GESP C++ 六级认证（一）

树结构与编码 - 代码实战

哈夫曼树 & 二叉树 & 编码

相关题目

- P5657 [CSP-S 2019] 格雷码
- P10109 [GESP202312 六级] 工作沟通
- P14076 [GESP202509 六级] 货物运输
- P10722 [GESP202406 六级] 二叉树

P5657 [CSP-S 2019] 格雷码 题解

题目分析

这道题要求我们根据给定的编号 k，找出对应的 n 位格雷码。格雷码的特点是相邻的两个二进制串只有一位不同，且第一个和最后一个也相邻。

思路

图片中的代码使用了格雷码的一个**数学性质**：

n 位格雷码的第 k 个码字可以通过公式 $k \oplus \lfloor k/2 \rfloor$ 计算得到

其中 $\oplus$ 表示按位异或运算。

为什么这个公式有效?

让我们通过例子来理解：对于 3 位格雷码（$n = 3$）：

- $k = 0$: $0 \oplus 0 = 0$ → 000
- $k = 1$: $1 \oplus 0 = 1$ → 001
- $k = 2$: $2 \oplus 1 = 3$ → 011
- $k = 3$: $3 \oplus 1 = 2$ → 010
- $k = 4$: $4 \oplus 2 = 6$ → 110
- $k = 5$: $5 \oplus 2 = 7$ → 111
- $k = 6$: $6 \oplus 3 = 5$ → 101
- $k = 7$: $7 \oplus 3 = 4$ → 100

```cpp
int n;
ull k;
int main() {
    ios::sync_with_stdio(false), cin.tie(nullptr);
    cin >> n >> k;
    // 公式法构造 k ^= k-1
    k ^= k >> 1;
    while (n)
        cout << ((k >> (n - 1)) & 1), n--;
    return 0;
}
```

为什么使用 `unsigned long long`？

- 题目数据范围：$1 \le n \le 64, 0 \le k < 2^n$
- 当 $n = 64$ 时，k 最大为 $2^{64} - 1$，这超出了普通 `long long` 的范围
- `unsigned long long` 可以表示 0 到 $2^{64} - 1$ 的整数

P10109 [GESP202312 六级] 工作沟通

题意

某公司有 N 名员工，编号从 0 到 $N-1$，其中 0 号员工是老板。除老板外，每个员工都有一个直接领导，形成树形结构。员工 x 可以管理员工 y 当且仅当 x 是 y 的祖先（包括 y 自己）。现在有 Q 场合作，每场合作有 m 名员工参与，需要找出一名能够管理所有参与员工的主持人（即所有参与员工的公共祖先），如果有多个满足条件的主持人，选择编号最大的那个。

数据范围：$3 \leq N \leq 300$，$Q \leq 100$，$2 \leq m \leq N$。

分析

本题的核心是求多个节点的公共祖先。公司结构是一棵以 0 号员工为根的树，每个员工的管理者就是其祖先节点（包括自己）。

关键观察：

- 员工 x 能够管理员工 y 当且仅当 x 是 y 的祖先
- 因此，主持人必须是所有参与员工的公共祖先
- 需要找出所有公共祖先中编号最大的

算法思路：

1. 预处理每个员工的父节点信息
2. 对于每次查询：
    - 先找出第一个员工的所有祖先（包括自己）
    - 依次与其他员工求祖先集合的交集
    - 在最终的交集中选择编号最大的员工作为主持人

正确性证明：

- 初始时，第一个员工的所有祖先都是候选集合
- 每次与其他员工求交集，保留同时是两者祖先的节点
- 最终集合中的节点都是所有参与员工的公共祖先
- 选择编号最大的满足题目要求

时间复杂度：

- 每次查询需要遍历最多 m 个员工，每个员工最多有 N 个祖先
- 总时间复杂度为 $O(Q \cdot m \cdot N)$，在给定数据范围内完全可行

参考代码

```
const int N = 1005;
int n, q, fa[N], query[N], m;

int main() {
    ios::sync_with_stdio(false), cin.tie(nullptr);
    cin >> n;
    fa[0] = -1; // 老板没有父节点
    // 读入每个员工的直接领导
    for (int i = 1; i < n; i++)
        cin >> fa[i];

    cin >> q;
    while (q--) {
        cin >> m;
        // 读入本次合作的所有员工
        for (int i = 0; i < m; i++)
            cin >> query[i];

        bool vis[N] = {0}; // 标记公共祖先集合
        // ...
```

```
        // 初始化：第一个员工的所有祖先都是候选
        int now = query[0];
        while (now != -1) {
            vis[now] = 1;
            now = fa[now];
        }

        // 与其他员工求交集
        for (int i = 1; i < m; i++) {
            bool tmp[N] = {0}; // 当前员工的祖先集合
            now = query[i];
            while (now != -1) {
                tmp[now] = 1;
                now = fa[now];
            }
            // 取交集：只有同时在两个集合中的节点才保留
            for (int j = 0; j < n; j++)
                vis[j] = vis[j] && tmp[j];
        }

        // 找编号最大的公共祖先
        int ans = -1;
        for (int j = 0; j < n; j++)
            if (vis[j] && j > ans)
                ans = j;

        cout << ans << endl;
    }
    return 0;
}
```

P10722 [GESP202406 六级] 二叉树

题意

给定一棵 n 个节点的二叉树，根节点编号为 1。每个节点初始为黑色或白色（用 01 串表示）。进行 q 次操作，每次操作选择一个节点，将以该节点为根的子树内所有节点的颜色反转（黑色变白色，白色变黑色）。求 q 次操作后每个节点的颜色。

数据范围：$n, q \leq 10^5$。

分析

1. **问题转化**：每次操作是对子树进行颜色翻转，这相当于对子树中所有节点的翻转次数加 1。由于颜色翻转具有周期性（翻转两次等于不翻转），我们只关心每个节点被翻转的总次数的奇偶性。
2. **观察**：如果直接模拟每次操作，时间复杂度为 $O(n \times q)$，无法通过。需要优化。
3. **优化思路**：利用树的性质，记录每个节点被直接操作的次数，然后通过 DFS 将操作信息传递给子节点。具体来说：
   - 用数组 `flag` 记录每个节点被直接操作的次数（模 2）
   - 从根节点开始 DFS，如果当前节点有翻转标记，就将标记传递给它的左右子节点
   - 最终每个节点的颜色 = 初始颜色 $\oplus$ 该节点的翻转标记

4. **正确性证明**：

- 对于任意节点，它的最终翻转次数等于从根节点到该节点路径上所有节点的直接操作次数之和（模 2）
- 在 DFS 过程中，当访问到节点 x 时：
  - 如果 `flag[x] = 1`，说明需要翻转以 x 为根的子树
  - 将标记传递给子节点，相当于记录"从根节点到当前路径上已有奇数次翻转"
  - 这样，每个叶子节点最终得到的标记就是从根节点到该叶子节点路径上所有操作的总效果

时间复杂度：$O(n)$，其中建树 $O(n)$，DFS 遍历 $O(n)$。

参考代码

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

struct Node {
    int l, r;       // 左右子节点编号
    bool color;     // 节点初始颜色
} tree[N];

bool flag[N];       // 记录每个节点的翻转标记
int n, q;

// DFS遍历树，传递翻转标记
void dfs(int x) {
    int left = tree[x].l, right = tree[x].r;

    // 如果当前节点有翻转标记，传递给子节点
    if (flag[x]) {
        if (left) flag[left] ^= 1;   // 翻转左子树
        if (right) flag[right] ^= 1; // 翻转右子树
    }

    // 递归处理子树
    if (left) dfs(left);
    if (right) dfs(right);
}
// ...
```

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

    // 建树
    for (int i = 2, parent; i <= n; i++) {
        cin >> parent;
        if (!tree[parent].l)
            tree[parent].l = i;  // 第一个子节点作为左孩子
        else
            tree[parent].r = i;  // 第二个子节点作为右孩子
    }

    // 读入初始颜色
    char ch;
    for (int i = 1; i <= n; i++)
        cin >> ch, tree[i].color = (ch == '1');

    // 处理操作
    cin >> q;
    int x;
    for (int i = 0; i < q; i++)
        cin >> x, flag[x] ^= 1;  // 标记该节点被操作（异或实现模2计数）

    // DFS传递标记
    dfs(1);

    // 输出最终颜色：初始颜色 ⊕ 翻转标记
    for (int i = 1; i <= n; i++)
        cout << (tree[i].color ^ flag[i]);
    return 0;
}
```

GESP C++ 六级认证（一）

树结构与编码 - 代码实战