2022 ICPC 46th World Finals Problem P

2023 ICPC 47th World Finals Problem G

Turning Red

1. 题目翻译

问题： 变红 (Turning Red)

时间限制：3 秒

梅的父母在过去一年里重新装修了他们的房子，但他们的照明系统非常复杂！房子里的每个房间都有一个LED灯，可以设置为红色、绿色或蓝色，如图所示。

[图片]

房子各处有许多按钮，每个按钮都连接着一个或多个灯。当一个按钮被按下时，所有连接到该按钮的**红灯会变成绿灯，绿灯会变成蓝灯，而蓝灯则会变成红灯**。每个按钮可以被按多次。由于房子是在交叉开关发明之前建造的，每个灯最多由两个按钮控制。

梅最喜欢的颜色是红色，所以她想**把所有的灯都变成红色**。她的父母担心按钮会磨损，要求她**尽量减少按下按钮的总次数**。

输入

输入的第一行包含两个正整数 l 和 b，其中 l ($1 \leq l \leq 2 \cdot 10^5$) 是灯的数量，$b$ ($0 \leq b \leq 2 \cdot l$) 是按钮的数量。第二行输入一个长度为 l 的字符串，每个字符是 `R` (红色)、`G` (绿色) 或 `B` (蓝色)，表示第 i 个灯的初始颜色。接下来的 b 行描述这些按钮。每行首先是一个整数 k ($1 \leq k \leq l$)，表示该按钮控制的灯的数量。随后是 k 个不同的整数，表示该按钮控制的灯的编号。灯的编号从1到 l (包含1和 l)。每个灯在所有按钮中最多出现两次。

输出

输出梅将所有灯变成红色所需的最少按钮按动次数。如果无法将所有灯都变成红色，则输出 "impossible"。

样例输入 1

```
8 6
GBRBRRRG
2 1 4
1 2
4 4 5 6 7
3 5 6 7
1 8
1 8
```

样例输出 1

```
8
```

2. 参考代码 (C++) 与题解逻辑介绍

官方题解概要：

> 我们可以将三种颜色建模为：红色 = 0，绿色 = 1，蓝色 = 2。那么按下按钮的效果是，该按钮控制的所有灯的颜色值模3加1。
> 令 x_i 为我们按下按钮 i 的（未知）次数。那么，对于一个由按钮 i 和 j 控制的灯 L，其必须变为红色的要求变成了方程 $c_L + x_i + x_j \equiv 0 \pmod 3$，其中 c_L 表示该灯的初始颜色。注意，如果 x_i 或 x_j 中的任何一个已知，这个方程就能让我们立即计算出另一个的值。这意味着，如果我们考虑一个隐含的图，其中两个按钮 i 和 j 相连（如果它们控制同一个灯），那么基于一个按钮的 x_i 值，我们可以在其整个连通分量内传播并计算出其他值。
> 这引出了以下线性时间算法：对于图的每个连通分量，任选一个按钮 i，尝试所有3种可能的 $x_i = 0, 1, 2$ 值，并传播结果。如果发现不一致（某个方程不满足），则该 x_i 值无效。否则，检查此分量内按钮按下的总次数（该分量内所有 x_i 的总和）。如果所有3种 x_i 的选择都无效，则无解；否则，选择导致此分量内按钮按下次数最少的那个。
> 问题中也可能有些灯仅由单个按钮控制，这会导致形如 $c_L + x_i \equiv 0 \pmod 3$ 的方程，从而直接确定 x_i。可以特殊处理这些情况并首先传播它们的值，但更不容易出错的做法可能是在上述算法的不一致性检查中包含它们。

题解逻辑详细阐释：

核心思想是将问题转化为在模3算术下的方程求解，并结合图的连通性来优化搜索。

1. **颜色表示与按钮效果**：
   - **红色 (R)** $\rightarrow 0$
   - **绿色 (G)** $\rightarrow 1$
   - **蓝色 (B)** $\rightarrow 2$
   - **目标**：所有灯的颜色都为 0。
   - **每按一次按钮**，连接灯的颜色值 $(\text{当前值} + 1) \pmod 3$。
   - **对于按钮 i**，我们关心的是它被按下的次数 $x_i \pmod 3$。因此，我们只需考虑 $x_i \in \{0, 1, 2\}$。
2. **灯程**：
   - **设灯 L 的初始颜色为 c_L**。
   - **如果灯 L 不受任何按钮控制**：若 $c_L \neq 0$，则不可能变为红色，输出 "impossible"。
   - **如果灯 L 只受按钮 B_i 控制（按下 x_i 次）**：
     - $(c_L + x_i) \pmod 3 = 0 \implies x_i = (-c_L) \pmod 3$。
     - 这直接确定了 x_i 的值。
   - **如果灯 L 受按钮 B_i (按 x_i 次) 和 B_j (按 x_j 次) 控制**：
     - $(c_L + x_i + x_j) \pmod 3 = 0$。
     - 如果 x_i 和 x_j 中有一个已知，另一个就能确定。
3. **构建“按钮图”**：
   - **图中的节点是按钮**。
   - **如果两个按钮 B_i 和 B_j 共同控制至少一个灯**，就在它们之间连一条边。
4. **算法步骤**：

- **初始化**：
  - `x[button_idx]`：存储按钮 `button_idx` 的按动次数，初始为 -1 (未知)。
  - `is_fixed[button_idx]`：记录按钮的按动次数是否已确定。
  - `total = 0`。
- **处理单灯 (确定初始值)**：
  - 遍历所有灯。如果灯 L 只被按钮 B_i 控制：
    - 计算所需的 $x_i = (-c_L) \pmod 3$。
    - 如果 `x[B_i]` 未知，则设 `x[B_i] = x_i`，标记 `is_fixed[B_i] = true`。
    - 如果 `x[B_i]` 已知但与计算值不同，则说明矛盾，输出 "impossible"。
- **处理连通分量**：
  - 使用一个 `visited[button_idx]` 数组来确保每个按钮只被处理一次（作为某个连通分量的一部分）。
  - 遍历所有按钮 $s = 0 \dots (\text{num\_buttons} - 1)$：
    - **Case A: 按钮 s 的值已通过单灯固定 (`is_fixed[s] == true`) 且尚未访问 (`!visited[s]`)**：
      - 这个按钮属于一个“部分确定”或“完全确定”的连通分量。
      - 从 s 开始进行BFS/DFS遍历（只在按钮图上走）。
      - 将 s 加入队列，标记 `visited[s] = true`。`total += x[s]`。
      - 当从已确定值的按钮 u (值为 x_u) 访问到邻居 v (通过灯 L_{uv}连接)：
        - 如果 v 也已固定值 x_v：检查 $(c_{L_{uv}} + x_u + x_v) \pmod 3 == 0$ 是否成立。若不成立，则 "impossible"。
        - 如果 v 未固定：计算 $x_v = (-c_{L_{uv}} - x_u) \pmod 3$。设 x[v] = x_v，`is_fixed[v] = true`。total += x_v。将 v 入队并标记 `visited[v] = true`。
    - **Case B: 按钮 s 的值未 (`is_fixed[s] == false`) 且尚未访问 (`!visited[s]`)**：
      - 这标志着一个新的“完全不确定”连通分量的开始。
      - **1. 首先**，通过BFS/DFS找到这个连通分量中的所有按钮节点 (这些按钮的 `is_fixed` 都是 false)，记为 `nodes`。同时将它们标记为 `visited`。
      - **2. 选择** `nodes` 中的第一个按钮作为起点 `start_node`。
      - **3.** `min_cost_for_this_component = infinity`。
      - **4.** `best_values_for_component = null` (用来存储使成本最小的方案下，该分量各按钮的按键次数)。
      - **5. 尝试三种可能**：对 `start_node` 尝试按动次数 `try_x_start = 0, 1, 2`。
        - **对于每种** `try_x_start`：
          - `current_component_cost = try_x_start`。
          - `temp_x_values = {}` (一个映射，存储此尝试下分量内按钮的临时按动次数)。
          - `temp_x_values[start_node] = try_x_start`
          - **使用BFS/DFS从** `start_node` **开始**，在 `nodes` 内部传播按动次数：
            - 当从按钮 u (临时值为 x'_u) 传播到邻居 v (通过灯 L_{uv} 连接)：

- 如果 v 也属于 `nodes` 且其值在 `temp_x_values` 中未知：计算 $x'_v = (-c_{L_{uv}} - x'_u) \pmod 3$。将其存入 `temp_x_values`，current_component_cost += x'_v，将 v 入队。
- 如果 v 属于 `nodes` 且其值在 `temp_x_values` 中已知 x''_v: 检查 $(c_{L_{uv}} + x'_u + x''_v) \pmod 3 == 0$ 是否成立。若不成立，则此次尝试 (`try_x_start`) 无效，跳出BFS，处理下一种 `try_x_start`。
- 如果 v **不属于** `nodes` (意味着 v 是一个已全局固定的按钮，其值为 `x[v]`): 检查 $(c_{L_{uv}} + x'_u + x[v]) \pmod 3 == 0$ 是否成立。若不成立，则此次尝试无效。

验证一致性：如果传播完成没有矛盾，还需验证所有涉及 `nodes` 中按钮的灯是否都满足条件。遍历所有灯 L_k：

- 计算灯 L_k 的两个（或一个）控制按钮的临时/固定按键次数总和 `effect_sum` (如果按钮在 `nodes` 内，用 `temp_x_values` 的值；否则用全局 `x` 的值)。
- 如果 $(c_{L_k} + \text{effect\_sum}) \pmod 3 \neq 0$，则当前 `try_x_start` 方案无效。

如果当前 `try_x_start` 方案有效且 `current_component_cost < min_cost_for_mponent`：

- 更新 `min_cost_for_this_component = current_component_cost`。
- 保存 `temp_x_values` 到 `best_values_for_component`。

6. 如果 `min_cost_for_this_component == infinity` (三种尝试都无效)，则输出 "impossible"。

7. `total += min_cost_for_this_component`。

8. 将 `best_values_for_component` 中的值更新到全局的 `x` 数组和 `is_fixed` 数组中。

检查所有灯是否变红：

所有按钮的按键次数 `x` 都确定后，最后再检查一次所有灯。对于每个灯 L_k：

- 计算其控制按钮的 `x` 值之和 `final_effect_sum`。
- 如果 $(c_{L_k} + \text{final\_effect\_sum}) \pmod 3 \neq 0$，输出 "impossible"。 (这一步理论上如果前面的逻辑都正确，应该是满足的，可作为校验)

输出 `total`。