

# icpc 算法模板

Catch-22

2022 年 7 月 14 日

## 目录

<b>1</b>	<b>数学</b>	<b>4</b>
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 . . . . .	8
1.6	数论分块 . . . . .	9
1.7	Möbius 反演 . . . . .	9
1.8	高斯消元 . . . . .	9
1.9	Miller Rabin 素数测试 . . . . .	10
<b>2</b>	<b>数据结构</b>	<b>11</b>
2.1	(带权) 并查集 . . . . .	11
2.2	Sparse Table . . . . .	12
2.3	01Trie . . . . .	12
2.4	树状数组 . . . . .	13
2.5	线段树 . . . . .	13
2.6	可持久化线段树 . . . . .	13
2.7	线段树合并 . . . . .	13
2.8	树链剖分 . . . . .	13
2.9	莫队 . . . . .	13
2.10	左偏树 . . . . .	13
<b>3</b>	<b>图论</b>	<b>15</b>
3.1	spfa . . . . .	15
3.2	dijkstra . . . . .	16
3.3	最小生成树 . . . . .	17
3.4	kruskal 重构树 . . . . .	19
3.5	强连通分量缩点 . . . . .	21
3.6	lca . . . . .	23
3.7	基环树 . . . . .	25
<b>4</b>	<b>动态规划</b>	<b>26</b>
4.1	数位 dp . . . . .	26
4.2	换根 dp . . . . .	26
<b>5</b>	<b>字符串</b>	<b>28</b>
5.1	KMP . . . . .	28
5.2	字符串 Hash . . . . .	28
5.3	Trie . . . . .	28
5.4	AC 自动机 . . . . .	29
5.5	SA . . . . .	31

目录	3
5.6 Manacher . . . . .	31
<b>6 其他</b>	<b>31</b>
6.1 glibc 内置函数 . . . . .	31
6.2 __int128 读写 . . . . .	31
6.3 单调栈 . . . . .	31
6.4 单调队列 . . . . .	32

# 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     for (int i = 1; i <= n; i++)
23         cout << inv[i] << endl;
24
25     return 0;
26 }

```

## 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;

```



```

30     }
31 }
32 }

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 }

```

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

```

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 {
22     for (int i = 0; i < N; i ++ )
23         for (int j = 0; j <= i; j ++ )
24             if(!j) C[i][j] = 1;
25             else C[i][j] = (C[i - 1][j] + C[i - 1][j - 1]) % mod;
26 }
27
28 //最常用
29 /*3. 预处理 fac[], invfac[]*/
30 /**
31  * //调用 :
32  * ll * fac[b] * invfac[a] % mod * invfac[b - a] % mod;
33  */
34 // O(N) 1e6 左右 看 N 大小
35 int fac[N], invfac[N];
36 void init() {
37     fac[0] = 1;
38     for (int i = 1; i < N; i ++ ) (ll)fac[i] = fac[i - 1] * i % mod;
39     invfac[N - 1] = qpow(fac[N - 1], mod - 2);
40     for (int i = N - 2; i >= 0; i -- )
41         invfac[i] = (ll)invfac[i + 1] * (i + 1) % mod;

```

```

42 }
43
44 /*4. lucas 定理 当 a, b 的值特别大 如 1e9 以上...1e18 等 */
45 int C(int a, int b) {
46     int res = 1;
47     for (int i = 1, j = a; i <= b; i ++, j --) {
48         res = (ll)res * j % p;
49         res = (ll)res * binpow(i, p - 2) % p;
50     }
51     return res;
52 }
53
54 ll lucas(ll a, ll b) { //p 为质 (模) 数
55     if(a < p && b < p) return C(a, b);
56     return (ll)C(a % p, b % p) * lucas(a / p, b / p) % p;
57 }

```

## 1.5 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 }

```

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

```

## 1.7 Möbius 反演

## 1.8 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 }

```

## 1.9 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 }

```

## 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  $O(1)$ ，空间复杂度  $O(n \log n)$

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

## 2.5 线段树

## 2.6 可持久化线段树

## 2.7 线段树合并

## 2.8 树链剖分

## 2.9 莫队

## 2.10 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 }

```

## 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 }

```

对于完全图的 *MST* 问题，一般考虑使用 *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。

```

1  // cf888G botuvka+01trie
2  #include <bits/stdc++.h>
3  using namespace std;
4  using ll = long long;
5
6  int main() {
7
8
9
10     return 0;
11 }

```

### 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 {
30     dep[0] = 0, dep[s] = 1;

```

```

31     queue<int> q;
32     q.push(s);
33     while(q.size())
34     {
35         int u = q.front(); q.pop();
36         for(int v:G[u])
37         {
38             if(dep[v] > dep[u] + 1)
39             {
40                 dep[v] = dep[u] + 1;
41                 q.push(v);
42                 fa[v][0] = u;
43                 for (int i = 1; i <= 20; i++)
44                     fa[v][i] = fa[fa[v][i - 1]][i - 1];
45             }
46         }
47     }
48 }
49
50 int lca(int a, int b)
51 {
52     if(dep[a] < dep[b]) swap(a, b);
53     for (int k = 20; k >= 0; k--)
54         if(dep[fa[a][k]] >= dep[b])
55             a = fa[a][k];
56     if(a == b) return a;
57     for (int k = 20; k >= 0; k--)
58         if(fa[a][k] != fa[b][k])
59             a = fa[a][k], b = fa[b][k];
60     return fa[a][0];
61 }
62
63 void build()
64 {
65     idx = n;
66     int cnt = 0;
67     for (int i = 1; i <= m; i++)
68     {
69         int u = edges[i].u, v = edges[i].v, w = edges[i].w;
70         int fu = find(u), fv = find(v);
71         if(fu != fv)
72         {
73             val[++idx] = w;
74             G[idx].pb(fu), G[idx].pb(fv);
75             G[fu].pb(idx), G[fv].pb(idx);
76             p[fu] = p[fv] = idx;
77             cnt++;
78         }
79         if(cnt >= n - 1) break;

```

```

80     }
81 }
82
83 int main()
84 {
85     scanf("%d %d %d", &n, &m, &k);
86     for (int i = 1; i <= m; i++)
87     {
88         int u, v, w; scanf("%d %d %d", &u, &v, &w);
89         edges[i] = {u, v, w};
90     }
91     sort(edges + 1, edges + m + 1);
92     for (int i = 1; i <= (n << 1); i++) p[i] = i;
93
94     build(); // kruskal 重构树
95
96     memset(dep, 0x3f, sizeof dep);
97     bfs(idk); //bfs 的根节点一定要是重构树的最高点
98
99     while(k -- )
100     {
101         int s, t;
102         scanf("%d %d", &s, &t);
103         if(find(s) != find(t)) puts("-1");
104         else
105             printf("%d\n", val[lca(s, t)]);
106     }
107     return 0;
108 }

```

### 3.5 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

```

```

17     for(int v:G[u]) {
18         if(!dfn[v]) {
19             tarjan(v);
20             low[u] = min(low[v], low[u]);
21         }
22         else if(ins[v])
23             low[u] = min(low[u], dfn[v]);
24     }
25
26     if(low[u] == dfn[u]) {
27         int y; ++scc_cnt;
28         do {
29             y = s.top(); s.pop();
30             ins[y] = false;
31             id[y] = scc_cnt;
32             W[scc_cnt] += w[y];
33         } while (y != u);
34     }
35 }
36
37 int sol() {
38
39     queue<int> q;
40     for (int i = 1; i <= scc_cnt; i++)
41         if(!ind[i]) {
42             q.push(i);
43             dist[i] = W[i];
44         }
45
46     while(q.size()) {
47         //cout << "cnt = " << ++cnt << endl;
48         int u = q.front(); q.pop();
49         for (int v:G2[u]) {
50
51             ⇨ //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
52             dist[v] = max(dist[v], dist[u] + W[v]);
53             if(--ind[v] == 0)
54                 q.push(v);
55         }
56     }
57
58     int ans = 0;
59     for (int i = 1; i <= scc_cnt; i++)
60         ans = max(ans, dist[i]);
61     return ans;
62 }
63
64 int main() {

```

```

65     ios::sync_with_stdio(false), cin.tie(0);
66     cin >> n >> m;
67     for (int i = 1; i <= n; i++)    cin >> w[i];
68     while(m--) {
69         int u, v;
70         cin >> u >> v;
71         G[u].pb(v);
72     }
73     for (int i = 1; i <= n; i++)
74         if(!dfn[i])
75             tarjan(i);
76     //缩点
77     for (int u = 1; u <= n; ++u) {
78         for(int v : G[u]) {
79             if(id[v] != id[u]) {
80                 G2[id[u]].pb(id[v]);
81                 ind[id[v]]++;
82                 //printf("ind[%d] = %d\n", id[v], ind[id[v]]);
83             }
84         }
85     }
86     // debug
87     // for (int i = 1; i <= scc_cnt; i++)
88     //     printf("ind[%d] = %d\n", i, ind[i]);
89     // for (int i = 1; i <= scc_cnt; i++)
90     // {
91     //     printf("%d->", i);
92     //     for (int v:G2[i])
93     //         printf("%d ", v);
94     //     puts("");
95     // }
96     printf("%d\n", sol());
97     return 0;
98 }

```

### 3.6 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 }

```

### 3.7 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 }

```

## 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 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 }

```

### 5.2 字符串 Hash

### 5.3 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 }

```

## 5.4 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() {
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 }

```

↪ // 但是下一步 `else` 做了一个优化 (类似

## 5.5 SA

## 5.6 Manacher

# 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
4 // Returns the number of trailing 0 (undefined when x == 0)
5 int __builtin_ctz(unsigned int x);
6
7 // Returns log_2(x)
8 int __lg(int x);
9
10 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 单调栈

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int N = 100010;
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 }
```

## 6.4 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 }
```