

ICPC TEMPLATE

widsnoy

2023 年 12 月 29 日

目录

1 图论	2
1.1 最近公共祖先	2
1.2 2-SAT 前缀优化建图	3
1.3 Kruskal 重构树的性质	3
1.4 广义圆方树	3
1.5 点分树	4
1.6 二分图	6
1.7 有向图最小路径覆盖问题	6
1.8 网络流	7
1.8.1 Dinic 最大流	7
1.8.2 最小费用最大流	8
1.8.3 最大闭权子图	9
1.9 树哈希	9
1.10 强联通分量	10
1.11 割点和桥	11
1.12 点双联通分量	11
1.13 边双联通分量	12
1.14 2-SAT	13

2	数据结构	14
2.1	李超线段树	14
2.2	Link Cut Tree	16
2.3	兔队线段树	18
3	字符串	19
3.1	哈希	19
3.1.1	最长回文子串	19
3.2	字典树	20
3.3	维护异或和	20
3.4	KMP	21
3.4.1	字符串最小周期	21
3.4.2	每个前缀的出现次数	22
3.4.3	一个字符串中本质不同子串的数目	22
3.5	AC 自动机	23
3.6	后缀数组	23
3.7	Manacher	24
3.8	Z 函数	25
4	数学	25
4.1	基本预处理	25
4.2	线性基	26
4.3	高斯消元	27
4.3.1	解线性方程组	27
4.3.2	求行阶梯矩阵	28
4.4	矩阵树定理	28
4.5	多项式	30

4.5.1	FFT	30
4.6	组合数学	31
4.6.1	小球放盒	31
5	MISC	32
5.1	完全平方数判断	32
5.2	维护区间 GCD 值	32

目录

1 图论

1.1 最近公共祖先

```

1 // 倍增
2 int faz[N][20], dep[N];
3 void dfs(int u, int fa) {
4     faz[u][0] = fa;
5     dep[u] = dep[fa] + 1;
6     for (int i = 1; i < 20; i++) faz[u][i] = faz[faz[u][i - 1]][i - 1];
7     for (int v : G[u]) if (v != fa) {
8         dfs(v, u);
9     }
10 }
11 int LCA(int u, int v) {
12     if (dep[u] < dep[v]) swap(u, v);
13     int d = dep[u] - dep[v];
14     for (int i = 0; i < 20; i++) if ((d >> i) & 1) u = faz[u][i];
15     if (v == u) return u;
16     for (int i = 19; i >= 0; i--) if (faz[u][i] != faz[v][i])
17         u = faz[u][i], v = faz[v][i];
18     return faz[u][0];
19 }
20
21 // 树剖
22 int dfc, dfn[N], rnk[N], siz[N], top[N], dep[N], son[N], faz[N];
23 void dfs1(int u, int fa) {
24     dep[u] = dep[fa] + 1;
25     siz[u] = 1;
26     son[u] = -1;
27     faz[u] = fa;
28     for (int v : G[u]) {
29         if (v == fa) continue;
30         dfs1(v, u);
31         siz[u] += siz[v];
32         if (son[u] == -1 || siz[son[u]] < siz[v]) son[u] = v;
33     }
34 }
35 void dfs2(int u, int fa, int tp) {
36     dfn[u] = ++dfc;
37     rnk[dfc] = u;
38     top[u] = tp;
39     if (son[u] != -1) dfs2(son[u], u, tp);
40     for (int v : G[u]) {
41         if (v == fa || v == son[u]) continue;

```

```

42     dfs2(v, u, v);
43 }
44 }
45 int LCA(int u, int v) {
46     while (top[u] != top[v]) {
47         if (dep[top[u]] > dep[top[v]])
48             u = faz[top[u]];
49         else
50             v = faz[top[v]];
51     }
52     return dep[u] > dep[v] ? v : u;
53 }

```

1.2 2-SAT 前缀优化建图

1. 当前点选择说明之前的前缀都未选择

之前的前缀选择说明当前点被选择

2. 之前的前缀选择说明当前前缀被选择

当前前缀未选择说明之前前缀未选择

3. 当前点选择说明当前前缀选择

当前前缀未选择说明当前点未选择

1.3 Kruskal 重构树的性质

不难发现，原图中两个点之间的所有简单路径上最大边权的最小值 = 最小生成树上两个点之间的简单路径上的最大值 = Kruskal 重构树上两点之间的 LCA 的权值。

也就是说，到点 x 的简单路径上最大边权的最小值 $\leq val$ 的所有点 y 均在 Kruskal 重构树上的某一棵子树内，且恰好为该子树的所有叶子节点。

我们在 Kruskal 重构树上找到 x 到根的路径上权值 $\leq val$ 的最浅的节点。显然这就是所有满足条件的节点所在的子树的根节点。

如果要求原图中两个点之间的所有简单路径上最小边权的最大值，则在跑 Kruskal 的过程中按边权大到小的顺序加边。

1.4 广义圆方树

```

1  int stk[N], n, m, top, cnt, low[N], dfn[N], dfc;
2  bool vis[N];
3  vector<int> G[N], T[N];
4
5  void tarjan(int u) {
6      stk[++top] = u;
7      low[u] = dfn[u] = ++dfc;
8      for (int v : G[u]) {
9          if (!dfn[v]) {
10             tarjan(v);
11             low[u] = min(low[u], low[v]);
12             if (low[v] == dfn[u]) {
13                 cnt++;
14                 for (int x = 0; x != v; --top) {
15                     x = stk[top];
16                     T[cnt].push_back(x);
17                     T[x].push_back(cnt);
18                     val[cnt]++;
19                 }
20                 T[cnt].push_back(u);
21                 T[u].push_back(cnt);
22                 val[cnt]++;
23             }
24             } else low[u] = min(low[u], dfn[v]);
25      }
26  }
27  int main() {
28      cnt = n;
29      for (int i = 1; i <= n; i++) if (!dfn[i]) {
30          tarjan(i);
31          --top;
32      }
33  }

```

1.5 点分树

需要注意，点分树上的路径与原来的树完全没有关系。

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4
5  const int mod = 998244353;
6  const int N = 2e5 + 5;

```

```

7
8 int n, q, val[N], rt, siz[N], SUM, mx[N];
9 int faz[N];
10 vector<int> T[N];
11 bool vis[N];
12
13 // 假设树剖已经写好
14
15 void getrt(int u, int fa) {
16     siz[u] = 1;
17     mx[u] = 0;
18     for (int v : t.G[u]) if (v != fa && !vis[v]) {
19         getrt(v, u);
20         siz[u] += siz[v];
21         mx[u] = max(mx[u], siz[v]);
22     }
23     mx[u] = max(mx[u], SUM - siz[u]);
24     if (mx[u] < mx[rt]) rt = u;
25 }
26 void Dfs(int u, int fa) {
27     vis[u] = 1;
28     for (int v : t.G[u]) if (v != fa && !vis[v]) {
29         rt = 0;
30         SUM = siz[v];
31         getrt(v, rt);
32         T[u].push_back(rt);
33         faz[rt] = u;
34         Dfs(rt, rt);
35     }
36 }
37
38 // 假设动态开点权值线段树已经写好
39
40 void dfs(int u, const int now) { // 预处理
41     add(rt1[now], 0, n, dist(faz[now], u), val[u]);
42     add(rt0[now], 0, n, dist(u, now), val[u]);
43     for (int v : T[u]) dfs(v, now);
44 }
45 void upd(int u, int d) {
46     int x = u;
47     while (true) {
48         add(rt1[u], 0, n, dist(x, faz[u]), d);
49         add(rt0[u], 0, n, dist(x, u), d);
50         if (u == root) break;

```



```

51     u = faz[u];
52 }
53 }
54 int qry(int u, int k) {
55     int ans = query(rt0[u], 0, n, 0, k), x = u;
56     while (u != root) {
57         if (k - dist(faz[u], x) >= 0) ans += query(rt0[faz[u]], 0, n, 0, k -
            dist(faz[u], x)) - query(rt1[u], 0, n, 0, k - dist(faz[u], x));
58         u = faz[u];
59     }
60     return ans;
61 }

```

1.6 二分图

最大匹配

```

1  int mch[maxn], vis[maxn];
2  std::vector<int> e[maxn];
3  bool dfs(const int u, const int tag) {
4      for (auto v : e[u]) {
5          if (vis[v] == tag) continue;
6          vis[v] = tag;
7          if (!mch[v] || dfs(mch[v], tag)) return mch[v] = u, 1;
8      }
9      return 0;
10 }
11 int main() {
12     int ans = 0;
13     for (int i = 1; i <= n; ++i) if (dfs(i, i)) ++ans;
14 }

```

1.7 有向图最小路径覆盖问题

```

1  int n, m;
2  bitset<N> f[N];
3  int vis[N], mch[N];
4
5  bool dfs(int u, int dfc) {
6      for (int v = 1; v <= n; v++) if (v != u && vis[v] != dfc && f[u][v]) {
7          vis[v] = dfc;
8          if (!mch[v] || dfs(mch[v], dfc)) return mch[v] = u, 1;
9      }
10     return 0;

```

```

11 }
12
13 void solve() {
14     memset(vis, 0, sizeof vis);
15     memset(mch, 0, sizeof mch);
16     for (int i = 1; i <= n; i++) f[i].reset();
17     for (int i = 1; i <= m; i++) {
18         int u, v;
19         scanf("%d %d", &u, &v);
20         f[u].set(v);
21     }
22     for (int k = 1; k <= n; k++) {
23         for (int i = 1; i <= n; i++) if (f[i][k]) f[i] |= f[k];
24     }
25     int res = n;
26     for (int i = 1; i <= n; i++) res -= dfs(i, i);
27     printf("%d\n", res);
28 }

```

1.8 网络流

1.8.1 Dinic 最大流

注意每次清空数组的范围是 s 到 t .

```

1 int head[N], cur[N], ecnt, d[N];
2 struct Edge {
3     int nxt, v, flow, cap;
4 }e[];
5 void add_edge(int u, int v, int flow, int cap) {
6     e[ecnt] = {head[u], v, flow, cap}; head[u] = ecnt++;
7     e[ecnt] = {head[v], u, flow, 0}; head[v] = ecnt++;
8 }
9 bool bfs() {
10     memset(vis, 0, sizeof vis);
11     std::queue<int> q;
12     q.push(s);
13     vis[s] = 1;
14     d[s] = 0;
15     while (!q.empty()) {
16         int u = q.front();
17         q.pop();
18         for (int i = head[u]; i != -1; i = e[i].nxt) {
19             int v = e[i].v;
20             if (vis[v] || e[i].flow >= e[i].cap) continue;

```

```

21         d[v] = d[u] + 1;
22         vis[v] = 1;
23         q.push(v);
24     }
25 }
26 return vis[t];
27 }
28 int dfs(int u, int a) {
29     if (u == t || !a) return a;
30     int flow = 0, f;
31     for (int& i = cur[u]; i != -1; i = e[i].nxt) {
32         int v = e[i].v;
33         if (d[u] + 1 == d[v] && (f = dfs(v, std::min(a, e[i].cap - e[i].flow))
34             ) > 0) {
35             e[i].flow += f;
36             e[i ^ 1].flow -= f;
37             flow += f;
38             a -= f;
39             if (!a) break;
40         }
41     }
42     return flow;
43 }

```

1.8.2 最小费用最大流

```

1  const int inf = 1e9;
2  int head[N], cur[N], ecnt, dis[N], s, t, n, m, mincost;
3  bool vis[N];
4  struct Edge {
5      int nxt, v, flow, cap, w;
6  }e[100002];
7  void add_edge(int u, int v, int flow, int cap, int w) {
8      e[ecnt] = {head[u], v, flow, cap, w}; head[u] = ecnt++;
9      e[ecnt] = {head[v], u, flow, 0, -w}; head[v] = ecnt++;
10 }
11 bool spfa(int s, int t) {
12     std::fill(vis + s, vis + t + 1, 0);
13     std::fill(dis + s, dis + t + 1, inf);
14     std::queue<int> q;
15     q.push(s);
16     dis[s] = 0;
17     vis[s] = 1;
18     while (!q.empty()) {

```

```

19     int u = q.front();
20     q.pop();
21     vis[u] = 0;
22     for (int i = head[u]; i != -1; i = e[i].nxt) {
23         int v = e[i].v;
24         if (e[i].flow < e[i].cap && dis[u] + e[i].w < dis[v]) {
25             dis[v] = dis[u] + e[i].w;
26             if (!vis[v]) vis[v] = 1, q.push(v);
27         }
28     }
29 }
30 return dis[t] != inf;
31 }
32 int dfs(int u, int a) {
33     if (vis[u]) return 0;
34     if (u == t || !a) return a;
35     vis[u] = 1;
36     int flow = 0, f;
37     for (int i = cur[u]; i != -1; i = e[i].nxt) {
38         int v = e[i].v;
39         if (dis[u] + e[i].w == dis[v] && (f = dfs(v, std::min(a, e[i].cap - e[
40             i].flow))) > 0) {
41             e[i].flow += f;
42             e[i ^ 1].flow -= f;
43             flow += f;
44             mincost += e[i].w * f;
45             a -= f;
46             if (!a) break;
47         }
48     }
49     vis[u] = 0;
50     return flow;
51 }

```

1.8.3 最大闭权子图

正权点向 S 连边，负权点向 T 连边。边权为点权的绝对值。原图的边容量设为 INF。

则最大收益为 $\sum_{v>0} v - mincost$

在最大闭权子图中的点是残量网络中 S 能到达的点。

1.9 树哈希

```
1  const ull mask = chrono::steady_clock::now().time_since_epoch().count();
2
3  ull shift(ull x) {
4      x ^= mask;
5      x ^= x << 13;
6      x ^= x >> 7;
7      x ^= x << 17;
8      x ^= mask;
9      return x;
10 }
11 int n;
12 ull H[N];
13 vector<int> G[N];
14 set<ull> s;
15
16 void dfs(int u, int fa) {
17     H[u] = 1;
18     for (int v : G[u]) {
19         if (v == fa) continue;
20         dfs(v, u);
21         H[u] += shift(H[v]);
22     }
23     s.emplace(H[u]);
24 }
```

1.10 强联通分量

```
1  int n, dfc, dfn[N], low[N], stk[N], top, idx[N], in_stk[N], scc_cnt;
2  vector<int> G[N];
3
4  void tarjan(int u) {
5      low[u] = dfn[u] = ++dfc;
6      stk[++top] = u;
7      in_stk[u] = 1;
8      for (int v : G[u]) {
9          if (!dfn[v]) {
10             tarjan(v);
11             low[u] = min(low[u], low[v]);
12         } else if (in_stk[v]) low[u] = min(dfn[v], low[u]);
13     }
14     if (low[u] == dfn[u]) {
15         int x;
16         scc_cnt++;
```

```

17         do {
18             x = stk[top--];
19             idx[x] = scc_cnt;
20             in_stk[x] = 0;
21         } while (x != u);
22     }
23 }
24
25 // 多测清空
26 dfc = scc_cnt = top = 0;
27 for (int i = 1; i <= tot; i++) low[i] = dfn[i] = idx[i] = in_stk[i] = 0;

```

1.11 割点和桥

```

1 int dfn[N], low[N], dfs_clock;
2 bool iscut[N], vis[N];
3 void dfs(int u, int fa) {
4     dfn[u] = low[u] = ++dfs_clock;
5     vis[u] = 1;
6     int child = 0;
7     for (int v : e[u]) {
8         if (v == fa) continue;
9         if (!dfn[v]) {
10             dfs(v, u);
11             low[u] = min(low[u], low[v]);
12             child++;
13             if (low[v] >= dfn[u]) iscut[u] = 1;
14         } else if (dfn[u] > dfn[v] && v != fa) low[u] = min(low[u], dfn[v]);
15         if (fa == 0 && child == 1) iscut[u] = 0;
16     }
17 }

```

1.12 点双联通分量

```

1 int bccno[N], bcc_cnt, siz_e[N], siz_p[N], dfs_clock, low[N], dfn[N], top;
2 pair<int, int> stk[N];
3 void dfs(int u, int fa) {
4     low[u] = dfn[u] = ++dfs_clock;
5     for(int i = head[u]; i; i = e[i].nxt) {
6         int v = e[i].v;
7         if(v == fa) continue;
8         if(!dfn[v]) {
9             stk[++top] = make_pair(u, v);

```

```

10     dfs(v, u);
11     low[u] = min(low[u], low[v]);
12     if(low[v] >= dfn[u]) {
13         bcc_cnt++;
14         while(true) {
15             int x = stk[top].first, y = stk[top].second;
16             top--;
17             siz_e[bcc_cnt]++;
18             if(bccno[x] != bcc_cnt) {bccno[x] = bcc_cnt; siz_p[bcc_cnt]++;}
19             if(bccno[y] != bcc_cnt) {bccno[y] = bcc_cnt; siz_p[bcc_cnt]++;}
20             if(x == u && y == v) break;
21         }
22     }
23     } else if(dfn[v] < dfn[u]) {stk[++top] = make_pair(u, v); low[u] = min
        (low[u], dfn[v]);}
24 }
25 }

```

1.13 边双联通分量

```

1  const int N = 5000 + 5;
2  int n, m, stk[N], top, ccno, sc[N];
3  int dfn[N], dfc, low[N];
4  int mp[N][N];
5  int in[N];
6  int head[N], ecnt;
7  struct Edge {
8      int nxt, v;
9  } e[N << 2];
10 void add_edge(int u, int v) {
11     e[ecnt] = {head[u], v}; head[u] = ecnt++;
12     e[ecnt] = {head[v], u}; head[v] = ecnt++;
13 }
14 void dfs(int u, int from) {
15     stk[++top] = u;
16     low[u] = dfn[u] = ++dfc;
17     for (int i = head[u]; i != -1; i = e[i].nxt) {
18         int v = e[i].v;
19         if (!dfn[v]) {
20             dfs(v, i);
21             low[u] = min(low[u], low[v]);
22         } else if ((i ^ 1) != from) low[u] = min(low[u], dfn[v]);

```

```

23     }
24     if (dfn[u] == low[u]) {
25         ccno++;
26         int x;
27         while (true) {
28             x = stk[top--];
29             sc[x] = ccno;
30             if (x == u) break;
31         }
32     }
33 }
34
35 void solve() {
36     memset(head, -1, sizeof head);
37     scanf("%d %d", &n, &m);
38     for (int i = 1; i <= m; i++) {
39         int u, v;
40         scanf("%d %d", &u, &v);
41         add_edge(u, v);
42     }
43     for (int i = 1; i <= n; i++) if (!dfn[i]) dfs(i, i);
44     for (int i = 1; i <= n; i++) {
45         for (int k = head[i]; k != -1; k = e[k].nxt) {
46             int j = e[k].v;
47             if (sc[i] != sc[j]) mp[sc[i]][sc[j]] = 1;
48         }
49     }
50
51     for (int i = 1; i <= ccno; i++) {
52         for (int j = 1; j <= ccno; j++) if (mp[i][j]) in[j]++;
53     }
54     int cnt = 0;
55     for (int i = 1; i <= ccno; i++) if (in[i] == 1) cnt++;
56     printf("%d\n", (cnt + 1) / 2);
57 }

```

1.14 2-SAT

$2 * u$ 代表不选择, $2 * u + 1$ 代表选择。

也可以求强连通分量。

如果对于一个 x ‘ $sccno$ ’比它的反状态 $x * 1$ 的 ‘ $sccno$ ’ 要小, 那么我们用 x 这个状态当做答

案，否则用它的反状态当做答案。

```

1  vector<int> G[N * 2];
2  bool mark[N * 2];
3  int stk[N], top;
4  void build_G() {
5      for (int i = 1; i <= n; i++) {
6          int u, v;
7          G[2 * u + 1].push_back(2 * v);
8          G[2 * v + 1].push_back(2 * u);
9      }
10 }
11 bool dfs(int u) {
12     if (mark[u ^ 1]) return false;
13     if (mark[u]) return true;
14     mark[u] = 1;
15     stk[++top] = u;
16     for (int v : G[u]) {
17         if (!dfs(v)) return false;
18     }
19     return true;
20 }
21 bool 2_sat() {
22     for (int i = 1; i <= n; i++) {
23         if (!mark[i * 2] && !mark[i * 2 + 1]) {
24             top = 0;
25             if (!dfs(2 * i)) {
26                 while (top) mark[stk[top--]] = 0;
27                 if (!dfs(2 * i + 1)) return 0;
28             }
29         }
30     }
31     return 1;
32 }

```

2 数据结构

2.1 李超线段树

```

1  struct Line {
2      ll k, b;
3  } lin[N];
4  int lcnt;
5  int add_line(ll k, ll b) {

```

```

6     lin[++lcnt] = {k, b};
7     return lcnt;
8 }
9 struct node {
10     int ls, rs, u;
11 } tr[N << 2];
12 int tot;
13 ll calc(int u, ll x) {
14     return lin[u].k * x + lin[u].b;
15 }
16 bool cmp(int u, int v, ll x) {
17     return calc(u, x) <= calc(v, x); // 如果要求最大值，只需要修改为大于等于
18 }
19 void pushdown(int &p, int l, int r, int v) {
20     if (!p) p = ++tot;
21     if (l == r) return;
22     int mid = (l + r) >> 1;
23     int &u = tr[p].u, b = cmp(v, u, mid);
24     if (b) swap(u, v);
25     int bl = cmp(v, u, l), br = cmp(v, u, r);
26     if (bl) pushdown(tr[p].ls, l, mid, v);
27     if (br) pushdown(tr[p].rs, mid + 1, r, v);
28 }
29 void update(int &p, int l, int r, int L, int R, int v) {
30     if (l > R || r < L) return;
31     if (!p) p = ++tot;
32     int mid = (l + r) >> 1;
33     if (l >= L && r <= R) return pushdown(p, l, r, v), void();
34     update(tr[p].ls, l, mid, L, R, v);
35     update(tr[p].rs, mid + 1, r, L, R, v);
36 }
37 ll query(int p, int l, int r, ll pos) {
38     if (!p) return 1e16;
39     ll res = calc(tr[p].u, pos);
40     int mid = (l + r) >> 1;
41     if (l == r) return res;
42     if (pos <= mid) {
43         res = min(res, query(tr[p].ls, l, mid, pos));
44     } else res = min(res, query(tr[p].rs, mid + 1, r, pos));
45     return res;
46 }
47
48 int main() {
49     lin[0].b = 1e16;

```

```
50     return 0;
51 }
```

2.2 Link Cut Tree

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const int N = 1e5 + 5;
5  int n, m, ch[N][2], f[N], s[N], r[N], v[N];
6
7  #define lc ch[x][0]
8  #define rc ch[x][1]
9
10 bool noroot(int x) {
11     return ch[f[x]][1] == x || ch[f[x]][0] == x;
12 }
13 void pushup(int x) {
14     s[x] = s[lc] ^ s[rc] ^ v[x];
15 }
16 void pushr(int x) {
17     swap(lc, rc);
18     r[x] ^= 1;
19 }
20 void pushdown(int x) {
21     if (r[x]) {
22         if (lc) pushr(lc);
23         if (rc) pushr(rc);
24         r[x] = 0;
25     }
26 }
27 void rotate(int x) {
28     int y = f[x], z = f[y], k = (ch[y][1] == x), w = ch[x][k ^ 1];
29     if (noroot(y)) ch[z][y == ch[z][1]] = x;
30     ch[x][k ^ 1] = y;
31     ch[y][k] = w;
32     f[w] = y;
33     f[y] = x;
34     f[x] = z;
35     pushup(y), pushup(x);
36 }
37 void update(int x) {
38     if (noroot(x)) update(f[x]);
39     pushdown(x);
```

```
40 }
41 bool get(int x) {
42     return ch[f[x]][1] == x;
43 }
44 void splay(int x) {
45     update(x);
46     for (int fa; fa = f[x], noroot(x); rotate(x)) {
47         if (noroot(fa)) rotate(get(x) == get(fa) ? fa : x);
48     }
49     pushup(x);
50 }
51 void access(int x) {
52     int p;
53     for (p = 0; x; p = x, x = f[x]) {
54         splay(x), ch[x][1] = p, pushup(x);
55     }
56 }
57 void makeroot(int x) {
58     access(x); splay(x);
59     pushr(x);
60 }
61 int findroot(int x) {
62     access(x);
63     splay(x);
64     while (lc) pushdown(x), x = lc;
65     splay(x);
66     return x;
67 }
68 void split(int x, int y) {
69     makeroot(x);
70     access(y); splay(y);
71 }
72 void link(int x, int y) {
73     makeroot(x);
74     if (findroot(y) != x) f[x] = y;
75 }
76 void cut(int x, int y) {
77     makeroot(x);
78     if (findroot(y) == x && f[y] == x && !ch[y][0]) {
79         f[y] = ch[x][1] = 0;
80         pushup(x);
81     }
82 }
83
```

```

84 int main() {
85     scanf("%d %d", &n, &m);
86     for (int i = 1; i <= n; i++) scanf("%d", &v[i]);
87     while (m--) {
88         int opt, x, y;
89         scanf("%d %d %d", &opt, &x, &y);
90         if (opt == 0) split(x, y), printf("%d\n", s[y]);
91         if (opt == 1) link(x, y);
92         if (opt == 2) cut(x, y);
93         if (opt == 3) splay(x), v[x] = y, pushup(x);
94     }
95     return 0;
96 }

```

2.3 兔队线段树

求有多少个严格前缀最大值。

线段树保存每个区间为子问题时右部分的答案 res (可以不需要信息可减), 和区间的最大值 mx 。

$calc$ 考虑一段区间之前有 x 大的数时, 区间此时前缀最大数的数目。

1. $x \geq val[lson]$, $ans = calc(rson)$
2. $x < val[lson]$, $ans = calc(lson) + res[p]$

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4
5  const int N = 1e5 + 5;
6  #define lson (p << 1)
7  #define rson ((p << 1) | 1)
8  #define mid ((l + r) >> 1)
9  int n, m;
10 struct node {
11     int s, a, b;
12 } tr[N << 2];
13 bool cmp(int a, int b, int c, int d) {
14     if (d == 0 && b == 0) return 0;
15     if (d == 0 && a == 0) return 0;
16     if (d == 0) return 1;
17     return a * 1ll * d > c * 1ll * b;
18 }
19 int calc(int p, int l, int r, int c, int d) {

```

```

20     if (l == r)
21         return cmp(tr[p].a, tr[p].b, c, d);
22     if (cmp(tr[lson].a, tr[lson].b, c, d)) {
23         return calc(lson, l, mid, c, d) + tr[p].s;
24     }
25     return calc(rson, mid + 1, r, c, d);
26 }
27 void modify(int p, int l, int r, int pos, int v) {
28     if (l == r) {
29         tr[p] = {0, v, pos};
30         return;
31     }
32     if (pos <= mid) modify(lson, l, mid, pos, v);
33     else modify(rson, mid + 1, r, pos, v);
34     if (cmp(tr[lson].a, tr[lson].b, tr[rson].a, tr[rson].b)) {
35         tr[p] = tr[lson];
36     } else tr[p] = tr[rson];
37     tr[p].s = calc(rson, mid + 1, r, tr[lson].a, tr[lson].b);
38 }
39
40 int main() {
41     scanf("%d %d", &n, &m);
42     while (m--) {
43         int x, y;
44         scanf("%d %d", &x, &y);
45         modify(1, 1, n, x, y);
46         printf("%d\n", calc(1, 1, n, 0, 0));
47     }
48     return 0;
49 }

```

3 字符串

3.1 哈希

3.1.1 最长回文子串

通过哈希同样可以 $O(n)$ 解决这个问题，具体方法就是记 R_i 表示以 i 作为结尾的最长回文的长度，那么答案就是 $\max_{i=1}^n R_i$ 。考虑到 $R_i \leq R_{i-1} + 2$ ，因此我们只需要暴力从 $R_{i-1} + 2$ 开始递减，直到找到第一个回文即可。记变量 z 表示当前枚举的 R_i ，初始时为 0，则 z 在每次 i 增大的时候都会增大 2，之后每次暴力循环都会减少 1，故暴力循环最多发生 $2n$ 次，总的时间复杂

度为 $O(n)$ 。

3.2 字典树

3.3 维护异或和

```

1  const int N = 526010, MX = 22;
2  int ch[N * MX][2], tot, rt[N], w[N * MX], xorv[N * MX], val[N];
3  ll ans;
4
5  void pushup(int u) {
6      w[u] = xorv[u] = 0;
7      if (ch[u][0]) {
8          w[u] += w[ch[u][0]];
9          xorv[u] ^= (xorv[ch[u][0]] << 1);
10     }
11     if (ch[u][1]) {
12         w[u] += w[ch[u][1]];
13         xorv[u] ^= (xorv[ch[u][1]] << 1) | (w[ch[u][1]] & 1);
14     }
15     w[u] &= 1;
16 }
17 void insert(int &o, ll ux, int dep) {
18     if (!o) o = ++tot;
19     if (dep > MX) return (void)(w[o]++);
20     insert(ch[o][ux & 1], ux >> 1, dep + 1);
21     pushup(o);
22 }
23 void addall(int o) {
24     swap(ch[o][0], ch[o][1]);
25     if (ch[o][0]) addall(ch[o][0]);
26     pushup(o);
27 }
28 int merge(int a, int b) {
29     if (!b || !a) return a + b;
30     xorv[a] ^= xorv[b];
31     w[a] += w[b];
32     ch[a][0] = merge(ch[a][0], ch[b][0]);
33     ch[a][1] = merge(ch[a][1], ch[b][1]);
34     return a;
35 }
36
37 vector<int> G[N];
38 int read() {
39     int w = 0, f = 1; char ch = getchar();

```

```

40     while (ch > '9' || ch < '0') {
41         if (ch == '-') f = -1;
42         ch = getchar();
43     }
44     while (ch >= '0' && ch <= '9') {
45         w = w * 10 + ch - 48;
46         ch = getchar();
47     }
48     return w * f;
49 }
50
51 void dfs(int u) {
52     for (auto v : G[u]) {
53         dfs(v);
54         rt[u] = merge(rt[u], rt[v]);
55     }
56     addall(rt[u]);
57     insert(rt[u], val[u], 0);
58     ans += (ll)xorv[rt[u]];
59 }
60
61 int main() {
62     int n = read();
63     for (int i = 1; i <= n; i++) val[i] = read();
64     for (int i = 2; i <= n; i++) G[read()].push_back(i);
65     dfs(1);
66     printf("%lld\n", ans);
67     return 0;
68 }

```

3.4 KMP

```

1  int n = strlen(s + 1);
2  for (int i = 2; i <= n; i++) {
3      int j = k[i - 1];
4      while (j != 0 && s[i] != s[j + 1]) j = k[j];
5      if (s[i] == s[j + 1]) k[i] = j + 1;
6      else k[i] = 0;
7  }

```

3.4.1 字符串最小周期

设 border 长度为 r

则 $s[i] = s[n - r + i]$

$|T| = n - r$

3.4.2 每个前缀的出现次数

1. 统计每个前缀在自身的出现次数

```
1 vector<int> ans(n + 1);
2 for (int i = 1; i <= n; i++) ans[k[i]]++;
3 for (int i = n; i >= 1; i--) ans[k[i]] += ans[i];
4 for (int i = 1; i <= n; i++) ans[i]++;
```

2. 统计每个前缀在其他串的出现次数

我们应用来自 Knuth-Morris-Pratt 的技巧：构造一个字符串 $s + \# + t$ 并计算其前缀函数。与第一个问题唯一的不同之处在于，我们只关心与字符串 t 相关的前缀函数值，即 $i \geq n + 1$ 的 $\pi[i]$ 。有了这些值之后，我们可以同样应用在第一个问题中的算法来解决该问题。

3.4.3 一个字符串中本质不同子串的数目

给定一个长度为 n 的字符串 s ，我们希望计算其本质不同子串的数目。

我们将迭代的解决该问题。换句话说，在知道了当前的本质不同子串的数目的情况下，我们要找出一种在 s 末尾添加一个字符后重新计算该数目的方法。

令 k 为当前 s 的本质不同子串数量。我们添加一个新的字符 c 至 s 。显然，会有一些新的子串以字符 c 结尾。我们希望对这些以该字符结尾且我们之前未曾遇到的子串计数。

构造字符串 $t = s + c$ 并将其反转得到字符串 t^\sim 。现在我们的任务变为计算有多少 t^\sim 的前缀未在 t^\sim 的其余任何地方出现。如果我们计算了 t^\sim 的前缀函数最大值 π_{\max} ，那么最长的出现在 s 中的前缀其长度为 π_{\max} 。自然的，所有更短的前缀也出现了。

因此，当添加了一个新字符后新出现的子串数目为 $|s| + 1 - \pi_{\max}$ 。

所以对于每个添加的字符，我们可以在 $O(n)$ 的时间内计算新子串的数目，故最终复杂度为 $O(n^2)$ 。

值得注意的是，我们也可以重新计算在头部添加一个字符，或者从尾或者头移除一个字符时的本质不同子串数目。

3.5 AC 自动机

```

1 namespace AC {
2     int ch[N][26], tot, fail[N], e[N];
3     void insert(const char *s) {
4         int u = 0, n = strlen(s + 1);
5         for (int i = 1; i <= n; i++) {
6             if (!ch[u][s[i] - 'a']) ch[u][s[i] - 'a'] = ++tot;
7             u = ch[u][s[i] - 'a'];
8         }
9         e[u] += 1;
10    }
11    void build() {
12        queue<int> q;
13        for (int i = 0; i <= 25; i++) if (ch[0][i]) q.push(ch[0][i]);
14        while (!q.empty()) {
15            int now = q.front(); q.pop();
16            for (int i = 0; i < 26; i++) {
17                if (ch[now][i]) fail[ch[now][i]] = ch[fail[now]][i], q.push(ch
18                    [now][i]);
19                else ch[now][i] = ch[fail[now]][i];
20            }
21        }
22        int query(const char *s) {
23            int u = 0, n = strlen(s + 1), res = 0;
24            for (int i = 1; i <= n; i++){
25                u = ch[u][s[i] - 'a'];
26                for (int j = u; j && e[j] != -1; j = fail[j]) {
27                    res += e[j];
28                    e[j] = -1;
29                }
30            }
31            return res;
32        }
33    }

```

3.6 后缀数组

```

1 const int N = 2e5 + 5;
2 int sa[N << 1], ork[N << 1], rk[N << 1], cnt[N], id[N << 1], M, n;
3 char s[N];
4
5 int main() {

```

```

6   scanf("%s", s + 1);
7   n = strlen(s + 1);
8   for (int i = n + 1; i <= (n << 1); i++) s[i] = s[i - n], M = max(M, (int)s
    [i]);
9   n <<= 1;
10  for (int i = 1; i <= n; i++) if ((int)(s[i]) > M) M = (int)(s[i]);
11  for (int i = 1; i <= n; i++) cnt[rk[i] = s[i]]++;
12  for (int i = 0; i <= M; i++) cnt[i] += cnt[i - 1];
13  for (int i = n; i; i--) sa[cnt[rk[i]]--] = i;
14  for (int w = 1, p; w < n; w <<= 1, M = p) {
15      p = 0;
16      for (int i = n; i > n - w; i--) id[++p] = i;
17      for (int i = 1; i <= n; i++) if (sa[i] > w) id[++p] = sa[i] - w;
18      for (int i = 0; i <= M; i++) cnt[i] = 0;
19      for (int i = 1; i <= n; i++) cnt[rk[i]]++;
20      for (int i = 1; i <= M; i++) cnt[i] += cnt[i - 1];
21      for (int i = n; i; i--) sa[cnt[rk[id[i]]]--] = id[i];
22      p = 0;
23      for (int i = 0; i <= n; i++) ork[i] = rk[i];
24      for (int i = 1; i <= n; i++) {
25          if (ork[sa[i]] == ork[sa[i - 1]] && ork[sa[i] + w] == ork[sa[i -
            1] + w]) rk[sa[i]] = p;
26          else rk[sa[i]] = ++p;
27      }
28      if (p == n) break;
29  }
30  for (int i = 1, k = 0; i <= n; i++) {
31      if (rk[i] == 1) continue;
32      if (k) k--;
33      while (s[i + k] == s[sa[rk[i] - 1] + k]) k++;
34      h[rk[i]] = k;
35  }
36  return 0;
37 }

```

3.7 Manacher

对于第 i 个字符为对称轴:

1. 如果回文串长为奇数, $d[2 * i] / 2$ 是半径加上自己的长度
2. 如果长为偶数, $d[2 * i - 1] / 2$ 是半径的长度, 方向向右.

```

1  int n, d[N * 2];
2  char s[N];

```

```

3
4 for (int i = 1; i <= n; i++) t[i * 2] = s[i], t[i * 2 - 1] = '#';
5 t[n * 2 + 1] = '#';
6 m = n * 2 + 1;
7 for (int i = 1, l = 0, r = 0; i <= m; i++) {
8     int k = i <= r ? min(d[r - i + 1], r - i + 1) : 1;
9     while (i + k <= m && i - k >= 1 && t[i + k] == t[i - k]) k++;
10    d[i] = k--;
11    if (i + k > r) r = i + k, l = i - k;
12 }

```

3.8 Z 函数

$$z[i] = lcp(suf_1, suf_i)$$

```

1 for (int i = 2, l = 0, r = 0; i <= n; i++) {
2     if (r >= i && r - i + 1 > z[i - l + 1]) {
3         z[i] = z[i - l + 1];
4     } else {
5         z[i] = max(0, r - i + 1);
6         while (z[i] < n - i + 1 && s[z[i] + 1] == s[i + z[i]]) ++z[i];
7     }
8     if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
9 }

```

4 数学

4.1 基本预处理

```

1 int fpow(int a, int b) {
2     int res = 1;
3     for (; b; b >>= 1, a = a * 1ll * a % mod) if (b & 1) res = res * 1ll * a %
4         mod;
5     return res;
6 }
7 ll exgcd(ll a, ll b, ll &x, ll &y) {
8     if (b) {
9         ll d = exgcd(b, a % b, y, x);
10        return y -= a / b * x, d;
11    } else return x = 1, y = 0, a;
12 }
13 int getinv(int v) {
14     //return fpow(v, mod - 2);
15 }

```

```

14     // ll x, y;
15     // exgcd(v, mod, x, y);
16     // return (x % mod + mod) % mod;
17 }
18 int fac[N], ifac[N];
19 void init_binom(int n) {
20     fac[0] = ifac[0] = 1
21     for (int i = 1; i <= n; i++) fac[i] = fac[i - 1] * 1ll * i % mod;
22     ifac[n] = getinv(fac[n]);
23     for (int i = n; i > 1; i--) ifac[i - 1] = ifac[i] * 1ll * i % mod;
24 }
25 int binom(int a, int b) {
26     if (b < 0 || a < 0 || b > a) return 0;
27     return fac[a] * 1ll * ifac[b] % mod * ifac[a - b] % mod;
28 }
29 int getphi(int x) {
30     int res = 1;
31     for (int i = 2; i * i <= x; i++) {
32         if (x % i == 0) {
33             x /= i;
34             res *= (i - 1);
35             while (x % i == 0) {
36                 x /= i;
37                 res *= i;
38             }
39         }
40     }
41     if (x > 1) res *= (x - 1);
42     return res;
43 }
44 int prime[N], pcnt;
45 bool isp[N];
46 int get_prime(int n) {
47     for (int i = 2; i <= n; i++) {
48         if (!isp[i]) prime[++pcnt] = i;
49         for (int j = 1; j <= pcnt && i * prime[j] <= n; j++) {
50             isp[prime[j] * i] = 1;
51             if (i % prime[j] == 0) break;
52         }
53     }
54 }

```

4.2 线性基

```

1 struct LinerBasis {
2     int a[20], pos[20];
3     void add(int v, int p) {
4         for (int i = 19; i >= 0; i--) if ((v >> i) & 1) {
5             if (a[i]) {
6                 if (p > pos[i]) {
7                     swap(p, pos[i]);
8                     swap(a[i], v);
9                 }
10                v ^= a[i];
11            } else {
12                a[i] = v;
13                pos[i] = p;
14                return;
15            }
16        }
17    }
18 } b[N];
19
20 LinerBasis operator + (LinerBasis a, LinerBasis b) {
21     for (int i = 19; i >= 0; i--) {
22         if (b.a[i]) a.add(b.a[i], b.pos[i]);
23     }
24     return a;
25 }

```

4.3 高斯消元

4.3.1 解线性方程组

```

1 void gauss() {
2     for (int i = 0; i < n; i++) {
3         int id = i;
4         for (int j = i + 1; j < n; j++) if (fabs(a[j][i]) > fabs(a[id][i])) id = j;
5         for (int j = i; j <= n; j++) swap(a[id][j], a[i][j]);
6         if (a[i][i] == 0) {
7             puts("No Solution");
8             return;
9         }
10        for (int j = 0; j < n; j++) {
11            if (j == i) continue;
12            double t = a[j][i] / a[i][i];
13            for (int k = i; k <= n; k++) a[j][k] -= a[i][k] * t;

```

```

14     }
15 }
16 for (int i = 0; i < n; i++) printf("%.2lf\n", a[i][n] / a[i][i]);
17 }

```

4.3.2 求行阶梯矩阵

```

1 bool gauss() {
2     int k = 1;
3     for (int i = 1; i <= m; i++) {
4         if (k > n) break;
5         if (a[k][i] == 0) {
6             for (int j = k + 1; j <= n; j++) if (a[j][i] != 0) {
7                 for (int l = 1; l <= m + 1; l++) swap(a[j][l], a[k][l]);
8                 break;
9             }
10        }
11        if (a[k][i] == 0) continue;
12        for (int j = k + 1; j <= n; j++) if (a[j][i] == 1) {
13            for (int l = i; l <= m + 1; l++) a[j][l] ^= a[k][l];
14        }
15        k++;
16    }
17    int flag = 1;
18    for (int i = k; i <= n; i++) if (a[i][m + 1] == 1) flag = 0;
19    return flag;
20 }

```

4.4 矩阵树定理

一、无向无环图

A 为邻接矩阵, $A[i][j] = i \rightarrow j$ 的边数

D 为度数矩阵, $D[i][i] = \sum_{j=1}^n A[i][j] = i$ 的度数, 其他位置为 0

基尔霍夫矩阵 $K = D - A$, 令 $K' = K$ 的去掉第 k 行第 k 列 (k 任意) 的 $n - 1$ 阶主子式

$\det(K') =$ 该无向图生成树个数

特别地, 完全图生成树个数是 n^{n-2}

二、加权

求所有生成树边权的乘积之和，需要把邻接矩阵中边的条数改为为边权和

度数矩阵改为 $D[i][i] = \sum_{j=1}^n A[i][j]$

三、有向图

对于有根外向树，需要把度数矩阵改为入度和， $D[i][i] = \sum_{j=1}^n A[j][i]$

对于有根内向树，需要把度数矩阵改为出度和， $D[i][i] = \sum_{j=1}^n A[i][j]$

类似地，求所有有向生成树边权的乘积之和，需要把邻接矩阵改为入/出边边权和

四、变形：边权和的和

求所有生成树边权和的和，给原先边权为 w 的边赋值为一次多项式 $wx + 1$ ，多项式乘法对 x^2

取模， $\prod(w_i x + 1)$ 的一次项系数即为 w_i 之和

```

1 struct P {
2     ll x,y; //x是一次项系数, y是常数项
3     P (ll x=0, ll y=0):x(x),y(y){}
4     friend P operator + (const P &u, const P &v) {
5         return P(add(u.x, v.x), add(u.y, v.y));
6     }
7     friend P operator - (const P &u, const P &v) {
8         return P(add(u.x, mod - v.x), add(u.y, mod - v.y));
9     }
10    friend P operator * (const P &u, const P &v) {
11        return P(add(mul(u.x, v.y), mul(u.y, v.x)), mul(u.y, v.y));
12    }
13    friend P operator / (const P &u, const P &v) {
14        ll inv=qpow(v.y, mod-2);
15        return P(add(mul(u.x, v.y), mod - mul(u.y, v.x)) * inv % mod * inv %
16            mod, mul(u.y, inv));
17    }
18 }
19 P g[N][N];
20 ll gauss(P g[N][N], int n){
21     P res(0,1);
22     for(int i=1; i<=n; ++i) {
23         int pos=-1;
24         for(int j=i; j<=n; ++j){
25             if(g[j][i].y){
26                 pos=j; break;
27             }
28         }
29     }
30 }

```



```

27     }
28     if(pos==-1)return 0;
29     swap(g[i],g[pos]);
30     if(pos!=i)res=res*P(0,mod-1);
31     res=res*g[i][i];
32     P inv=P(0,1)/g[i][i];
33     for(int j=i+1;j<=n;++j){
34         P t=g[j][i]*inv;
35         for(int k=n;k>=i;--k)g[j][k]=g[j][k]-t*g[i][k];
36     }
37 }
38 return res.x;
39 }

```

4.5 多项式

4.5.1 FFT

```

1  typedef long long ll;
2  typedef complex<double> cp;
3  const int N = 6e5 + 5;
4  const double pi = acos(-1.0);
5  int n, m, len = 1, l, rev[N], x[N], y[N];
6  cp a[N * 2], b[N];
7
8  void fft(cp *a, int n, int inv) {
9      for (int i = 0; i < n; i++) if (rev[i] < i) swap(a[i], a[rev[i]]);
10     for (int k = 1; k < n; k <= 1) {
11         cp wn(cos(pi / k), inv * sin(pi / k));
12         for (int i = 0; i < n; i += k * 2) {
13             cp w(1, 0);
14             for (int j = 0; j < k; j++, w *= wn) {
15                 cp x = a[i + j], y = a[i + j + k] * w;
16                 a[i + j] = x + y, a[i + j + k] = x - y;
17             }
18         }
19     }
20     if (inv < 0) for (int i = 0; i < len; i++) a[i] /= n;
21 }

```

4.6 组合数学

4.6.1 小球放盒

第二类斯特林数（斯特林子集数） $\left\{ \begin{matrix} n \\ k \end{matrix} \right\}$ ，也可记做 $S(n, k)$ ，表示将 n 个两两不同的元素，划分为 k 个互不区分的非空子集的方案数。

$$\left\{ \begin{matrix} n \\ k \end{matrix} \right\} = \left\{ \begin{matrix} n-1 \\ k-1 \end{matrix} \right\} + k \left\{ \begin{matrix} n-1 \\ k \end{matrix} \right\}$$

边界是 $\left\{ \begin{matrix} n \\ 0 \end{matrix} \right\} = [n = 0]$ 。

假设小球个数为 n ，盒子个数为 m

1. 小球无标号，盒子有标号，不允许空盒。

即求解方程 $\sum_{i=1}^m x_i = n$ 解的个数

即 $\binom{n-1}{m-1}$

2. 小球无标号，盒子有标号，允许空盒。

令 $y_i = x_i + 1$

即求解方程 $\sum_{i=1}^m y_i = n$ 解的个数

即 $\binom{n+m-1}{m-1}$

3. 小球有标号，盒子有标号，允许空盒。

即 m^n

4. 小球有标号，盒子有标号，不允许空盒。

$$m! \times \left\{ \begin{matrix} n \\ m \end{matrix} \right\}$$

5. 小球有标号，盒子无标号，不允许空盒。

$$\left\{ \begin{matrix} n \\ m \end{matrix} \right\}$$

6. 小球有标号，盒子无标号，允许空盒。

$$\sum_{i=1}^m \left\{ \begin{matrix} n \\ i \end{matrix} \right\}$$

7. 小球无标号，盒子无标号，允许空盒。

设 $f[i][j]$ 表示 i 个球放入 j 个盒子的方案数。

1. $i = 0$ 或者 $j = 1$, 方案数为 1

2. $i < j$, $f[i][j] = f[i][i]$

3. $i \geq j$, $f[i][j] = f[i-j][j] + f[i][j-1]$

8. 小球无标号，盒子无标号，不允许空盒。

用 7 的结论，提前在每个盒子放 1 个球。

方案数就是 $f[n-m][m]$

5 MISC

5.1 完全平方数判断

```

1  typedef unsigned long long ull;
2  int sqrt1(ull x) {
3      ull y = sqrt(x);
4      return y * y == x;
5  }
6  constexpr ull calc_table(int k) {
7      ull table = 0;
8      for (int i = 0; i < 64; i++)
9          table |= 1ull << (i * i % (1 << k));
10     return table;
11 }
12 int sqrt4(ull x) {
13     constexpr int k = 6;
14     constexpr auto table = calc_table(k);
15     ull y = x % (1 << k);
16     if ((table >> y) & 1) return sqrt1(x);
17     return 0;
18 }

```

5.2 维护区间 GCD 值

```

1  int main() {

```

```
2     for (int i = 1; i <= n; i++) {
3         v.push_back({i, a[i]});
4         for (int j = (int)(v.size()) - 2; j >= 0; j--) {
5             v[j].second = gcd(v[j].second, a[i]);
6             if (v[j].second == v[j + 1].second) v.erase(v.begin() + j + 1);
7         }
8         mp[v[(int)(v.size()) - 1].second] += i - v[(int)(v.size()) - 1].first
          + 1;
9         for (int j = (int)(v.size()) - 2; j >= 0; j--) {
10             mp[v[j].second] += v[j + 1].first - v[j].first;
11         }
12     }
13 }
```