proj2 技术文档

陈永鸿

2025 年 12 月 23 日

笔者首先将会对解题思路进行简要概述，介绍使用的核心方法，随后对梳理算法流程，接着介绍各个函数的作用和使用的数据结构，最后说明实现难点。

1 解题思路概要

本题需要在无向图中找出任意两个顶点的 k 条最短路径，同时实现黑名单要求和中继点要求。

传统的 Dijkstra 算法只能找到单源最短路径，但因为其算法的贪心性质，无法直接满足 k 条路径的需求。为了解决这一问题，笔者参考了 Yen's 算法 (Yen, J. Y. (1971)) 算法来实现 k 条最短路径的查找。同时，为了加快运行效率，笔者调用 <queue> 库的优先队列，对 Dijkstra 算法进行了堆优化。

最后，为了满足黑名单要求，笔者使用一个全局禁用节点数组 `global_banned`，用于记录哪些节点被 ban。在 Dijkstra 算法和 Yen's 算法的路径搜索过程中，每次扩展到新节点时，都会判断该节点是否在禁用列表中（即 `global_banned[v]` 是否为真）。

为了满足中继点要求，笔者通过变量 `maxTransLimit` 控制路径中允许经过的最大边数。在 Dijkstra 算法的状态中，增加了“已用边数”这一维度，并在扩展新节点时判断当前路径的边数是否超过限制。如果超过，则不再继续扩展该路径。Yen's 算法在生成新路径时，也会根据当前已用边数和剩余可用边数，动态调整路径搜索的深度，确保所有输出路径都满足最大中转数的要求。

2 算法流程梳理

本文实现的 k 条最短路径算法基于 Yen's 算法，主要流程如下：

1. 假设给定起点 st 和终点 en，以及所需的最短路径条数 k，首先使用 Dijkstra 算法找到从 st 到 en 的第一条最短路径 P_1。
2. 如果要计算第 $i+1$ 条最短路径 P_{i+1}，对第 i 条最短路径 P_i，依次枚举 P_i 上除终点 en 外的每一个节点作为偏离节点 b：
   (a) 取 P_i 中从 st 到 b 的子路径作为根路径（root path）。
   (b) 临时禁用根路径上的其他节点，防止产生环路。
   (c) 禁用 P_i 在 b 处的出边，避免生成与已知路径重复的路径。
   (d) 从偏离节点 b 出发，使用 Dijkstra 算法计算 b 到 en 的最短路径（在上述禁用约束下）。
   (e) 若存在可行路径,则将根路径与偏离路径拼接,得到一条新的候选路径,并加入候选优先队列 $candidate$。
3. 若候选优先队列非空，则从中取出总代价最小的路径，作为新的最短路径 P_{i+1}，加入结果集合。
4. 重复上述过程，直到找到 k 条最短路径或候选优先队列为空为止。
5. 在整个过程中，始终保证每条新路径不与已找到的路径重复，且所有路径均满足黑名单和中继点等约束条件。

对于 Dijkstra 算法的实现，主要流程如下：

1. 计算本次搜索允许的最大边数 limit：若用户未设置 `edge_limit_override` 则使用全局 `maxTransLimit+1`（未设置时为节点数）。
2. 初始化距离数组 `dist[num_ver][limit+1]` 为大值，前驱数组 `parent[num_ver][limit+1]` 为默认值，起点状态 `dist[st][0]=0` 并将状态 `{st,0,0}` 压入优先队列（小根堆）。
3. 循环弹出优先队列中距离最小的状态 `{pos,dist,num_of_edges}`：
   (a) 若弹出状态的距离大于记录的 `dist[pos][num_of_edges]` 则跳过（过时状态）。
   (b) 若 `pos==en`，记录最终距离和边数并结束搜索（找到在限制内的最短路径）。
   (c) 若已用边数 `num_of_edges >= limit` 则不再扩展该状态。
   (d) 否则枚举所有邻边 `pos->v`：
   - 跳过被全局禁用或临时禁用的节点（但允许终点为例外），跳过被临时禁用的边。
   - 若通过该边可以在 `n+1` 条边内使 `dist[v][n+1]` 变小，则更新 `dist` 与 `parent`，并将新状态压入优先队列。
4. 若循环结束未找到终点，则返回不可达结果（`total_dist = -1`）。否则使用 `parent` 从终点回溯到起点构造路径节点序列，随后反转并返回路径结构体。

3 主要函数介绍与数据结构说明

- **Edge 结构体**：用于表示图中的一条边，包含两个成员变量：
  - `to`：目标节点编号（整数）。
  - `weight`：边的权重（整数）。
- **Path 结构体**：用于描述一条路径，包含：
  - `total_dist`：路径的总权重（整数）。
  - `vers`：路径经过的节点编号序列（整数向量）。
  - 重载了比较运算符（`<`、`>`、`==`），便于在优先队列中按照路径长度排序和判重。
- **State 结构体**：用于 Dijkstra 算法中的优先队列状态，包含：
  - `pos`：当前顶点的编号（整数）。
  - `dist`：当前到达该顶点的距离（整数）。
  - `num_of_edges`：到达该顶点时经过的边数（整数）。
  - 重载了比较运算符（`<`、`>`、`==`），便于优先队列按距离排序。
- **shortest_path 函数**：实现带节点禁用、边禁用和最大中转限制的 Dijkstra 算法。主要参数包括起点、终点、临时禁用节点数组、临时禁用边数组和中转限制。返回一条最短路径（Path 结构体）。
- **kShortestPaths 函数**：实现 Yen's 算法，输入起点、终点和 K，依次生成 K 条最短路径。内部维护已找到路径集合和候选路径优先队列，动态调整禁用节点和边，确保路径不重复且满足约束。
- **get_pos 函数**：根据节点名称（字符串）查找其在节点列表中的编号，便于字符串与编号的相互转换。
- **reverse_vec 函数**：对路径节点编号序列进行反转，用于还原从终点回溯得到的路径。

- **is_same_path 函数**：用于判断两条路径是否完全相同（用于路径去重）。实现方式为先比较路径长度，若长度不同直接返回不相同；若长度相同则逐一比较路径上的每个节点编号，全部相同则判为同一路径。
- **全局变量**：
  - `adj`：邻接表，存储每个节点的所有出边（`vector<Edge>` 数组）。
  - `ver_list`：节点名称列表（字符串数组），用于编号与名称的映射。
  - `global_banned`：全局禁用节点数组，记录哪些节点被 ban。
  - `maxTransLimit`：最大中转数限制，控制路径中允许经过的最大边数。
  - 节点数 `num_ver`、边数 `num_edge` 等。

4 难点与创新点说明

笔者将列出难点与对应的解决方案（创新点）：

1. **难点一：时间复杂度约束**
   本题要求在 3s 内输出要求路径，这对算法的时间复杂度有要求。笔者最先采用带剪枝的 DFS 算法，但在面临 15 个顶点的完全图时就已经超时。随后笔者参考了 Yen's 算法，并且将其中的 Dijkstra 算法进行了堆优化，最终算法的时间复杂度为 $O(KV(E+V)\log V)$。

2. **难点二：黑名单与中继点要求的实现**
   为了满足中继点要求，笔者通过变量 `maxTransLimit` 控制路径中允许经过的最大边数。在 Dijkstra 算法的状态中，增加了“已用边数”这一维度，并在扩展新节点时判断当前路径的边数是否超过限制。如果超过，则不再继续扩展该路径。Yen's 算法在生成新路径时，也会根据当前已用边数和剩余可用边数，动态调整路径搜索的深度，确保所有输出路径都满足最大中转数的要求。

3. **难点三：路径去重**
   在 Yen's 算法中，生成的候选路径可能会重复。为了避免重复路径的输出，笔者实现了 `is_same_path` 函数，用于判断两条路径是否完全相同（用于路径去重），实现方法已经在主要函数介绍中说明。

4. **难点四：生成路径出现遗漏**
   在实际写代码与 debug 的过程中，笔者发现在某些情况下，生成的候选路径会遗漏一些可能的路径。经过分析，发现这是原本设置了 maxTrans 限制时，Yen 算法中的分支搜索仍然会使用全部全局边预算运行 Dijkstra 算法。如果最短分支路径导致组合路径的边数超过允许的范围，则该路径会被丢弃，并且不会考虑任何替代分支路径（在剩余边预算范围内），这会导致错过有效的路径。
   修复方法：将每条分支路径的 Dijkstra 算法限制在剩余的边预算范围内（global_limit − edges_already_used_by_root）。同时，将此边预算传递给初始的最短路径调用（这就是为什么 shortest_path 函数要接受一个 edge_limit_override）。

5. **难点五：生成路径带环**
   在实现过程中，笔者发现生成的路径中会出现环路，导致路径不符合要求。经过分析，发现这是因为在 Yen's 算法中，临时禁用节点的实现不够完善，具体来说是没有禁止起点导致环路。
   修复方法：在 Yen's 算法中，除了禁用根路径上的其他节点外，还需要确保起点节点被禁用，防止路径回到起点形成环路。