
        link1(stk[top - 1], stk[top]);
        --top;
    }
}

void dfs1(int u, int p) {
    fa1[u] = p;
    val[u] = 0;
    len[u] = dep[u] - dep[p];
    for (const auto& v : adj1[u]) if (v != p) dfs1(v, u);
}

int main() {
    // 多组清空操作
    for (int i = 1; i <= n; ++i) {
        adj0[i].clear();
        adj1[i].clear();
    }
    euler_clock = 0;

    // 读入原树 link0 加边
    // 读入处理关键节点存入vector key，包含1和链的端点和他们的lca的父节点（lca如果为1就不加）。
    dfs0(1, 0);
    lca_init();

    vector<int> key(1, 1);
    for (auto& q : query) {
        cin >> q.u >> q.v;
        key.emplace_back(q.u);
        key.emplace_back(q.v);
        int p = lca(q.u, q.v);
        if (p != 1) key.emplace_back(fa0[p]);
    }

    build(key);
    dfs1(1, 0);
    return 0;
}
```

3.6 PersistentDS

3.6.1 主席树区间 k 大

```
// const int maxn = 100005;
int n, m;
int a[maxn];
int root[maxn];
int cnt = 0;
vector<int> b;
struct node {
    int l, r, val;
}p[maxn * 40];

void update(int l, int r, int pre, int &now, int pos) {
    now = ++cnt;
    p[now] = p[pre];
    p[now].val++;
    if (l == r) {
        return;
    }
    int mid = l + r >> 1;
    if (pos <= mid) update(l, mid, p[pre].l, p[now].l, pos);
    else update(mid + 1, r, p[pre].r, p[now].r, pos);
}

int query(int l, int r, int x, int y, int k) {
    if (l == r) return b[l - 1];
    int mid = l + r >> 1;
    int temp = p[p[y].l].val - p[p[x].l].val;
    if (k <= temp) return query(l, mid, p[x].l, p[y].l, k);
    return query(mid + 1, r, p[x].r, p[y].r, k - temp);
}

int main(int argc,char *argv[])
{
    while (scanf("%d%d", &n, &m) != EOF) {
        b.clear();
        cnt = 0;
        for (int i = 1; i <= n; ++i) scanf("%d", &a[i]), b.push_back(a[i]);
        sort(b.begin(), b.end());
        b.erase(unique(b.begin(), b.end()), b.end());
        for (int i = 1; i <= n; ++i) {
            update(1, b.size(), root[i - 1], root[i], lower_bound(b.begin(), b.end(), a[i]) - b.begin() + 1);
        }
        int L, R, k;
        while (m--) {
            scanf("%d%d%d", &L, &R, &k);
            printf("%d\n", query(1, b.size(), root[L - 1], root[R], k));
        }
    }
    return 0;
}
```

3.6.2 可持久化数组

```
/*1、操作将u, v合并 2、操作回退 */
const int maxn = 2e5+5;
int n, m, sz;
int root[maxn],ls[maxn*40],rs[maxn*40],v[maxn*40],deep[maxn*40];
int has[maxn];

void build(int &k, int l, int r) {
    if (!k)k = ++sz;
    if (l == r) {
        v[k] = l;
        return;
    }
    int mid = (l + r) >> 1;
    build(ls[k], l, mid);
    build(rs[k], mid + 1, r);
}

void modify(int l, int r, int x, int &y, int pos, int val) {
    y = ++sz;
    if (l == r) {
        v[y] = val;
        deep[y] = deep[x];
        return;
    }
    ls[y] = ls[x];
    rs[y] = rs[x];
    int mid = (l + r) >> 1;
    if (pos <= mid)
        modify(l, mid, ls[x], ls[y], pos, val);
    else modify(mid + 1, r, rs[x], rs[y], pos, val);
}

int query(int k, int l, int r, int pos) {
    if (l == r)return k;
    int mid = (l + r) >> 1;
    if (pos <= mid)return query(ls[k], l, mid, pos);
    else return query(rs[k], mid + 1, r, pos);
}

void add(int k, int l, int r, int pos) {
    if (l == r) {
        deep[k]++;
        return;
    }
    int mid = (l + r) >> 1;
    if (pos <= mid)add(ls[k], l, mid, pos);
    else add(rs[k], mid + 1, r, pos);
}

int find(int k, int x) {
    int p = query(k, 1, n, x);
    if (x == v[p])return p;
    return find(k, v[p]);
}

int main() {
    int T = read();
    while (T--) {
        sz = 0;
        memset(root, 0, sizeof root);
        memset(ls, 0, sizeof ls);
        memset(rs, 0, sizeof rs);
        n = read();
        has[0] = n;
        m = read();
        build(root[0], 1, n);
        int f, k, a, b;
        for (int i = 1; i <= m; i++) {
            f = read();
            if (f == 1) {
                root[i] = root[i - 1];
                has[i] = has[i - 1];
                a = read();
                b = read();
                int p = find(root[i], a), q = find(root[i], b);
                if (v[p] == v[q])continue;
                has[i]--;
                if (deep[p] > deep[q])swap(p, q);
                modify(1, n, root[i - 1], root[i], v[p], v[q]);
                if (deep[p] == deep[q])add(root[i], 1, n, v[q]);
            } else if (f == 2) {
                k = read();
                root[i] = root[k];
                has[i] = has[k];
            }
            printf("%d\n", has[i]);
        }
    }
    return 0;
}
```

3.7 Tree

3.7.1 LCA

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

// 普通倍增lca
int n, dep[maxn], fa[maxn][30];
vector<int> edge[maxn];

void dfs(int u, int pre) {
    dep[u] = dep[pre] + 1, fa[u][0] = pre;
    for(int i = 1; (1 << i) <= n; i ++)
        fa[u][i] = fa[fa[u][i - 1]][i - 1];
    for(auto v : edge[u]) if(v != pre) dfs(v, u);
}

int LCA(int u, int v) {
    if(dep[u] < dep[v]) swap(u, v);
    int d = dep[u] - dep[v];
    for(int i = 0; (1 << i) <= d; i ++)
        if((1 << i) & d) u = fa[u][i];
    if(u == v) return u;
    for(int i = 20; i >= 0; i --)
        if(fa[u][i] != fa[v][i])
            u = fa[u][i], v = fa[v][i];
    return fa[u][0];
}


// 欧拉序lca
// pow2 = 19
// maxn = 1 << pow2
int st[maxn << 1][pow2 + 1], dep[maxn], euler[maxn], euler_clock,  fa[maxn];
void dfs(int u, int p) {
    fa[u] = p;
    dep[u] = dep[p] + 1;
    st[++euler_clock][0] = u;
    euler[u] = euler_clock;
    for (const auto& v : adj0[u]) if (v != p) {
            dfs(v, u);
            st[++euler_clock][0] = u;
        }
}
void lca_init() {
    for (int i = 0; i != 31 - __builtin_clz(euler_clock); ++i)
        for (int j = 1; j + (1 << (i + 1)) <= euler_clock; ++j)
            st[j][i + 1] = upper(st[j][i], st[j + (1 << i)][i]);
}
inline int lca(int u, int v) {
    if (u == v) return u;
    u = euler[u];
    v = euler[v];
    if (u > v) swap(u, v);
    int temp = 31 - __builtin_clz(++v - u);
    return upper(st[u][temp], st[v - (1 << temp)][temp]);
}

// dfs(1, 0);
// lca_init();

// 另有树剖lca详见hld模板
```

3.7.2 前向星

```
// 清零 head 和 tot
const int maxm = 4e5+5;
int ver[maxm], Next[maxm], head[maxn], edge[maxm];
void addEdge(int u, int v, int w){
    ver[++tot]=v;
    Next[tot]=head[u];
    head[u]=tot;
    edge[tot]=w;
}

for(int i = head[u]; i; i=Next[i])
```

3.7.3 点分治

```
int n, k;

// 清零 head 和 tot
const int maxm = maxn * 2;
int ver[maxm], Next[maxm], head[maxn], edge[maxm];
int tot;
void addEdge(int u, int v, int w){
    ver[++tot]=v;
    Next[tot]=head[u];
    head[u]=tot;
    edge[tot]=w;
}

int sz[maxn], vis[maxn];
int rt, mxsz, has;

void getrt(int u, int pre) {
    sz[u] = 1;
    int mxnow = 0;
    for (int i = head[u]; i; i = Next[i]) {
        int v = ver[i];
        if (v == pre || vis[v]) continue;
        getrt(v, u);
        sz[u] += sz[v];
        mxnow = max(mxnow, sz[v]);
    }
    mxnow = max(mxnow, has - sz[u]);
    if (mxnow < mxsz) {
        mxsz = mxnow, rt = u;
    }
```

```
}

int dl[maxn], r;
int val[maxn];

void getdis(int u, int pre) {
    dl[r++] = val[u];
    for (int i = head[u]; i; i = Next[i]) {
        int v = ver[i];
        if (v == pre || vis[v]) continue;
        val[v] = val[u] + edge[i];
        getdis(v, u);
    }
}

ll cal(int u, int pre) {
    r = 0;
    val[u] = pre;
    getdis(u, 0);
    ll sum = 0;
    sort(dl, dl + r);
    r --;
    int l = 0;
    while (l < r) {
        if (dl[l] + dl[r] > k) r --;
        else sum += r - l, l ++;
    }
    return sum;
}

ll res = 0;
void dfs(int u) {
    res += cal(u, 0);
    vis[u] = 1;
    for (int i = head[u]; i; i = Next[i]) {
        int v = ver[i];
        if (vis[v]) continue;
        res -= cal(v, edge[i]);
        has = sz[v];
        mxsz = 0x3f3f3f3f;
        getrt(v, 0);
        dfs(rt);
    }
}

int main(int argc, char* argv[]) {
    while (scanf("%d%d", &n, &k) != EOF && (n || k)) {
        tot = 0; memset(head, 0, sizeof head);
        memset(vis, 0, sizeof vis);
        res = 0;
        for (int i = 1, u, v, w; i < n; ++i) {
            scanf("%d%d%d", &u, &v, &w);
            addEdge(u, v, w);
            addEdge(v, u, w);
        }
        mxsz = 0x3f3f3f3f;
        has = n;
        getrt(1, 0);
        dfs(rt);
        printf("%lld\n", res);
    }
    return 0;
}
```

3.8 Others

3.8.1 BITinNM

```
struct Fenwick_Tree {
#define type int
    type bit[maxn][maxn];
    int n, m;
    void init(int _n, int _m) {
        n = _n;
        m = _m;
        mem(bit, 0);
    }
    int lowbit(int x) { return x & (-x); }
    void update(int x, int y, type v) {
        int i, j;
        for (i = x; i <= n; i += lowbit(i)) {
            for (j = y; j <= m; j += lowbit(j)) {
                bit[i][j] += v;
            }
        }
    }
    type get(int x, int y) {
        type i, j, res = 0;
        for (i = x; i > 0; i -= lowbit(i)) {
            for (j = y; j > 0; j -= lowbit(j)) {
                res += bit[i][j];
            }
        }
        return res;
    }
    type query(int x1, int x2, int y1, int y2) {
        x1--;
        y1--;
        return get(x2, y2) - get(x1, y2) - get(x2, y1) + get(x1, y1);
    }
#undef type
} tr;

// 二维区间前缀和写法 (非树状数组)
inline void range_add(int xa, int ya, int xb, int yb) { add(xa, ya, 1), add(xa, yb + 1, -1), add(xb + 1, ya, -1), add(xb + 1, yb + 1, 1); }
inline ll range_ask(int xa, int ya, int xb, int yb){ return ask(xb, yb) - ask(xb, ya - 1) - ask(xa - 1, yb) + ask(xa - 1, ya - 1); }
inline void build() {
    for (int i = 1; i < n + 5; ++i) {
        for (int j = 1; j < m + 5; ++j) {
            if (st[i][j] > 1) st[i][j] = 1;
            st[i][j] += st[i - 1][j] + st[i][j - 1] - st[i - 1][j - 1];
        }
    }
}

// 二维树状数组区间加与求和
```

```
ll t1[maxn][maxn], t2[maxn][maxn], t3[maxn][maxn], t4[maxn][maxn];
void add(ll x, ll y, ll z){
    for(int X = x; X <= n; X += X & -X)
        for(int Y = y; Y <= m; Y += Y & -Y){
            t1[X][Y] += z;
            t2[X][Y] += z * x;
            t3[X][Y] += z * y;
            t4[X][Y] += z * x * y;
        }
}
ll ask(ll x, ll y){
    ll res = 0;
    for(int i = x; i; i -= i & -i)
        for(int j = y; j; j -= j & -j)
            res += (x + 1) * (y + 1) * t1[i][j]
                - (y + 1) * t2[i][j]
                - (x + 1) * t3[i][j]
                + t4[i][j];
    return res;
}

// 区间加，询问单点：直接维护前缀差分数组，求单点=普通求前缀和
```

3.8.2 静态区间 k 大划分树

```
// const int maxn = 100010;
int tree[20][maxn];
// 读入sorted并排序，赋值给tree的第0层
int sorted[maxn];
int toleft[20][maxn];
// 保存左子树的和
// ll sum[20][maxn];

// 1, n, 0
void build(int l, int r, int dep) {
    if (l == r) return;
    // sum[dep][0] = 0;
    toleft[dep][0] = 0;
    int mid = l + r >> 1;
    int same = mid - l + 1;
    for (int i = l; i <= r; ++i) {
        if (tree[dep][i] < sorted[mid]) same--;
    }
    int lpos = l, rpos = mid + 1;
    for (int i = l; i <= r; ++i) {
        // sum[dep][i] = sum[dep][i - 1];
        if (tree[dep][i] < sorted[mid]) {
            // sum[dep][i] += tree[dep][i];
            tree[dep + 1][lpos++] = tree[dep][i];
        }
        else if (tree[dep][i] == sorted[mid] && same > 0) {
            // sum[dep][i] += tree[dep][i];
            tree[dep + 1][lpos++] = tree[dep][i];
            same --;
        } else tree[dep + 1][rpos ++] = tree[dep][i];
        toleft[dep][i] = toleft[dep][l - 1] + lpos - l;
    }
    build(l, mid, dep + 1);
    build(mid + 1, r, dep + 1);
}

//(1~k-1)的数的和，注意每次查询前初始化
// ll ress = 0;

// L = 1, R = n， dep = 0, l,r是查询区间
int query(int L, int R, int l, int r, int dep, int k) {
    if (l == r) return tree[dep][l];
    int mid = (L + R) >> 1;
    int cnt = toleft[dep][r] - toleft[dep][l - 1];
    if (cnt >= k) {
        int newl = L + toleft[dep][l - 1] - toleft[dep][L - 1];
        int newr = newl + cnt - 1;
        return query(L, mid, newl, newr, dep + 1, k);
    } else {
        int newr = r + toleft[dep][R] - toleft[dep][r];
        int newl = newr - (r - l - cnt);
        // ress += sum[dep][r] - sum[dep][l - 1];
        return query(mid + 1, R, newl, newr, dep + 1, k - cnt);
    }
}


scan(n), scan(m);
for (int i = 1; i <= n; ++i) {
    scan(sorted[i]);
    tree[0][i] = sorted[i];
}
sort(sorted + 1, sorted + 1 + n);
build(1, n, 0);
int l, r, k;
while (m--) {
    scan(l), scan(r), scan(k);
    printf("%d\n", query(1, n, l, r, 0, k));
}
```

4 String

4.1 KMP

4.1.1 KMP

```
// nxt[0]表示失配到完全不匹配
int nxt[maxm];

void getNext(char *s, int len) {
    int i = 0, j = -1;
    nxt[i] = j;
    while (i < len) {
        if (j == -1 || s[i] == s[j]) nxt[++i] = ++j;
        else j = nxt[j];
    }
}

// a为原串，b为模式串，下标从0开始，找第一个出现模式串的位置（起点为1），找不到返回-1
int KMP(char *a, char *b, int n, int m) {
    getNext(b, m);
    int i = 0, j = 0;
    while (i < n && j < m) {
        if (j == -1 || a[i] == b[j]) ++i, ++j;
        else j = nxt[j];
    }
    return j == m ? i - m + 1 : -1;
}
```

4.1.2 exKMP

```
const int maxn = 1e5 + 10;
int nex[maxn], extend[maxn];

//预处理计算Next数组
void getNext(char *str)
{
    int i = 0, j, po, len = strlen(str);
    nex[0] = len;     //初始化nex[0]
    while (str[i] == str[i + 1] && i + 1 < len) i++;   //计算nex[1]
    nex[1] = i;
    po = 1;   //初始化po的位置
    for (int i = 2; i < len; i++)
    {
        if (nex[i - po] + i < nex[po] + po)  //第一种情况，可以直接得到nex[i]的值
            nex[i] = nex[i - po];
        else    //第二种情况，要继续匹配才能得到nex[i]的值
        {
            j = nex[po] + po - i;
            if (j < 0) j = 0;    //如果i>po+nex[po],则要从头开始匹配
            while (i + j < len && str[j] == str[j + i]) j++;
            nex[i] = j;
            po = i;   //更新po的位置
        }
    }
}

void EXKMP(char *s1, char *s2)
{
    int i = 0, j, po, len = strlen(s1), l2 = strlen(s2);
    getNext(s2);
    while (s1[i] == s2[i] && i < l2 && i < len) i++;
    extend[0] = i;
    po = 0;
    for (int i = 1; i < len; i++)
    {
        if (nex[i - po] + i < extend[po] + po)
            extend[i] = nex[i - po];
        else
        {
            j = extend[po] + po - i;
            if (j < 0) j = 0;
            while (i + j < len && j < l2 && s1[j + i] == s2[j]) j++;
            extend[i] = j;
            po = i;
        }
    }
}
```

4.2 Trie

4.2.1 Trie

```
const int maxn = 2e6 + 10;

int trie[maxn][30], tot;
bool flag[maxn];

void insert_ch(char *str)
{
    int len = strlen(str);
    int root = 0;
    for (int i = 0; i < len; i++)
    {
        int id = str[i] - 'a';
        if (!trie[root][id]) trie[root][id] = ++tot;
        root = trie[root][id];
    }
    flag[root] = true;
}

bool find_ch(char *str)
{
    int len = strlen(str);
    int root = 0;
    for (int i = 0; i < len; i++)
    {
        int id = str[i] - 'a';
        if (!trie[root][id]) return false;
        root = trie[root][id];
    }
    return true;
}
```

4.2.2 Persistence Trie

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

```

int a[maxn], rt[maxn], n;

struct Trie
{
    int tot;
    int child[maxn * 32][2], sum[maxn *32];
    int insert(int x, int val)
    {
        int tmp, y;
        tmp = y= ++tot;
        for(int i = 30; i >= 0; --i)
        {
            child[y][0] = child[x][0];
            child[y][1] = child[x][1];
            sum[y] = sum[x] + 1;
            int t = val >> i & 1;
            x = child[x][t];
            child[y][t] = ++tot;
            y = child[y][t];
        }
        sum[y] = sum[x] + 1;
        return tmp;
    }
    int query(int l, int r, int val)
    {
        int tmp = 0;
        for(int i =30; i >= 0; --i)
        {
            int t = val >> i & 1;
            if(sum[child[r][t^1]] - sum[child[l][t^1]]) tmp += (1<<i), r = child[r][t^1], l = child[l][t ^ 1];
            else r = child[r][t], l = child[l][t];
        }
        return tmp;
    }
}trie;
```

4.2.3 01Trie

```
struct Trie {
    int tree[maxn*20][2], tot;
    int flag[maxn*20];

    void insert_ch(int x) {
        int root = 0;
        flag[0]++;
        for (int i = 30; i >= 0; --i) {
            int id = (x >> i) & 1;
            if (!tree[root][id]) {
                tree[root][id] = ++tot;
                tree[tree[root][id]][0] = tree[tree[root][id]][1] = 0;
                flag[tree[root][id]] = flag[tree[tree[root][id]][0]] = flag[tree[tree[root][id]][1]] = 0;
            }
            root = tree[root][id];
            flag[root]++;
        }
    }

    void del(int x) {
        int root = 0;
        flag[0]--;
        for (int i = 30; i >= 0; --i) {
            int id = (x >> i) & 1;
            assert(tree[root][id]);
            if (flag[tree[root][id]] == 1) {
                flag[tree[root][id]] = 0;
                tree[root][id] = 0;
                return;
            }
            root = tree[root][id];
            flag[root]--;
        }
    }

    int find_ch(int x, int flag = 0) { // flag 0 最小异或值, 1 最大异或值
        int root = 0;
        int res = 0;
        for (int i = 30; i >= 0; --i) {
            int id = ((x >> i) & 1);
            if (flag) id = !id;
            if (tree[root][id]) {
                root = tree[root][id];
                res = res << 1 | id;
            } else {
                root = tree[root][!id];
                res = res << 1 | (!id);
            }
        }
        return res;
    }

    void init() {
        tree[0][0] = tree[0][1] = 0;
        tot = 0;
    }
};
```

4.3 Manachar

4.3.1 Manacher

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

char s[maxn];

char tmp[maxn << 1];
int Len[maxn << 1];

int init(char *str)
{
    int len = strlen(str);
    tmp[0] = '@';
    for (int i = 1; i <= 2 * len; i += 2)
    {
        tmp[i] = '#';
        tmp[i + 1] = str[i / 2];
    }
    tmp[2 * len + 1] = '#';
    tmp[2 * len + 2] = '$';
    tmp[2 * len + 3] = 0;
    return 2 * len + 1;
}

int manacher(char *str)
{
    int mx = 0, ans = 0, pos = 0;
    int len = init(str);
    for (int i = 1; i <= len; i++)
    {
        if (mx > i) Len[i] = min(mx - i, Len[2 * pos - i]);
        else Len[i] = 1;
        while (tmp[i - Len[i]] == tmp[i + Len[i]]) Len[i]++;
        if (Len[i] + i > mx) mx = Len[i] + i, pos = i;
    }
}
```

4.4 Aho-Corasick Automation

4.4.1 AC Automation

```
class AC_automation
{
public:
    int trie[maxn][26], cnt;
    int tag[maxn];
    int fail[maxn], num[maxn], res[maxn], in[maxn], Map[maxn];

    void init()
    {
        memset(trie, 0, sizeof trie);
        memset(tag, 0, sizeof tag);
        memset(fail, 0, sizeof fail);
        cnt = 0;
    }

    void insert(char *str, int id)
    {
        int root = 0;
        for (int i = 0; str[i]; i++)
        {
            int id = str[i] - 'a';
            if (!trie[root][id]) trie[root][id] = ++cnt;
            root = trie[root][id];
        }
        if(!tag[root]) tag[root] = id;
        Map[id] = tag[root];
    }

    void build()
    {
        queue<int> que;
        for (int i = 0; i < 26; i++) if (trie[0][i]) que.push(trie[0][i]);
        while (!que.empty())
        {
            int k = que.front();
            que.pop();
            for (int i = 0; i < 26; i++)
            {
                if (trie[k][i])
                {
                    fail[trie[k][i]] = trie[fail[k]][i];
                    que.push(trie[k][i]);
                    in[fail[trie[k][i]]] ++;
                } else trie[k][i] = trie[fail[k]][i];
            }
        }
    }

    void toposort()
    {
        queue<int> que;
        for(int i = 1; i <= cnt; i ++) if(in[i] == 0) que.push(i);
        while(!que.empty())
        {
            int u = que.front(); que.pop();
            res[tag[u]] = num[u];
            int v = fail[u]; in[v] --;
            num[v] += num[u];
            if(in[v] == 0) que.push(v);
        }
    }

    void query(char *str, int n)
    {
        int u = 0, len = strlen(s);
        for(int i = 0; i < len; i ++)
            u = trie[u][str[i] - 'a'], num[u] ++;
        toposort();
        for(int i = 1; i <= n; i ++) printf("%d\n", res[Map[i]]);
    }
} AC;
```

4.5 Suffix Array

4.5.1 Suffix Array

```
char s[maxn];
int sa[maxn], t[maxn], t2[maxn], c[maxn], n;

//build_sa(n + 1, 130), sa, height下标从1开始,rk下标从0开始
void build_sa(int n, int m)
{
    int *x = t, *y = t2;
    for(int i = 0; i < m; i++) c[i] = 0;
    for(int i = 0; i < n; i++) c[x[i] = s[i]]++;
    for(int i = 1; i < m; i++) c[i] += c[i - 1];
    for(int i = n - 1; i >= 0; i--) sa[--c[x[i]]] = i;
    for(int k = 1; k <= n; k <<= 1)
    {
        int p = 0;
        for(int i = n - k; i < n; i++) y[p++] = i;
        for(int i = 0; i < n; i++) if(sa[i] >= k) y[p++] = sa[i] - k;
        for(int i = 0; i < m; i++) c[i] = 0;
        for(int i = 0; i < n; i++) c[x[y[i]]]++;
        for(int i = 0; i < m; i++) c[i] += c[i - 1];
        for(int i = n - 1; i >= 0; i--) sa[--c[x[y[i]]]] = y[i];
        swap(x, y);
        p = 1; x[sa[0]] = 0;
        for(int i = 1; i < n; i++)
            x[sa[i]] = y[sa[i - 1]] == y[sa[i]] && y[sa[i - 1] + k] == y[sa[i] + k] ? p - 1 : p++;
        if(p >= n) break;
        m = p;
    }
}

int rk[maxn], height[maxn];

void getHeight()
{
    for(int i = 1; i <= n; i++) rk[sa[i]] = i;
    for(int i = 0, k = 0; i < n; i++)
    {
        if(k) k--;
        int j = sa[rk[i] - 1];
        while(s[i + k] == s[j + k]) k++;
        height[rk[i]] = k;
    }
}

int dp[maxn][20];

void RMQ()
{
    for(int i = 1; i <= n; i ++) dp[i][0] = height[i];
    for(int j = 1; (1 << j) < maxn; j ++)
        for(int i = 1; i + (1 << j) - 1 <= n; i ++)
            dp[i][j] = min(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);
}

int query(int l, int r)
{
    int k = 0;
    while((1 << (k + 1)) <= r - l + 1) k ++;
    return min(dp[l][k], dp[r - (1 << k) + 1][k]);
}

int lcp(int x, int y)
{
    x = rk[x], y = rk[y];
    if(x > y) swap(x, y);
    return query(x + 1, y);
}
```

4.6 PalindromicTree

4.6.1 PalindromicTree

```
const int maxn = 2e6+6;
const int N = 26;
const int mod = 51123987;

struct Palindromic_Tree {
```

```
//    vector<pair<int, int> > next[maxn];
    int next[maxn][N];//next指针，next指针和字典树类似，指向的串为当前串两端加上同一个字符构成
    int fail[maxn]{};//fail指针，失配后跳转到fail指针指向的节点
    int cnt[maxn]{}; //表示节点i表示的本质不同的串的个数（建树时求出的不是完全的，最后count()函数跑一遍以后才是正确的)
    int num[maxn]{}; //表示以节点i表示的最长回文串的最右端点为回文串结尾的回文串个数
    int len[maxn]{};//len[i]表示节点i表示的回文串的长度（一个节点表示一个回文串)
    int S[maxn]{};//存放添加的字符
    int last{};//指向新添加一个字母后所形成的最长回文串表示的节点。
    int n{};//表示添加的字符个数。
    int p{};//表示添加的节点个数。
    //0向前加，1向后加字符
    //int last[2];
    //int lpos, rpos;

    int newnode(int l) {//新建节点
//        next[p].clear();
        for (int i = 0; i < N; ++i) next[p][i] = 0;
        cnt[p] = 0;
        num[p] = 0;
        len[p] = l;
        return p++;
    }

    void init() {//初始化
        n = last = p = 0;
        newnode(0);
        newnode(-1);
        S[n] = -1;//开头放一个字符集中没有的字符，减少特判
        fail[0] = 1;
        // lpos 为字符串最大长度
        // last[0] = last[1] = 0;
        // lpos = 100000, rpos = lpos - 1;
        // S[lpos - 1] = S[rpos + 1] = -1;
    }

    int get_fail(int x) {//和KMP一样，失配后找一个尽量最长的
        // op 0 向前， 1 向后
        // if (op == 0) while (S[lpos + len[x] + 1] != S[lpos]) x = fail[x];
        // else while(S[rpos - len[x] - 1] != S[rpos]) x = fail[x];
        while (S[n - len[x] - 1] != S[n]) x = fail[x];
        return x;
    }

//    int find(int u, int c) {
//        vector<pair<int, int> > & x = next[u];
//        int sz = x.size();
//        for(int i = 0; i < sz; ++i) {
//            if(x[i].first == c) return x[i].second;
//        }
//        return 0;
//    }

    int add(int c) {
        // 注意清空左右字符
        // if (op == 0) S[--lpos] = c, S[lpos - 1] = -1;
        // else S[++rpos] = c, S[rpos + 1] = -1;
        S[++n] = c;
        int cur = get_fail(last);//通过上一个回文串找这个回文串的匹配位置
```

```
//        int x = find(cur, c);
//        if (!x) {
        if (!next[cur][c]) {//如果这个回文串没有出现过，说明出现了一个新的本质不同的回文串
            int now = newnode(len[cur] + 2);//新建节点
//            x = now;
//            fail[now] = find(get_fail(fail[cur]), c);
//            next[cur].emplace_back(make_pair(c, now));
            fail[now] = next[get_fail(fail[cur])][c];//和AC自动机一样建立fail指针，以便失配后跳转
            next[cur][c] = now;
            num[now] = num[fail[now]] + 1;
        }
//        last = x;
        // 修改最终长度
        // if (len[last[op]] == rpos - lpos + 1) last[op ^ 1] = last[op];
        last = next[cur][c];
        cnt[last]++;
        return num[last];
    }

    void count() {
        for (int i = p - 1; i >= 0; --i) cnt[fail[i]] += cnt[i];
        //父亲累加儿子的cnt，因为如果fail[v]=u，则u一定是v的子回文串！
    }
} solve;

char s[maxn];

// 求相交回文串数量
ll a[maxn], b[maxn];
int main() {
    solve.init();
    int n;
    scanf("%d", &n);
    scanf("%s", s);
    for (int i = 0; i < n; ++i) a[i] = solve.add(s[i] - 'a');
    solve.init();
    for (int i = n - 1; i >= 0; --i) b[i] = (b[i + 1] + solve.add(s[i] - 'a')) % mod;
    ll res = (b[0] * (b[0] - 1) / 2) % mod;
    for (int i = 0; i < n; ++i) res = ((res - (a[i] * b[i + 1]) + mod) % mod) % mod;
    printf("%lld\n", res);
    return 0;
}
```

4.7 Hash

4.7.1 hash

```
// hash常用素数
// 61, 83, 113, 151, 211
// 91815541, 38734667, 68861641
// 917120411, 687840301, 386910137, 515880193
// 1222827239, 1610612741

typedef unsigned long long ull;
struct mhash {
    // 自然溢出无模数 805306457
    ull base[maxn];
    ull hash_index[maxn];
    ull seed; //31, 131
    void inithash(ull seedt = 31) {
        base[0] = 1;
        seed = seedt;
        for (int i = 1; i < maxn; ++i) base[i] = base[i - 1] * seed;
    }
    void H(char *p, int n) { // from 1 to n
        hash_index[0] = 0;
        for (int i = 1; i <= n; ++i) hash_index[i] = hash_index[i - 1] * seed + p[i] - 'a';
    }
    ull gethash(int s, int e) {
        return hash_index[e] - hash_index[s - 1] * base[e - s + 1];
    }
};

// 26个素数，解决加法hash
int prime[] = {34183,13513,152993,13591,19687,350869,111187,766091,769297,
               633469,752273,298651,617191,880421,136067,
               1408397,726899,458921,2133701,2599847,2730947,4696343,10267237,
               18941059,34078909,69208409};
```

4.7.2 doubleHash

```
namespace Hash{

    template<class __A,class __B>
    class Hash{
    private:
        static const int size=2000000;
        __B *hash; __A *O; int sz;
    public:
        Hash(int hash_size=size){ sz=hash_size;
            hash=(__B *)malloc(sizeof(__B)*sz);
            O=(__A *)malloc(sizeof(__A)*sz);
            memset(O,0xff,sizeof(__A)*sz);
        }~Hash(){free(O);free(hash);}
        __B &operator [](const __A &_O){
            int loc=_O%sz;
            while(~O[loc]&&O[loc]!=_O){
                ++loc;
                if(loc>sz)loc=0;
            }if(!~O[loc])O[loc]=_O;
            return hash[loc];
        }
        void clear(){memset(O,0xff,sizeof(__A)*sz);}
    };

    struct StringDoubleHashResult{
        int32_t *H1,*H2,c_len,len;
        StringDoubleHashResult(int32_t sz=0){
            len=sz; c_len=0; //cur_len;
            if(len<=0){
                H1=H2=0;
                return;
            }
            H1=(int32_t *)malloc(sizeof(int32_t)*sz);
            H2=(int32_t *)malloc(sizeof(int32_t)*sz);
        }
        ~StringDoubleHashResult(){}
        void clear(){free(H1);free(H2);len=0;H1=H2=0;}
        void resize(int new_len){
            int32_t *T1=(int32_t *)malloc(sizeof(int32_t)*new_len);
            int32_t *T2=(int32_t *)malloc(sizeof(int32_t)*new_len);
            for(int i=0;i<c_len;++i)T1[i]=H1[i],T2[i]=H2[i];
            free(H1);free(H2); H1=T1; H2=T2; len=new_len;
        }
        void erase(int ers_len){//erase suffix
            c_len-=ers_len;
            if(c_len<0)c_len=0;
        }
        //erase prefix not better than reculc
    };

    namespace hash_random{
        const int mod_tot=5;
        const int mod[]={1000000009,1000000007,998244353,917120411,515880193};
    };

    class StringDoubleHash{
    private:
        static const int enable_random=1;
        int32_t sz,HA1,HA2;
        long long B,C;
        int32_t *H1,*H2;
    public:
        StringDoubleHash(int32_t SZ=2e6+5,int32_t ha1=-1,int32_t ha2=-1,int32_t b=-1,int32_t c=-1){
            sz=SZ;
            if(enable_random){
                std::mt19937 rnd(time(0)+19990630);
                int z1= rnd() % hash_random::mod_tot;
                int z2= (z1 +rnd()%(hash_random::mod_tot - 1) + 1) % hash_random::mod_tot;
                if(ha1<0)ha1=hash_random::mod[z1];
                if(ha2<0)ha2=hash_random::mod[z2];
                if(b<0)b=rnd()%114514+23333;
                if(c<0)c=rnd()%1919810+23333;
            } else {
                if(ha1<0)ha1=1e9+7;
                if(ha2<0)ha2=1e9+9;
                if(b<0)b=114514;
                if(c<0)c=1919810;
            }
            HA1=ha1; HA2=ha2; B=b; C=c;
            //cerr<<HA1<<" "<<HA2<<" "<<B<<" "<<C<<endl;
            H1=(int32_t *)malloc(sizeof(int32_t)*sz);
            H2=(int32_t *)malloc(sizeof(int32_t)*sz);
            init_hash_val();
        }
        ~StringDoubleHash(){free(H1);free(H2);}
        void init_hash_val(){
            H1[0]=H2[0]=1;
            for(int32_t i=1;i<sz;++i){
                H1[i]=(H1[i-1]*B)%HA1;
                H2[i]=(H2[i-1]*B)%HA2;
            }
        }
        template <class _Tp>
        StringDoubleHashResult culc_hash(const _Tp &s,int32_t len,int32_t tot_len=-1){
            if(tot_len<0)tot_len=len;
            StringDoubleHashResult R(tot_len);
            if(len<=0)return R;
            R.H1[0]=(s[0]+C)%HA1;
            R.H2[0]=(s[0]+C)%HA2;
            for(int32_t i=1;i<len;++i){
                R.H1[i]=(R.H1[i-1]*B+s[i]+C)%HA1;
                R.H2[i]=(R.H2[i-1]*B+s[i]+C)%HA2;
            }
            R.c_len=len;
            return R;
        }
        // s is the char* first, len is the append length
        template <class _Tp>
        void append(StringDoubleHashResult &R,const _Tp &s,int32_t len){
            if(len<=0)return;
            int t_len=R.len;
            while(R.c_len+len>t_len)t_len<<=1;
            if(t_len>R.len)R.resize(t_len);
            for(int32_t i=R.c_len;i<R.c_len+len;++i){
                if(i==0){
                    R.H1[i]=(s[i-R.c_len]+C)%HA1;
                    R.H2[i]=(s[i-R.c_len]+C)%HA2;
                } else {
                    R.H1[i]=(R.H1[i-1]*B+s[i-R.c_len]+C)%HA1;
                    R.H2[i]=(R.H2[i-1]*B+s[i-R.c_len]+C)%HA2;
                }
            }
            R.c_len+=len;
        }
        void append(StringDoubleHashResult &R, char s){
            int t_len=R.len;
            while(R.c_len+1>t_len)t_len<<=1;
            if(t_len>R.len)R.resize(t_len);
            for(int32_t i=R.c_len;i<R.c_len+1;++i){
                if(i==0){
                    R.H1[i]=(s+C)%HA1;
                    R.H2[i]=(s+C)%HA2;
                } else {
                    R.H1[i]=(R.H1[i-1]*B+s+C)%HA1;
                    R.H2[i]=(R.H2[i-1]*B+s+C)%HA2;
                }
            }
            R.c_len+=1;
        }
        //return hash [l,r)
        ll gethash(const StringDoubleHashResult &R, int32_t l,int32_t r){
            if(l>r||l<0||r-->R.c_len)return -1;//fail
            ll v1=l>0?R.H1[l-1]*(long long)H1[r-l+1]%HA1:0;
            ll v2=l>0?R.H2[l-1]*(long long)H2[r-l+1]%HA2:0;
            v1=R.H1[r]-v1; v2=R.H2[r]-v2;
            if(v1<0)v1+=HA1; if(v2<0)v2+=HA2;
            return v1<<32|v2;
        }
        //merge two hashes as one(s1+s2), but need s2's length
        ll merge_hash(const long long &hs1,const long long &hs2,int lenr){
            int32_t m1=hs1>>32,m2=hs1&0xffffffffLL;
            int32_t m3=hs2>>32,m4=hs2&0xffffffffLL;
            m1=m1*(long long)H1[lenr]%HA1+m3;
            if(m1>=HA1)m1-=HA1;
```

```
            m2=m2*(long long)H2[lenr]%HA2+m4;
            if(m2>=HA2)m2-=HA2;
            return (long long)m1<<32|m2;
        }
    };
};
```

4.7.3 二维 hash

```
#define ull unsigned long long
const int maxn = 1005;
ull hs[maxn][maxn];
char a[maxn][maxn];
int n, m;
ull base1 = 131, base2 = 13331;
ull pwb1[maxn] = {1}, pwb2[maxn] = {1};

void init() {
    for (int i = 1; i < maxn; ++i) {
        pwb1[i] = pwb1[i - 1] * base1;
        pwb2[i] = pwb2[i - 1] * base2;
    }
}

void Hash() {
    for(int i=1;i<=n;i++)
        for(int j=1;j<=m;j++)
            hs[i][j]=hs[i][j-1]*base1+a[i][j] - 'a';
    for(int i=1;i<=n;i++)
        for(int j=1;j<=m;j++)
            hs[i][j]+=hs[i-1][j]*base2;
}

// 右下角(i,j)，行列长度n,m
ull getHs(int i, int j, int lenn, int lenm) {
    return hs[i][j] - hs[i - lenn][j] * pwb2[lenn] -
            hs[i][j - lenm] * pwb1[lenm] +
            hs[i - lenn][j - lenm] * pwb2[lenn] * pwb1[lenm];
}
```

4.7.4 树 hash 同构

```
// n=1e5的话base开2e6+9，可以输出看到top不比n小即可
const int base = 2e6+9;
// vis大小要开到素数大小， turn表示当前树的编号，p是预处理数组
int vis[base + 1], top, turn, p[base + 1];
// 程序开头调用一次
void init() {
    top = 0;
    for (int i = 2; i <= base; ++i) {
        if (!vis[i]) {
            p[++top] = i;
        }
        for (int j = 1; j <= top && i * p[j] <= base; ++j) {
            vis[i * p[j]] = 1;
            if (i % p[j] == 0) break;
        }
    }
    assert(top >= maxn);
}

vector<int> edge[maxn];
// h[x]表示x这棵子树的hash值， g[x]表示以x为根的hash值
int h[maxn], g[maxn], sz[maxn];

struct TreeHash {
    int n;
    // 如果树比较多，在类内部开edge可能会炸内存，可以改到外面做前向星
    // 除了hs是答案其他都可以改到外部，只有edge需要清零
    // vector<int> edge[maxn];
    // int h[maxn], g[maxn], sz[maxn];
    vector<int> hs;

    void init(int n_ = 0) {
        n = n_;
        hs.clear();
    }

    void dfs1(int u, int pre) {
        sz[u] = 1;
        h[u] = 1;
        for (auto v : edge[u]) {
            if (v == pre) continue;
            dfs1(v, u);
            h[u] = (h[u] + 1ll * h[v] * p[sz[v]] % mod) % mod;
            sz[u] += sz[v];
        }
    }

    void dfs2(int u, int pre, int V, int needres = 1) {
        g[u] = (h[u] + 1ll * V * p[n - sz[u]] % mod) % mod;
        if (needres) hs.push_back(g[u]);
        for (auto v : edge[u]) {
            if (v == pre) continue;
            dfs2(v, u, (g[u] - 1ll * h[v] * p[sz[v]] % mod + mod) % mod);
        }
    }

    void work(int needres = 1) {
        // 无根树选一个不存在的点当pre即可，当多棵无根树判重时需要sort
        dfs1(1, 0);
        dfs2(1, 0, 0, needres);
        sort(hs.begin(), hs.end());
    }
};

// 获取删掉某叶子节点后以与该叶子节点相邻点开头的hash值
// int res = (hs[edge[i][0]] - 2 + mod) % mod;
```

4.8 Suffix Automation

4.8.1 SAM

```
const int maxn = 2e4 + 10;

struct SuffixAutomation
{
```

```
int last, cnt;
int ch[maxn << 1][26], fa[maxn << 1], len[maxn << 1], pos[maxn << 1];
int sz[maxn << 1], a[maxn << 1], c[maxn << 1];

void init()
{
    last = cnt = 1;
    memset(ch[1], 0, sizeof ch[1]);
    fa[1] = len[1] = 0;
}

int inline newnode(int idx)
{
    ++cnt;
    memset(ch[cnt], 0, sizeof ch[cnt]);
    fa[cnt] = len[cnt] = 0;
    pos[cnt] = idx;
    return cnt;
}

void ins(int c)
{
    int p = last , np = newnode(pos[last] + 1);
    last = np, len[np] = len[p] + 1;
    for(; p && !ch[p][c]; p = fa[p]) ch[p][c] = np;
    if(!p) fa[np] = 1;
    else
    {
        int q = ch[p][c];
        if(len[p] + 1 == len[q]) fa[np] = q;
        else
        {
            int nq = newnode(pos[p] + 1);
            len[nq] = len[p] + 1;
            memcpy(ch[nq], ch[q], sizeof ch[q]);
            fa[nq] = fa[q], fa[q] = fa[np] = nq;
            for(; ch[p][c] == q; p = fa[p]) ch[p][c] = nq;
        }
    }
    sz[np] = 1;
}

int solve(int n)
{
    /*求两个串的LCS:
        对一个字符串建立SAM，记录一个当前匹配的长度Len和当前节点v，枚举另一个字符串的每个字符;
        如果p有字符v的转移边出边，则使Len加一，并使p转移到出边指向的节点上;
        否则不断向父节点上跳，直到当前节点有字符p的转移出边，或者跳到根节点;
    */
    int p = 1, ans = 0, now_len = 0;
    for(int i = 0; s2[i]; i ++)
    {
        if(ch[p][s2[i] - 'a']) p = ch[p][s2[i] - 'a'], now_len ++;
        else
        {
            for(;p && !ch[p][s2[i] -'a'] ; p = fa[p]) ;
            if(p == 0) now_len = 0, p = 1;
            else now_len = len[p] + 1, p = ch[p][s2[i] - 'a'];
        }
```

```
            ans = max(now_len, ans);
        }
    }

    void Toposort()
    {
        long long ans = 0;
        for(int i = 1; i <= cnt; i ++) c[len[i]] ++;
        for(int i = 1; i <= cnt; i ++) c[i] += c[i - 1];
        for(int i = 1; i <= cnt; i ++) a[c[len[i]] --] = i;
        for(int i = cnt; i; i --) sz[fa[a[i]]] += sz[a[i]];
    }
}sam;
```

4.9 Others

4.9.1 最小表示法

```
// 0起始
int Gao(char a[], int len) {
    int i = 0, j = 1, k = 0;
    while (i < len && j < len && k < len) {
        int cmp = a[(j + k) % len] - a[(i + k) % len];
        if (cmp == 0) k++;
        else {
            if (cmp > 0) j += k + 1;
            else i += k + 1;
            if (i == j) j ++;
            k = 0;
        }
    }
    return min(i, j);
}
```

5 dp

5.1 BitDP

5.1.1 数位 dp 计和

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;
const int mod = 998244353;
pair<ll, ll> dp[20][1<<10];
bool vis[20][1<<10];
int k;
int t[20];
ll base[20];

pair<ll, ll> dfs(int pos, int state, bool limit, bool lead) {
    if (pos == -1) return __builtin_popcount(state) <= k ? make_pair(1, 0) : make_pair(0, 0);
    if (!limit && !lead && vis[pos][state]) return dp[pos][state];
    int up = limit ? t[pos] : 9;
    pair<ll, ll> res = {0, 0};
    for (int i = 0; i <= up; ++i) {
        int n_s = state;
        if (lead && i == 0) n_s = 0;
        else n_s = state | (1 << i);
        auto tmp = dfs(pos - 1, n_s, limit && i == t[pos], lead && i == 0);
        ll pre = 1ll * i * base[pos] % mod;
        (res.first += tmp.first) %= mod;
        (res.second += tmp.second + pre * tmp.first) %= mod;
    }
    if (!limit && !lead) dp[pos][state] = res, vis[pos][state] = 1;
    return res;
}

ll solve(ll x) {
    int pos = 0;
    do {
        t[pos ++] = x % 10;
    } while (x /= 10);
    return dfs(pos - 1, 0, true, true).second;
}

int main(int argc,char *argv[])
{
    base[0] = 1;
    for (int i = 1; i < 20; ++i) base[i] = base[i - 1] * 10;
    ll l, r;
    scanf("%lld%lld%d", &l, &r, &k);
    printf("%lld\n", (solve(r) - solve(l - 1) + mod) % mod);
    return 0;
}
```

5.1.2 两个数数位 dp

```
// 二进制数位dp, 求a $\in$ 1 ~ x 和 b $\in$ 1~y, 满足 $a & b > c || a ^ b < c$的对数
ll dp[maxn][2][2][2][2];
int a[maxn], b[maxn], c[maxn];


```

```cpp
void cal(int *xt, ll x) {
    int has = 0;
    while (x) {
        xt[has++] = x % 2;
        x /= 2;
    }
}

ll dfs(int pos, int o1, int o2, int lim1, int lim2) {
    if (pos < 0) return 1;
    ll &t = dp[pos][o1][o2][lim1][lim2];
    if (t != -1) return t;
    int up1 = o1 ? a[pos] : 1;
    int up2 = o2 ? b[pos] : 1;
    ll res = 0;
    for (int i = 0; i <= up1; ++i) {
        for (int j = 0; j <= up2; ++j) {
            int t1 = i & j;
            int t2 = i ^ j;
            if (lim1 && t1 > c[pos]) continue;
            if (lim2 && t2 < c[pos]) continue;
            res += dfs(pos - 1, o1 && i == up1, o2 && j == up2, lim1 && t1 == c[pos], lim2 && t2 == c[pos]);
        }
    }
    return t = res;
}

ll solve(ll x, ll y, ll z) {
    memset(dp, -1ll, sizeof dp);
    for (int i = 0; i < 33; ++i) a[i] = b[i] = c[i] = 0;
    cal(a, x);
    cal(b, y);
    cal(c, z);
    return dfs(32, 1, 1, 1, 1);
}

int main(int argc, char *argv[]) {
    int T;
    scanf("%d", &T);
    ll x, y, z;
    for (int kase = 1; kase <= T; ++kase) {
        scanf("%lld%lld%lld", &x, &y, &z);
        ll res = solve(x, y, z);
        res -= max(0ll, y - z + 1);
        res -= max(0ll, x - z + 1);
        printf("%lld\n", x * y - res);
    }
    return 0;
}
```

5.2 Subsequence

5.2.1 MaxSum

```cpp
// 传入序列a和长度n，返回最大子序列和
int MaxSeqSum(int a[], int n)
{
    int rt = 0, cur = 0;
    for (int i = 0; i < n; i++)
        cur += a[i], rt = max(cur, rt), cur = max(0, cur);
    return rt;
}
```

5.2.2 LIS

```
// 简单写法(下标从0开始,只返回长度)
int dp[N];
int LIS(int a[], int n)
{
    memset(dp, 0x3f, sizeof(dp));
    for (int i = 0; i < n; i++) *lower_bound(dp, dp + n, a[i]) = a[i];
    return lower_bound(dp, dp + n, INF) - dp;
}

// 小常数nlogn求序列用树状数组维护dp即可
// dp[i] = max(dp[j]) + 1 (j < i && a[j] < a[i])
```

5.2.3 LongestCommonIncrease

```
// 序列下标从1开始
int LCIS(int a[], int b[], int n, int m)
{
    memset(dp, 0, sizeof(dp));
    for (int i = 1; i <= n; i++)
    {
        int ma = 0;
        for (int j = 1; j <= m; j++)
        {
            dp[i][j] = dp[i - 1][j];
            if (a[i] > b[j]) ma = max(ma, dp[i - 1][j]);
            if (a[i] == b[j]) dp[i][j] = ma + 1;
        }
    }
    return *max_element(dp[n] + 1, dp[n] + 1 + m);
}
```

5.2.4 LCS

```
#include <stdio.h>
#include <string.h>

#define M 30005
#define SIZE 128
#define WORDMAX 3200
#define BIT 32

char s1[M], s2[M];
int nword;
unsigned int str[SIZE][WORDMAX];
unsigned int tmp1[WORDMAX], tmp2[WORDMAX];

void pre(int len)
{
    int i, j;
    memset(str, 0, sizeof(str));
    for(i = 0; i < len; i ++)
        str[s1[i]][i / BIT] |= 1 << (i % BIT);
}

void cal(unsigned int *a, unsigned int *b, char ch)
{
    int i, bottom = 1, top;
    unsigned int x, y;
    for(i = 0; i < nword; i ++)
    {
        y = a[i];
        x = y | str[ch][i];
        top = (y >> (BIT - 1)) & 1;
        y = (y << 1) | bottom;
        if(x < y) top = 1;
        b[i] = x & ((x - y) ^ x);
        bottom = top;
    }
}

int bitcnt(unsigned int *a)
{
    int i, j, res = 0, t;
    unsigned int b[5] = {0x55555555, 0x33333333, 0x0f0f0f0f, 0x00ff00ff, 0x0000ffff}, x;
    for(i = 0; i < nword; i ++)
    {
        x = a[i];
        t = 1;
        for(j = 0; j < 5; j ++, t <<= 1)
            x = (x & b[j]) + ((x >> t) & b[j]);
        res += x;
    }
    return res;
}

void process()
{
    int i, j, len1, len2;
    unsigned int *a, *b, *t;
    len1 = strlen(s1);
    len2 = strlen(s2);
    nword = (len1 + BIT - 1) / BIT;
    pre(len1);
    memset(tmp1, 0, sizeof(tmp1));
    a = &tmp1[0];
    b = &tmp2[0];
    for(i = 0; i < len2; i ++)
    {
        cal(a, b, s2[i]);
        t = a; a = b; b = t;
    }
    printf("%d\n", bitcnt(a));
}

int main()
{
    while(scanf("%s%s", s1, s2) != EOF)
        process();
    return 0;
```

```
}
```

5.3 Others

问题 设 $f(i) = \min(y[k] - s[i] \times x[k]), k \in [1, i-1]$, 现在要求出所有 $f(i), i \in [1, n]$

考虑两个决策 j 和 k，如果 j 比 k 优，则

$$y[j] - s[i] \times x[j] < y[k] - s[i] \times x[k]$$

化简得:

$$\frac{y_j - y_k}{x_j - x_k} < s_i$$

不等式左边是个斜率，我们把它设为 $\operatorname{slope}(j, k)$

我们可以维护一个单调递增的队列，为什么呢?

因为如果 $\operatorname{slope}(q[i-1], q[i]) > \operatorname{slope}(q[i], q[i+1])$，那么当前者成立时，后者必定成立。即 $q[i]$ 决策优于 $q[i-1]$ 决策时，$q[i+1]$ 必然优于 $q[i]$，因此 $q[i]$ 就没有存在的必要了。所以我们要维护递增的队列。

那么每次的决策点 i，都要满足

$$\begin{cases} \operatorname{slope}(q[i-1], q[i]) < s[i] \\ \operatorname{slope}(q[i], q[i+1]) \geq s[i] \end{cases}$$

一般情况去二分这个 i 即可。

如果 $s[i]$ 是单调不降的，那么对于决策 j 和 $k(j < k)$ 来说，如果决策 k 优于决策 j，那么对于 $i \in [k+1, n]$，都存在决策 k 优于决策 j，因此决策 j 就可以舍弃了。这样的话我们可以用单调队列进行优化，可以少个 log。

单调队列滑动窗口最大值

```
// k为滑动窗口的大小，数列下标从1开始，d为序列长度+1
deque<int> q;
for (int i = 0, j = 0; i + k <= d; i++)
{
    while (j < i + k)
    {
        while (!q.empty() && a[q.back()] < a[j]) q.pop_back();
        q.push_back(j++);
    }
    while (q.front() < i) q.pop_front();
    // a[q.front()]为当前滑动窗口的最大值
}
```

5.3.1 矩阵快速幂

```
struct Matrix {
    int sz;
    // int n, m;
    ll a[maxn][maxn];
    Matrix(int sz_ = 0):sz(sz_) {
        memset(a, 0, sizeof a);
    }
    void pr() {
        printf("*\n");
        for(int i = 0; i < sz; ++i) {
            for (int j = 0; j < sz; ++j) {
                printf("%lld ", a[i][j]);
            }
            printf("\n");
        }
    }
    void tr() {
        for (int i = 0; i < sz; ++i) {
            for (int j = i + 1; j < sz; ++j) {
                swap(a[i][j], a[j][i]);
            }
        }
    }
}res, t1;

void init() {
    ;
}

Matrix mul(Matrix a, Matrix b)
{
    Matrix res(a.sz);
    // if (a.m != b.n) return res;
    for(int i = 0; i < res.sz; i++) // a.n
        for(int j = 0; j < res.sz; j++) // b.m
            for(int k = 0; k < res.sz; k++) // a.m, b.n
                (res.a[i][j] +=a.a[i][k] * b.a[k][j] % mod) %= mod;
    return res;
}

Matrix pow(ll n)
{
    init();
    //for(int i = 0; i < cur; i++) res.a[i][i] = 1;
    while(n > 0) {
        if(n & 1) res = mul(res, t1);
        t1 = mul(t1, t1);
        n >>= 1;
    }
    return res;
}
```

6 Others

6.1 mint 类

```cpp
const int mod = 998244353;

struct mint {
  int n;
  mint(int n_ = 0) : n(n_) {}
};

mint operator+(mint a, mint b) { return (a.n += b.n) >= mod ? a.n - mod : a.n; }
mint operator-(mint a, mint b) { return (a.n -= b.n) < 0 ? a.n + mod : a.n; }
mint operator*(mint a, mint b) { return 1LL * a.n * b.n % mod; }
mint &operator+=(mint &a, mint b) { return a = a + b; }
mint &operator-=(mint &a, mint b) { return a = a - b; }
mint &operator*=(mint &a, mint b) { return a = a * b; }
ostream &operator<<(ostream &o, mint a) { return o << a.n; }
```

6.2 不重叠区间贪心

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

const int maxn = 5e5+5;
pair<int, int> a[maxn];
int main() {
    int n;
    cin >> n;
    for (int i = 1; i <= n; ++i) {
        cin >> a[i].second >> a[i].first;
    }
    sort(a + 1, a + 1 + n);
    int res = 1;
    int tmp = a[1].first;
//    printf("%d %d\n", a[1].second, a[1].first);
    for (int i = 2; i <= n; ++i) {
        if (a[i].second > tmp) {
            res ++;
//            printf("%d %d\n", a[i].second, a[i].first);
            tmp = a[i].first;
        }
    }
    printf("%d\n", res);
    return 0;
}
```

6.3 BigInt 类

```cpp
const double PI = acos(-1.0);
struct Complex{
    double x,y;
    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);
    }
};
void change(Complex y[],int len){
    int i,j,k;
    for(int i = 1,j = len/2;i<len-1;i++){
        if(i < j)    swap(y[i],y[j]);
        k = len/2;
        while(j >= k){
            j = j - k;
            k = k/2;
        }
        if(j < k)    j+=k;
    }
}
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;
        }
    }
}
class BigInt
{
#define Value(x, nega) ((nega) ? -(x) : (x))
#define At(vec, index) ((index) < vec.size() ? vec[(index)] : 0)
    static int absComp(const BigInt &lhs, const BigInt &rhs)
    {
        if (lhs.size() != rhs.size())
            return lhs.size() < rhs.size() ? -1 : 1;
        for (int i = lhs.size() - 1; i >= 0; --i)
            if (lhs[i] != rhs[i])
                return lhs[i] < rhs[i] ? -1 : 1;
        return 0;
    }
    using Long = long long;
    const static int Exp = 9;
    const static Long Mod = 1000000000;
    mutable std::vector<Long> val;
    mutable bool nega = false;
    void trim() const
    {
        while (val.size() && val.back() == 0)
            val.pop_back();
        if (val.empty())
            nega = false;
    }
    int size() const { return val.size(); }
    Long &operator[](int index) const { return val[index]; }
    Long &back() const { return val.back(); }
    BigInt(int size, bool nega) : val(size), nega(nega) {}
    BigInt(const std::vector<Long> &val, bool nega) : val(val), nega(nega) {}

public:
    friend std::ostream &operator<<(std::ostream &os, const BigInt &n)
    {
        if (n.size())
        {
            if (n.nega)
                putchar('-');
            for (int i = n.size() - 1; i >= 0; --i)
            {
                if (i == n.size() - 1)
                    printf("%lld", n[i]);
                else
                    printf("%0*lld", n.Exp, n[i]);
            }
        }
        else
            putchar('0');
        return os;
    }
    friend BigInt operator+(const BigInt &lhs, const BigInt &rhs)
    {
        BigInt ret(lhs);
        return ret += rhs;
    }
    friend BigInt operator-(const BigInt &lhs, const BigInt &rhs)
    {
        BigInt ret(lhs);
        return ret -= rhs;
    }
    BigInt(Long x = 0)
    {
        if (x < 0)
            x = -x, nega = true;
        while (x >= Mod)
            val.push_back(x % Mod), x /= Mod;
        if (x)
            val.push_back(x);
    }
    BigInt(const char *s)
    {
        int bound = 0, pos;
        if (s[0] == '-')
            nega = true, bound = 1;
        Long cur = 0, pow = 1;
        for (pos = strlen(s) - 1; pos >= Exp + bound - 1; pos -= Exp, val.push_back(cur), cur = 0, pow = 1)
            for (int i = pos; i > pos - Exp; --i)
                cur += (s[i] - '0') * pow, pow *= 10;
        for (cur = 0, pow = 1; pos >= bound; --pos)
            cur += (s[pos] - '0') * pow, pow *= 10;
        if (cur)
            val.push_back(cur);
    }
    BigInt &operator=(const char *s){
        BigInt n(s);
        *this = n;
        return n;
    }
    BigInt &operator=(const Long x){
        BigInt n(x);
        *this = n;
        return n;
    }
    friend std::istream &operator>>(std::istream &is, BigInt &n){
        string s;
        is >> s;
        n=(char*)s.data();
        return is;
    }
    BigInt &operator+=(const BigInt &rhs)
    {
        const int cap = std::max(size(), rhs.size()) + 1;
        val.resize(cap);
        int carry = 0;
        for (int i = 0; i < cap - 1; ++i)
        {
            val[i] = Value(val[i], nega) + Value(At(rhs, i), rhs.nega) + carry, carry = 0;
            if (val[i] >= Mod)
                val[i] -= Mod, carry = 1;
            else if (val[i] < 0)
                val[i] += Mod, carry = -1;
        }
        if ((val.back() = carry) == -1) //assert(val.back() == 1 or 0 or -1)
        {
            nega = true, val.pop_back();
            bool tailZero = true;
            for (int i = 0; i < cap - 1; ++i)
            {
                if (tailZero && val[i])
                    val[i] = Mod - val[i], tailZero = false;
                else
                    val[i] = Mod - 1 - val[i];
            }
        }
        trim();
        return *this;
    }
    friend BigInt operator-(const BigInt &rhs)
    {
        BigInt ret(rhs);
        ret.nega ^= 1;
        return ret;
    }
    BigInt &operator-=(const BigInt &rhs)
    {
        rhs.nega ^= 1;
        *this += rhs;
        rhs.nega ^= 1;
        return *this;
    }
    friend BigInt operator*(const BigInt &lhs, const BigInt &rhs)
    {
        int len=1;
        BigInt ll=lhs,rr=rhs;
        ll.nega = lhs.nega ^ rhs.nega;
        while(len<2*lhs.size()||len<2*rhs.size())len<<=1;
        ll.val.resize(len),rr.val.resize(len);
        Complex x1[len],x2[len];
        for(int i=0;i<len;i++){
            Complex nx(ll[i],0.0),ny(rr[i],0.0);
            x1[i]=nx;
            x2[i]=ny;
        }
        fft(x1,len,1);
        fft(x2,len,1);
        for(int i = 0 ; i < len; i++)
            x1[i] = x1[i] * x2[i];
        fft( x1 , len , -1 );
        for(int i = 0 ; i < len; i++)
            ll[i] = int( x1[i].x + 0.5 );
        for(int i = 0 ; i < len; i++){
            ll[i+1]+=ll[i]/Mod;
            ll[i]%=Mod;
        }
        ll.trim();
        return ll;
    }
    friend BigInt operator*(const BigInt &lhs, const Long &x){
        BigInt ret=lhs;
        bool negat = ( x < 0 );
        Long xx = (negat) ? -x : x;
        ret.nega ^= negat;
        ret.val.push_back(0);
        ret.val.push_back(0);
        for(int i = 0; i < ret.size(); i++)
            ret[i]*=xx;
        for(int i = 0; i < ret.size(); i++){
            ret[i+1]+=ret[i]/Mod;
            ret[i] %= Mod;
        }
        ret.trim();
        return ret;
    }
    BigInt &operator*=(const BigInt &rhs) { return *this = *this * rhs; }
    BigInt &operator*=(const Long &x) { return *this = *this * x; }
    friend BigInt operator/(const BigInt &lhs, const BigInt &rhs)
    {
        static std::vector<BigInt> powTwo{BigInt(1)};
        static std::vector<BigInt> estimate;
        estimate.clear();
        if (absComp(lhs, rhs) < 0)
            return BigInt();
        BigInt cur = rhs;
        int cmp;
        while ((cmp = absComp(cur, lhs)) <= 0)
        {
            estimate.push_back(cur), cur += cur;
            if (estimate.size() >= powTwo.size())
                powTwo.push_back(powTwo.back() + powTwo.back());
        }
        if (cmp == 0)
            return BigInt(powTwo.back().val, lhs.nega ^ rhs.nega);
        BigInt ret = powTwo[estimate.size() - 1];
        cur = estimate[estimate.size() - 1];
        for (int i = estimate.size() - 1; i >= 0 && cmp != 0; --i)
            if ((cmp = absComp(cur + estimate[i], lhs)) <= 0)
                cur += estimate[i], ret += powTwo[i];
        ret.nega = lhs.nega ^ rhs.nega;
        return ret;
    }
    friend BigInt operator/(const BigInt &num,const Long &x){
        bool negat = ( x < 0 );
        Long xx = (negat) ? -x : x;
        BigInt ret;
        Long k = 0;
        ret.val.resize( num.size() );
        ret.nega = (num.nega ^ negat);
        for(int i = num.size() - 1 ;i >= 0; i--){
            ret[i] = ( k * Mod + num[i]) / xx;
            k = ( k * Mod + num[i]) % xx;
        }
        ret.trim();
        return ret;
    }
    bool operator==(const BigInt &rhs) const
    {
        return nega == rhs.nega && val == rhs.val;
    }
    bool operator!=(const BigInt &rhs) const { return nega != rhs.nega || val != rhs.val; }
    bool operator>=(const BigInt &rhs) const { return !(*this < rhs); }
    bool operator>(const BigInt &rhs) const { return !(*this <= rhs); }
    bool operator<=(const BigInt &rhs) const
    {
        if (nega && !rhs.nega)
            return true;
        if (!nega && rhs.nega)
            return false;
        int cmp = absComp(*this, rhs);
        return nega ? cmp >= 0 : cmp <= 0;
    }
    bool operator<(const BigInt &rhs) const
    {
        if (nega && !rhs.nega)
            return true;
        if (!nega && rhs.nega)
            return false;
        return (absComp(*this, rhs) < 0) ^ nega;
    }
    void swap(const BigInt &rhs) const
    {
        std::swap(val, rhs.val);
        std::swap(nega, rhs.nega);
    }
};
BigInt ba,bb;
int main(){
    cin>>ba>>bb;
    cout << ba + bb << '\n';//和
    cout << ba - bb << '\n';//差
    cout << ba * bb << '\n';//积
    BigInt d;
    cout << (d = ba / bb) << '\n';//商
    cout << ba - d * bb << '\n';//余
    return 0;
}
```

6.4 date

```
string dayOfWeek[] = {"Mo", "Tu", "We", "Th", "Fr", "Sa", "Su"};
// converts Gregorian date to integer (Julian day number)
int DateToInt (int m, int d, int y){
    return
        1461 * (y + 4800 + (m - 14) / 12) / 4 +
        367 * (m - 2 - (m - 14) / 12 * 12) / 12 -
        3 * ((y + 4900 + (m - 14) / 12) / 100) / 4 +
        d - 32075;
}

// converts integer (Julian day number) to Gregorian date: month/day/year
void IntToDate (int jd, int &m, int &d, int &y){
    int x, n, i, j;
    x = jd + 68569;
    n = 4 * x / 146097;
    x -= (146097 * n + 3) / 4;
    i = (4000 * (x + 1)) / 1461001;
    x -= 1461 * i / 4 - 31;
    j = 80 * x / 2447;
    d = x - 2447 * j / 80;
    x = j / 11;
    m = j + 2 - 12 * x;
    y = 100 * (n - 49) + i + x;
}
// converts integer (Julian day number) to day of week
string IntToDay (int jd){
    return dayOfWeek[jd % 7];
}
```

6.5 Frac 类

```
struct Frac {
    ll a, b;
    void getJian() {
        ll gcd = abs(__gcd(a, b));
        a /= gcd;
        b /= gcd;
        if (b < 0) {
            a = -a;
            b = -b;
        }
    }
    Frac(ll a_ = 1, ll b_ = 1) {
        a = a_;
        b = b_;
        getJian();
    }
    Frac add(const Frac& oth) {
        ll bt = b * oth.b;
        ll at = a * oth.b + oth.a * b;
        return Frac(at, bt);
    }
    Frac multi(const Frac& oth) {
        a *= oth.a;
        b *= oth.b;
        getJian();
        return *this;
    }
    bool operator < (const Frac& oth) const {
        return a * oth.b < b * oth.a;
    }
    bool operator == (const Frac& oth) const {
        return a * oth.b == b * oth.a;
    }
    bool operator <= (const Frac& oth) const {
        return a * oth.b <= b * oth.a;
    }
};
```

6.6 模拟退火 (最小圆覆盖)

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

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

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

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

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

6.7 string 类

```
const int maxn = 1005;
struct String{
    int nex[maxn];
    char x[maxn];
    int len;
    int getLength() {
        return len;
    }
    void getNext() {
        int n = len, i = 0, j = -1;
        nex[0] = -1;
        while (i < n) {
            if (j == -1 || x[i] == x[j]) nex[++i] = ++j;
            else j = -1;
        }
    }
    void input() {
        scanf("%s", x);
        len = strlen(x);
    }
    void inputAndCal() {
        scanf("%s", x);
        len = strlen(x);
        getNext();
    }
    void show() {
        printf("%s\n", x);
    }
    bool operator < (const String&oth) const {
        return strcmp(x, oth.x) < 0;
    }
    char operator [] (const int a) const {
        return x[a];
    }
    bool substring(String b) {//b is the substring of a
        int m = len, n = b.getLength();
        int i = 0, j = 0;
        while (i < m && j < n) {
            if (j == -1 || x[i] == b[j]) ++i, ++j;
            else j = b.nex[j];
            if (j == n) return true;
        }
        return false;
    }
};
```

6.8 前缀异或和

```
ll xor_sum(ll n) {
    ll t=n&3;
    if (t&1) return t/2ull^1;
    return t/2ull^n;
}
```

6.9 约瑟夫环第 k 个

```
ll kth(ll n, ll m, ll k) { // n个人，m间隔，第k个出列的人
    if (m == 1) return k;
    ll res = (m - 1) % (n - k + 1);
    for (ll i = n - k + 2, stp = 0; i <= n; i += stp, res += stp * m) {
        if (res + m >= i) {
            res = (res + m) % i;
            i++;
            stp = 0;
        } else {
            stp = (i - res - 2) / (m - 1);
            if (i + stp > n) {
                res += (n - (i - 1)) * m;
                break;
            }
        }
    }
    return res + 1;
}

ll dieInXturn(int n, int k, int x) { // n个人，m间隔，第k个人出列时间
    ll tmp = 0;
    while (n) {
        x = (x + n) % n;
        if (k > n)x += (k - x - 1 + n - 1) / n * n;
        if ((x + 1) % k == 0) {
            tmp += (x + 1) / k;
            break;
        } else {
            if (k > n) {
                tmp += x / k;
                ll ttmp = x;
                x = x - (x / n + 1) * (x / k) + (x + n) / n * n - k;
                n -= ttmp / k;

            } else {
                tmp += n / k;
                x = x - x / k;
                x += n - n / k * k;
                n -= n / k;
            }
        }
    }
    return tmp;
}
```

6.10 二分

```
// a为二分数组，x为需要查找的数，返回最左端和最右端
pair<int, int> F(vector<int> a, int x) {
    int l = 0, r = a.size() - 1;
    int lres = -1;
    while (l <= r) {
        int mid = l + r >> 1;
        int tt = a[mid];
        if (tt >= x) {
            r = mid - 1;
        } else if (tt < x) {
            l = mid + 1;
        }
    }
    if (l >= a.size() || a[l] != x) return make_pair(-1, -1);
    lres = l;
    l = 0, r = a.size() - 1;
    while (l <= r) {
        int mid = l + r >> 1;
        int tt = a[mid];
        if (tt > x) {
            r = mid - 1;
        } else if (tt <= x) {
            l = mid + 1;
        }
    }
    return make_pair(lres, r);
}
```

6.11 猛男 IO 挂

```
const int LEN = 100000;
struct fastio {
    int it, len;
    char s[LEN + 5];
    fastio() {
        it = len = 0;
    }
    char get() {
        if (it < len) return s[it++];
        it = 0, len = fread(s, 1, LEN, stdin);
        return len ? s[it++] : EOF;
    }
    bool notend() {
        char c;
        for (c = get(); c == ' ' || c == '\n'; c = get());
        if (it) it--;
        return c != EOF;
    }
    void put(char c) {
        if (it == LEN) fwrite(s, 1, LEN, stdout), it = 0;
        s[it++] = c;
    }
    void flush() {
        fwrite(s, 1, it, stdout);
    }
} buff, bufo;
inline int getint() {
    char c;
    int res = 0, sig = 1;
    for (c = buff.get(); c < '0' || c > '9'; c = buff.get()) if (c == '-') sig = -1;
    for (; c >= '0' && c <= '9'; c = buff.get()) res = res * 10 + (c - '0');
    return sig * res;
}
inline ll getll() {
    char c;
    ll res = 0, sig = 1;
    for (c = buff.get(); c < '0' || c > '9'; c = buff.get()) if (c == '-') sig = -1;
    for (; c >= '0' && c <= '9'; c = buff.get()) res = res * 10 + (c - '0');
    return sig * res;
}
inline void putint(int x, char suf) {
    if (!x) bufo.put('0');
    else {
        if (x < 0) bufo.put('-'), x = -x;
        int k = 0;
        char s[15];
        while (x) {
            s[++k] = x % 10 + '0';
            x /= 10;
        }
        for (; k; k--) bufo.put(s[k]);
    }
    bufo.put(suf);
}
inline void putll(ll x, char suf) {
    if (!x) bufo.put('0');
    else {
        if (x < 0) bufo.put('-'), x = -x;
        int k = 0;
        char s[25];
        while (x) {
            s[++k] = x % 10 + '0';
            x /= 10;
        }
        for (; k; k--) bufo.put(s[k]);
    }
    bufo.put(suf);
}
inline char get_char() {
    char c;
    for (c = buff.get(); c == ' ' || c == '\n'; c = buff.get());
    return c;
}
```

6.12 贪心结论

```
// n个区间，挪到使得某个点被所有区间覆盖需要的最少步数时，选择的点是所有区间端点的中位数（mid~mid+1答案
    都是一样的）


// 不重叠区间贪心
pair<int, int> a[maxn];
int main() {
    int n;
    cin >> n;
    for (int i = 1; i <= n; ++i) {
        cin >> a[i].second >> a[i].first;
    }
    sort(a + 1, a + 1 + n);
    int res = 1;
    int tmp = a[1].first;
//    printf("%d %d\n", a[1].second, a[1].first);
    for (int i = 2; i <= n; ++i) {
        if (a[i].second > tmp) {
            res ++;
//            printf("%d %d\n", a[i].second, a[i].first);
            tmp = a[i].first;
        }
    }
    printf("%d\n", res);
    return 0;
}
```

6.13 builtin

```
__builtin_popcount(unsigned int n) // 1的个数
__builtin_parity(unsigned int n) // 奇数个1返回1，偶数个返回0
__builtin_ctz(unsigned int n) // 判断n的二进制末尾后面0的个数
__builtin_clz(unsigned int n) //返回前导0的个数
```

6.14 n 以内 k 因子的个数

```
// 返回1~n中k因子的个数
ll dig(ll n, ll k) {
    ll res = 0;
    while (n > 0) {
        res += n / k;
        n /= k;
    }
    return res;
}
```