FOI2023 图论 2 练习赛题解

张志心

Zhejiang University

2023 年 7 月 13 日

1. T1. 环城旅行
2. T2. 阳光运动
3. T3. 分头行动
4. T4. 逆向生长

T1. 环城旅行

简要题意:
给一张图，求最大极差环。(环的定义是边不能重复走，但点可以)
$1 \le n, m \le 10^5$

T1. 环城旅行 - Solution

- 结论：当两条边在 ** 同一个边双连通分量中 ** 时，它们可以在一个环上。

T1. 环城旅行 - Solution

- 结论：当两条边在 ** 同一个边双连通分量中 ** 时，它们可以在一个环上。
- 设 (x, y) 和 (u, v) 在同一个边双连通分量中。我们添加辅助点 z 和 w，并连接 $(x, z), (y, z), (u, w), (v, w)$，这样 z 和 w 也在这个分量中。根据定义，z 和 w 之间必有边不相交的两条路径，且这两条路径一定包含新加入的这四条边。这两条路径形成了一个环。将新加入的这四条边换成 (x, y) 和 (u, v)，仍形成环，即构造出方案。

T1. 环城旅行 - Solution

- 结论：当两条边在 ** 同一个边双连通分量中 ** 时，它们可以在一个环上。
- 设 (x, y) 和 (u, v) 在同一个边双连通分量中。我们添加辅助点 z 和 w，并连接 $(x, z), (y, z), (u, w), (v, w)$，这样 z 和 w 也在这个分量中。根据定义，z 和 w 之间必有边不相交的两条路径，且这两条路径一定包含新加入的这四条边。这两条路径形成了一个环。将新加入的这四条边换成 (x, y) 和 (u, v)，仍形成环，即构造出方案。
- 因此我们求出所有边双连通分量并找到极差最大分量即可。

- 结论：当两条边在 ** 同一个边双连通分量中 ** 时，它们可以在一个环上。
- 设 (x, y) 和 (u, v) 在同一个边双连通分量中。我们添加辅助点 z 和 w，并连接 $(x, z), (y, z), (u, w), (v, w)$，这样 z 和 w 也在这个分量中。根据定义，z 和 w 之间必有边不相交的两条路径，且这两条路径一定包含新加入的这四条边。这两条路径形成了一个环。将新加入的这四条边换成 (x, y) 和 (u, v)，仍形成环，即构造出方案。
- 因此我们求出所有边双连通分量并找到极差最大分量即可。
- 复杂度 $O(n + m)$。

T2. 阳光运动

简要题意:
给一张带边权的无向图，求从 0 出发并回到 0 ，经过每个点至少一次且长度在 $[L, R]$ 之间的路径数。
$n \le 14, m \le 50, L \le R \le 200.$

T2. 阳光运动 - Solution

- 测试点 $1 \sim 10$：暴力搜索；

T2. 阳光运动 - Solution

- 测试点 $1 \sim 10$：暴力搜索；
- 设 $f(S, L, u) =$ 已经过的点集为 S，经过的长度为 L，当前节点为 u 的方案数。

- 测试点 $1 \sim 10$：暴力搜索；
- 设 $f(S, L, u) =$ 已经过的点集为 S，经过的长度为 L，当前节点为 u 的方案数。

$$f(s, L, u) \to f(s \cup \{v\}, L + w(u, v), v), u \in S \tag{1}$$

- 测试点 $1 \sim 10$：暴力搜索；
- 设 $f(S, L, u) =$ 已经过的点集为 S，经过的长度为 L，当前节点为 u 的方案数。

$$f(s, L, u) \rightarrow f(s \cup \{v\}, L + w(u, v), v), u \in S \tag{1}$$

- 时间复杂度 $O(2^n mR)$，空间复杂度 $O(2^n nR)$。

注意 dp 的枚举顺序，不好的顺序往往导致效率变低。

T2. 阳光运动 - Code

```
int all = (1<<(n+1)) - 1;
f[1][0][0] = 1;
for(int S = 1; S <= all; ++S)
    for(int i = 0; i <= R; ++i)
        for(int u = 0; u <= n; ++u) if(f[S][i][u])
        for(auto& [v,w] : e[u]) if(i + w <= R)
            inc(f[S|1<<v][i+w][v], f[S][i][u]);
int ans = 0;
for(int i = L; i <= R; ++i)
inc(ans, f[all][i][0]);
```

T3. 分头行动

简要题意:
一棵树有 n 个节点，两人分别选择其中 m_1, m_2 个节点，每次可以移动一个人从一个点到其相邻节点，保证两人距离始终不超过 d。求使得两个人从 1 号点开始各自访问完所有他们一定要访问的节点之后回到 1 号节点的最小步数。
$n \leq 5 \times 10^5$.

T3. 分头行动 - 部分分

- $n, m_1, m_2 \le 12$：暴力搜索；

T3. 分头行动 - 部分分

- $n, m_1, m_2 \le 12$：暴力搜索；
- $n \le 1000, m_1, m_2 \le 5$，此时需要访问的点很少，因此两个人的路径相对确定，可以通过观察样例找到规律，并结合适当搜索得到答案；

- $n, m_1, m_2 \le 12$：暴力搜索；
- $n \le 1000, m_1, m_2 \le 5$，此时需要访问的点很少，因此两个人的路径相对确定，可以通过观察样例找到规律，并结合适当搜索得到答案；
- $\mathrm{d} = 1$：两个人必须始终呆在一起，除非当前节点的儿子节点是一个人要访问的，而另一个人不用访问，这个时候可以只需要挪动一个人。

T3. 分头行动 - 部分分

- $n, m_1, m_2 \le 12$：暴力搜索；
- $n \le 1000, m_1, m_2 \le 5$，此时需要访问的点很少，因此两个人的路径相对确定，可以通过观察样例找到规律，并结合适当搜索得到答案；
- $\mathsf{d} = 1$：两个人必须始终呆在一起，除非当前节点的儿子节点是一个人要访问的，而另一个人不用访问，这个时候可以只需要挪动一个人。
- $\mathsf{d} = \mathsf{n}$：两个人分开算。把一个人需要经过的点的集合按照 dfs 序排好，在 dfs 的时候计算路径和。

T3. 分头行动 - Solution1

- 设把所有需要经过的点按照 dfs 序排序。

T3. 分头行动 - Solution1

- 设把所有需要经过的点按照 dfs 序排序。
- 设当前两人分别位于 u 和 v，假设下一步第一个人要去 t。

- 设把所有需要经过的点按照 dfs 序排序。
- 设当前两人分别位于 u 和 v，假设下一步第一个人要去 t。
  1. 若 $dis(v, t) < d$，则第二个人无需移动，第一个人自己走到 t 即可。

T3. 分头行动 - Solution1

- 设把所有需要经过的点按照 dfs 序排序。
- 设当前两人分别位于 u 和 v，假设下一步第一个人要去 t。
  1. 若 $dis(v,t) < d$，则第二个人无需移动，第一个人自己走到 t 即可。
  2. 否则分类讨论：
     - 若 v 是 t 的祖先，则第二个人走到 t 的 d 级祖先；

T3. 分头行动 - Solution1

- 设把所有需要经过的点按照 dfs 序排序。
- 设当前两人分别位于 u 和 v，假设下一步第一个人要去 t。
  1. 若 $dis(v,t) < d$，则第二个人无需移动，第一个人自己走到 t 即可。
  2. 否则分类讨论：
     - 若 v 是 t 的祖先，则第二个人走到 t 的 d 级祖先；
     - 若 t 是 v 的祖先，则第二个人走到 v 的 $dis(v,t) - d$ 级祖先；

T3. 分头行动 - Solution1

- 设把所有需要经过的点按照 dfs 序排序。
- 设当前两人分别位于 u 和 v，假设下一步第一个人要去 t。
  1. 若 $dis(v,t) < d$，则第二个人无需移动，第一个人自己走到 t 即可。
  2. 否则分类讨论：
     - 若 v 是 t 的祖先，则第二个人走到 t 的 d 级祖先；
     - 若 t 是 v 的祖先，则第二个人走到 v 的 $dis(v,t)-d$ 级祖先；
     - 若二者无祖孙关系，则讨论 v 和 t 到 $lca(v,t)$ 的距离；

- 设把所有需要经过的点按照 dfs 序排序。
- 设当前两人分别位于 u 和 v，假设下一步第一个人要去 t。
  1. 若 $dis(v, t) < d$，则第二个人无需移动，第一个人自己走到 t 即可。
  2. 否则分类讨论：
     - 若 v 是 t 的祖先，则第二个人走到 t 的 d 级祖先；
     - 若 t 是 v 的祖先，则第二个人走到 v 的 $dis(v, t) - d$ 级祖先；
     - 若二者无祖孙关系，则讨论 v 和 t 到 $lca(v, t)$ 的距离；
     - 若 $dis(t, lca(v, t)) > d$，则同样走到 t 的 d 级祖先，否则走到 v 的 $dis(v, t) - d$ 级祖先。

复杂度 $O(n \log n)$。

- 根据提示，如果一个人需要到达 x，如果 x 的深度超过了 d，那么另一个人一定要到达 x 的 d 级祖先。(反之显然不对)。

- 根据提示，如果一个人需要到达 x，如果 x 的深度超过了 d，那么另一个人一定要到达 x 的 d 级祖先。(反之显然不对)。
- 一个人经过一条边的次数要么是 2 次（来回各一次)，要么是 0 次。
- 两个人按照 dfs 序依次访问所有他们一定要访问的点即可。

- 根据提示，如果一个人需要到达 x，如果 x 的深度超过了 d，那么另一个人一定要到达 x 的 d 级祖先。(反之显然不对)。
- 一个人经过一条边的次数要么是 2 次（来回各一次），要么是 0 次。
- 两个人按照 dfs 序依次访问所有他们一定要访问的点即可。
- 一定要访问的点：自己本身要访问的点 + 对方要访问的点的 d 级祖先。

复杂度 $O(n\log n)$。

倍增求所有关键点的 d 级祖先加入到对方要访问的点中。令 $b[i], c[i] = 0/1$ 表示这个点是否要被第一个人/第二个人访问。
使用 dfs 求解该问题的答案如下：

```
int dfs2(int u, int f) {
    int ans = 0;
    for(int v : e[u]) if(v != f) {
        ans += dfs2(v, u);
        if(b[v]) ans += 2;
        if(c[v]) ans += 2;
        b[u] |= b[v];
        c[u] |= c[v];
    }
    return ans;
}
```

1. T1. 环城旅行
2. T2. 阳光运动
3. T3. 分头行动
4. T4. 逆向生长

简要题意:
给你一棵逆向生长的树（一开始只有叶子节点），有两个操作:
1. 添加一个节点，并且与其儿子节点连边;
2. 询问两个节点的 LCA。
强制在线。
$N, q \leq 2 \times 10^5$。（原本为 $N \leq 10^6$，标算复杂度为 $O(N \log N)$。）

使用并查集来维护当前两个点是否在一个连通块里，如果不在，答案为 0。

使用并查集来维护当前两个点是否在一个连通块里，如果不在，答案为0。

- 对于 $N, q \le 5 \times 10^3$，直接使用朴素求解 LCA 的方法即可；

使用并查集来维护当前两个点是否在一个连通块里，如果不在，答案为0。

- 对于 $N, q \leq 5 \times 10^3$，直接使用朴素求解 LCA 的方法即可；
- 对于 $N, q \leq 3 \times 4$，可以采用分块的思想，比如没过 $\sqrt{q}$ 次操作重新建一个倍增表。求解 LCA 的时候，首先现在倍增表上寻找，如果超过了倍增表，就用朴素方法向上寻找；

使用并查集来维护当前两个点是否在一个连通块里，如果不在，答案为 0。

- 对于 $N, q \leq 5 \times 10^3$，直接使用朴素求解 LCA 的方法即可；
- 对于 $N, q \leq 3 \times 4$，可以采用分块的思想，比如没过 $\sqrt{q}$ 次操作重新建一个倍增表。求解 LCA 的时候，首先现在倍增表上寻找，如果超过了倍增表，就用朴素方法向上寻找；
- A = 1, B = 0，此时可以使用离线做法，首先得到每个点出现的时间戳，那么可以直接求解 $LCA(x, y)$，如果 LCA 出现的时间比询问晚，那么答案为 0，否则为 LCA。

T4. 逆向生长 - Solution

动态维护倍增数组。

T4. 逆向生长 - Solution

动态维护倍增数组。

新加入一个点，我们要修改它的所有 2^k 级孩子节点 son 的 $fa[son][k]$。

动态维护倍增数组。
新加入一个点，我们要修改它的所有 2^k 级孩子节点 son 的 $fa[son][k]$。
开 $k \times \log(N)$ 个 vector 来 e[x][k] 来维护 x 的所有 k 级后代。

动态维护倍增数组。
新加入一个点，我们要修改它的所有 2^k 级孩子节点 son 的 $fa[son][k]$。
开 $k \times \log(N)$ 个 vector 来 e[x][k] 来维护 x 的所有 k 级后代。
维护方法：新加入一个节点 x，首先更新它的孩子节点 y 的 $fa[y][0]$，并且将 y 其加入 $e[x][0]$，然后更新所有 y 的 $e[y][0]$ 的节点 z 的 $fa[z][1]$，并且将 z 加入 $e[x][2]$ ……

动态维护倍增数组。

新加入一个点，我们要修改它的所有 2^k 级孩子节点 son 的 $fa[son][k]$。

开 $k \times \log(N)$ 个 vector 来 e[x][k] 来维护 x 的所有 k 级后代。

维护方法：新加入一个节点 x，首先更新它的孩子节点 y 的 $fa[y][0]$，并且将 y 其加入 $e[x][0]$，然后更新所有 y 的 $e[y][0]$ 的节点 z 的 $fa[z][1]$，并且将 z 加入 $e[x][2]$ ……

对于 $e[x][k]$ 中的节点 u，找到 $e[u][k]$ 中的节点 v，将其加入 $e[x][k+1]$，并且更新 $fa[v][k+1] = x$。

动态维护倍增数组。
新加入一个点，我们要修改它的所有 2^k 级孩子节点 son 的 $fa[son][k]$。
开 $k \times \log(N)$ 个 vector 来 e[x][k] 来维护 x 的所有 k 级后代。
维护方法：新加入一个节点 x，首先更新它的孩子节点 y 的 $fa[y][0]$，并且将 y 其加入 $e[x][0]$，然后更新所有 y 的 $e[y][0]$ 的节点 z 的 $fa[z][1]$，并且将 z 加入 $e[x][2]$ ……
对于 $e[x][k]$ 中的节点 u，找到 $e[u][k]$ 中的节点 v，将其加入 $e[x][k+1]$，并且更新 $fa[v][k+1] = x$。
复杂度 $O(n \log n)$。

代码实现

```
void dfs(int x, int y, int i) {
    for(auto son : e[y][i]) {
        fa[son][i + 1] = x;
        e[x][i + 1].push_back(son);
        dfs(x, son, i + 1);
    }
}
```