icpc 算法模板

Catch-22

2022 年 8 月 29 日

目录

1 数学

1.1 求逆元

注意考虑 x 是 mod 倍数的情况

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

ll inv(ll x) { return qpow(x, mod - 2); }

const int N = 1e6 + 10;
// 线性递推求逆元 [1, n] 的所有数关于 p 的逆元
int inv[N];
void init_inv () {
    int n, p;
    cin >> n >> p;
    inv[0] = 0, inv[1] = 1;
    for (int i = 2; i <= n; i++)
        inv[i] = (ll)(p - p / i) * inv[p % i] % p;//为了保证大于零加了个 p

    return 0;
}
```

1.2 扩展欧几里德算法

bezout 定理：设 a, b 为正整数，则关于 x, y 的方程 $ax + by = c$ 有整数解当且仅当 c 是 $gcd(a, b)$ 的倍数。
返回结果：$ax + by = gcd(a, b)$ 的一组解 (x, y)
时间复杂度：$\mathcal{O}(nlogn)$

```
//拓欧解线性同余方程 a*x=b(mod m)
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
int a, b, m, n;

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

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

int main() {
    int x, y;
    cin >> n;
    while(n -- ) {
        cin >> a >> b >> m;
        int d = exgcd(a, m, x, y);  // d = gcd(a, m)s
        if(b % d != 0) puts("impossible");  //bezout 定理: 有解的条件, gcd(a, m) | b
        else printf("%lld\n", (ll)x * (b/d) % m);
    }
    return 0;
}
```

1.3 BSGS

find smallest non-negative x s.t. $a^x = b \bmod p$, or -1(assume $0^0 = 1$)

```
// find smallest non-negative x s.t. a^x = b mod p, or -1(assume 0^0 = 1)
int babyStepGiantStep(int a, int b, int p) {
    a %= p; b %= p;
    if (p == 1 || b == 1) return 0;
    int cnt = 0, t = 1;
    for (int g = __gcd(a, p); g != 1; g = __gcd(a, p)) {
        if (b % g) return -1;
        p /= g;
        ++cnt;
        b /= g;
        t = 1LL * t * (a / g) % p;
        if (b == t) return cnt;
    }
    std::map<int, int> mp;
    int m = ceil(std::sqrt(p));
    int base = b;
    for (int i = 0; i != m; ++i) {
        mp[base] = i;
        base = 1LL * base * a % p;
    }
    base = powMod(a, m, p);
    for (int i = 1; i <= m; ++i) {
        t = 1LL * t * base % p;
        if (mp.count(t))
            return (1LL * i * m - mp[t]) % p + cnt;
    }
    return -1;
}
// https://www.luogu.com.cn/problem/P4195
```

1.4 筛法

筛质数

```cpp
#include<bits/stdc++.h>
using namespace std;
using ll = long long;
const int N = 1e7 + 10;
// minp[i] 为 i 的最小素因子  http://oj.daimayuan.top/course/10/problem/733
int primes[N], pcnt, minp[N]; // 可用于 log 级别分解质因数
bool vis[N]; //合数 true
int n, q;
//linear
void get_prime(int n) {
  for(int i = 2; i <= n; i ++) {
      if(!vis[i]) primes[ ++ pcnt] = i, minp[i] = i;
      for(int j = 1; j <= pcnt && i * primes[j] <= n; ++ j) {
          vis[i * primes[j]] = 1;
            minp[primes[j] * i] = primes[j];
            if(i % primes[j] == 0) break;
      }
    }
}

//about linear :O(nloglogn)
bool isprime[N];
inline void getprime(int n) {
    for (int i = 2; i <= n; i++) isprime[i] = 1;
    for (int i = 2; i <= n; i++) {
        if(isprime[i]) {
            primes[++pcnt] = i;
            if((ll)i*i<=n)
            for (int j = i * i; j <= n; j+=i){
                isprime[j] = 0;
            }
        }
    }
}
```

筛欧拉函数

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

/*phi compute
根据给定 n 计算 phi(n) O(aqrt(n))
核心公式 phi(n) = n*(1-1/p1)*(1 - 1/p2)*...
*/
int get_phi(int n) {
    int res = n;
    for (int i = 2; i <= n / i; i++) {
```

```
        if(n % i == 0) {
            res = res / i * (i - 1); // res *= (1 - 1/n)
            while(n % i == 0)   n /= i;
        }
    }
    if(n > 1)  res = res / n * (n - 1);
    return res;
}

using ll = long long;
const int N = 1e6 + 10;

int phi[N], prime[N];
bool vis[N]; //合数 true

void sel_phi(int n) {
    int cnt = 0;
    phi[1] = 1;
    for (int i = 2; i <= n; i ++) {
        if(!vis[i])  {
            prime[cnt++] = i;
            phi[i] = i - 1;
        }
        for (int j = 0; prime[j] <= n / i; j ++) {
            vis[prime[j] * i] = true;
            if(i % prime[j] == 0) {
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            }
            else
                phi[prime[j] * i] = phi[i] * (prime[j] - 1);
        }
    }
}
```

筛莫比乌斯函数

```
#include <bits/stdc++.h>
using namespace std;
const int N = 50010;
int mu[N], p[N]; // p 为素数数组
bool flg[N];
void init() {
    int tot = 0;  mu[1] = 1;
    for (int i = 2; i < N; ++i) {
        if (!flg[i]) {
            p[++tot] = i;
            mu[i] = -1;
        }
        for (int j = 1; j <= tot && i * p[j] < N; ++j) {
            flg[i * p[j]] = 1;
```

```
            if (i % p[j] == 0) {
                mu[i * p[j]] = 0;
                break;
            }
            mu[i * p[j]] = -mu[i];
        }
    }
    // 常用 mu 前缀和
    // for (int i = 1; i <= N; ++i) mu[i] += mu[i - 1];
}
```

1.5 组合数

1. $C_n^m = C_n^{n-m}$
2. $C_n^m = C_{n-1}^m + C_{n-1}^{m-1}$
3. $C_n^0 + C_n^1 + \cdots + C_n^n = 2^n$
4. $lucas: \ \ C_n^m \equiv C_{n \ \ mod \ \ p}^{m \ \ mod \ \ p} * C_{n/p}^{m/p}$

多重集组合数:

设 $S = \{n_1 \cdot a_1, n_2 \cdot a_2, \cdots n_k \cdot a_k\}$ 是一个由 n_1 个 a_1,n_2 个 a_2,$\cdots$,n_k 个 a_k 组成的多重集。设 $n = \sum_{i=1}^k n_i$，对于任意整数 $r \le n$，从 S 中取出 r 个元素组成一个多重集（不考虑顺序），产生的不同多重集的数量为:

$$C_{k+r-1}^{k-1} - \sum_{i=1}^{k} C_{k+r-n_i-2}^{k-1} + \sum_{1\le i<j\le k} C_{k+r-n_i-n_j-3}^{k-1} - \cdots + (-1)^k C_{k+r-\sum_{i=1}^k n_i-(k+1)}^{k-1}$$

多重集排列数:

多重集 $S = \{n_1 \cdot a_1, n_2 \cdot a_2, \cdots n_k \cdot a_k\}$ 生成的排列是 $\frac{(\sum_{i=1}^k n_i)!}{n_1! \cdot n_2! \cdots n_k!}$

```
//求组合数的几种方法
//不确定的时候都开 long long
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const int mod = 1e9 + 7, N = 1e6 + 10;
//C(a, b) a 上 b 下

/*1. 依照定义 适用于 a, b 很小的时候（几十）*/
int C(ll a, int b) /* a 上 b 下 */{
    if(a < b) return 0;
    int up = 1, down = 1;
    for (ll i = a; i > a - b; i -- ) up = i % mod * up % mod; //up *= i
    for (int j = 1; j <= b; j ++) down = (ll)j * down % mod; // down *= j
    return (ll)up * qpow(down, mod - 2) % mod; //   (up/down)
}

/*2. 递推 杨辉三角  a, b 在 2000 这个数量级 */
```

```
//O(N^2) 1e6~1e7
void init() {
    for (int i = 0; i < N; i ++)
        for (int j = 0; j <= i; j ++)
            if(!j) C[i][j] = 1;
            else C[i][j] = (C[i - 1][j] + C[i - 1][j - 1]) % mod;
}

//最常用
/*3. 预处理 fac[], invfac[]*/
/**
 * //调用 :
 * 1ll * fac[b] * invfac[a] % mod * invfac[b - a] % mod;
 */
// O(N) 1e6 左右 看 N 大小
int fac[N], invfac[N];
void init() {
    fac[0] = 1;
    for (int i = 1; i < N; i ++)  (ll)fac[i] = fac[i - 1]*i% mod;
    invfac[N - 1] = qpow(fac[N - 1], mod - 2);
    for (int i = N - 2; i >= 0; i --)
        invfac[i] = (ll)invfac[i + 1] * (i + 1) % mod;
}

/*4. lucas 定理 当 a, b 的值特别大 如 1e9 以上...1e18 等 */
int C(int a, int b) {
    int res = 1;
    for (int i = 1, j = a; i <= b; i ++, j --) {
        res = (ll)res * j % p;
        res = (ll)res * binpow(i, p - 2) % p;
    }
    return res;
}

ll lucas(ll a, ll b) {//p 为质 (模) 数
    if(a < p && b < p) return C(a, b);
    return (ll)C(a % p, b % p) * lucas(a / p, b / p) % p;
}
```

1.6 容斥原理

S_i 为有限集，$|S|$ 为 S 的大小（元素个数），则：

$$|\bigcup_{i=1}^{n} S_i| = \sum_{i=1}^{n} |S_i| - \sum_{1\le i<j\le n} |S_i \cap S_j| + \sum_{1\le i<j<k\le n} |S_i \cap S_j \cap S_k| + \cdots + (-1)^{n+1}|S_1 \cap \cdots \cap S_n|$$

```
// 容斥原理
// 给定素数集合 A(大小为 k), 求 [L, R] 中素数集合的任意元素的倍数的个数
// 1<=L<=R<=10^18,1<=k<=20,2<=ai<=100
```

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

int main() {
  ll l, r, k, f[25];
  cin >> l >> r >> k;
  for (int i = 0; i < k; i++) cin >> f[i];

  ll ans = 0;

  for (int i = 1; i < 1 << k; i ++) {// 枚举集合中全部的非空子集
    ll cnt = 0, a = r, b = l - 1;  // cnt 用来表示所取的数的个数
    for (int j = 0; j < k; j ++) {
      if(i >> j & 1) {
        cnt++;
        a /= f[j], b /= f[j];
      }
    }
    if(cnt & 1) ans += (a - b);
    else ans -= (a - b);
  }
  cout << ans << endl;
  return 0;
}
```

1.7 数论分块

考虑和式：$\sum_{i=1}^{n} f(i)\lfloor\frac{n}{i}\rfloor$，由于 $\lfloor\frac{n}{i}\rfloor$ 的值成一个块状分布，故可以一块一块运算。我们先求出 $f(i)$ 的前缀和，每次以 $[l,r]=[l,\lfloor\frac{n}{\lfloor\frac{n}{i}\rfloor}\rfloor]$ 为一块分块求出贡献累加到结果中。(常配合莫反使用)

常见转换：

- $\lceil\frac{a}{b}\rceil=\lfloor\frac{a-1}{b}+1\rfloor$
- $a \ \ mod \ \ b=a-\lfloor\frac{a}{b}\rfloor*b$

```cpp
// for(int i = st;i<=ed;i++) ans += num/i
ll block(ll st, ll ed, ll num) {
  //sum(num/i i in [st,ed])
  ll L = 0, res = 0;
  ed = min(ed, num);
  for (ll i = st; i <= ed; i = L + 1) {
        L = min(ed, num / (num / i)); //该区间的最后一个数
        res += (L - i + 1) * (num / i); //区间 [i,L] 的 num/i 都是一个值
        // res += (s(L) - s(i-1)) * (num/i); //s(i) 为 f(i) 前缀和
    }
  return res;
}
```

1.8 Möbius 反演

μ 为莫比乌斯函数，定义为

$$\mu(x) = \begin{cases} 1 & n = 1 \\ 0 & n\text{含有平方因子} \\ (-1)^k & k\text{为 n 本质不同的质因子个数} \end{cases}$$

性质：

$$\sum_{d|n} \mu(d) = \begin{cases} 1 & n = 1 \\ 0 & n \neq 1 \end{cases}$$

证：设 $n = \prod_{i=1}^{k} p_i^{c_i}, n' = \prod_{i=1}^{k} p_i$

那么 $\sum_{d|n} \mu(d) = \sum_{d|n'} \mu(d) = \sum_{i=0}^{k} C_k^i \cdot (-1)^i = (1 + (-1))^k = 1$

反演：

形式一：

$$f(n) = \sum_{d|n} g(d) \Leftrightarrow g(n) = \sum_{d|n} \mu(d) f(\frac{n}{d})$$

证：

$$\sum_{d|n} \mu(d) f\left(\frac{n}{d}\right) = \sum_{d|n} \mu(d) \sum_{k|\frac{n}{d}} g(k) = \sum_{k|n} g(k) \sum_{d|\frac{n}{k}} \mu(d) = g(n)$$

用 $\sum_{d|n} g(d)$ 来替换 $f(\frac{n}{d})$，再变换求和顺序。最后一步变换的依据：$\sum_{d|n} \mu(d) = [n = 1]$，因此在 $\frac{n}{k} = 1$ 时第二个和式的值才为。此时 $n = k$，故原式等价于 $\sum_{k|n} [n = k] \cdot g(k) = g(n)$

形式二：

$$f(n) = \sum_{n|d} g(d) \Leftrightarrow g(n) = \sum_{n|d} \mu(\frac{d}{n}) f(d)$$

1.9 高斯消元

```
#include<bits/stdc++.h>
using namespace std;
const int N = 110;
const double eps = 1e-6;
int n;
double a[N][N];

int gauss() {
    int c, r;
    for(c = 0, r = 0; c < n; c ++) {
        int t = r;
        for(int i = r; i < n; i ++)//找到首元素最大
            if(fabs(a[i][c]) > fabs(a[t][c]))
                t = i;

        if(fabs(a[t][c]) < eps) continue;

        for(int i = c; i <= n; i ++) swap(a[t][i], a[r][i]);
```

```
        for(int i = n; i >= c; i --) a[r][i] /= a[r][c];
        for(int i = r + 1; i < n; i ++)
            if(fabs(a[i][c]) > eps)
                for(int j = n; j >= c; j --)
                    a[i][j] -= a[r][j] * a[i][c];
        r ++;
    }
    if(r < n) {
        for(int i = r; i < n; i ++)
            if(fabs(a[i][n]) > eps)
                return 2;
        return 1;
    }

    for(int i = n - 1; i >= 0; i --)
        for(int j = i + 1; j < n; j ++)
            a[i][n] -= a[i][j] * a[j][n];

    return 0;//有唯一解
}

int main() {
    cin >> n ;
    for(int i = 0; i < n; i ++)
        for(int j = 0; j < n + 1; j ++)
            cin >> a[i][j];

    int t = gauss();
    if(t == 0)
        for(int i = 0; i < n; i ++) printf("%.2f\n", a[i][n]);
    else if(t == 1)
        puts("Infinite group solutions");
    else puts("No solution");

    return 0;
}
```

1.10 Miller Rabin 素数测试

```
//loj143  prime test
#include <bits/stdc++.h>
using namespace std;
using ull = unsigned long long;
using ll = long long;
/*  O(sqrt(n))
bool is_prime(ll x)
{
    if(x < 2) return false;
    for(ll i = 2; i <= x / i; ++i)
```

```
        if(x % i == 0) return false;
    return true;
}
*/
//常常是大素数测试，要用到 int128
inline ll qmul(ll a, ll b, ll p) { return (ll)((__int128)a * b % p); }
ll qpow(ll a, ll b, ll p) {
    ll res = 1;
    while(b) {
        if(b & 1) res = qmul(res, a, p);
        a = qmul(a, a, p);
        b >>= 1;
    }
    return res;
}
const int test_time = 8;

bool mr_test(ll n) {
    if(n < 3 || n % 2 == 0)  return n == 2;
    ll a = n - 1, b = 0;
    while(a % 2 == 0)  a /= 2, ++b;

    for (int i = 1, j; i <= test_time; ++i)  {
        ll x = rand() % (n - 2) + 2, v = qpow(x, a, n);
        if(v == 1)  continue;
        for (j = 0; j < b; ++j) {
            if(v == n - 1) break;
            v = qmul(v, v, n);
        }
        if(j >= b) return 0;
    }
    return 1;
}

int main() {
    srand(time(0));
    ll x;
    while(cin >> x) {
        if(mr_test(x)) puts("Y");
        else puts("N");
    }
    return 0;
}
```

1.11 FFT

```
#include <bits/stdc++.h>
#include <any>
#define rep(i, a, b) for (int i = (a); i <= (b); i ++)
```

```
using namespace std;

namespace FFT {
const double PI = acos(-1);
using C = complex<double>;
vector<int> rev;
vector<C> roots{C(0, 0), C(1, 0)};
void dft(vector<C>& a) {
    int n = (int)a.size();
    if ((int)rev.size() != n) {
        int k = __builtin_ctz(n) - 1;
        rev.resize(n);
        for (int i = 0; i < n; ++i) {
            rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
        }
    }
    if ((int)roots.size() < n) {
        int k = __builtin_ctz(roots.size());
        roots.resize(n);
        while ((1 << k) < n) {
            C e = polar(1.0, PI / (1 << k));
            for (int i = 1 << (k - 1); i < (1 << k); ++i) {
                roots[2 * i] = roots[i];
                roots[2 * i + 1] = roots[i] * e;
            }
        ++k;
        }
    }
    for (int i = 0; i < n; ++i) if (rev[i] < i) {
        swap(a[i], a[rev[i]]);
    }
    for (int k = 1; k < n; k *= 2) {
        for (int i = 0; i < n; i += 2 * k) {
            for (int j = 0; j < k; ++j) {
                auto u = a[i + j], v = a[i + j + k] * roots[k + j];
                a[i + j] = u + v;
                a[i + j + k] = u - v;
            }
        }
    }
}
void idft(vector<C>& a) {
  int n = (int)a.size();
  reverse(a.begin() + 1, a.end());
  dft(a);
  for (auto& x : a) x /= n;
}
} // namespace FFT

```

```
vector<int> mul(const vector<int> &A, const vector<int> &B) {
  int n = max(A.size(), B.size()), tot = max(1, n * 2 - 1);
  int sz = 1 << __lg(tot * 2 - 1);
  vector<complex<double>> C(sz);
  for (int i = 0; i < A.size(); ++i) C[i].real(A[i]);
  for (int i = 0; i < B.size(); ++i) C[i].imag(B[i]);
  FFT::dft(C);
  for (auto &x : C) x *= x;
  FFT::idft(C);
  vector<int> ans(A.size() + B.size() - 1);
  for (int i = 0; i < ans.size(); ++i) ans[i] = int(C[i].imag() / 2 + 0.2);
  return ans;
}

int main() {

  cin.tie(nullptr)->sync_with_stdio(false);
  int n, m;
  cin >> n >> m;
  vector<int> a(n + 1), b(m + 1);
  for (auto &x : a) cin >> x;
  for (auto &x : b) cin >> x;
  auto c = mul(a, b);
  for (auto &x : c) cout << x << ' ';
  cout << '\n';
    std::any a = 34;
    return 0;

}
```

2 数据结构

2.1 （带权）并查集

```
const int N = 1e5 + 10;
int fa[N], n, m, d[N];

int find(int x) {return x == fa[x] ? x : fa[x] = find(fa[x]);}
// 对于带权并查集，一般的 find 函数写作：
int find(int x) {
    if(x == fa[x]) return x;
    int rt = find(fa[x]); //这和下面一行顺序很重要
    d[x] += d[fa[x]]; //可以改成 d[x] ^= d[fa[x]], 根据权值意义的需要修改
    return fa[x] = rt;
}

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

```
}
```

2.2 Sparse Table

时间复杂度 $\mathcal{O}(1)$，空间复杂度 $\mathcal{O}(nlogn)$
静态区间查询可重复贡献信息，如“区间最值”、“区间按位和”、“区间按位或”、“区间 GCD”

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

const int N = 1e5 + 10;
int f[N][21], n, m;
int a[N];
//f[i][j] 表示闭区间 [i, i + 2^j - 1] 的最大值

void init_st() {
    // cout << __lg(N) << endl;
    for (int j = 0; j < 21; j ++)
        for (int i = 1; i + (1 << j) - 1 <= n; i++)//区间长度是 2^j 所以要减一
            if(!j)  f[i][j] = a[i];
            else
                f[i][j] = max(f[i][j - 1], f[i + (1 << j - 1)][j - 1]);
}

int query(int l, int r) {
    int k = __lg(r - l + 1);
    return max(f[l][k], f[r - (1 << k) + 1][k]);
}
```

2.3 01Trie

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 10,M = N * 31;
int a[N];
int son[M][2], idx;

void insert(int x) {
    int p = 0;
    for (int i = 30; i >= 0; --i) {
        int u = ( (x>>i) & 1);
        if(!son[p][u]) son[p][u] = ++idx;
        p = son[p][u];
    }
}
// 集合内和 x 异或的最大值
int query(int x) {
  int p = 0, res = 0;
  for (int i = 30; i >= 0; --i) {
    int u = (x >> i) & 1;
```

```
        if(son[p][u ^ 1]) p = son[p][u ^ 1], res |= (1 << i);
        else p = son[p][u];
            // 集合内和 x 异或的最小值
            // if(son[p][u])  p = son[p][u];
            // else res |= (1 << i), p = son[p][u ^ 1];
    }
    return res;
}

int main() {
    int n, res = 0;
    cin >> n;
    for(int i = 0; i < n; i++)  cin >> a[i];
    for(int i = 0; i < n; i++) {
        insert(a[i]);
        res = max(res, query(a[i]));
    }
    cout << res;
    return 0;
}
```

2.4 树状数组

```
const int N = 1e5 + 10;
int tr[N], a[N];
inline int lowbit(int x) {return x & -x;}

int query(int x) {
    int res = 0;
    for (int i = x; i; i -= lowbit(i)) res += tr[i];
    return res;
}

void add(int x, int val) {
    for(int i = x; i <= n; i += lowbit(i))
        tr[i] += val;
}

//fenwich-tree 写区间修改，区间查询
//记录两个数组 b[i] = a[i] - a[i - 1]; c[i] = i * b[i];
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 1e5 + 10;
int a[N], b[N];
ll t1[N], t2[N]; //维护 b[i], b[i] * i 的前缀和
int n, m;

void add(ll tr[], int x, ll c) {
```

```
    for (int i = x; i <= n; i += lowbit(i))
        tr[i] += c;
}

ll query(ll tr[], int x) {
    ll res = 0;
    for (int i = x; i; i-= lowbit(i))
        res += tr[i];
    return res;
}

ll preSum(int x) { return query(t1, x) * (x + 1) - query(t2, x); }

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i<=n; i++) scanf("%d", &a[i]);

    for (int i = 1; i <= n; i++) {
        int b = a[i] - a[i - 1];
        add(t1, i, b);
        add(t2, i, (ll)b * i);
    }
    while(m -- ) {
        char op[2];
        int l, r, d;
        scanf("%s%d%d", op, &l, &r);
        if(*op == 'Q')
            printf("%lld\n", preSum(r) - preSum(l - 1));
        else {
            scanf("%d", &d);
            //a[l] += d;
            add(t1, l ,d), add(t2, l, l * d);
            add(t1, r + 1, -d), add (t2, r + 1, (r + 1) * -d);
        }
    }
    return 0;
}
```

2.5 线段树

```
//常见维护
/**
 * 区间和, 最值
 * 维护最大连续字段和 （维护 lmax, rmax, tmax）
 * 维护区间平方和
 * 区间修改成之指定数 维护 sum, lazy(指定数值), bool changed;
 * 区间内开根号: 由于六次根号 1e12（向下取整）即得到 1, 所以可以暴力修改
 * 区间内数字同时乘以一个数 如下:
 */
```

```cpp
#include<bits/stdc++.h>
using namespace std;
using ll = long long;
const int N = 1e5 + 10;
int n, m, mod;
int a[N];

struct node {
    int l, r;
    int sum, add, mul;
} t[4 * N];

void eval(node &t, int add, int mul) {
    t.sum = ((ll)t.sum * mul + (ll)(t.r - t.l + 1) * add) % mod;
    t.mul = (ll)t.mul * mul % mod;
    t.add = ((ll)t.add * mul + add) % mod;
}

void pushup(int p) {
    t[p].sum = (t[p << 1].sum + t[p << 1 | 1].sum) % mod;
}

void pushdown(int p) {
    eval(t[p << 1], t[p].add, t[p].mul);
    eval(t[p << 1 | 1], t[p].add, t[p].mul);

    t[p].add = 0, t[p].mul = 1;
}

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

void modify(int p, int l, int r, int add, int mul) {
    if(t[p].l >= l && t[p].r <= r)  eval(t[p], add, mul);
    else {
        pushdown(p);
        int mid = t[p].l + t[p].r >> 1;
        if(l <= mid)  modify(p << 1, l, r, add, mul);
        if(r > mid)  modify(p << 1 | 1, l, r, add, mul);
        pushup(p);
```

```
    }
}

int query(int p, int l, int r) {
    if(t[p].l >= l && t[p].r <= r)  return t[p].sum;

    pushdown(p);
    int res = 0;
    int mid = t[p].l + t[p].r >> 1;

    if(l <= mid) res += query(p << 1, l, r);
    if(r > mid)  res += query(p << 1 | 1, l, r);
    res %= mod;
    return res;
}

int main() {
    scanf("%d%d", &n, &mod);
    for (int i = 1; i <= n; i++)
        scanf("%d", &a[i]);
    build(1, 1, n);

    scanf("%d", &m);
    while(m -- ) {
        int op, l, r, d;
        scanf("%d%d%d", &op, &l, &r);
        if(op == 1) {
            scanf("%d", &d);
            modify(1, l, r, 0, d);
        }
        else if(op == 2) {
            scanf("%d", &d);
            modify(1, l, r, d, 1);
        }
        else
            printf("%d\n", query(1, l, r));
    }
    return 0;
}
```

扫描线：(面积)

```
//p1502 线段树扫描线算法
#include<bits/stdc++.h>
using namespace std;
using ll = long long;
const ll N = 1e4 + 10;
struct L {
    ll x, y1, y2;
    ll c;
    //当左矩形的右边界与右矩形的左边界重合时，该线上的点应属于能被两个窗户都能看见的状态所以先加
```

```
    bool operator<(const L &rhs) const { return x == rhs.x ? c < rhs.c : x < rhs.x; }
}line[2 * N];

ll n, w, h, m;
ll b[2 * N]; //离散化前的 y 轴

struct node {
    ll l, r;
    ll maxv, add;
} t[8 * N];


void pushdown(ll p) {
    node &root = t[p], &nl = t[p << 1], &nr = t[p << 1 | 1];
    if(root.add) {
        nl.add += root.add, nl.maxv += root.add;
        nr.add += root.add, nr.maxv += root.add;
        root.add = 0;
    }
}

void pushup(ll p) {
    t[p].maxv = max(t[p << 1].maxv, t[p << 1 | 1].maxv);
}

void modify(ll p, ll l, ll r, ll c) {
    if(t[p].l >= l && t[p].r <= r) {
        t[p].maxv += c;
        t[p].add += c;
        return;
    }
    pushdown(p);
    ll mid = t[p].l + t[p].r >> 1;
    if(l <= mid)  modify(p << 1, l, r, c);
    if(r > mid) modify(p << 1 | 1, l, r, c);
    pushup(p);

}

void build(ll p, ll l, ll r) {
    if(l == r) {
        t[p] = {l, r, 0, 0};
        return;
    }
    t[p].l = l, t[p].r = r;
    ll mid = l + r >> 1;
    build(p << 1, l, mid);
    build(p << 1 | 1, mid + 1, r);
    //pushup(p);//初始化都是 0 不用 pushup()
```

```
}

int main() {
    ll T;
    scanf("%lld", &T);
    while( T -- ) {
        memset(line, 0, sizeof(line));
        memset(b, 0, sizeof(b));
        memset(t, 0, sizeof(t));

        scanf("%lld%lld%lld", &n, &w, &h);
        for (ll i = 1, j = 0; i <= n; i++) {
            ll x, y, l;
            scanf("%lld%lld%lld", &x, &y, &l);
            line[i] = {x, y, y + h - 1, l};
            line[i + n] = {x + w - 1, y, y + h - 1, -l};
            b[ ++ j] = y;
            b[ ++ j] = y + h - 1;
        }
        n <<= 1;
        sort(b + 1, b + 1 + n);
        m = unique(b + 1, b + 1 + n) - b - 1;//unique 得到 end() 迭代器
        sort(line + 1, line + 1 + n);

        for (ll i = 1; i <= n; i++) {
            line[i].y1 = lower_bound(b + 1, b + m + 1, line[i].y1) - b - 1;
            line[i].y2 = lower_bound(b + 1, b + m + 1, line[i].y2) - b - 1;
        }
        build(1, 1, m - 1);

        ll res = 0;
        for (ll i = 1; i <= n; i++) {
            modify(1, line[i].y1, line[i].y2, line[i].c);
            res = max(res, t[1].maxv);
        }
        printf("%d\n", res);
    }
    return 0;
}
```

2.6 可持久化线段树

```
//luogu 3824 kth-number
#include <bits/stdc++.h>
using namespace std;
const int N = 2e5 + 10, M = (N << 2) + 17 * N;

struct node {
    int l, r;
```

```cpp
    int cnt;
} t[M];
int idx, a[N];
vector<int> num;
int find(int x) { return lower_bound(num.begin(), num.end(), x) - num.begin(); }

int insert(int now, int l, int r, int x) {
    int p = ++ idx;
    t[p] = t[now];
    if (l == r) {
        t[p].cnt ++ ;
        return p;
    }
    int mid = l + r >> 1;
    if(x <= mid) t[p].l = insert(t[now].l, l, mid, x);
    else t[p].r = insert(t[now].r, mid + 1, r, x);
    t[p].cnt = t[t[p].l].cnt + t[t[p].r].cnt;

    return p;
}

int build(int l, int r) {
    int p = ++ idx;
    if (l == r) return p;
    int mid = l + r >> 1;
    t[p].l = build(l, mid), t[p].r = build(mid + 1, r);
    return p;
}

int query(int x, int y, int l, int r, int k) {
    if(l == r) return l;
    int cnt = t[t[y].l].cnt - t[t[x].l].cnt;
    int mid = l + r >> 1;
    if(k <= cnt) return query(t[x].l, t[y].l, l, mid, k);
    else return query(t[x].r, t[y].r, mid + 1, r, k - cnt);
}

int n, m, root[N];

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i ++ ) {
        scanf("%d", &a[i]);
        num.push_back(a[i]);
    }

    sort(num.begin(), num.end());
    num.erase(unique(num.begin(), num.end()), num.end());

```

```
    root[0] = build(0, num.size() - 1);

    for (int i = 1; i <= n; i ++ )
        root[i] = insert(root[i - 1], 0, num.size() - 1, find(a[i]));
    while (m -- ) {
        int l, r, k;
        scanf("%d%d%d", &l, &r, &k);
        printf("%d\n", num[query(root[l - 1], root[r], 0, num.size() - 1, k)]);
    }

    return 0;
}
```

2.7 线段树合并

```
int merge(int p, int q, int l, int r) {
    if(!p || !q) return p + q;
    if(l == r) {
        //维护信息，一般是 t[p].val += t[q].val 等
        // t[p].val.first += t[q].val.first;
        return p;
    }
    int mid = l + r >> 1;
    t[p].l = merge(t[p].l, t[q].l, l, mid);
    t[p].r = merge(t[p].r, t[q].r, mid + 1, r);
    // pushup();
    //  t[p].val = max(t[t[p].l].val, t[t[p].r].val);
    return p;
}
```

2.8 树链剖分

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

using ll = long long;
const int N = 1e5 + 10;

struct node {
  int l, r;
  ll add, sum;
} t[N << 2];
int n, m, w[N], nw[N];
vector<int> G[N];

int dep[N], top[N], son[N], dfn[N], sz[N], fa[N], cnt;
///////////////////////线段树部分////////////////////////

```

```
void pushdown(int p) {
  auto &rt = t[p], &nl = t[p << 1], &nr = t[p << 1 | 1];
  if(rt.add) {
    nl.add += rt.add, nl.sum += (ll)(nl.r - nl.l + 1) * rt.add;
    nr.add += rt.add, nr.sum += (ll)(nr.r - nr.l + 1) * rt.add;
    rt.add = 0;
  }
}

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

void build(int p, int l, int r) {
  t[p] = {l, r, 0, nw[l]};
  if(l == r) return;

  int mid = l + r >> 1;
  build(p << 1, l, mid);
  build(p << 1 | 1, mid + 1, r);
  pushup(p);
}

ll query(int p, int l, int r) {
  if(t[p].l >= l && t[p].r <= r) return t[p].sum;

  pushdown(p);
  int mid = t[p].l + t[p].r >> 1;
  ll res = 0;
  if(l <= mid)  res += query(p << 1, l, r);
  if(r > mid)  res += query(p << 1 | 1, l ,r);
  //pushup(p);
  return res;
}

void modify(int p, int l, int r, int k) {
  if(t[p].l >= l && t[p].r <= r) {
    t[p].sum += (t[p].r - t[p].l + 1) * k;
    t[p].add += k;
    return;
  }

  pushdown(p);
  int mid = t[p].l + t[p].r >> 1;
  if(l <= mid) modify(p << 1, l, r, k);
  if(r > mid) modify(p << 1 | 1, l , r, k);
  pushup(p);
}

//////////////////////树剖部分////////////////////////
//第一次 dfs 维护 sz, 重儿子,  dep[], fa[]
```

```
void dfs1(int u, int fath) {
  sz[u] = 1, dep[u] = dep[fath] + 1, fa[u] = fath;
  for(int v:G[u]) {
    if(v == fath) continue;
    dfs1(v, u);
    sz[u] += sz[v];
    if(sz[son[u]] < sz[v]) son[u] = v;
  }
}
//第二次 dfs，维护 dfs 序，
void dfs2(int u, int tp) {
  dfn[u] = ++cnt, nw[cnt] = w[u], top[u] = tp;
  if(!son[u]) return;
  dfs2(son[u], tp); //递归重儿子
  //维护轻儿子信息
  for(int v:G[u]) {
    if(v == fa[u] || v == son[u]) continue;
    dfs2(v, v);
  }
}

void modify_path(int u, int v, int k) {
  while(top[u] != top[v]) {
    if(dep[top[u]] < dep[top[v]]) swap(u, v);
    modify(1, dfn[top[u]], dfn[u], k);
    u = fa[top[u]];
  }
  if(dep[u] < dep[v]) swap(u, v);
  modify(1, dfn[v], dfn[u], k);
}

void modify_tree(int u, int k) {
  modify(1, dfn[u], dfn[u] + sz[u] - 1, k);
}

ll query_tree(int u) {
  return query(1, dfn[u], dfn[u] + sz[u] - 1);
}

ll query_path(int u, int v) {
  ll res = 0;
  while(top[u] != top[v]) {
    if(dep[top[u]] < dep[top[v]]) swap(u, v);
    res += query(1, dfn[top[u]], dfn[u]);
    u = fa[top[u]];
  }
  if(dep[u] < dep[v]) swap(u, v);
  res += query(1, dfn[v], dfn[u]);
  return res;
```

```
}

////////////////////////////////////////////////////////////
int main() {

  scanf("%d", &n);
  for(int i = 1; i <= n; i ++) scanf("%d", &w[i]);
  for(int i = 1; i < n; i ++) {
    int u, v; scanf("%d%d", &u, &v);
    G[u].pb(v), G[v].pb(u);
  }
  dfs1(1, 0);
  dfs2(1, 1);

  build(1, 1, n);

  scanf("%d", &m);
  while(m -- ) {
    int op, u, v, k;
    scanf("%d%d", &op, &u);
    if(op == 1) {
      scanf("%d%d", &v, &k);
      modify_path(u, v, k);
    }
    else if(op == 2) {
      scanf("%d", &k);
      modify_tree(u, k);
    }
    else if(op == 3) {
      scanf("%d", &v);
      printf("%lld\n", query_path(u, v));
    }
    else
      printf("%lld\n", query_tree(u));
  }
  return 0;
}
```

2.9 左偏树

支持操作（以维护最小值为例）:

1. 找到最小值 $\mathcal{O}(1)$
2. 删除最小值 $\mathcal{O}(logn)$
3. 插入一个值 $\mathcal{O}(logn)$
4. 合并两个堆 $\mathcal{O}(logn)$

```
#include <bits/stdc++.h>
#define endl '\n'
```

```cpp
using namespace std;
const int N = 2e5 + 10;
int val[N], lson[N], rson[N], dis[N];
int fa[N], idx, n;
int find(int x) { return x == fa[x] ? x : fa[x] = find(fa[x]); }

bool cmp(int x, int y) { return val[x] == val[y] ? x < y : val[x] < val[y]; }

int merge(int x, int y) {
    if(!x || !y) return x + y;
    if(cmp(y, x)) swap(x, y);
    rson[x] = merge(rson[x], y);
    if(dis[rson[x]] > dis[lson[x]]) swap(lson[x], rson[x]);
    dis[x] = dis[rson[x]] + 1;
    return x;
}

int main() {
    ios::sync_with_stdio(false), cin.tie(0);
    cin >> n;
    val[0] = 2e9;
    while(n --) {
        int op, x, y;  cin >> op;
        if(op == 1) {
            cin >> x;
            val[++idx] = x;
            fa[idx] = idx;
            dis[idx] = 1;
        }
        else if(op == 2) {
            cin >> x >> y;
            x = find(x), y = find(y);
            if(x != y) {
                if(cmp(y, x)) swap(x, y); //x 为较小的
                fa[y] = x;
                merge(x, y);
            }
        }
        else if(op == 3) {
            cin >> x;
            cout << val[find(x)] << endl;
        }
        else { // 删除 x 所在堆的最小值
            cin >> x; x = find(x);
            if(cmp(rson[x], lson[x])) swap(lson[x], rson[x]);
            fa[x] = lson[x], fa[lson[x]] = lson[x];
            merge(lson[x], rson[x]);
        }
    }
```

```
    return 0;
}
```

2.10 莫队

普通莫队：

```
#include <bits/stdc++.h>
#define endl '\n'
#define rep(i, a, b) for (int i = (a); i <= (b); i ++)
using namespace std;

const int N = 5e4 + 10, M = 2e5 + 10, S = 1e6 + 10; //值域

int n, A[N], ans[M], cnt[S], m, sq, cur;

struct query {
    int l, r, id;
    bool operator<(const query &rhs) const { //奇偶化排序
        if (l / sq != rhs.l / sq)
            return l < rhs.l;
        if (l / sq & 1)
            return r < rhs.r;
        return r > rhs.r;
    }
} q[M];

void add(int p) {
    if(cnt[A[p]] == 0) cur++;
    cnt[A[p]]++;
}

void del(int p) {
    cnt[A[p]]--;
    if(cnt[A[p]] == 0) cur--;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    cin >> n;
    sq = sqrt(n);
    rep(i, 1, n) cin >> A[i];
    cin >> m;
    rep(i, 1, m) {
        int l, r;
        cin >> l >> r;
        q[i] = {l, r, i};
    }
```

```
    sort(q + 1, q + 1 + m);

    int l = 1, r = 0;
    rep(i, 1, m) {
        while(l > q[i].l) add(--l);
        while(r < q[i].r) add(++r);
        while(l < q[i].l) del(l++);
        while(r > q[i].r) del(r--);
        ans[q[i].id] = cur;
    }
    rep(i, 1, m)  cout << ans[i] << endl;

    return 0;
}
```

待修莫队：

```
#include <bits/stdc++.h>
#define endl '\n'
#define rep(i, a, b) for (int i = (a); i <= (b); i ++)
using namespace std;

const int N = 134000, S = 1e6 + 10; //值域

int n, m, mq, mc, len, cur;
int w[N], cnt[S], ans[N];
struct Query {
    int id, l, r, tim;
}q[N];
struct Modify {
    int pos, val;
} c[N];

int get(int x) {
    return x / len;
}

bool cmp(const Query& a, const Query& b) {
    int al = get(a.l), ar = get(a.r);
    int bl = get(b.l), br = get(b.r);
    if (al != bl) return al < bl;
    if (ar != br) return ar < br;
    return a.tim < b.tim;
}

void add(int val) {
    if(cnt[val] == 0) cur++;
    cnt[val]++;
}

void del(int val) {
```

```
    cnt[val]--;
    if(cnt[val] == 0) cur--;
}
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cin >> n >> m;
    rep(i, 1, n) cin >> w[i];

    rep (i, 1, m) {
        char op[2];
        int a, b;
        cin >> op >> a >> b;
        if (*op == 'Q') mq ++, q[mq] = {mq, a, b, mc};
        else c[ ++ mc] = {a, b};
    }

    len = cbrt((double)n * max(1 , mc)) + 1;
    sort(q + 1, q + mq + 1, cmp);

    int l = 1, r = 0, t = 0;
    rep(i, 1, mq) {
        auto [id, ql, qr, qt] = q[i];
        while (l < ql) del(w[l++]);
        while (l > ql) add(w[--l]);
        while (r < qr) add(w[++r]);
        while (r > qr) del(w[r--]);
        while (t < qt) {
            t ++ ;
            if (ql <= c[t].pos && qr >= c[t].pos) {
                del(w[c[t].pos]);
                add(c[t].val);
            }
            swap(w[c[t].pos], c[t].val);
        }
        while (t > qt) {
            if (ql <= c[t].pos && qr >= c[t].pos) {
                del(w[c[t].pos]);
                add(c[t].val);
            }
            swap(w[c[t].pos], c[t].val);
            t--;
        }
        ans[id] = cur;
    }

    rep(i, 1, mq) printf("%d\n", ans[i]);
    return 0;
}
```

3 图论

3.1 spfa

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

const int N = 1e5 + 10, inf = 0x3f3f3f3f;
struct node{int v, w;};
vector<node> G[N];
int dis[N], n, m;
bool inq[N];

void spfa() {
    memset(dis, 0x3f, sizeof dis);
    dis[1] = 0;
    inq[1] = 1;
    queue<int> q;
    q.push(1);
    while(q.size()) {
        int u = q.front(); q.pop();
        inq[u] = 0;
        for(auto [v, w]:G[u]) {
            if(dis[v] > w + dis[u]) {
                dis[v] = dis[u] + w;
                if(!inq[v])
                    q.push(v), inq[v] = true;
            }
        }
    }
}

int main() {
    cin >> n >> m;
    while(m -- ) {
        int u, v, w;
        cin >> u >> v >> w;
        G[u].pb({v, w});
    }
    spfa();
    if(dis[n] == inf)    cout << "impossible";
    else         cout << dis[n];
    return 0;
}
```

3.2 dijkstra

稀疏图 dijkstra：

```
//acwing 849
#include <bits/stdc++.h>
using namespace std;
const int N = 510, inf = 0x3f3f3f3f;
int dis[N], G[N][N], n, m;
bool vis[N];

void dij() {
    memset(dis, 0x3f, sizeof dis);
    dis[1] = 0;
    for (int j = 0; j < n; j ++) {
        int minv = inf, pos = -1;
        for(int i = 1; i <= n; i ++)
            if (!vis[i] && minv > dis[i])
                minv = dis[i], pos = i;

        if(pos == -1) break;
        vis[pos] = 1;
        for (int i = 1; i <= n; i ++)
            if(!vis[i] && dis[pos] + G[pos][i] < dis[i])
                dis[i] = dis[pos] + G[pos][i];
    }
}

int main() {
    cin >> n >> m;
    scanf("%d %d", &n, &m);
    memset(G, 0x3f, sizeof(G));
    while(m --) {
        int u, v, w; scanf("%d %d %d", &u, &v, &w);
        G[u][v] = min(G[u][v], w);
    }

    dij();

    cout << (dis[n] == inf ? -1 : dis[n]);
    return 0;
}
```

稠密图 dijkstra：

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

using P = pair<int, int>;
const int N = 151000, inf = 0x3f3f3f3f;
struct node{int v, w;};
vector<node> G[N];
```

```cpp
int dis[N], n, m;
bool vis[N];

void dij() {
    memset(dis, 0x3f, sizeof dis);
    priority_queue<P, vector<P>, greater<P>> q;
    q.push({0, 1});
    while(q.size()) {
        auto t = q.top(); q.pop();
        int u = t.se, d = t.fi;
        if(vis[u]) continue;
        vis[u] = true;
        for(auto [v, w] : G[u]) {
            if(dis[v] > d + w) {
                dis[v] = d + w;
                q.push({dis[v], v});
            }
        }
    }
}

int main() {
    ios::sync_with_stdio(false);
    cin >> n >> m;
    while(m -- ) {
        int u, v, w; cin >> u >> v >> w;
        G[u].pb({v, w});
    }
    dij();
    cout << (dis[n] == inf ? -1 : dis[n]);
    return 0;
}
```

3.3 最小生成树

```cpp
// kruskal
const int N = 1e5 + 10;
struct edge {
    int u, v, w;
    bool operator<(const edge &rhs) const { return w < rhs.w; }
} edges[N];

int fa[N], n, m;
int find(int x) { return x == fa[x] ? x : fa[x] = find(fa[x]); }

int kruskal() {
    cin >> n >> m;
    int u, v, w, ans = 0;
    for (int i = 1; i <= m; i ++) {
```

```
        cin >> u >> v >> w;
        edges[i] = {u, v, w};
    }
    sort(edges + 1, edges + 1 + m);
    for (int i = 1; i <= n; i ++) fa[i] = i;
    for (int i = 1; i <= m; i ++) {
        auto [u, v, w] = edges[i];
        u = find(u), v = find(v);
        if(u == v) continue;
        fa[u] = v;
        ans += w;
    }
    return ans;
}

//prim
const int N = 510, inf = 0x3f3f3f3f;
int G[N][N], dis[N];
int n, m;
bool vis[N];

int prim() {
    int res = 0;
    memset(dis, 0x3f, sizeof dis);
    dis[1] = 0;   //随便选一点进入 mst 集合
    for(int j = 0; j < n; j ++) {
        int minv = inf, pos = -1;
        for(int i = 1; i <= n; i ++)
            if(!vis[i] && dis[i] < minv)
                pos = i, minv = dis[i];

        if(pos == -1) return inf;
        vis[pos] = true;
        res += dis[pos];

        for(int i = 1; i <= n; i ++)
            if(!vis[i] && dis[i] > G[pos][i])
                dis[i] = G[pos][i];
    }
    return res;
}
```

另外，对于完全图的 MST 问题，可以考虑使用 $Boruvka$ 算法。我们要在 $nlogn$ 或 $nlog^2n$ 时间内求出每个连通块最小的连接的边，而这个边权一般可通过点权以一定方式求出。通常不用直接写出，运用该思想求解。

3.4 kruskal 重构树

```
//kruskal 重构树

```

```
//性质:
//两个点之间的所有简单路径上最大边权的最小值
//  = 最小生成树上两个点之间的简单路径上的最大值
//  = Kruskal 重构树上两点之间的 LCA 的权值。
//loj136
#include <bits/stdc++.h>
#define pb push_back
using namespace std;

const int N = 1010 << 1, M = 3e5 + 10;
int n, m, k, val[N];// kruskal 重构树的点权
int idx;  //重构树的节点数

struct Edge{
    int u, v, w;
    bool operator<(const Edge &rhs) const { return w < rhs.w; }
}edges[M];

vector<int> G[N];

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

int dep[N], fa[N][21];

void bfs(int s) {
    dep[0] = 0, dep[s] = 1;
    queue<int> q;
    q.push(s);
    while(q.size()) {
        int u = q.front(); q.pop();
        for(int v:G[u]) {
            if(dep[v] > dep[u] + 1) {
                dep[v] = dep[u] + 1;
                q.push(v);
                fa[v][0] = u;
                for (int i = 1; i <= 20; i ++)
                    fa[v][i] = fa[fa[v][i - 1]][i - 1];
            }
        }
    }
}

int lca(int a, int b) {
    if(dep[a] < dep[b]) swap(a, b);
    for (int k = 20; k >= 0; k --)
        if(dep[fa[a][k]] >= dep[b])
            a = fa[a][k];
    if(a == b) return a;
```

```
    for (int k = 20; k >= 0; k --)
        if(fa[a][k] != fa[b][k])
            a = fa[a][k], b = fa[b][k];
    return fa[a][0];
}

void build() {
    idx = n;
    int cnt = 0;
    for (int i = 1; i <= m; i ++) {
        int u = edges[i].u, v = edges[i].v, w = edges[i].w;
        int fu = find(u), fv = find(v);
        if(fu != fv) {
            val[++idx] = w;
            G[idx].pb(fu), G[idx].pb(fv);
            G[fu].pb(idx), G[fv].pb(idx);
            p[fu] = p[fv] = idx;
            cnt++;
        }
        if(cnt >= n - 1) break;
    }
}

int main() {
    scanf("%d %d %d", &n, &m, &k);
    for (int i = 1; i <= m; i ++) {
        int u, v, w;  scanf("%d %d %d", &u, &v, &w);
        edges[i] = {u, v, w};
    }
    sort(edges + 1, edges + m + 1);
    for (int i = 1; i <= (n << 1); i ++)  p[i] = i;

    build(); // kruskal 重构树

    memset(dep, 0x3f, sizeof dep);
    bfs(idx);  //bfs 的根节点一定要是重构树的最高点

    while(k -- ) {
        int s, t;
        scanf("%d %d", &s, &t);
        if(find(s) != find(t))    puts("-1");
        else
            printf("%d\n", val[lca(s, t)]);
    }
    return 0;
}
```

3.5 二分图匹配

二分图匹配的模型有两个要素：

1. 节点能分成独立的两个集合，每个集合内部有 0 条边
2. 每个节点只能与 1 条匹配边相连

二分图最小覆盖模型特点是：每条边有 2 个端点，二者至少选择一个。
könig 定理：二分图最小点覆盖包含的点数等于二分图最大匹配数包含的边数。
图的最大独立集：点集 S 中任意两点之间都没有边相连。其大小等于 $n-$ 最大匹配数。(n 是二分图总点数)

```
/* 染色法判断二分图
bool vis[N];
int col[N], flag = 1, n, m;
void dfs(int u, int t) {
    if (vis[u]) {
        if (col[u] != t) flag = 0;
        return;
    }
    vis[u] = 1; col[u] = t;
    for (int v : g[u]) {
        dfs(v, t ^ 1);
    }
}
bool isbit() {//是否为二分图
    for (int u = 1; u <= n; u++) {
        if (!vis[u]) dfs(u, 0);
    }
    return flag;
}
*/
int G[N][M];  // 左半部 n, 右半部 m
int n, m, p[M], vis[M];
bool match(int u) {
    for (int i = 1; i <= m; i ++) {
        if(G[u][i] && !vis[i]) {
            vis[i] = true;
            if(p[i] == 0 || match(p[i])) {
                p[i] = u;  return true;
            }
        }
    }
    return false;
}
int main() {
    /* 建图 */
    int res = 0;
    for (int i = 1; i <= n; i ++) {
        memset(vis, 0, sizeof vis);
```

```
        if(match(i)) res++;
    }
    return 0;
}
```

3.6 强连通分量缩点

时间复杂度 $O(m+n)$，反向枚举 `scc_cnt` 即是新图拓扑序。

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

const int N = 1e4 + 10;
vector<int> G[N], G2[N];
stack<int> s;
int n, m, tim, scc_cnt;
int w[N], dfn[N], low[N], id[N];
int dist[N], ind[N], W[N];
bool ins[N];

void tarjan(int u) {
    low[u] = dfn[u] = ++tim;
    s.push(u);  ins[u] = true;
    for(int v:G[u]) {
        if(!dfn[v]) {
            tarjan(v);
            low[u] = min(low[v], low[u]);
        }
        else if(ins[v])
            low[u] = min(low[u], dfn[v]);
    }
    if(low[u] == dfn[u]) {
        int y;  ++scc_cnt;
        do {
            y = s.top(); s.pop();
            ins[y] = false;
            id[y] = scc_cnt;
            W[scc_cnt] += w[y];
        } while (y != u);
    }
}

int sol() {
    queue<int> q;
    for (int i = 1; i <= scc_cnt; i++)
        if(!ind[i]) {
            q.push(i);
            dist[i] = W[i];
        }
```

```cpp

    while(q.size()) {
        //cout << "cnt = " << ++cnt << endl;
        int u = q.front();  q.pop();
        for (int v:G2[u]) {

            ↪ //当有重边时， dist[v] 被更新的值始终不变，即 dist[v] = dist[u] + W[v]; 所以不会影响
            dist[v] = max(dist[v], dist[u] + W[v]);
            if(--ind[v] == 0)
                q.push(v);
        }
    }

    int ans = 0;
    for (int i = 1; i <= scc_cnt; i++)
        ans = max(ans, dist[i]);
    return ans;
}


int main() {
    ios::sync_with_stdio(false), cin.tie(0);
    cin >> n >> m;
    for (int i = 1; i <= n; i ++)    cin >> w[i];
    while(m--) {
        int u, v;
        cin >> u >> v;
        G[u].pb(v);
    }
    for (int i = 1; i <= n; i ++)
        if(!dfn[i])
            tarjan(i);
    //缩点
    for (int u = 1; u <= n; ++u) {
        for(int v : G[u]) {
            if(id[v] != id[u]) {
                G2[id[u]].pb(id[v]);
                ind[id[v]]++;
                //printf("ind[%d] = %d\n",id[v], ind[id[v]]);
            }
        }
    }
    // debug
    // for (int i = 1; i <= scc_cnt; i++)
    //     printf("ind[%d] = %d\n",i, ind[i]);
    // for (int i = 1; i <= scc_cnt; i++)
    // {

    //     printf("%d->", i);
```

```
    //     for (int v:G2[i])
    //         printf("%d ", v);
    //     puts("");
    // }
    printf("%d\n", sol());
    return 0;
}
```

3.7 无向图的双连通分量

桥：

```
// 一个有桥的连通图，如何把它通过加边变成边双连通图?
// 1. 求出所有的桥，然后删除这些桥边，剩下的每个连通块都是一个双连通子图。
// 把每个双连通子图收缩为一个顶点 ，再把桥边加回来，最后的这图一定是一棵树，边连通度为 1。
// 2 统计出树中度为 1 的节点的个数，即为叶节点的个数，记为 cnt。
// 3. 则至少在树上添加 (cnt+1)/2 条边，就能 使树达到边二连通，所以至少添加的边数就是 (cnt+1)/2。
#include <bits/stdc++.h>
#define pb push_back
using namespace std;
const int N = 5010;
int n, m;
vector<int> G[N];
int low[N], dfn[N], id[N], deg[N];
int dcc_cnt, tim, stk[N], top;
vector<int> bridge[N];

void tarjan(int u, int fa) {
    low[u] = dfn[u] = ++tim;
    stk[++top] = u;

    for (int i = 0; i < G[u].size(); i++) {
        int v = G[u][i];
        if(!dfn[v]) {
            tarjan(v, u);
            low[u] = min(low[v], low[u]);
            if(dfn[u] < low[v])
                bridge[u].pb(v), bridge[v].pb(u);
        }
        else if(fa != v)
            low[u] = min(low[u], dfn[v]);
    }
    if (dfn[u] == low[u]) {
        int y;
        ++dcc_cnt;
        do {
            y = stk[top--];
            id[y] = dcc_cnt;
        } while (u != y);
    }
```

```
}

int main() {
    ios::sync_with_stdio(false), cin.tie(0);
    cin >> n >> m;
    while(m -- ) {
        int u, v;
        cin >> u >> v;
        G[u].pb(v), G[v].pb(u);
    }
    tarjan(1, -1);
    for (int u = 1; u <= n; u++)
        deg[id[u]] += bridge[u].size();
    int cnt = 0;
    for (int i = 1; i <= dcc_cnt; i ++)
        if(deg[i] == 1)
            cnt++;
    cout << (cnt + 1) / 2;
    return 0;
}
```

割点:

```
#include<bits/stdc++.h>
#define pb push_back
using namespace std;
const int N = 2e4 + 10;
vector<int> G[N], cut;
int tim, n, m, root;
int dfn[N], low[N];

void tarjan(int u) {
    low[u] = dfn[u] = ++tim;
    int tot = 0;
    for(int v:G[u]) {
        if(!dfn[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
            if (dfn[u] <= low[v])
                tot++;
        }
        else
            low[u] = min(low[u], dfn[v]);
    }
    if ( (tot > 0 && u != root) || (tot > 1 && u == root))
        cut.pb(u);

}

int main() {
    cin >> n >> m;
```

```
    while(m --) {
        int u, v;
        cin >> u >> v;
        G[u].pb(v), G[v].pb(u);
    }

    for (root = 1; root <= n; root ++)
        if(!dfn[root])
            tarjan(root);

    //不用 sort，就开一个 bool cut[N];
    sort(cut.begin(), cut.end());
    printf("%d\n", cut.size());
    for(int v : cut)
        printf("%d ", v);

    return 0;
}
```

3.8 lca

```
/*
求 lca： 1. 倍增  2. 树剖  3.tarjan 离线

lca 用处
1. 树上两点之间的距离 （多维护一个 dist 数组， dis[u] + dis[v] - 2 * dis[lca(u, v)]）
2. 树上两条路径是否相交 （如果两条路径相交，那么一定有一条路径的 LCA 在另一条路径上）
*/
//acwing1171 树上距离
#include <bits/stdc++.h>
#define pb push_back
#define endl '\n'
using namespace std;
const int N = 1e4 + 10;

struct node{int v, w;};
vector<node> G[N];
int fa[N][19], dep[N], dis[N];
int n, m;

void bfs(int s) {
    memset(dep, 0x3f, sizeof dep);
    dep[0] = 0, dep[s] = 1;
    dis[s] = 0;
    queue<int> q; q.push(s);
    while(q.size()) {
        int u = q.front(); q.pop();
        for(auto [v, w] : G[u]) {
            if(dep[v] > dep[u] + 1) {
```

```cpp
                    dis[v] = dis[u] + w;
                    dep[v] = dep[u] + 1;
                    fa[v][0] = u;
                    q.push(v);
                    for(int i = 1; i < 19; ++i)
                        fa[v][i] = fa[fa[v][i - 1]][i - 1];
                }
            }
        }
}

int lca(int a, int b) {
    if(dep[a] < dep[b]) swap(a, b);
    for(int k = 18; k >= 0; k--)
        if(dep[fa[a][k]] >= dep[b])
            a = fa[a][k];
    if(a == b) return a;

    for(int k = 18; k >= 0; --k)
        if(fa[a][k] != fa[b][k])
            a = fa[a][k], b = fa[b][k];
    return fa[a][0];
}

int main() {
    ios::sync_with_stdio(false), cin.tie(0);
    cin >> n >> m;
    for(int i = 1; i < n ; i ++) {
        int u, v, w;  cin >> u >> v >> w;
        G[u].pb({v, w}), G[v].pb({u, w});
    }
    bfs(1);
    while(m -- ) {
        int u, v; cin >> u >> v;
        int anc = lca(u, v);
        cout << dis[u] + dis[v] - 2 * dis[anc] << endl;
    }
    return 0;
}
```

3.9 基环树

基环树的性质：点数等于边数；度数是点数两倍。一般题目中出现“从一个点到另一个点建一条边”，“N 个点通过恰好 N 条双向道路连接起来，不存在任何两条道路连接了相同的两个点”等类似信息可以判定该图是基环树森林。以下是求基环树（森林）直径（和）代码

```cpp
//基环树森林求直径和最大
#include <bits/stdc++.h>
#define endl '\n'
#define pb push_back
```

```cpp
using ll = long long;
using namespace std;
const int N = 1e6 + 10, M = N << 1;
int h[N], e[M], w[M], ne[M], idx;
ll s[N], sum[M], d[M]; //环上的前缀和数组, 破环成链后两倍的前缀和
bool ins[N], vis[N];
int n, cir[M], ed[M], cnt; //cnt 环的个数
int fa[N], fw[N]; //父节点, 反向权值
int q[M];
ll ans;

void add(int a, int b, int c) {
    e[idx] = b, ne[idx] = h[a], w[idx] = c, h[a] = idx++;
}

//深搜 + 栈 找环
void dfs(int u, int from) {
    vis[u] = ins[u] = true;
    for (int i = h[u]; ~i; i = ne[i]) {
        //如果是反向边则跳过, 必须用边来判断, 这样才能确定是通过反向变回到父节点
        if (i == (from ^ 1)) continue;
        int v = e[i];
        fa[v] = u, fw[v] = w[i];
        if (!vis[v]) dfs(v, i);
        else  if(ins[v]) {
            cnt++;
            ed[cnt] = ed[cnt - 1];
            ll tot = w[i];
            for (int k = u; k != v; k = fa[k])  {
                s[k] = tot;
                tot += fw[k];
                cir[++ ed[cnt]] = k;
            }
            s[v] = tot, cir[++ ed[cnt]] = v;
        }
    }
    ins[u] = false;
}

// 求以 u 为根节点的子树的最大深度
ll dfs_d(int u)  {
    vis[u] = true;
    ll d0 = 0, d1 = 0; //最大距离, 次大距离
    for (int i = h[u]; ~i; i = ne[i]) {
        int v = e[i];
        if (vis[v]) continue;
        ll d = dfs_d(v) + w[i];
        if (d >= d0) d1 = d0, d0 = d;
        else if (d > d1) d1 = d;
```

```
    }
    ans = max(ans, d1 + d0);
    return d0;
}

int main() {
    ios::sync_with_stdio(false), cin.tie(0);
    cin >> n;
    memset(h, -1, sizeof h);
    for (int u = 1; u <= n; u ++) {
        int v; ll w; cin >> v >> w;
        add(u, v, w), add(v, u, w);
    }

    for (int i = 1; i <= n; i ++)
        if(!vis[i])
            dfs(i, -1);

    memset(vis, 0, sizeof vis);
    for (int i = 1; i <= n; i ++) vis[cir[i]] = 1; //标记环上所有点

    ll res = 0;
    for (int i = 1; i <= cnt; i ++) {
        ans = 0; // 当前基环树的直径
        int sz = 0; // 当前基环树的环的大小
        for (int j = ed[i - 1] + 1; j <= ed[i]; j ++) {
            int k = cir[j];
            d[sz] = dfs_d(k); // 求以当前点为根的子树的最大深度
            sum[sz] = s[k];
            sz++;
        }
        // 破环成链，前缀和数组和 d[] 数组延长一倍
        for (int j = 0; j < sz; j ++)
            d[sz + j] = d[j], sum[sz + j] = sum[j] + sum[sz - 1];

        // 做一遍滑动窗口，比较依据是 d[k] - sum[k]
        int hh = 0, tt = -1;
        for (int j = 0; j < sz * 2; j++) {
            while (hh <= tt && q[hh] <= j - sz) hh++;
            if (hh <= tt) ans = max(ans, d[j] + sum[j] + d[q[hh]] - sum[q[hh]]);
            while (hh <= tt && d[j] - sum[j] >= d[q[tt]] - sum[q[tt]]) tt--;
            q[ ++ tt] = j;
        }
        res += ans;
    }
    cout << res << endl;
    return 0;
}
```

3.10 dinic

```
#include <bits/stdc++.h>
#define pb push_back
using namespace std;
using ll = long long;
const int N = 1e4 + 10;
const ll inf = 0x3f3f3f3f3f3f3f3f;
int n, m, s, t, dep[N];
struct node {int v, cap, rec;};
vector<node> G[N];

bool bfs() {
    queue<int> q;
    q.push(s);
    memset(dep, -1, sizeof dep);
    dep[s] = 0;
    while (q.size()) {
        int u = q.front(); q.pop();
        for(auto [v, cap, rev] : G[u])
            if(dep[v] == -1 && cap)
                dep[v] = dep[u] + 1, q.push(v);
    }
    return dep[t] != -1;
}

ll dfs(int u, ll lim) {
    if(u == t || lim == 0) return lim;
    ll tot_flow = 0;
    for(auto& [v, cap, rev] : G[u]) {
        if(dep[v] == dep[u] + 1 && cap > 0) {
            ll d = dfs(v, min(lim, (ll)cap));
            cap -= d, G[v][rev].cap += d;
            lim -= d, tot_flow += d;
            if(lim == 0)  return tot_flow;
        }
    }
    if(lim != 0) dep[u] = -1;
    return tot_flow;
}

ll dinic() {
    ll max_flow = 0;
    while(bfs())
        max_flow += dfs(s, inf);
    return max_flow;
}

int main() {
    scanf("%d%d%d%d", &n, &m, &s, &t);
```

```cpp
    while(m --) {
        int u, v, cap;  scanf("%d%d%d", &u, &v, &cap);
        G[u].pb({v, cap, G[v].size()});
        G[v].pb({u, 0, G[u].size() - 1});
    }
    printf("%lld\n", dinic());
    return 0;
}
```

4 动态规划

4.1 数位 dp

4.2 换根 dp

换根 dp 一般时间复杂度为 $\mathcal{O}(n)$，需要对树处理得到大规模答案，如对每个点得到一个答案。

```cpp
// 求树上 对某个点来说包含他的连通点集个数
#include <bits/stdc++.h>
#define pb push_back
#define endl '\n'
using ll = long long;
using namespace std;
const int N = 1e6 + 10, mod = 1e9 + 7;

ll f[N], ans[N], n;
vector<int> G[N];

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

void dfs(int u, int fa) {
    f[u] = 1;
    for (auto v:G[u]) {
        if(v == fa) continue;
        dfs(v, u);
        f[u] = f[u] * (f[v] + 1) % mod;
    }
}

/*
考虑换根，ans[u] 记为以 u 为根，和整棵树其他点能形成的所有子树数量。(即最终答案)
```

```
换根方程: ans[v]=( ans[u]/(f[v]+1) +1)*f[v]
解释: u 点答案除以 v 点贡献 (f[v] + 1) 为与 v 无关的 u 点答案, +1 后为其余点对 v 点贡献, 再乘上 f[v]

有一个很坑的地方, 就是 (f[v]+1) 求逆元可能得到 0 (f[v] 可能为 mod-1), 这时相当于除于 0, 出错
当逆元 inv 为 0 时, ans[u] 实际是由在树形 dp 的时候求出的 f[u], 而 f[u] 又等于 (他所有儿子 f 的值 +1) 的乘积。
所以 ans[u] / (f[v]+1) 又可以变成 u 的其他儿子的乘积: u 除 v 外的其他儿子记 brother。
(f[brother_1]+1) * (f[brother_2] + 1) * ..... 他的所有兄弟的值乘积。
*/

void dp(int u, int fa) {
    for (int v:G[u]) {
        if(v == fa) continue;
        ll inv = qpow(f[v] + 1, mod - 2);
        if(inv) ans[v] = (ans[u] * inv % mod + 1) % mod * f[v] % mod;
        else {
            ll t = 1;
            for (auto other:G[u]) {
                if(other == v || other == fa) continue;
                t = t * (f[other] + 1) % mod;
            }
            ans[v] = (t + 1) * f[v] % mod;
        }
        dp(v, u);
    }
}

int main() {
    cin >> n;
    for (int i = 1; i < n; i ++) {
        int u, v; cin >> u >> v;
        G[u].pb(v), G[v].pb(u);
    }
    dfs(1, 0);
    ans[1] = f[1];
    dp(1, 0);

    for (int i = 1; i <= n; i ++) cout << ans[i] << endl;
    return 0;
}
```

5 字符串

5.1 字符串 Hash

```
#include <bits/stdc++.h>
using namespace std;
struct Hash {
    using ull = unsigned long long;
    const int base = 131;
```

```cpp
    int siz;
    vector<ull> pow_base, hash_val; // or p, h due to time budget
    Hash() { }
    Hash(const string &s) {
        siz = s.size();
        pow_base.resize(siz);
        hash_val.resize(siz);
        pow_base[0] = 1;
        hash_val[0] = s[0];
        for (int i = 1; i < siz; i++){
            pow_base[i] = pow_base[i - 1] * base;
            hash_val[i] = hash_val[i - 1] * base + s[i];
        }
    }
    // 下标 0 开始，闭区间
    ull operator[](const array<int, 2>& range) const {
        // if(r < l || l > n) return 0; //根据题目需要处理边界情况
        auto l = range[0], r = range[1];
        if(l == 0) return hash_val[r];
        return hash_val[r] - hash_val[l - 1] * pow_base[r - l + 1];
    }

    ull get(int l, int r) {
        return this->operator[]({l, r});
    }
};

struct doubleHash {
    using ll = long long;
    int size;
    array<int, 2> mod = {2000000011, 2000000033}, base = {20011, 20033};
    vector<array<ll, 2>> hash, pow_base;
    doubleHash() { }
    doubleHash(const string& s) {
        size = s.size();
        hash.resize(size);
        pow_base.resize(size);
        pow_base[0][0] = pow_base[0][1] = 1;
        hash[0][0] = hash[0][1] = s[0];
        for(int i = 1; i < size; i++){
            hash[i][0] = (hash[i - 1][0] * base[0] + s[i]) % mod[0];
            hash[i][1] = (hash[i - 1][1] * base[1] + s[i]) % mod[1];
            pow_base[i][0] = pow_base[i - 1][0] * base[0] % mod[0];
            pow_base[i][1] = pow_base[i - 1][1] * base[1] % mod[1];
        }
    }
    array<ll, 2> operator[](const array<int, 2>& range) const {
        auto l = range[0], r = range[1];
        if(l == 0) return hash[r];
```

```
        return {
            (hash[r][0] - hash[l - 1][0] * pow_base[r - l + 1][0] % mod[0] + mod[0]) %
            ↪ mod[0],
            (hash[r][1] - hash[l - 1][1] * pow_base[r - l + 1][1] % mod[1] + mod[1]) %
            ↪ mod[1]};
    }
    //double hash to A hash_val
    ll get(int l, int r) {
        auto h = this->operator[]({l, r});
        return h[0] * 1000000000ll + h[1];
    }
};
int main() {}
```

5.2 Trie

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

const int N = 1e5 + 10;
char str[N];
int son[N][26], cnt[N], idx;

void insert(char *str) {
    int p = 0;
    for (int i = 0; str[i]; i ++) {
        int u = str[i] - 'a';
        if(!son[p][u])  son[p][u] = ++idx;
        p = son[p][u];
    }
    ++cnt[p];
}

int query(char *str) {
    int p = 0;
    for (int i = 0; str[i]; ++i) {
        int u = str[i] - 'a';
        if(!son[p][u]) return 0;
        p = son[p][u];
    }
    return cnt[p];
}
```

5.3 KMP

```
//poj2406
#include <bits/stdc++.h>
using namespace std;
const int N = 1e6 + 10;
```

```
char s[N];
int nxt[N], n;
/*
//区间 l->r 的 kmp
    nxt[l] = 0;
    for (int i = l + 1; i <= r; i ++) {
        int j = nxt[i - 1];
        while(j && s[i] != s[l + j])   j = nxt[l + j - 1];
        if(s[i] == s[j + l])  j++;
        nxt[i] = j;
    }
*/
void get_nxt() {
    nxt[1] = 0;
    for (int i = 2, j = 0; i <= n; i ++) {
        while(j && s[i] != s[j + 1]) j = nxt[j];
        if(s[i] == s[j + 1]) j++;
        nxt[i] = j;
    }
}

int main() {
    while(~scanf("%s", s + 1)) {
        if(s[1] == '.')  break;
        n = strlen(s + 1);
        get_nxt();
        int period = n - nxt[n];
        if(n % period == 0) printf("%d\n", n / period);
        else  puts("1");
    }
    return 0;
}
```

5.4 Z-algorithm

- 给出字符串 a, b，求 a 的每个后缀与 b 的 LCP：
  设 \$ 为字符集外字符，求 $b+\$+a$ 的 Z 函数，则 a 的后缀 $a[i..]$ 与 b 的 LCP 为 $Z(|b|+1+i)$ 。

- 求 s 的每个前缀的出现次数：
  求 s 的 Z 函数。对于每一个 i ，如果 $Z(i)$ 不等于 0，说明长度为 $Z(i), Z(i)-1, \cdots, 1$ 的前缀在此处各出现了一次，所以求一个后缀和即可。在这个问题中一般令 $Z(0)=|s|$。

```
for (int i = n + 1; i < 2 * n + 1; ++i)
    S[z[i]]++;
for (int i = n; i >= 1; --i)
    S[i] += S[i + 1];
```

- 求 s 的所有 border：

KMP 就可以，也可以用 Z 算法。求 s 的 Z 函数。对于每一个 i，如果 $i+Z(i)=|s|$ ，说明这个 Z-Box 对应一个 border。（注：与 KMP 不同，这里只是求所有 border，不是求所有前缀的 border）

```
//给定两个字符串 a,b,
// 要求出两个数组：b 的 z 函数数组 z、
// b 与 a 的每一个后缀的 LCP 长度数组 p。
#include <bits/stdc++.h>
#define rep(i, a, b) for (int i = (a); i < (b); i ++)
#define sz(a) int((a).size())
using namespace std;
using ll = long long;
const int N = 2e7;
ll ansz, ansp;
string a, b;

// Zfunction
int z[N << 1];
void getz(string s) {
    int l = 0;
    for (int i = 1; i <= s.size(); i ++) {
        if(l + z[l] > i) z[i] = min(z[i - l], l + z[l] - i);
        while(i + z[i] < s.size() && s[z[i]] == s[i + z[i]])  z[i]++;
        if(i + z[i] > l + z[l]) l = i;
    }
    // rep(i,0,s.size()) cout<<z[i]<<" ";cout<<'\n';
}


int main(){
    ios::sync_with_stdio(0);
    cin.tie(0),cout.tie(0);
    cin >> a >> b, getz(b + a);
    ansz ^= 1ll * (sz(b)+1)*(0+1);
    rep(i,1,sz(b)) ansz^=1ll*(min(z[i],sz(b)-i)+1)*(i+1);
    rep(i,0,sz(a)) ansp^=1ll*(min(z[i+sz(b)],sz(b))+1)*(i+1);
    cout << ansz << '\n'
         << ansp << '\n';
    return 0;
}
```

5.5 AC 自动机

```
//luogu3808
#include <bits/stdc++.h>
using namespace std;

const int N = 1e6 + 10;
int n;
char s[N];
```

```cpp

namespace ac
{

int tr[N][26], fail[N], idx;
queue<int> q;
int cnt[N];

void insert(char* s) {
    int p = 0;
    for (int i = 1; s[i]; ++i) {
        int u = s[i] - 'a';
        if(!tr[p][u])  tr[p][u] = ++idx;
        p = tr[p][u];
    }
    ++cnt[p];
}

void build() {
    for (int i = 0; i < 26; ++i)
        if(tr[0][i]) q.push(tr[0][i]);

    while(q.size()) {
        int u = q.front(); q.pop();
        for (int i = 0; i < 26;  i++) {
            if(tr[u][i])
                fail[tr[u][i]] = tr[fail[u]][i], q.push(tr[u][i]); //原本这个 tr[fail[u]][i] 可能不存在（为 0）
                                                                   // 但是下一步 else 做了一个优化（类似
            else
                tr[u][i] = tr[fail[u]][i];
        }
    }
}

int query(char *s) {
    int u = 0, res = 0;
    for (int i = 1; s[i]; ++i) {
        u = tr[u][s[i] - 'a'];
        for (int j = u; j && cnt[j] != -1; j = fail[j])
            res += cnt[j], cnt[j] = -1;
    }
    return res;
}

}

int main() {
```

```
    scanf("%d", &n);
    for (int i = 1; i <= n; i ++) {
        scanf("%s", s + 1);
        ac::insert(s);
    }
    ac::build();
    scanf("%s", s + 1);
    printf("%d\n", ac::query(s));
    return 0;
}
```

5.6 SA

$lcp(i,j)$ 表示后缀 i,j 的最长公共前缀（的长度）

$height$ 数组定义：$ht[i] = lcp(sa[i], sa[i-1])$

性质：$lcp(sa[i], sa[j]) = min\{ht[i+1..j]\}$

由此，求两子串（排名为 i,j）最长公共前缀就转化为了 RMQ 问题（求 $ht[i+1]$ 到 $ht[j]$ 的最小值）。

本质不同的子串：$\frac{n*(n+1)}{2} - \sum_{i=2}^{n} ht[i]$

ht 数组连续一段不小于 h 的区间长度代表长 h 的这个子串的出现次数

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

class SuffixArray {
//得到的 sa[], rk[] 下标从 0 开始，ht 下标从 1 开始（因为是长度）
private:
    int n, m;
    vector<int> x, y, cnt;
    void radixSort() {
        for (int i = 0; i < m; ++ i) cnt[i] = 0;
        for (int i = 0; i < n; ++ i) cnt[x[i]] ++;
        for (int i = 1; i < m; ++ i) cnt[i] += cnt[i - 1];
        for (int i = n - 1; i >= 0; -- i) sa[-- cnt[x[y[i]]]] = y[i];
    }
public:
    vector<int> sa, rk, ht;

    SuffixArray(const string &s) :
        n(s.size()), m(256), //m 为字符集最大数量
        x(n), y(n), cnt(max(n, m)),
        sa(n), rk(n), ht(n) {
        init_sa(s);
        init_ht(s);
    }
    void init_sa(const string &s) {
        for (int i = 0; i < n; ++ i) {
            x[i] = s[i];
            y[i] = i;
```

```
        }
        radixSort();
        for (int w = 1; w <= n; w <<= 1) {
            int p = 0;
            for (int i = n - w; i < n; ++ i) y[p ++] = i;
            for (int i = 0; i < n; ++ i)
                if (sa[i] >= w) y[p ++] = sa[i] - w;

            radixSort();
            swap(x, y);
            x[sa[0]] = 0;
            p = 1;
            auto cmp = [&](int i, int j) {
                if (i < n && j < n) return y[i] == y[j];
                return i >= n && j >= n;
            };
            for (int i = 1; i < n; ++ i)
                x[sa[i]] = (cmp(sa[i], sa[i - 1]) && cmp(sa[i - 1] + w, sa[i] + w))
                ? p - 1 : p++;

            if (p >= n) break;
            m = p;
        }
        for (int i = 0; i < n; ++ i) rk[sa[i]] = i;
    }
    void init_ht(const string &s) {
        for (int i = 0, k = 0; i < n; ++ i) {
            if (rk[i] == 0) continue ;
            if (k) k --;
            int j = sa[rk[i] - 1];
            while (i + k < n && j + k < n && s[i + k] == s[j + k]) k ++;
            ht[rk[i]] = k;
        }
    }
};

void solve() {
    string s;
    cin >> s;
    SuffixArray f(s);
    for (int i = 0; i < s.size(); ++ i)
        cout << f.sa[i] + 1 << " \n"[i + 1 == s.size()];
    for (int i = 0; i < s.size(); ++ i)
        cout << f.ht[i] << " \n"[i + 1 == s.size()];
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
```

```cpp
    int T; cin >> T;
    while (T -- ) {
        solve();
    }
    return 0;
}
```

5.7 Manacher

用 $Manacher+hash$ 可以求出所有本质不同的回文子串 (存 hash 值)，时间复杂度 $\mathcal{O}(|s|)$。但是不用于求每个本质不同回文子串出现次数相关统计，因为统计出现次数时，$while(l <= r)$ 中不可以 $break$，复杂度 n^2

```cpp
auto p = manacher(s);
Hash hs(s); //or doubleHash
set<ull> res; // ll when doubleHash
for (int mid = 1; mid < p.size() - 1; mid ++) {
    //枚举回文子串的左右端点
    int l = (mid - p[mid] + 1) / 2, r = (mid + p[mid] - 1) / 2;
    while(l <= r) {
        if(res.count(hash.get(l, r))) break;
        res.insert(hash.get(l++, r--));
    }
}
```

```cpp
#include<bits/stdc++.h>
using namespace std;
// return p, p[i] 表示修改后的串中以 i 为中心的最长回文半径
vector<int> manacher(const string& _s) {
    vector<int> p(_s.size() * 2 + 1);
    string s(_s.size() * 2 + 1, '$');
    for (int i = 0; i < _s.size(); i++) s[2 * i + 1] = _s[i];
    for(int i = 0, maxr = 0, mid = 0; i < s.size(); i++) {
        if(i < maxr) p[i] = min(p[mid * 2 - i], maxr - i);
        while(i - p[i] - 1 >= 0 && i + p[i] + 1 < s.size()
            && s[i - p[i] - 1] == s[i + p[i] + 1])
            ++p[i];
        if(i + p[i] > maxr) maxr = i + p[i], mid = i;
    }
    return p;
}


int main() {
    string s;
    cin >> s;
    auto p = manacher(s);
    // for (int i = 0; i < p.size(); i ++) {
```

```
    //     cout << p[i] << " \n"[i == p.size() - 1];
    // }
    cout << (*max_element(p.begin(), p.end())) << endl;
}
```

6 其他

6.1 glibc 内置函数

```
// Returns the number of 1-bits in x.
int __builtin_popcount(unsigned int x);
int __builtin_popcountll(unsigned long long x);

// Returns the number of trailing 0 (undefined when x == 0)
int __builtin_ctz(unsigned int x);
int __builtin_ctzll(unsigned long long x);

// Returns log_2(x)
int __lg(int x);

int __gcd(int x, int y);
```

6.2 __int128 读写

```
inline __int128 read(){
    __int128 x = 0, f = 1;
    char ch = getchar();
    while (ch<'0' || ch>'9') { if(ch == '-') f = -1; ch = getchar();}
    while (ch >= '0' && ch <= '9') { x = x * 10 + ch - '0'; ch = getchar(); }
    return x * f;
}

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

6.3 整数二分

```
// 区间 [l, r] 被划分成 [l, ans] 和 [ans + 1, r] 时使用:
int bsearch_1(int l, int r) {
    while (l < r) {
        int mid = l + r >> 1;
        if (check(mid)) r = mid;    // check() 判断 mid 是否满足性质
        else l = mid + 1;
    }
    return l;
}
```

```cpp
// 区间 [l, r] 被划分成 [l, ans - 1] 和 [ans, r] 时使用：
int bsearch_2(int l, int r) {
    while (l < r) {
        int mid = l + r + 1 >> 1;
        if (check(mid)) l = mid;
        else r = mid - 1;
    }
    return l;
}
```

6.4 单调栈

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1000100;
//单调栈，记录每个数左边比他小（大）的第一个数（也可以记录其下标）
int stk[N], tt, a[N];

int main() {
    ios::sync_with_stdio(false), cin.tie(0), cout.tie(0);
    int n;  cin >> n;
    for (int i = 1; i <= n; i ++)  cin >> a[i];

    for (int i = 1; i <= n; i++) {
        while(tt && stk[tt] >= a[i]) tt--;
        if(tt) cout << stk[tt] << ' ';
        else cout << -1 << ' ';
        stk[++tt] = a[i];
    }
    return 0;
}
```

6.5 单调队列

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N = 1e6 + 10;
int a[N], q[N],n, k;
//滑动窗口
int main() {
    cin >> n >> k;
    for(int i = 0; i < n; i++)   cin >> a[i];
    int hh = 0, tt = -1;
    for(int i = 0; i < n; i++) {
        //判断队头是否已经划出窗口
        if( hh <= tt && i - k + 1  > q[hh]) hh++;
        while(hh <= tt && /* 后面改成要维护的最小值 */a[q[tt]] >= a[i]) tt -- ;//求区间最小
        q[ ++ tt ] = i;
        if(i >= k-1) printf("%d ",a[q[hh]]);
```

```

    }
    return 0;
}
```

6.6 GospersHack

生成 n 元集合所有 k 元子集的算法。这个算法复杂度与答案个数是同阶的，比暴力枚举 2^n 个数然后分别算 $popcount$ 要好。

```cpp
void GospersHack(int k, int n) {
    int cur = (1 << k) - 1;
    int limit = (1 << n);
    while (cur < limit) {
        // do something
        int lb = cur & -cur;
        int r = cur + lb;
        cur = ((r ^ cur) >> __builtin_ctz(lb) + 2) | r;
        // 或: cur = (((r ^ cur) >> 2) / lb) | r;
    }
}
```

6.7 C++17-STL

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

/*
lambda expression
*/

/* structure binding */
{
    // Graph with weighted edges
    vector<vector<pair<int, int>>> G(n);
    // somewhere in dfs,
    for (auto [v, w] : G[u]) {
        ...
    }
}

/* discrete manipulate*/
{
    // vector<int> v. vv = v;
    sort(v.begin(), v.end());
    v.erase(unique(v.begin(), v.end()), v.end());

}

```

```
/*
some useful function in STL
*/
{
    int n = 256;
    vector<int> v(n);
    // generate v as 0, 1, 2, ...
    // often used when you initialize DSU or generate a permutation
    iota(v.begin(), v.end(), 0);

    ll res = accumulate(v.begin(), v.end(), 1LL,
                        [](int a, int b){return (ll)a*b;});

    gcd() and lcm(); // not __gcd() version

    // max_element and min_element which can be used with user-defined op
    cout << *max_element(v.begin(), v.end() /*[]()->bool{}*/);

    //std::clamp
    // Returns x if it is in the interval [low, high] or,
    // otherwise, the nearest value.No more max of min of max of...
    cout << clamp(7, 0,  10); //7
    cout << clamp(7, 0,  5);  //5
    cout << clamp(7, 10, 50); //10

    /*don't do*/ max(ans, max(t1, t2));
    /*just*/ max({ans, t1, t2}); // initalizer list


    /* partial_sum and adjacent_difference 前缀和与差分 */
    vector<int> a(n, 2), b(n, 2);
    partial_sum(a.begin(), a.end(), a.begin());
    partial_sum(a.begin(), a.end(), b.begin());

    adjacent_difference(v.begin(), v.end(), v.begin());
}

/*
template< class R, class... Args >
class function<R(Args...)>;
*/
{
    // a callable obj
    function<int(int, int)> dfs = [&](int u, int fa) {
        ...
    };
}

/* Set operations (on sorted ranges)*/
```

```
{
    set<int> s1, s2;
    set<int> ans;
    set_intersection(s1.begin(), s1.end(),
                      s2.begin(), s2.end(),
                      std::inserter(ans, ans.begin()));
                      // 若为 vector，可以用 back_inserter;
}

/*
Initializer in if and switch:
*/
{
    set<int> s;
    if (auto [iter, ok] = s.insert(42); ok) {
        //...
    }
    else {
        //`ok` and `iter` are available here
    }
    //But not here
}

/*
about string
*/
{
    substr(npos, count);
}

int main() {
}
```