

icpc 算法模板

Catch-22

2022 年 8 月 29 日

目录

1	数学	4
1.1	求逆元	4
1.2	扩展欧几里德算法	4
1.3	筛法	5
1.4	组合数	7
1.5	容斥原理	9
1.6	数论分块	9
1.7	Möbius 反演	10
1.8	高斯消元	10
1.9	Miller Rabin 素数测试	12
1.10	FFT	13
2	数据结构	15
2.1	(带权) 并查集	15
2.2	Sparse Table	15
2.3	01Trie	16
2.4	树状数组	16
2.5	线段树	18
2.6	可持久化线段树	22
2.7	线段树合并	23
2.8	树链剖分	24
2.9	左偏树	27
2.10	莫队	28
3	图论	31
3.1	spfa	31
3.2	dijkstra	32
3.3	最小生成树	34
3.4	kruskal 重构树	35
3.5	二分图匹配	37
3.6	强连通分量缩点	38
3.7	无向图的双连通分量	40
3.8	lca	42
3.9	基环树	44
3.10	dinic	46
4	动态规划	47
4.1	数位 dp	47
4.2	换根 dp	47

5	字符串	49
5.1	字符串 Hash	49
5.2	Trie	50
5.3	KMP	51
5.4	Z-algorithm	52
5.5	AC 自动机	53
5.6	SA	54
5.7	Manacher	56
6	其他	57
6.1	glibc 内置函数	57
6.2	__int128 读写	57
6.3	整数二分	58
6.4	单调栈	58
6.5	单调队列	59
6.6	GospersHack	59
6.7	C++17-STL	59

1 数学

1.1 求逆元

注意考虑 x 是 mod 倍数的情况

```

1  ll qpow(ll a, ll b) {
2      ll res = 1;
3      while(b) {
4          if(b & 1) res = res * a % mod;
5          a = a * a % mod;
6          b >>= 1;
7      }
8      return res;
9  }
10
11 ll inv(ll x) { return qpow(x, mod - 2); }
12
13 const int N = 1e6 + 10;
14 // 线性递推求逆元 [1, n] 的所有数关于 p 的逆元
15 int inv[N];
16 void init_inv () {
17     int n, p;
18     cin >> n >> p;
19     inv[0] = 0, inv[1] = 1;
20     for (int i = 2; i <= n; i++)
21         inv[i] = (ll)(p - p / i) * inv[p % i] % p; // 为了保证大于零加了个 p
22
23     return 0;
24 }
```

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： $\mathcal{O}(n \log n)$

```

1  // 拓欧解线性同余方程  $a \cdot x \equiv b \pmod m$ 
2  #include <bits/stdc++.h>
3  using namespace std;
4  using ll = long long;
5  int a, b, m, n;
6
7  int exgcd(int a, int b, int &x, int &y) {
8      if(b == 0) {
9          x = 1, y = 0;
10         return a;
11     }
12     int d = exgcd(b, a % b, y, x);
```

```

13     y -= a/b * x;
14     return d;
15 }
16
17 int main() {
18     int x, y;
19     cin >> n;
20     while(n -- ) {
21         cin >> a >> b >> m;
22         int d = exgcd(a, m, x, y); // d = gcd(a, m)
23         if(b % d != 0) puts("impossible"); //bezout 定理: 有解的条件, gcd(a, m) | b
24         else printf("%lld\n", (1ll)x * (b/d) % m);
25     }
26     return 0;
27 }

```

1.3 BSGS

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

```

1 // find smallest non-negative x s.t. a^x = b mod p, or -1 (assume 0^0 = 1)
2 int babyStepGiantStep(int a, int b, int p) {
3     a %= p; b %= p;
4     if (p == 1 || b == 1) return 0;
5     int cnt = 0, t = 1;
6     for (int g = __gcd(a, p); g != 1; g = __gcd(a, p)) {
7         if (b % g) return -1;
8         p /= g;
9         ++cnt;
10        b /= g;
11        t = 1LL * t * (a / g) % p;
12        if (b == t) return cnt;
13    }
14    std::map<int, int> mp;
15    int m = ceil(sqrt(p));
16    int base = b;
17    for (int i = 0; i != m; ++i) {
18        mp[base] = i;
19        base = 1LL * base * a % p;
20    }
21    base = powMod(a, m, p);
22    for (int i = 1; i <= m; ++i) {
23        t = 1LL * t * base % p;
24        if (mp.count(t))
25            return (1LL * i * m - mp[t]) % p + cnt;
26    }
27    return -1;
28 }
29 // https://www.luogu.com.cn/problem/P4195

```

1.4 筛法

筛质数

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4  const int N = 1e7 + 10;
5  // minp[i] 为 i 的最小素因子 http://oj.daimayuan.top/course/10/problem/733
6  int primes[N], pcnt, minp[N]; // 可用于 Log 级别分解质因数
7  bool vis[N]; //合数 true
8  int n, q;
9  //Linear
10 void get_prime(int n) {
11     for(int i = 2; i <= n; i++) {
12         if(!vis[i]) primes[ ++ pcnt] = i, minp[i] = i;
13         for(int j = 1; j <= pcnt && i * primes[j] <= n; ++ j) {
14             vis[i * primes[j]] = 1;
15             minp[primes[j] * i] = primes[j];
16             if(i % primes[j] == 0) break;
17         }
18     }
19 }
20
21 //about Linear :O(nloglogn)
22 bool isprime[N];
23 inline void getprime(int n) {
24     for (int i = 2; i <= n; i++) isprime[i] = 1;
25     for (int i = 2; i <= n; i++) {
26         if(isprime[i]) {
27             primes[++pcnt] = i;
28             if((ll)i*i<=n)
29                 for (int j = i * i; j <= n; j+=i){
30                     isprime[j] = 0;
31                 }
32         }
33     }
34 }

```

筛欧拉函数

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  /*phi compute
5  根据给定 n 计算 phi(n) O(aqrt(n))
6  核心公式 phi(n) = n*(1-1/p1)*(1 - 1/p2)*...
7  */
8  int get_phi(int n) {
9      int res = n;
10     for (int i = 2; i <= n / i; i++) {

```

```

11         if(n % i == 0) {
12             res = res / i * (i - 1); // res *= (1 - 1/n)
13             while(n % i == 0)    n /= i;
14         }
15     }
16     if(n > 1) res = res / n * (n - 1);
17     return res;
18 }
19
20 using ll = long long;
21 const int N = 1e6 + 10;
22
23 int phi[N], prime[N];
24 bool vis[N]; //合数 true
25
26 void sel_phi(int n) {
27     int cnt = 0;
28     phi[1] = 1;
29     for (int i = 2; i <= n; i++) {
30         if(!vis[i]) {
31             prime[cnt++] = i;
32             phi[i] = i - 1;
33         }
34         for (int j = 0; prime[j] <= n / i; j++) {
35             vis[prime[j] * i] = true;
36             if(i % prime[j] == 0) {
37                 phi[i * prime[j]] = phi[i] * prime[j];
38                 break;
39             }
40             else
41                 phi[prime[j] * i] = phi[i] * (prime[j] - 1);
42         }
43     }
44 }

```

筛莫比乌斯函数

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  const int N = 50010;
4  int mu[N], p[N]; // p 为素数数组
5  bool flg[N];
6  void init() {
7      int tot = 0; mu[1] = 1;
8      for (int i = 2; i < N; ++i) {
9          if (!flg[i]) {
10             p[++tot] = i;
11             mu[i] = -1;
12         }
13         for (int j = 1; j <= tot && i * p[j] < N; ++j) {
14             flg[i * p[j]] = 1;

```

```

15         if (i % p[j] == 0) {
16             mu[i * p[j]] = 0;
17             break;
18         }
19         mu[i * p[j]] = -mu[i];
20     }
21 }
22 // 常用 mu 前缀和
23 // for (int i = 1; i <= N; ++i) mu[i] += mu[i - 1];
24 }

```

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*: $C_n^m \equiv C_{n \bmod p}^{m \bmod p} * C_{n/p}^{m/p}$

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 $r \leq n$, 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:

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

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 $\frac{(\sum_{i=1}^k n_i)!}{n_1! \cdot n_2! \cdots n_k!}$

```

1 //求组合数的几种方法
2 //不确定的时候都开 long long
3 #include <bits/stdc++.h>
4 using namespace std;
5 using ll = long long;
6 const int mod = 1e9 + 7, N = 1e6 + 10;
7 //C(a, b) a 上 b 下
8
9 /*1. 依照定义 适用于 a, b 很小的时候 (几十) */
10 int C(ll a, int b) /* a 上 b 下 */{
11     if(a < b) return 0;
12     int up = 1, down = 1;
13     for (ll i = a; i > a - b; i -- ) up = i % mod * up % mod; //up *= i
14     for (int j = 1; j <= b; j ++ ) down = (ll)j * down % mod; // down *= j
15     return (ll)up * qpow(down, mod - 2) % mod; // (up/down)
16 }
17
18 /*2. 递推 杨辉三角 a, b 在 2000 这个数量级 */

```



```

19 //O(N^2) 1e6~1e7
20 void init() {
21     for (int i = 0; i < N; i++)
22         for (int j = 0; j <= i; j++)
23             if(!j) C[i][j] = 1;
24             else C[i][j] = (C[i - 1][j] + C[i - 1][j - 1]) % mod;
25 }
26
27 //最常用
28 /*3. 预处理 fac[], invfac[]*/
29 /**
30  * //调用 :
31  * 1ll * fac[b] * invfac[a] % mod * invfac[b - a] % mod;
32  */
33 // O(N) 1e6 左右 看 N 大小
34 int fac[N], invfac[N];
35 void init() {
36     fac[0] = 1;
37     for (int i = 1; i < N; i++) (ll)fac[i] = fac[i - 1]*i% mod;
38     invfac[N - 1] = qpow(fac[N - 1], mod - 2);
39     for (int i = N - 2; i >= 0; i--)
40         invfac[i] = (ll)invfac[i + 1] * (i + 1) % mod;
41 }
42
43 /*4. Lucas 定理 当 a, b 的值特别大 如 1e9 以上...1e18 等 */
44 int C(int a, int b) {
45     int res = 1;
46     for (int i = 1, j = a; i <= b; i++, j--) {
47         res = (ll)res * j % p;
48         res = (ll)res * binpow(i, p - 2) % p;
49     }
50     return res;
51 }
52
53 ll lucas(ll a, ll b) { //p 为质 (模) 数
54     if(a < p && b < p) return C(a, b);
55     return (ll)C(a % p, b % p) * lucas(a / p, b / p) % p;
56 }

```

1.6 容斥原理

S_i 为有限集, $|S|$ 为 S 的大小 (元素个数), 则:

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

```

1 // 容斥原理
2 // 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
3 // 1<=L<=R<=10^18, 1<=k<=20, 2<=ai<=100

```

```

4  #include <bits/stdc++.h>
5  using ll = long long;
6  using namespace std;
7
8  int main() {
9      ll l, r, k, f[25];
10     cin >> l >> r >> k;
11     for (int i = 0; i < k; i++) cin >> f[i];
12
13     ll ans = 0;
14
15     for (int i = 1; i < 1 << k; i++) { // 枚举集合中全部的非空子集
16         ll cnt = 0, a = r, b = l - 1; // cnt 用来表示所取的数的个数
17         for (int j = 0; j < k; j++) {
18             if(i >> j & 1) {
19                 cnt++;
20                 a /= f[j], b /= f[j];
21             }
22         }
23         if(cnt & 1) ans += (a - b);
24         else ans -= (a - b);
25     }
26     cout << ans << endl;
27     return 0;
28 }

```

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 $[l, r] = [l, \lfloor \frac{n}{\lfloor \frac{n}{l} \rfloor}]$ 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:

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

```

1  // for(int i = st; i <= ed; i++) ans += num/i
2  ll block(ll st, ll ed, ll num) {
3      //sum(num/i i in [st,ed])
4      ll L = 0, res = 0;
5      ed = min(ed, num);
6      for (ll i = st; i <= ed; i = L + 1) {
7          L = min(ed, num / (num / i)); //该区间的最后一个数
8          res += (L - i + 1) * (num / i); //区间 [i,L] 的 num/i 都是一个值
9          // res += (s(L) - s(i-1)) * (num/i); //s(i) 为 f(i) 前缀和
10     }
11     return res;
12 }

```

1.8 Möbius 反演

μ 为莫比乌斯函数, 定义为

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

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:

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

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 1。此时 $n = k$, 故原式等价于 $\sum_{k|n} [n=k] \cdot g(k) = g(n)$

形式二:

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

1.9 高斯消元

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  const int N = 110;
4  const double eps = 1e-6;
5  int n;
6  double a[N][N];
7
8  int gauss() {
9      int c, r;
10     for(c = 0, r = 0; c < n; c++) {
11         int t = r;
12         for(int i = r; i < n; i++) //找到首元素最大
13             if(fabs(a[i][c]) > fabs(a[t][c]))
14                 t = i;
15
16         if(fabs(a[t][c]) < eps) continue;
17
18         for(int i = c; i <= n; i++) swap(a[t][i], a[r][i]);

```

```

19     for(int i = n; i >= c; i --) a[r][i] /= a[r][c];
20     for(int i = r + 1; i < n; i ++)
21         if(fabs(a[i][c]) > eps)
22             for(int j = n; j >= c; j --)
23                 a[i][j] -= a[r][j] * a[i][c];
24     r ++;
25 }
26 if(r < n) {
27     for(int i = r; i < n; i ++)
28         if(fabs(a[i][n]) > eps)
29             return 2;
30     return 1;
31 }
32
33 for(int i = n - 1; i >= 0; i --)
34     for(int j = i + 1; j < n; j ++)
35         a[i][n] -= a[i][j] * a[j][n];
36
37 return 0; //有唯一解
38 }
39
40 int main() {
41     cin >> n ;
42     for(int i = 0; i < n; i ++)
43         for(int j = 0; j < n + 1; j ++)
44             cin >> a[i][j];
45
46     int t = gauss();
47     if(t == 0)
48         for(int i = 0; i < n; i ++) printf("%.2f\n", a[i][n]);
49     else if(t == 1)
50         puts("Infinite group solutions");
51     else puts("No solution");
52
53     return 0;
54 }

```

1.10 Miller Rabin 素数测试

```

1 //loj143 prime test
2 #include <bits/stdc++.h>
3 using namespace std;
4 using ull = unsigned long long;
5 using ll = long long;
6 /* O(sqrt(n))
7 bool is_prime(ll x)
8 {
9     if(x < 2) return false;
10    for(ll i = 2; i <= x / i; ++i)

```

```

11         if(x % i == 0) return false;
12     return true;
13 }
14 */
15 //常常是大素数测试, 要用到 int128
16 inline ll qmul(ll a, ll b, ll p) { return (ll)((__int128)a * b % p); }
17 ll qpow(ll a, ll b, ll p) {
18     ll res = 1;
19     while(b) {
20         if(b & 1) res = qmul(res, a, p);
21         a = qmul(a, a, p);
22         b >>= 1;
23     }
24     return res;
25 }
26 const int test_time = 8;
27
28 bool mr_test(ll n) {
29     if(n < 3 || n % 2 == 0) return n == 2;
30     ll a = n - 1, b = 0;
31     while(a % 2 == 0) a /= 2, ++b;
32
33     for (int i = 1, j; i <= test_time; ++i) {
34         ll x = rand() % (n - 2) + 2, v = qpow(x, a, n);
35         if(v == 1) continue;
36         for (j = 0; j < b; ++j) {
37             if(v == n - 1) break;
38             v = qmul(v, v, n);
39         }
40         if(j >= b) return 0;
41     }
42     return 1;
43 }
44
45 int main() {
46     srand(time(0));
47     ll x;
48     while(cin >> x) {
49         if(mr_test(x)) puts("Y");
50         else puts("N");
51     }
52     return 0;
53 }

```

1.11 FFT

```

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

```

```

4 using namespace std;
5
6 namespace FFT {
7     const double PI = acos(-1);
8     using C = complex<double>;
9     vector<int> rev;
10    vector<C> roots{C(0, 0), C(1, 0)};
11    void dft(vector<C>& a) {
12        int n = (int)a.size();
13        if ((int)rev.size() != n) {
14            int k = __builtin_ctz(n) - 1;
15            rev.resize(n);
16            for (int i = 0; i < n; ++i) {
17                rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
18            }
19        }
20        if ((int)roots.size() < n) {
21            int k = __builtin_ctz(roots.size());
22            roots.resize(n);
23            while ((1 << k) < n) {
24                C e = polar(1.0, PI / (1 << k));
25                for (int i = 1 << (k - 1); i < (1 << k); ++i) {
26                    roots[2 * i] = roots[i];
27                    roots[2 * i + 1] = roots[i] * e;
28                }
29                ++k;
30            }
31        }
32        for (int i = 0; i < n; ++i) if (rev[i] < i) {
33            swap(a[i], a[rev[i]]);
34        }
35        for (int k = 1; k < n; k *= 2) {
36            for (int i = 0; i < n; i += 2 * k) {
37                for (int j = 0; j < k; ++j) {
38                    auto u = a[i + j], v = a[i + j + k] * roots[k + j];
39                    a[i + j] = u + v;
40                    a[i + j + k] = u - v;
41                }
42            }
43        }
44    }
45    void idft(vector<C>& a) {
46        int n = (int)a.size();
47        reverse(a.begin() + 1, a.end());
48        dft(a);
49        for (auto& x : a) x /= n;
50    }
51 } // namespace FFT
52

```

```

53
54 vector<int> mul(const vector<int> &A, const vector<int> &B) {
55     int n = max(A.size(), B.size()), tot = max(1, n * 2 - 1);
56     int sz = 1 << __lg(tot * 2 - 1);
57     vector<complex<double>> C(sz);
58     for (int i = 0; i < A.size(); ++i) C[i].real(A[i]);
59     for (int i = 0; i < B.size(); ++i) C[i].imag(B[i]);
60     FFT::dft(C);
61     for (auto &x : C) x *= x;
62     FFT::idft(C);
63     vector<int> ans(A.size() + B.size() - 1);
64     for (int i = 0; i < ans.size(); ++i) ans[i] = int(C[i].imag() / 2 + 0.2);
65     return ans;
66 }
67
68 int main() {
69
70     cin.tie(nullptr)->sync_with_stdio(false);
71     int n, m;
72     cin >> n >> m;
73     vector<int> a(n + 1), b(m + 1);
74     for (auto &x : a) cin >> x;
75     for (auto &x : b) cin >> x;
76     auto c = mul(a, b);
77     for (auto &x : c) cout << x << ' ';
78     cout << '\n';
79     std::any a = 34;
80     return 0;
81
82 }

```

2 数据结构

2.1 (带权) 并查集

```

1  const int N = 1e5 + 10;
2  int fa[N], n, m, d[N];
3
4  int find(int x) {return x == fa[x] ? x : fa[x] = find(fa[x]);}
5  // 对于带权并查集，一般的 find 函数写作：
6  int find(int x) {
7      if(x == fa[x]) return x;
8      int rt = find(fa[x]); //这和下面一行顺序很重要
9      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
10     return fa[x] = rt;
11 }
12
13 void init() {
14     for (int i = 1; i <= n; i++) fa[i] = i;

```

```
15 }
```

2.2 Sparse Table

时间复杂度 $\mathcal{O}(1)$, 空间复杂度 $\mathcal{O}(n \log n)$

静态区间查询可重复贡献信息, 如“区间最值”、“区间接位和”、“区间接位或”、“区间 GCD”

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const int N = 1e5 + 10;
5  int f[N][21], n, m;
6  int a[N];
7  //f[i][j] 表示闭区间 [i, i + 2^j - 1] 的最大值
8
9  void init_st() {
10     // cout << __lg(N) << endl;
11     for (int j = 0; j < 21; j++)
12         for (int i = 1; i + (1 << j) - 1 <= n; i++) //区间长度是 2^j 所以要减一
13             if (!j) f[i][j] = a[i];
14             else
15                 f[i][j] = max(f[i][j - 1], f[i + (1 << j - 1)][j - 1]);
16 }
17
18 int query(int l, int r) {
19     int k = __lg(r - l + 1);
20     return max(f[l][k], f[r - (1 << k) + 1][k]);
21 }
```

2.3 01Trie

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  const int N = 1e5 + 10, M = N * 31;
4  int a[N];
5  int son[M][2], idx;
6
7  void insert(int x) {
8     int p = 0;
9     for (int i = 30; i >= 0; --i) {
10         int u = (x >> i) & 1;
11         if (!son[p][u]) son[p][u] = ++idx;
12         p = son[p][u];
13     }
14 }
15 // 集合内和 x 异或的最大值
16 int query(int x) {
17     int p = 0, res = 0;
18     for (int i = 30; i >= 0; --i) {
19         int u = (x >> i) & 1;
```



```

20     if(son[p][u ^ 1]) p = son[p][u ^ 1], res |= (1 << i);
21     else p = son[p][u];
22         // 集合内和 x 异或的最小值
23         // if(son[p][u]) p = son[p][u];
24         // else res |= (1 << i), p = son[p][u ^ 1];
25     }
26     return res;
27 }
28
29 int main() {
30     int n, res = 0;
31     cin >> n;
32     for(int i = 0; i < n; i++) cin >> a[i];
33     for(int i = 0; i < n; i++) {
34         insert(a[i]);
35         res = max(res, query(a[i]));
36     }
37     cout << res;
38     return 0;
39 }

```

2.4 树状数组

```

1  const int N = 1e5 + 10;
2  int tr[N], a[N];
3  inline int lowbit(int x) {return x & -x;}
4
5  int query(int x) {
6      int res = 0;
7      for (int i = x; i; i -= lowbit(i)) res += tr[i];
8      return res;
9  }
10
11 void add(int x, int val) {
12     for(int i = x; i <= n; i += lowbit(i))
13         tr[i] += val;
14 }
15
16 //fenwick-tree 写区间修改, 区间查询
17 //记录两个数组 b[i] = a[i] - a[i - 1]; c[i] = i * b[i];
18 #include <bits/stdc++.h>
19 using namespace std;
20 typedef long long ll;
21 const int N = 1e5 + 10;
22 int a[N], b[N];
23 ll t1[N], t2[N]; //维护 b[i], b[i] * i 的前缀和
24 int n, m;
25
26 void add(ll tr[], int x, ll c) {

```

```

27     for (int i = x; i <= n; i += lowbit(i))
28         tr[i] += c;
29 }
30
31 ll query(ll tr[], int x) {
32     ll res = 0;
33     for (int i = x; i; i -= lowbit(i))
34         res += tr[i];
35     return res;
36 }
37
38 ll preSum(int x) { return query(t1, x) * (x + 1) - query(t2, x); }
39
40 int main() {
41     scanf("%d%d", &n, &m);
42     for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
43
44     for (int i = 1; i <= n; i++) {
45         int b = a[i] - a[i - 1];
46         add(t1, i, b);
47         add(t2, i, (ll)b * i);
48     }
49     while(m -- ) {
50         char op[2];
51         int l, r, d;
52         scanf("%s%d%d", op, &l, &r);
53         if(*op == 'Q')
54             printf("%lld\n", preSum(r) - preSum(l - 1));
55         else {
56             scanf("%d", &d);
57             //a[l] += d;
58             add(t1, l, d), add(t2, l, l * d);
59             add(t1, r + 1, -d), add(t2, r + 1, (r + 1) * -d);
60         }
61     }
62     return 0;
63 }

```

2.5 线段树

```

1 //常见维护
2 /**
3  * 区间和，最值
4  * 维护最大连续字段和 （维护 lmax, rmax, tmax）
5  * 维护区间平方和
6  * 区间修改成之指定数 维护 sum, lazy(指定数值), bool changed;
7  * 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
8  * 区间内数字同时乘以一个数 如下：
9  */

```

```

10  #include<bits/stdc++.h>
11  using namespace std;
12  using ll = long long;
13  const int N = 1e5 + 10;
14  int n, m, mod;
15  int a[N];
16
17  struct node {
18      int l, r;
19      int sum, add, mul;
20  } t[4 * N];
21
22  void eval(node &t, int add, int mul) {
23      t.sum = ((ll)t.sum * mul + (ll)(t.r - t.l + 1) * add) % mod;
24      t.mul = (ll)t.mul * mul % mod;
25      t.add = ((ll)t.add * mul + add) % mod;
26  }
27
28  void pushup(int p) {
29      t[p].sum = (t[p << 1].sum + t[p << 1 | 1].sum) % mod;
30  }
31
32  void pushdown(int p) {
33      eval(t[p << 1], t[p].add, t[p].mul);
34      eval(t[p << 1 | 1], t[p].add, t[p].mul);
35
36      t[p].add = 0, t[p].mul = 1;
37  }
38
39  void build(int p, int l, int r) {
40      if(l == r) {
41          t[p] = {l, r, a[l], 0, 1};
42          return;
43      }
44      t[p] = {l, r, 0, 0, 1};
45      int mid = l + r >> 1;
46      build(p << 1, l, mid);
47      build(p << 1 | 1, mid + 1, r);
48      pushup(p);
49  }
50
51  void modify(int p, int l, int r, int add, int mul) {
52      if(t[p].l >= l && t[p].r <= r) eval(t[p], add, mul);
53      else {
54          pushdown(p);
55          int mid = t[p].l + t[p].r >> 1;
56          if(l <= mid) modify(p << 1, l, r, add, mul);
57          if(r > mid) modify(p << 1 | 1, l, r, add, mul);
58          pushup(p);

```

```

59     }
60 }
61
62 int query(int p, int l, int r) {
63     if(t[p].l >= l && t[p].r <= r) return t[p].sum;
64
65     pushdown(p);
66     int res = 0;
67     int mid = t[p].l + t[p].r >> 1;
68
69     if(l <= mid) res += query(p << 1, l, r);
70     if(r > mid) res += query(p << 1 | 1, l, r);
71     res %= mod;
72     return res;
73 }
74
75 int main() {
76     scanf("%d%d", &n, &mod);
77     for (int i = 1; i <= n; i++)
78         scanf("%d", &a[i]);
79     build(1, 1, n);
80
81     scanf("%d", &m);
82     while(m -- ) {
83         int op, l, r, d;
84         scanf("%d%d%d", &op, &l, &r);
85         if(op == 1) {
86             scanf("%d", &d);
87             modify(1, 1, r, 0, d);
88         }
89         else if(op == 2) {
90             scanf("%d", &d);
91             modify(1, 1, r, d, 1);
92         }
93         else
94             printf("%d\n", query(1, l, r));
95     }
96     return 0;
97 }

```

扫描线: (面积)

```

1 //p1502 线段树扫描线算法
2 #include<bits/stdc++.h>
3 using namespace std;
4 using ll = long long;
5 const ll N = 1e4 + 10;
6 struct L {
7     ll x, y1, y2;
8     ll c;
9     //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

```

```

10     bool operator<(const L &rhs) const { return x == rhs.x ? c < rhs.c : x < rhs.x; }
11 }line[2 * N];
12
13 ll n, w, h, m;
14 ll b[2 * N]; //离散化前的 y 轴
15
16 struct node {
17     ll l, r;
18     ll maxv, add;
19 } t[8 * N];
20
21
22 void pushdown(ll p) {
23     node &root = t[p], &nl = t[p << 1], &nr = t[p << 1 | 1];
24     if(root.add) {
25         nl.add += root.add, nl.maxv += root.add;
26         nr.add += root.add, nr.maxv += root.add;
27         root.add = 0;
28     }
29 }
30
31 void pushup(ll p) {
32     t[p].maxv = max(t[p << 1].maxv, t[p << 1 | 1].maxv);
33 }
34
35 void modify(ll p, ll l, ll r, ll c) {
36     if(t[p].l >= l && t[p].r <= r) {
37         t[p].maxv += c;
38         t[p].add += c;
39         return;
40     }
41     pushdown(p);
42     ll mid = t[p].l + t[p].r >> 1;
43     if(l <= mid) modify(p << 1, l, r, c);
44     if(r > mid) modify(p << 1 | 1, l, r, c);
45     pushup(p);
46
47 }
48
49 void build(ll p, ll l, ll r) {
50     if(l == r) {
51         t[p] = {l, r, 0, 0};
52         return;
53     }
54     t[p].l = l, t[p].r = r;
55     ll mid = l + r >> 1;
56     build(p << 1, l, mid);
57     build(p << 1 | 1, mid + 1, r);
58     //pushup(p); //初始化都是 0 不用 pushup()

```

```

59 }
60
61 int main() {
62     ll T;
63     scanf("%lld", &T);
64     while( T -- ) {
65         memset(line, 0, sizeof(line));
66         memset(b, 0, sizeof(b));
67         memset(t, 0, sizeof(t));
68
69         scanf("%lld%lld%lld", &n, &w, &h);
70         for (ll i = 1, j = 0; i <= n; i++) {
71             ll x, y, l;
72             scanf("%lld%lld%lld", &x, &y, &l);
73             line[i] = {x, y, y + h - 1, l};
74             line[i + n] = {x + w - 1, y, y + h - 1, -1};
75             b[ ++ j] = y;
76             b[ ++ j] = y + h - 1;
77         }
78         n <<= 1;
79         sort(b + 1, b + 1 + n);
80         m = unique(b + 1, b + 1 + n) - b - 1; //unique 得到 end() 迭代器
81         sort(line + 1, line + 1 + n);
82
83         for (ll i = 1; i <= n; i++) {
84             line[i].y1 = lower_bound(b + 1, b + m + 1, line[i].y1) - b - 1;
85             line[i].y2 = lower_bound(b + 1, b + m + 1, line[i].y2) - b - 1;
86         }
87         build(1, 1, m - 1);
88
89         ll res = 0;
90         for (ll i = 1; i <= n; i++) {
91             modify(1, line[i].y1, line[i].y2, line[i].c);
92             res = max(res, t[1].maxv);
93         }
94         printf("%d\n", res);
95     }
96     return 0;
97 }

```

2.6 可持久化线段树

```

1 //Luogu 3824 kth-number
2 #include <bits/stdc++.h>
3 using namespace std;
4 const int N = 2e5 + 10, M = (N << 2) + 17 * N;
5
6 struct node {
7     int l, r;

```

```

8     int cnt;
9 } t[M];
10 int idx, a[N];
11 vector<int> num;
12 int find(int x) { return lower_bound(num.begin(), num.end(), x) - num.begin(); }
13
14 int insert(int now, int l, int r, int x) {
15     int p = ++ idx;
16     t[p] = t[now];
17     if (l == r) {
18         t[p].cnt ++ ;
19         return p;
20     }
21     int mid = l + r >> 1;
22     if(x <= mid) t[p].l = insert(t[now].l, l, mid, x);
23     else t[p].r = insert(t[now].r, mid + 1, r, x);
24     t[p].cnt = t[t[p].l].cnt + t[t[p].r].cnt;
25
26     return p;
27 }
28
29 int build(int l, int r) {
30     int p = ++ idx;
31     if (l == r) return p;
32     int mid = l + r >> 1;
33     t[p].l = build(l, mid), t[p].r = build(mid + 1, r);
34     return p;
35 }
36
37 int query(int x, int y, int l, int r, int k) {
38     if(l == r) return l;
39     int cnt = t[t[y].l].cnt - t[t[x].l].cnt;
40     int mid = l + r >> 1;
41     if(k <= cnt) return query(t[x].l, t[y].l, l, mid, k);
42     else return query(t[x].r, t[y].r, mid + 1, r, k - cnt);
43 }
44
45 int n, m, root[N];
46
47 int main() {
48     scanf("%d%d", &n, &m);
49     for (int i = 1; i <= n; i ++ ) {
50         scanf("%d", &a[i]);
51         num.push_back(a[i]);
52     }
53
54     sort(num.begin(), num.end());
55     num.erase(unique(num.begin(), num.end()), num.end());
56

```

```

57     root[0] = build(0, num.size() - 1);
58
59     for (int i = 1; i <= n; i ++ )
60         root[i] = insert(root[i - 1], 0, num.size() - 1, find(a[i]));
61     while (m -- ) {
62         int l, r, k;
63         scanf("%d%d%d", &l, &r, &k);
64         printf("%d\n", num[query(root[l - 1], root[r], 0, num.size() - 1, k)]);
65     }
66
67     return 0;
68 }

```

2.7 线段树合并

```

1  int merge(int p, int q, int l, int r) {
2      if(!p || !q) return p + q;
3      if(l == r) {
4          //维护信息, 一般是 t[p].val += t[q].val 等
5          // t[p].val.first += t[q].val.first;
6          return p;
7      }
8      int mid = l + r >> 1;
9      t[p].l = merge(t[p].l, t[q].l, l, mid);
10     t[p].r = merge(t[p].r, t[q].r, mid + 1, r);
11     // pushup();
12     // t[p].val = max(t[t[p].l].val, t[t[p].r].val);
13     return p;
14 }

```

2.8 树链剖分

```

1  #include<bits/stdc++.h>
2  #define pb push_back
3  using namespace std;
4
5  using ll = long long;
6  const int N = 1e5 + 10;
7
8  struct node {
9      int l, r;
10     ll add, sum;
11 } t[N << 2];
12 int n, m, w[N], nw[N];
13 vector<int> G[N];
14
15 int dep[N], top[N], son[N], dfn[N], sz[N], fa[N], cnt;
16 ///////////////////////////////////////////////////线段树部分////////////////////////////////////
17

```



```

18 void pushdown(int p) {
19     auto &rt = t[p], &nl = t[p << 1], &nr = t[p << 1 | 1];
20     if(rt.add) {
21         nl.add += rt.add, nl.sum += (ll)(nl.r - nl.l + 1) * rt.add;
22         nr.add += rt.add, nr.sum += (ll)(nr.r - nr.l + 1) * rt.add;
23         rt.add = 0;
24     }
25 }
26
27 void pushup(int p) { t[p].sum = t[p << 1].sum + t[p << 1 | 1].sum; }
28
29 void build(int p, int l, int r) {
30     t[p] = {l, r, 0, nw[l]};
31     if(l == r) return;
32
33     int mid = l + r >> 1;
34     build(p << 1, l, mid);
35     build(p << 1 | 1, mid + 1, r);
36     pushup(p);
37 }
38
39 ll query(int p, int l, int r) {
40     if(t[p].l >= l && t[p].r <= r) return t[p].sum;
41
42     pushdown(p);
43     int mid = t[p].l + t[p].r >> 1;
44     ll res = 0;
45     if(l <= mid) res += query(p << 1, l, r);
46     if(r > mid) res += query(p << 1 | 1, l, r);
47     //pushup(p);
48     return res;
49 }
50
51 void modify(int p, int l, int r, int k) {
52     if(t[p].l >= l && t[p].r <= r) {
53         t[p].sum += (t[p].r - t[p].l + 1) * k;
54         t[p].add += k;
55         return;
56     }
57
58     pushdown(p);
59     int mid = t[p].l + t[p].r >> 1;
60     if(l <= mid) modify(p << 1, l, r, k);
61     if(r > mid) modify(p << 1 | 1, l, r, k);
62     pushup(p);
63 }
64
65 //////////////////////////////////////////////////树剖部分//////////////////////////////////////
66 //第一次 dfs 维护 sz, 重儿子, dep[], fa[]

```

```

67 void dfs1(int u, int fath) {
68     sz[u] = 1, dep[u] = dep[fath] + 1, fa[u] = fath;
69     for(int v:G[u]) {
70         if(v == fath) continue;
71         dfs1(v, u);
72         sz[u] += sz[v];
73         if(sz[son[u]] < sz[v]) son[u] = v;
74     }
75 }
76 //第二次 dfs, 维护 dfs 序,
77 void dfs2(int u, int tp) {
78     dfn[u] = ++cnt, nw[cnt] = w[u], top[u] = tp;
79     if(!son[u]) return;
80     dfs2(son[u], tp); //递归重儿子
81     //维护轻儿子信息
82     for(int v:G[u]) {
83         if(v == fa[u] || v == son[u]) continue;
84         dfs2(v, v);
85     }
86 }
87
88 void modify_path(int u, int v, int k) {
89     while(top[u] != top[v]) {
90         if(dep[top[u]] < dep[top[v]]) swap(u, v);
91         modify(1, dfn[top[u]], dfn[u], k);
92         u = fa[top[u]];
93     }
94     if(dep[u] < dep[v]) swap(u, v);
95     modify(1, dfn[v], dfn[u], k);
96 }
97
98 void modify_tree(int u, int k) {
99     modify(1, dfn[u], dfn[u] + sz[u] - 1, k);
100 }
101
102 ll query_tree(int u) {
103     return query(1, dfn[u], dfn[u] + sz[u] - 1);
104 }
105
106 ll query_path(int u, int v) {
107     ll res = 0;
108     while(top[u] != top[v]) {
109         if(dep[top[u]] < dep[top[v]]) swap(u, v);
110         res += query(1, dfn[top[u]], dfn[u]);
111         u = fa[top[u]];
112     }
113     if(dep[u] < dep[v]) swap(u, v);
114     res += query(1, dfn[v], dfn[u]);
115     return res;

```

```

116 }
117
118 //////////////////////////////////////
119 int main() {
120
121     scanf("%d", &n);
122     for(int i = 1; i <= n; i++) scanf("%d", &w[i]);
123     for(int i = 1; i < n; i++) {
124         int u, v; scanf("%d%d", &u, &v);
125         G[u].pb(v), G[v].pb(u);
126     }
127     dfs1(1, 0);
128     dfs2(1, 1);
129
130     build(1, 1, n);
131
132     scanf("%d", &m);
133     while(m -- ) {
134         int op, u, v, k;
135         scanf("%d%d", &op, &u);
136         if(op == 1) {
137             scanf("%d%d", &v, &k);
138             modify_path(u, v, k);
139         }
140         else if(op == 2) {
141             scanf("%d", &k);
142             modify_tree(u, k);
143         }
144         else if(op == 3) {
145             scanf("%d", &v);
146             printf("%lld\n", query_path(u, v));
147         }
148         else
149             printf("%lld\n", query_tree(u));
150     }
151     return 0;
152 }

```

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 $\mathcal{O}(\log n)$
3. 插入一个值 $\mathcal{O}(\log n)$
4. 合并两个堆 $\mathcal{O}(\log n)$

```

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

```

```

3  using namespace std;
4  const int N = 2e5 + 10;
5  int val[N], lson[N], rson[N], dis[N];
6  int fa[N], idx, n;
7  int find(int x) { return x == fa[x] ? x : fa[x] = find(fa[x]); }
8
9  bool cmp(int x, int y) { return val[x] == val[y] ? x < y : val[x] < val[y]; }
10
11 int merge(int x, int y) {
12     if(!x || !y) return x + y;
13     if(cmp(y, x)) swap(x, y);
14     rson[x] = merge(rson[x], y);
15     if(dis[rson[x]] > dis[lson[x]]) swap(lson[x], rson[x]);
16     dis[x] = dis[rson[x]] + 1;
17     return x;
18 }
19
20 int main() {
21     ios::sync_with_stdio(false), cin.tie(0);
22     cin >> n;
23     val[0] = 2e9;
24     while(n --) {
25         int op, x, y; cin >> op;
26         if(op == 1) {
27             cin >> x;
28             val[++idx] = x;
29             fa[idx] = idx;
30             dis[idx] = 1;
31         }
32         else if(op == 2) {
33             cin >> x >> y;
34             x = find(x), y = find(y);
35             if(x != y) {
36                 if(cmp(y, x)) swap(x, y); //x 为较小的
37                 fa[y] = x;
38                 merge(x, y);
39             }
40         }
41         else if(op == 3) {
42             cin >> x;
43             cout << val[find(x)] << endl;
44         }
45         else { // 删除 x 所在堆的最小值
46             cin >> x; x = find(x);
47             if(cmp(rson[x], lson[x])) swap(lson[x], rson[x]);
48             fa[x] = lson[x], fa[lson[x]] = lson[x];
49             merge(lson[x], rson[x]);
50         }
51     }

```

```

52     return 0;
53 }

```

2.10 莫队

普通莫队:

```

1  #include <bits/stdc++.h>
2  #define endl '\n'
3  #define rep(i, a, b) for (int i = (a); i <= (b); i++)
4  using namespace std;
5
6  const int N = 5e4 + 10, M = 2e5 + 10, S = 1e6 + 10; //值域
7
8  int n, A[N], ans[M], cnt[S], m, sq, cur;
9
10 struct query {
11     int l, r, id;
12     bool operator<(const query &rhs) const { //奇偶化排序
13         if (l / sq != rhs.l / sq)
14             return l < rhs.l;
15         if (l / sq & 1)
16             return r < rhs.r;
17         return r > rhs.r;
18     }
19 } q[M];
20
21 void add(int p) {
22     if(cnt[A[p]] == 0) cur++;
23     cnt[A[p]]++;
24 }
25
26 void del(int p) {
27     cnt[A[p]]--;
28     if(cnt[A[p]] == 0) cur--;
29 }
30
31 int main() {
32     ios::sync_with_stdio(false);
33     cin.tie(nullptr);
34
35     cin >> n;
36     sq = sqrt(n);
37     rep(i, 1, n) cin >> A[i];
38     cin >> m;
39     rep(i, 1, m) {
40         int l, r;
41         cin >> l >> r;
42         q[i] = {l, r, i};
43     }

```

```

44     sort(q + 1, q + 1 + m);
45
46     int l = 1, r = 0;
47     rep(i, 1, m) {
48         while(l > q[i].l) add(--l);
49         while(r < q[i].r) add(++r);
50         while(l < q[i].l) del(l++);
51         while(r > q[i].r) del(r--);
52         ans[q[i].id] = cur;
53     }
54     rep(i, 1, m) cout << ans[i] << endl;
55
56     return 0;
57 }

```

待修莫队:

```

1  #include <bits/stdc++.h>
2  #define endl '\n'
3  #define rep(i, a, b) for (int i = (a); i <= (b); i++)
4  using namespace std;
5
6  const int N = 134000, S = 1e6 + 10; //值域
7
8  int n, m, mq, mc, len, cur;
9  int w[N], cnt[S], ans[N];
10 struct Query {
11     int id, l, r, tim;
12 } q[N];
13 struct Modify {
14     int pos, val;
15 } c[N];
16
17 int get(int x) {
18     return x / len;
19 }
20
21 bool cmp(const Query& a, const Query& b) {
22     int al = get(a.l), ar = get(a.r);
23     int bl = get(b.l), br = get(b.r);
24     if (al != bl) return al < bl;
25     if (ar != br) return ar < br;
26     return a.tim < b.tim;
27 }
28
29 void add(int val) {
30     if(cnt[val] == 0) cur++;
31     cnt[val]++;
32 }
33
34 void del(int val) {

```

```

35     cnt[val]--;
36     if(cnt[val] == 0) cur--;
37 }
38 int main() {
39     ios::sync_with_stdio(false);
40     cin.tie(nullptr);
41     cin >> n >> m;
42     rep(i, 1, n) cin >> w[i];
43
44     rep (i, 1, m) {
45         char op[2];
46         int a, b;
47         cin >> op >> a >> b;
48         if (*op == 'Q') mq ++, q[mq] = {mq, a, b, mc};
49         else c[ ++ mc] = {a, b};
50     }
51
52     len = cbrt((double)n * max(1, mc)) + 1;
53     sort(q + 1, q + mq + 1, cmp);
54
55     int l = 1, r = 0, t = 0;
56     rep(i, 1, mq) {
57         auto [id, ql, qr, qt] = q[i];
58         while (l < ql) del(w[l++]);
59         while (l > ql) add(w[--l]);
60         while (r < qr) add(w[++r]);
61         while (r > qr) del(w[r--]);
62         while (t < qt) {
63             t ++ ;
64             if (ql <= c[t].pos && qr >= c[t].pos) {
65                 del(w[c[t].pos]);
66                 add(c[t].val);
67             }
68             swap(w[c[t].pos], c[t].val);
69         }
70         while (t > qt) {
71             if (ql <= c[t].pos && qr >= c[t].pos) {
72                 del(w[c[t].pos]);
73                 add(c[t].val);
74             }
75             swap(w[c[t].pos], c[t].val);
76             t--;
77         }
78         ans[id] = cur;
79     }
80
81     rep(i, 1, mq) printf("%d\n", ans[i]);
82     return 0;
83 }

```

3 图论

3.1 spfa

```

1  #include <bits/stdc++.h>
2  #define pb push_back
3  using namespace std;
4
5  const int N = 1e5 + 10, inf = 0x3f3f3f3f;
6  struct node{int v, w;};
7  vector<node> G[N];
8  int dis[N], n, m;
9  bool inq[N];
10
11 void spfa() {
12     memset(dis, 0x3f, sizeof dis);
13     dis[1] = 0;
14     inq[1] = 1;
15     queue<int> q;
16     q.push(1);
17     while(q.size()) {
18         int u = q.front(); q.pop();
19         inq[u] = 0;
20         for(auto [v, w]:G[u]) {
21             if(dis[v] > w + dis[u]) {
22                 dis[v] = dis[u] + w;
23                 if(!inq[v])
24                     q.push(v), inq[v] = true;
25             }
26         }
27     }
28 }
29
30 int main() {
31     cin >> n >> m;
32     while(m -- ) {
33         int u, v, w;
34         cin >> u >> v >> w;
35         G[u].pb({v, w});
36     }
37     spfa();
38     if(dis[n] == inf)    cout << "impossible";
39     else                cout << dis[n];
40     return 0;
41 }

```

3.2 dijkstra

稀疏图 dijkstra:


```

1 //acwing 849
2 #include <bits/stdc++.h>
3 using namespace std;
4 const int N = 510, inf = 0x3f3f3f3f;
5 int dis[N], G[N][N], n, m;
6 bool vis[N];
7
8 void dij() {
9     memset(dis, 0x3f, sizeof dis);
10    dis[1] = 0;
11    for (int j = 0; j < n; j++) {
12        int minv = inf, pos = -1;
13        for (int i = 1; i <= n; i++)
14            if (!vis[i] && minv > dis[i])
15                minv = dis[i], pos = i;
16
17        if (pos == -1) break;
18        vis[pos] = 1;
19        for (int i = 1; i <= n; i++)
20            if (!vis[i] && dis[pos] + G[pos][i] < dis[i])
21                dis[i] = dis[pos] + G[pos][i];
22    }
23 }
24
25 int main() {
26     cin >> n >> m;
27     scanf("%d %d", &n, &m);
28     memset(G, 0x3f, sizeof(G));
29     while (m--) {
30         int u, v, w; scanf("%d %d %d", &u, &v, &w);
31         G[u][v] = min(G[u][v], w);
32     }
33
34     dij();
35
36     cout << (dis[n] == inf ? -1 : dis[n]);
37     return 0;
38 }

```

稠密图 dijkstra:

```

1 #include <bits/stdc++.h>
2 #define pb push_back
3 #define fi first
4 #define se second
5 using namespace std;
6
7 using P = pair<int, int>;
8 const int N = 151000, inf = 0x3f3f3f3f;
9 struct node {int v, w;};
10 vector<node> G[N];

```

```

11 int dis[N], n, m;
12 bool vis[N];
13
14 void dij() {
15     memset(dis, 0x3f, sizeof dis);
16     priority_queue<P, vector<P>, greater<P>> q;
17     q.push({0, 1});
18     while(q.size()) {
19         auto t = q.top(); q.pop();
20         int u = t.se, d = t.fi;
21         if(vis[u]) continue;
22         vis[u] = true;
23         for(auto [v, w] : G[u]) {
24             if(dis[v] > d + w) {
25                 dis[v] = d + w;
26                 q.push({dis[v], v});
27             }
28         }
29     }
30 }
31
32 int main() {
33     ios::sync_with_stdio(false);
34     cin >> n >> m;
35     while(m -- ) {
36         int u, v, w; cin >> u >> v >> w;
37         G[u].pb({v, w});
38     }
39     dij();
40     cout << (dis[n] == inf ? -1 : dis[n]);
41     return 0;
42 }

```

3.3 最小生成树

```

1 // kruskal
2 const int N = 1e5 + 10;
3 struct edge {
4     int u, v, w;
5     bool operator<(const edge &rhs) const { return w < rhs.w; }
6 } edges[N];
7
8 int fa[N], n, m;
9 int find(int x) { return x == fa[x] ? x : fa[x] = find(fa[x]); }
10
11 int kruskal() {
12     cin >> n >> m;
13     int u, v, w, ans = 0;
14     for (int i = 1; i <= m; i ++ ) {

```

```

15         cin >> u >> v >> w;
16         edges[i] = {u, v, w};
17     }
18     sort(edges + 1, edges + 1 + m);
19     for (int i = 1; i <= n; i++) fa[i] = i;
20     for (int i = 1; i <= m; i++) {
21         auto [u, v, w] = edges[i];
22         u = find(u), v = find(v);
23         if(u == v) continue;
24         fa[u] = v;
25         ans += w;
26     }
27     return ans;
28 }
29
30 //prim
31 const int N = 510, inf = 0x3f3f3f3f;
32 int G[N][N], dis[N];
33 int n, m;
34 bool vis[N];
35
36 int prim() {
37     int res = 0;
38     memset(dis, 0x3f, sizeof dis);
39     dis[1] = 0;    //随便选一点进入 mst 集合
40     for(int j = 0; j < n; j++) {
41         int minv = inf, pos = -1;
42         for(int i = 1; i <= n; i++)
43             if(!vis[i] && dis[i] < minv)
44                 pos = i, minv = dis[i];
45
46         if(pos == -1) return inf;
47         vis[pos] = true;
48         res += dis[pos];
49
50         for(int i = 1; i <= n; i++)
51             if(!vis[i] && dis[i] > G[pos][i])
52                 dis[i] = G[pos][i];
53     }
54     return res;
55 }

```

另外，对于完全图的 *MST* 问题，可以考虑使用 *Boruvka* 算法。我们要在 $n \log n$ 或 $n \log^2 n$ 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

```

1 //kruskal 重构树
2

```

```

3 //性质:
4 //两个点之间的所有简单路径上最大边权的最小值
5 // = 最小生成树上两个点之间的简单路径上的最大值
6 // = Kruskal 重构树上两点之间的 LCA 的权值。
7 //Loj136
8 #include <bits/stdc++.h>
9 #define pb push_back
10 using namespace std;
11
12 const int N = 1010 << 1, M = 3e5 + 10;
13 int n, m, k, val[N]; // kruskal 重构树的点权
14 int idx; //重构树的节点数
15
16 struct Edge{
17     int u, v, w;
18     bool operator<(const Edge &rhs) const { return w < rhs.w; }
19 }edges[M];
20
21 vector<int> G[N];
22
23 int p[N];
24 int find(int x) { return x == p[x] ? x : p[x] = find(p[x]); }
25
26 int dep[N], fa[N][21];
27
28 void bfs(int s) {
29     dep[0] = 0, dep[s] = 1;
30     queue<int> q;
31     q.push(s);
32     while(q.size()) {
33         int u = q.front(); q.pop();
34         for(int v:G[u]) {
35             if(dep[v] > dep[u] + 1) {
36                 dep[v] = dep[u] + 1;
37                 q.push(v);
38                 fa[v][0] = u;
39                 for (int i = 1; i <= 20; i++)
40                     fa[v][i] = fa[fa[v][i - 1]][i - 1];
41             }
42         }
43     }
44 }
45
46 int lca(int a, int b) {
47     if(dep[a] < dep[b]) swap(a, b);
48     for (int k = 20; k >= 0; k--)
49         if(dep[fa[a][k]] >= dep[b])
50             a = fa[a][k];
51     if(a == b) return a;

```

```

52     for (int k = 20; k >= 0; k --)
53         if(fa[a][k] != fa[b][k])
54             a = fa[a][k], b = fa[b][k];
55     return fa[a][0];
56 }
57
58 void build() {
59     idx = n;
60     int cnt = 0;
61     for (int i = 1; i <= m; i ++) {
62         int u = edges[i].u, v = edges[i].v, w = edges[i].w;
63         int fu = find(u), fv = find(v);
64         if(fu != fv) {
65             val[++idx] = w;
66             G[idx].pb(fu), G[idx].pb(fv);
67             G[fu].pb(idx), G[fv].pb(idx);
68             p[fu] = p[fv] = idx;
69             cnt++;
70         }
71         if(cnt >= n - 1) break;
72     }
73 }
74
75 int main() {
76     scanf("%d %d %d", &n, &m, &k);
77     for (int i = 1; i <= m; i ++) {
78         int u, v, w; scanf("%d %d %d", &u, &v, &w);
79         edges[i] = {u, v, w};
80     }
81     sort(edges + 1, edges + m + 1);
82     for (int i = 1; i <= (n << 1); i ++) p[i] = i;
83
84     build(); // kruskal 重构树
85
86     memset(dep, 0x3f, sizeof dep);
87     bfs(idx); //bfs 的根节点一定要是重构树的最高点
88
89     while(k -- ) {
90         int s, t;
91         scanf("%d %d", &s, &t);
92         if(find(s) != find(t)) puts("-1");
93         else
94             printf("%d\n", val[lca(s, t)]);
95     }
96     return 0;
97 }

```

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 $n - \text{最大匹配数}$ 。（ n 是二分图总点数）

```

1  /* 染色法判断二分图
2  bool vis[N];
3  int col[N], flag = 1, n, m;
4  void dfs(int u, int t) {
5      if (vis[u]) {
6          if (col[u] != t) flag = 0;
7          return;
8      }
9      vis[u] = 1; col[u] = t;
10     for (int v : g[u]) {
11         dfs(v, t ^ 1);
12     }
13 }
14 bool isbit() { // 是否为二分图
15     for (int u = 1; u <= n; u++) {
16         if (!vis[u]) dfs(u, 0);
17     }
18     return flag;
19 }
20 */
21 int G[N][M]; // 左半部 n, 右半部 m
22 int n, m, p[M], vis[M];
23 bool match(int u) {
24     for (int i = 1; i <= m; i++) {
25         if (G[u][i] && !vis[i]) {
26             vis[i] = true;
27             if (p[i] == 0 || match(p[i])) {
28                 p[i] = u; return true;
29             }
30         }
31     }
32     return false;
33 }
34 int main() {
35     /* 建图 */
36     int res = 0;
37     for (int i = 1; i <= n; i++) {
38         memset(vis, 0, sizeof vis);

```

```

39         if(match(i)) res++;
40     }
41     return 0;
42 }

```

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。

```

1  #include<bits/stdc++.h>
2  #define pb push_back
3  using namespace std;
4
5  const int N = 1e4 + 10;
6  vector<int> G[N], G2[N];
7  stack<int> s;
8  int n, m, tim, scc_cnt;
9  int w[N], dfn[N], low[N], id[N];
10 int dist[N], ind[N], W[N];
11 bool ins[N];
12
13 void tarjan(int u) {
14     low[u] = dfn[u] = ++tim;
15     s.push(u); ins[u] = true;
16     for(int v:G[u]) {
17         if(!dfn[v]) {
18             tarjan(v);
19             low[u] = min(low[v], low[u]);
20         }
21         else if(ins[v])
22             low[u] = min(low[u], dfn[v]);
23     }
24     if(low[u] == dfn[u]) {
25         int y; ++scc_cnt;
26         do {
27             y = s.top(); s.pop();
28             ins[y] = false;
29             id[y] = scc_cnt;
30             W[scc_cnt] += w[y];
31         } while (y != u);
32     }
33 }
34
35 int sol() {
36     queue<int> q;
37     for (int i = 1; i <= scc_cnt; i++)
38         if(!ind[i]) {
39             q.push(i);
40             dist[i] = W[i];
41         }

```

```

42
43 while(q.size()) {
44     //cout << "cnt = " << ++cnt << endl;
45     int u = q.front(); q.pop();
46     for (int v:G2[u]) {
47
48         ↪ //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
49         dist[v] = max(dist[v], dist[u] + W[v]);
50         if(--ind[v] == 0)
51             q.push(v);
52     }
53 }
54
55 int ans = 0;
56 for (int i = 1; i <= scc_cnt; i++)
57     ans = max(ans, dist[i]);
58 return ans;
59 }
60
61 int main() {
62     ios::sync_with_stdio(false), cin.tie(0);
63     cin >> n >> m;
64     for (int i = 1; i <= n; i++) cin >> w[i];
65     while(m--) {
66         int u, v;
67         cin >> u >> v;
68         G[u].pb(v);
69     }
70     for (int i = 1; i <= n; i++)
71         if(!dfn[i])
72             tarjan(i);
73     //缩点
74     for (int u = 1; u <= n; ++u) {
75         for(int v : G[u]) {
76             if(id[v] != id[u]) {
77                 G2[id[u]].pb(id[v]);
78                 ind[id[v]]++;
79                 //printf("ind[%d] = %d\n", id[v], ind[id[v]]);
80             }
81         }
82     }
83     // debug
84     // for (int i = 1; i <= scc_cnt; i++)
85     //     printf("ind[%d] = %d\n", i, ind[i]);
86     // for (int i = 1; i <= scc_cnt; i++)
87     // {
88     //     printf("%d->", i);

```



```

90     //     for (int v:G2[i])
91     //         printf("%d ", v);
92     //     puts("");
93     // }
94     printf("%d\n", sol());
95     return 0;
96 }

```

3.7 无向图的双连通分量

桥:

```

1 // 一个有桥的连通图，如何把它通过加边变成边双连通图？
2 // 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。
3 // 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。
4 // 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。
5 // 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。
6 #include <bits/stdc++.h>
7 #define pb push_back
8 using namespace std;
9 const int N = 5010;
10 int n, m;
11 vector<int> G[N];
12 int low[N], dfn[N], id[N], deg[N];
13 int dcc_cnt, tim, stk[N], top;
14 vector<int> bridge[N];
15
16 void tarjan(int u, int fa) {
17     low[u] = dfn[u] = ++tim;
18     stk[++top] = u;
19
20     for (int i = 0; i < G[u].size(); i++) {
21         int v = G[u][i];
22         if (!dfn[v]) {
23             tarjan(v, u);
24             low[u] = min(low[v], low[u]);
25             if (dfn[u] < low[v])
26                 bridge[u].pb(v), bridge[v].pb(u);
27         }
28         else if (fa != v)
29             low[u] = min(low[u], dfn[v]);
30     }
31     if (dfn[u] == low[u]) {
32         int y;
33         ++dcc_cnt;
34         do {
35             y = stk[top--];
36             id[y] = dcc_cnt;
37         } while (u != y);
38     }

```

```

39 }
40
41 int main() {
42     ios::sync_with_stdio(false), cin.tie(0);
43     cin >> n >> m;
44     while(m -- ) {
45         int u, v;
46         cin >> u >> v;
47         G[u].pb(v), G[v].pb(u);
48     }
49     tarjan(1, -1);
50     for (int u = 1; u <= n; u++)
51         deg[id[u]] += bridge[u].size();
52     int cnt = 0;
53     for (int i = 1; i <= dcc_cnt; i++)
54         if(deg[i] == 1)
55             cnt++;
56     cout << (cnt + 1) / 2;
57     return 0;
58 }

```

割点:

```

1  #include<bits/stdc++.h>
2  #define pb push_back
3  using namespace std;
4  const int N = 2e4 + 10;
5  vector<int> G[N], cut;
6  int tim, n, m, root;
7  int dfn[N], low[N];
8
9  void tarjan(int u) {
10     low[u] = dfn[u] = ++tim;
11     int tot = 0;
12     for(int v:G[u]) {
13         if(!dfn[v]) {
14             tarjan(v);
15             low[u] = min(low[u], low[v]);
16             if (dfn[u] <= low[v])
17                 tot++;
18         }
19         else
20             low[u] = min(low[u], dfn[v]);
21     }
22     if ( (tot > 0 && u != root) || (tot > 1 && u == root))
23         cut.pb(u);
24 }
25 }
26
27 int main() {
28     cin >> n >> m;

```

```

29     while(m --) {
30         int u, v;
31         cin >> u >> v;
32         G[u].pb(v), G[v].pb(u);
33     }
34
35     for (root = 1; root <= n; root ++){
36         if(!dfn[root])
37             tarjan(root);
38
39         //不用 sort, 就开一个 bool cut[N];
40         sort(cut.begin(), cut.end());
41         printf("%d\n", cut.size());
42         for(int v : cut)
43             printf("%d ", v);
44
45         return 0;
46     }

```

3.8 lca

```

1  /*
2  求 lca: 1. 倍增 2. 树剖 3.tarjan 离线
3
4  lca 用处
5  1. 树上两点之间的距离 (多维护一个 dist 数组,  $dis[u] + dis[v] - 2 * dis[lca(u, v)]$ )
6  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)
7  */
8  //acwing1171 树上距离
9  #include <bits/stdc++.h>
10 #define pb push_back
11 #define endl '\n'
12 using namespace std;
13 const int N = 1e4 + 10;
14
15 struct node{int v, w;};
16 vector<node> G[N];
17 int fa[N][19], dep[N], dis[N];
18 int n, m;
19
20 void bfs(int s) {
21     memset(dep, 0x3f, sizeof dep);
22     dep[0] = 0, dep[s] = 1;
23     dis[s] = 0;
24     queue<int> q; q.push(s);
25     while(q.size()) {
26         int u = q.front(); q.pop();
27         for(auto [v, w] : G[u]) {
28             if(dep[v] > dep[u] + 1) {

```

```

29         dis[v] = dis[u] + w;
30         dep[v] = dep[u] + 1;
31         fa[v][0] = u;
32         q.push(v);
33         for(int i = 1; i < 19; ++i)
34             fa[v][i] = fa[fa[v][i - 1]][i - 1];
35     }
36 }
37 }
38 }
39
40 int lca(int a, int b) {
41     if(dep[a] < dep[b]) swap(a, b);
42     for(int k = 18; k >= 0; k--)
43         if(dep[fa[a][k]] >= dep[b])
44             a = fa[a][k];
45     if(a == b) return a;
46
47     for(int k = 18; k >= 0; --k)
48         if(fa[a][k] != fa[b][k])
49             a = fa[a][k], b = fa[b][k];
50     return fa[a][0];
51 }
52
53 int main() {
54     ios::sync_with_stdio(false), cin.tie(0);
55     cin >> n >> m;
56     for(int i = 1; i < n; i++) {
57         int u, v, w; cin >> u >> v >> w;
58         G[u].pb({v, w}), G[v].pb({u, w});
59     }
60     bfs(1);
61     while(m--) {
62         int u, v; cin >> u >> v;
63         int anc = lca(u, v);
64         cout << dis[u] + dis[v] - 2 * dis[anc] << endl;
65     }
66     return 0;
67 }

```

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

```

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

```

```

5  using ll = long long;
6  using namespace std;
7  const int N = 1e6 + 10, M = N << 1;
8  int h[N], e[M], w[M], ne[M], idx;
9  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
10 bool ins[N], vis[N];
11 int n, cir[M], ed[M], cnt; //cnt 环的个数
12 int fa[N], fw[N]; //父节点, 反向权值
13 int q[M];
14 ll ans;
15
16 void add(int a, int b, int c) {
17     e[idx] = b, ne[idx] = h[a], w[idx] = c, h[a] = idx++;
18 }
19
20 //深搜 + 栈 找环
21 void dfs(int u, int from) {
22     vis[u] = ins[u] = true;
23     for (int i = h[u]; ~i; i = ne[i]) {
24         //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
25         if (i == (from ^ 1)) continue;
26         int v = e[i];
27         fa[v] = u, fw[v] = w[i];
28         if (!vis[v]) dfs(v, i);
29         else if (ins[v]) {
30             cnt++;
31             ed[cnt] = ed[cnt - 1];
32             ll tot = w[i];
33             for (int k = u; k != v; k = fa[k]) {
34                 s[k] = tot;
35                 tot += fw[k];
36                 cir[++ ed[cnt]] = k;
37             }
38             s[v] = tot, cir[++ ed[cnt]] = v;
39         }
40     }
41     ins[u] = false;
42 }
43
44 // 求以 u 为根节点的子树的最大深度
45 ll dfs_d(int u) {
46     vis[u] = true;
47     ll d0 = 0, d1 = 0; //最大距离, 次大距离
48     for (int i = h[u]; ~i; i = ne[i]) {
49         int v = e[i];
50         if (vis[v]) continue;
51         ll d = dfs_d(v) + w[i];
52         if (d >= d0) d1 = d0, d0 = d;
53         else if (d > d1) d1 = d;

```

```

54     }
55     ans = max(ans, d1 + d0);
56     return d0;
57 }
58
59 int main() {
60     ios::sync_with_stdio(false), cin.tie(0);
61     cin >> n;
62     memset(h, -1, sizeof h);
63     for (int u = 1; u <= n; u++) {
64         int v; ll w; cin >> v >> w;
65         add(u, v, w), add(v, u, w);
66     }
67
68     for (int i = 1; i <= n; i++)
69         if (!vis[i])
70             dfs(i, -1);
71
72     memset(vis, 0, sizeof vis);
73     for (int i = 1; i <= n; i++) vis[cir[i]] = 1; //标记环上所有点
74
75     ll res = 0;
76     for (int i = 1; i <= cnt; i++) {
77         ans = 0; // 当前基环树的直径
78         int sz = 0; // 当前基环树的环的大小
79         for (int j = ed[i - 1] + 1; j <= ed[i]; j++) {
80             int k = cir[j];
81             d[sz] = dfs_d(k); // 求以当前点为根的子树的最大深度
82             sum[sz] = s[k];
83             sz++;
84         }
85         // 破环成链，前缀和数组和 d[] 数组延长一倍
86         for (int j = 0; j < sz; j++)
87             d[sz + j] = d[j], sum[sz + j] = sum[j] + sum[sz - 1];
88
89         // 做一遍滑动窗口，比较依据是 d[k] - sum[k]
90         int hh = 0, tt = -1;
91         for (int j = 0; j < sz * 2; j++) {
92             while (hh <= tt && q[hh] <= j - sz) hh++;
93             if (hh <= tt) ans = max(ans, d[j] + sum[j] + d[q[hh]] - sum[q[hh]]);
94             while (hh <= tt && d[j] - sum[j] >= d[q[tt]] - sum[q[tt]]) tt--;
95             q[++tt] = j;
96         }
97         res += ans;
98     }
99     cout << res << endl;
100     return 0;
101 }

```

3.10 dinic

```

1  #include <bits/stdc++.h>
2  #define pb push_back
3  using namespace std;
4  using ll = long long;
5  const int N = 1e4 + 10;
6  const ll inf = 0x3f3f3f3f3f3f3f3f;
7  int n, m, s, t, dep[N];
8  struct node {int v, cap, rev;};
9  vector<node> G[N];
10
11 bool bfs() {
12     queue<int> q;
13     q.push(s);
14     memset(dep, -1, sizeof dep);
15     dep[s] = 0;
16     while (q.size()) {
17         int u = q.front(); q.pop();
18         for(auto [v, cap, rev] : G[u])
19             if(dep[v] == -1 && cap)
20                 dep[v] = dep[u] + 1, q.push(v);
21     }
22     return dep[t] != -1;
23 }
24
25 ll dfs(int u, ll lim) {
26     if(u == t || lim == 0) return lim;
27     ll tot_flow = 0;
28     for(auto& [v, cap, rev] : G[u]) {
29         if(dep[v] == dep[u] + 1 && cap > 0) {
30             ll d = dfs(v, min(lim, (ll)cap));
31             cap -= d, G[v][rev].cap += d;
32             lim -= d, tot_flow += d;
33             if(lim == 0) return tot_flow;
34         }
35     }
36     if(lim != 0) dep[u] = -1;
37     return tot_flow;
38 }
39
40 ll dinic() {
41     ll max_flow = 0;
42     while(bfs())
43         max_flow += dfs(s, inf);
44     return max_flow;
45 }
46
47 int main() {
48     scanf("%d%d%d", &n, &m, &s, &t);

```

```

49     while(m --) {
50         int u, v, cap; scanf("%d%d%d", &u, &v, &cap);
51         G[u].pb({v, cap, G[v].size()});
52         G[v].pb({u, 0, G[u].size() - 1});
53     }
54     printf("%lld\n", dinic());
55     return 0;
56 }

```

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。

```

1  // 求树上 对某个点来说包含他的连通点集个数
2  #include <bits/stdc++.h>
3  #define pb push_back
4  #define endl '\n'
5  using ll = long long;
6  using namespace std;
7  const int N = 1e6 + 10, mod = 1e9 + 7;
8
9  ll f[N], ans[N], n;
10 vector<int> G[N];
11
12 ll qpow(ll a, ll b) {
13     ll res = 1;
14     while(b) {
15         if(b & 1) res = res * a % mod;
16         a = a * a % mod;
17         b >>= 1;
18     }
19     return res;
20 }
21
22 void dfs(int u, int fa) {
23     f[u] = 1;
24     for (auto v:G[u]) {
25         if(v == fa) continue;
26         dfs(v, u);
27         f[u] = f[u] * (f[v] + 1) % mod;
28     }
29 }
30
31 /*
32 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)

```



```

33 换根方程:  $ans[v] = (ans[u] / (f[v] + 1) + 1) * f[v]$ 
34 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  $f[v]$ 
35
36 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  $mod - 1$ ), 这时相当于除以  $0$ , 出错
37 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。
38 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  $brother$ 。
39  $(f[brother_1] + 1) * (f[brother_2] + 1) * \dots$  他的所有兄弟的值乘积。
40  $*/$ 
41
42 void dp(int u, int fa) {
43     for (int v:G[u]) {
44         if(v == fa) continue;
45         ll inv = qpow(f[v] + 1, mod - 2);
46         if(inv) ans[v] = (ans[u] * inv % mod + 1) % mod * f[v] % mod;
47         else {
48             ll t = 1;
49             for (auto other:G[u]) {
50                 if(other == v || other == fa) continue;
51                 t = t * (f[other] + 1) % mod;
52             }
53             ans[v] = (t + 1) * f[v] % mod;
54         }
55         dp(v, u);
56     }
57 }
58
59 int main() {
60     cin >> n;
61     for (int i = 1; i < n; i++) {
62         int u, v; cin >> u >> v;
63         G[u].pb(v), G[v].pb(u);
64     }
65     dfs(1, 0);
66     ans[1] = f[1];
67     dp(1, 0);
68
69     for (int i = 1; i <= n; i++) cout << ans[i] << endl;
70     return 0;
71 }

```

5 字符串

5.1 字符串 Hash

```

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

```

```

6     int siz;
7     vector<ull> pow_base, hash_val; // or p, h due to time budget
8     Hash() { }
9     Hash(const string &s) {
10         siz = s.size();
11         pow_base.resize(siz);
12         hash_val.resize(siz);
13         pow_base[0] = 1;
14         hash_val[0] = s[0];
15         for (int i = 1; i < siz; i++){
16             pow_base[i] = pow_base[i - 1] * base;
17             hash_val[i] = hash_val[i - 1] * base + s[i];
18         }
19     }
20     // 下标 0 开始, 闭区间
21     ull operator[](const array<int, 2>& range) const {
22         // if(r < l || l > n) return 0; //根据题目需要处理边界情况
23         auto l = range[0], r = range[1];
24         if(l == 0) return hash_val[r];
25         return hash_val[r] - hash_val[l - 1] * pow_base[r - l + 1];
26     }
27
28     ull get(int l, int r) {
29         return this->operator[]({l, r});
30     }
31 };
32
33 struct doubleHash {
34     using ll = long long;
35     int size;
36     array<int, 2> mod = {2000000011, 2000000033}, base = {20011, 20033};
37     vector<array<ll, 2>> hash, pow_base;
38     doubleHash() { }
39     doubleHash(const string& s) {
40         size = s.size();
41         hash.resize(size);
42         pow_base.resize(size);
43         pow_base[0][0] = pow_base[0][1] = 1;
44         hash[0][0] = hash[0][1] = s[0];
45         for(int i = 1; i < size; i++){
46             hash[i][0] = (hash[i - 1][0] * base[0] + s[i]) % mod[0];
47             hash[i][1] = (hash[i - 1][1] * base[1] + s[i]) % mod[1];
48             pow_base[i][0] = pow_base[i - 1][0] * base[0] % mod[0];
49             pow_base[i][1] = pow_base[i - 1][1] * base[1] % mod[1];
50         }
51     }
52     array<ll, 2> operator[](const array<int, 2>& range) const {
53         auto l = range[0], r = range[1];
54         if(l == 0) return hash[r];

```

```

55     return {
56         (hash[r][0] - hash[l - 1][0] * pow_base[r - l + 1][0] % mod[0] + mod[0]) %
        ↪ mod[0],
57         (hash[r][1] - hash[l - 1][1] * pow_base[r - l + 1][1] % mod[1] + mod[1]) %
        ↪ mod[1]);
58     }
59     //double hash to A hash_val
60     ll get(int l, int r) {
61         auto h = this->operator[]({l, r});
62         return h[0] * 1000000000ll + h[1];
63     }
64 };
65 int main() {}

```

5.2 Trie

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const int N = 1e5 + 10;
5  char str[N];
6  int son[N][26], cnt[N], idx;
7
8  void insert(char *str) {
9      int p = 0;
10     for (int i = 0; str[i]; i++) {
11         int u = str[i] - 'a';
12         if (!son[p][u]) son[p][u] = ++idx;
13         p = son[p][u];
14     }
15     ++cnt[p];
16 }
17
18 int query(char *str) {
19     int p = 0;
20     for (int i = 0; str[i]; ++i) {
21         int u = str[i] - 'a';
22         if (!son[p][u]) return 0;
23         p = son[p][u];
24     }
25     return cnt[p];
26 }

```

5.3 KMP

```

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

```

```

5  char s[N];
6  int nxt[N], n;
7  /*
8  //区间 l->r 的 kmp
9      nxt[l] = 0;
10     for (int i = l + 1; i <= r; i++) {
11         int j = nxt[i - 1];
12         while(j && s[i] != s[l + j]) j = nxt[l + j - 1];
13         if(s[i] == s[l + j]) j++;
14         nxt[i] = j;
15     }
16 */
17 void get_nxt() {
18     nxt[1] = 0;
19     for (int i = 2, j = 0; i <= n; i++) {
20         while(j && s[i] != s[j + 1]) j = nxt[j];
21         if(s[i] == s[j + 1]) j++;
22         nxt[i] = j;
23     }
24 }
25
26 int main() {
27     while(~scanf("%s", s + 1)) {
28         if(s[1] == '.') break;
29         n = strlen(s + 1);
30         get_nxt();
31         int period = n - nxt[n];
32         if(n % period == 0) printf("%d\n", n / period);
33         else puts("1");
34     }
35     return 0;
36 }

```

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 $Z(i), Z(i) - 1, \dots, 1$ 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）

```

1 //给定两个字符串 a,b,
2 // 要求出两个数组: b 的 z 函数数组 z、
3 // b 与 a 的每一个后缀的 LCP 长度数组 p。
4 #include <bits/stdc++.h>
5 #define rep(i, a, b) for (int i = (a); i < (b); i++)
6 #define sz(a) int((a).size())
7 using namespace std;
8 using ll = long long;
9 const int N = 2e7;
10 ll ansz, ansp;
11 string a, b;
12
13 // Zfunction
14 int z[N << 1];
15 void getz(string s) {
16     int l = 0;
17     for (int i = 1; i <= s.size(); i++) {
18         if (l + z[l] > i) z[i] = min(z[i - 1], l + z[l] - i);
19         while (i + z[i] < s.size() && s[z[i]] == s[i + z[i]]) z[i]++;
20         if (i + z[i] > l + z[l]) l = i;
21     }
22     // rep(i,0,s.size()) cout<<z[i]<<" ";cout<<'\\n';
23 }
24
25
26 int main(){
27     ios::sync_with_stdio(0);
28     cin.tie(0),cout.tie(0);
29     cin >> a >> b, getz(b + a);
30     ansz ^= 1ll * (sz(b)+1)*(0+1);
31     rep(i,1,sz(b)) ansz^=1ll*(min(z[i],sz(b)-i)+1)*(i+1);
32     rep(i,0,sz(a)) ansp^=1ll*(min(z[i+sz(b)],sz(b))+1)*(i+1);
33     cout << ansz << '\\n'
34         << ansp << '\\n';
35     return 0;
36 }

```

5.5 AC 自动机

```

1 //Luogu3808
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 const int N = 1e6 + 10;
6 int n;
7 char s[N];

```

```

8
9 namespace ac
10 {
11
12 int tr[N][26], fail[N], idx;
13 queue<int> q;
14 int cnt[N];
15
16 void insert(char* s) {
17     int p = 0;
18     for (int i = 1; s[i]; ++i) {
19         int u = s[i] - 'a';
20         if(!tr[p][u]) tr[p][u] = ++idx;
21         p = tr[p][u];
22     }
23     ++cnt[p];
24 }
25
26 void build() {
27     for (int i = 0; i < 26; ++i)
28         if(tr[0][i]) q.push(tr[0][i]);
29
30     while(q.size()) {
31         int u = q.front(); q.pop();
32         for (int i = 0; i < 26; i++) {
33             if(tr[u][i])
34                 fail[tr[u][i]] = tr[fail[u]][i], q.push(tr[u][i]);
35                 ↪ //原本这个 tr[fail[u]][i] 可能不存在 (为 0)
36
37                 else
38                     tr[u][i] = tr[fail[u]][i];
39         }
40     }
41
42 int query(char *s) {
43     int u = 0, res = 0;
44     for (int i = 1; s[i]; ++i) {
45         u = tr[u][s[i] - 'a'];
46         for (int j = u; j && cnt[j] != -1; j = fail[j])
47             res += cnt[j], cnt[j] = -1;
48     }
49     return res;
50 }
51
52 }
53
54 int main() {

```

↪ // 但是下一步 `else` 做了一个优化 (类似

```

55     scanf("%d", &n);
56     for (int i = 1; i <= n; i++) {
57         scanf("%s", s + 1);
58         ac::insert(s);
59     }
60     ac::build();
61     scanf("%s", s + 1);
62     printf("%d\n", ac::query(s));
63     return 0;
64 }

```

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

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  class SuffixArray {
5  //得到的 sa[], rk[] 下标从 0 开始, ht 下标从 1 开始 (因为是长度)
6  private:
7      int n, m;
8      vector<int> x, y, cnt;
9      void radixSort() {
10         for (int i = 0; i < m; ++ i) cnt[i] = 0;
11         for (int i = 0; i < n; ++ i) cnt[x[i]] ++;
12         for (int i = 1; i < m; ++ i) cnt[i] += cnt[i - 1];
13         for (int i = n - 1; i >= 0; -- i) sa[-- cnt[x[y[i]]]] = y[i];
14     }
15 public:
16     vector<int> sa, rk, ht;
17
18     SuffixArray(const string &s) :
19         n(s.size()), m(256), //m 为字符集最大数量
20         x(n), y(n), cnt(max(n, m)),
21         sa(n), rk(n), ht(n) {
22         init_sa(s);
23         init_ht(s);
24     }
25     void init_sa(const string &s) {
26         for (int i = 0; i < n; ++ i) {
27             x[i] = s[i];
28             y[i] = i;

```

```

29     }
30     radixSort();
31     for (int w = 1; w <= n; w <= 1) {
32         int p = 0;
33         for (int i = n - w; i < n; ++ i) y[p ++] = i;
34         for (int i = 0; i < n; ++ i)
35             if (sa[i] >= w) y[p ++] = sa[i] - w;
36
37         radixSort();
38         swap(x, y);
39         x[sa[0]] = 0;
40         p = 1;
41         auto cmp = [&](int i, int j) {
42             if (i < n && j < n) return y[i] == y[j];
43             return i >= n && j >= n;
44         };
45         for (int i = 1; i < n; ++ i)
46             x[sa[i]] = (cmp(sa[i], sa[i - 1]) && cmp(sa[i - 1] + w, sa[i] + w))
47                 ? p - 1 : p++;
48
49         if (p >= n) break;
50         m = p;
51     }
52     for (int i = 0; i < n; ++ i) rk[sa[i]] = i;
53 }
54 void init_ht(const string &s) {
55     for (int i = 0, k = 0; i < n; ++ i) {
56         if (rk[i] == 0) continue ;
57         if (k) k --;
58         int j = sa[rk[i] - 1];
59         while (i + k < n && j + k < n && s[i + k] == s[j + k]) k ++;
60         ht[rk[i]] = k;
61     }
62 }
63 };
64
65 void solve() {
66     string s;
67     cin >> s;
68     SuffixArray f(s);
69     for (int i = 0; i < s.size(); ++ i)
70         cout << f.sa[i] + 1 << " \n"[i + 1 == s.size()];
71     for (int i = 0; i < s.size(); ++ i)
72         cout << f.ht[i] << " \n"[i + 1 == s.size()];
73 }
74
75 int main() {
76     ios::sync_with_stdio(false);
77     cin.tie(nullptr);

```



```

78     int T; cin >> T;
79     while (T -- ) {
80         solve();
81     }
82     return 0;
83 }

```

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

```

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--));
    }
}

```

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  // return p, p[i] 表示修改后的串中以 i 为中心的最长回文半径
4  vector<int> manacher(const string& _s) {
5      vector<int> p(_s.size() * 2 + 1);
6      string s(_s.size() * 2 + 1, '$');
7      for (int i = 0; i < _s.size(); i++) s[2 * i + 1] = _s[i];
8      for(int i = 0, maxr = 0, mid = 0; i < s.size(); i++) {
9          if(i < maxr) p[i] = min(p[mid * 2 - i], maxr - i);
10         while(i - p[i] - 1 >= 0 && i + p[i] + 1 < s.size()
11             && s[i - p[i] - 1] == s[i + p[i] + 1])
12             ++p[i];
13         if(i + p[i] > maxr) maxr = i + p[i], mid = i;
14     }
15     return p;
16 }
17
18
19 int main() {
20     string s;
21     cin >> s;
22     auto p = manacher(s);
23     // for (int i = 0; i < p.size(); i++) {

```

```

24     //     cout << p[i] << " \n"[i == p.size() - 1];
25     // }
26     cout << (*max_element(p.begin(), p.end())) << endl;
27 }

```

6 其他

6.1 glibc 内置函数

```

1  // Returns the number of 1-bits in x.
2  int __builtin_popcount(unsigned int x);
3  int __builtin_popcountll(unsigned long long x);
4
5  // Returns the number of trailing 0 (undefined when x == 0)
6  int __builtin_ctz(unsigned int x);
7  int __builtin_ctzll(unsigned long long x);
8
9  // Returns log2(x)
10 int __lg(int x);
11
12 int __gcd(int x, int y);

```

6.2 __int128 读写

```

1  inline __int128 read(){
2      __int128 x = 0, f = 1;
3      char ch = getchar();
4      while (ch < '0' || ch > '9') { if(ch == '-') f = -1; ch = getchar(); }
5      while (ch >= '0' && ch <= '9') { x = x * 10 + ch - '0'; ch = getchar(); }
6      return x * f;
7  }
8
9  inline void print(__int128 x) {
10     if(x < 0) { putchar('-'); x = -x; }
11     if(x > 9) print(x / 10);
12     putchar(x % 10 + '0');
13 }

```

6.3 整数二分

```

1  // 区间 [l, r] 被划分成 [l, ans] 和 [ans + 1, r] 时使用:
2  int bsearch_1(int l, int r) {
3      while (l < r) {
4          int mid = l + r >> 1;
5          if (check(mid)) r = mid;    // check() 判断 mid 是否满足性质
6          else l = mid + 1;
7      }
8      return l;
9  }

```

```

10 // 区间 [l, r] 被划分成 [l, ans - 1] 和 [ans, r] 时使用:
11 int bsearch_2(int l, int r) {
12     while (l < r) {
13         int mid = l + r + 1 >> 1;
14         if (check(mid)) l = mid;
15         else r = mid - 1;
16     }
17     return l;
18 }

```

6.4 单调栈

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  const int N = 1000100;
4  //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)
5  int stk[N], tt, a[N];
6
7  int main() {
8      ios::sync_with_stdio(false), cin.tie(0), cout.tie(0);
9      int n; cin >> n;
10     for (int i = 1; i <= n; i++) cin >> a[i];
11
12     for (int i = 1; i <= n; i++) {
13         while(tt && stk[tt] >= a[i]) tt--;
14         if(tt) cout << stk[tt] << ' ';
15         else cout << -1 << ' ';
16         stk[++tt] = a[i];
17     }
18     return 0;
19 }

```

6.5 单调队列

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  const int N = 1e6 + 10;
4  int a[N], q[N], n, k;
5  //滑动窗口
6  int main() {
7      cin >> n >> k;
8      for(int i = 0; i < n; i++) cin >> a[i];
9      int hh = 0, tt = -1;
10     for(int i = 0; i < n; i++) {
11         //判断队头是否已经划出窗口
12         if( hh <= tt && i - k + 1 > q[hh]) hh++;
13         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
14         q[ ++ tt ] = i;
15         if(i >= k-1) printf("%d ",a[q[hh]]);

```

```

16
17     }
18     return 0;
19 }

```

6.6 GossersHack

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。

```

1 void GossersHack(int k, int n) {
2     int cur = (1 << k) - 1;
3     int limit = (1 << n);
4     while (cur < limit) {
5         // do something
6         int lb = cur & -cur;
7         int r = cur + lb;
8         cur = ((r ^ cur) >> __builtin_ctz(lb) + 2) | r;
9         // 或: cur = (((r ^ cur) >> 2) / lb) | r;
10    }
11 }

```

6.7 C++17-STL

```

1 #include <bits/stdc++.h>
2 using ll = long long;
3 using namespace std;
4
5 /*
6  Lambda expression
7  */
8
9 /* structure binding */
10 {
11     // Graph with weighted edges
12     vector<vector<pair<int, int>>> G(n);
13     // somewhere in dfs,
14     for (auto [v, w] : G[u]) {
15         ...
16     }
17 }
18
19 /* discrete manipulate*/
20 {
21     // vector<int> v. vv = v;
22     sort(v.begin(), v.end());
23     v.erase(unique(v.begin(), v.end()), v.end());
24
25 }
26

```

```

27  /*
28  some useful function in STL
29  */
30  {
31      int n = 256;
32      vector<int> v(n);
33      // generate v as 0, 1, 2, ...
34      // often used when you initialize DSU or generate a permutation
35      iota(v.begin(), v.end(), 0);
36
37      ll res = accumulate(v.begin(), v.end(), 1LL,
38                          [](int a, int b){return (ll)a*b;});
39
40      gcd() and lcm(); // not __gcd() version
41
42      // max_element and min_element which can be used with user-defined op
43      cout << *max_element(v.begin(), v.end() /*[]()->bool{*/});
44
45      //std::clamp
46      // Returns x if it is in the interval [low, high] or,
47      // otherwise, the nearest value.No more max of min of max of...
48      cout << clamp(7, 0, 10); //7
49      cout << clamp(7, 0, 5); //5
50      cout << clamp(7, 10, 50); //10
51
52      /*don't do*/ max(ans, max(t1, t2));
53      /*just*/ max({ans, t1, t2}); // initializer list
54
55
56      /* partial_sum and adjacent_difference 前缀和与差分 */
57      vector<int> a(n, 2), b(n, 2);
58      partial_sum(a.begin(), a.end(), a.begin());
59      partial_sum(a.begin(), a.end(), b.begin());
60
61      adjacent_difference(v.begin(), v.end(), v.begin());
62  }
63
64  /*
65  template< class R, class... Args >
66  class function<R(Args...)>;
67  */
68  {
69      // a callable obj
70      function<int(int, int)> dfs = [&](int u, int fa) {
71          ...
72      };
73  }
74
75  /* Set operations (on sorted ranges)*/

```

```
76 {
77     set<int> s1, s2;
78     set<int> ans;
79     set_intersection(s1.begin(), s1.end(),
80                     s2.begin(), s2.end(),
81                     std::inserter(ans, ans.begin()));
82     // 若为 vector, 可以用 back_inserter;
83 }
84
85 /*
86  Initializer in if and switch:
87  */
88 {
89     set<int> s;
90     if (auto [iter, ok] = s.insert(42); ok) {
91         //...
92     }
93     else {
94         //`ok` and `iter` are available here
95     }
96     //But not here
97 }
98
99 /*
100 about string
101 */
102 {
103     substr(npos, count);
104 }
105
106 int main() {
107 }
```