$Proj_2$

22340048　区子楠

目录

1 解题思路

1.1 初始化图

Algorithm 1 Graph Initialization —— Main Function

1: **Function:** `initNewGraph(int vertex_num, int edge_num)`
2: **Step1:** 调用 `addVertices(vertex_num)` 得到顶点表
3: **Step2:** 初始化只有顶点的 `new_graph`
4: **Step3:** 调用 `addEdge(new_graph,edge_num)` 往图里面加边和权值
5: **Step4:** 返回 `new_graph`
6: **Return:** `AMGraph<T>* new_graph`

Algorithm 2 Add Edges

1: **Function:** `addEdge(AMGraph<T>* new_graph, int edge_num)`:
2: **Return:** `Nothing`

Algorithm 3 Add Vertices

1: **Function:** `addVertices(int vertex_num)`:
2: **Return:** `T* vertex_list`

1.2 交互界面逻辑实现

Algorithm 4 Type in Command until Type in "quit"

```
1: Initialize: string command
2: while true do
3:     command ← input from user
4:   if command = "quit" then
5:       Break
6:   else
7:       Solve(command, Graph)
8:   end if
9: end while
10: Output: "Program ended!"
```

1.3 输入命令/输出结果逻辑实现

Algorithm 5 Solve Command for Graph Operations —— Main Function

```
1: Input: string command, AMGraph<T>* Graph
2: if command = "ban" then
3:     ban_vertex ← input from user
4:     Graph->addVertexIntoBlackList(ban_vertex)
5:     Graph->printVertexInBlackList()
6: else
7:     if command = "unban" then
8:         unban_vertex ← input from user
9:         Graph->removeVertexOutOfBlackList(unban_vertex)
10:        Graph->printVertexInBlackList()
11:    else
12:        if command = "maxTrans" then
13:            max_trans_num ← input from user
14:            Graph->limitMaxTransitVertexAs(max_trans_num)
15:        else
16:            if command = "paths" then
17:                start_vertex, end_vertex ← input from user
18:                top_k_shortest_path ← input from user
19:                Graph->findTopKShortestPath(start_vertex,
   end_vertex, top_k_shortest_path)
20:                Graph->printFinalOutcome()
21:            else
22:                Print "Invalid command!"
23:            end if
24:        end if
25:    end if
26: end if
```

Algorithm 6 Add Vertex into Black List

1: **Input:** Ban_Vertex
2: `ban_vertex_index` ← `verList.locateEle(Ban_Vertex)`
3: **if** ban_vertex_index = -1 **then**
4: **Print** message.
5: **Return**
6: **else**
7: **Print** message.
8: **Put** Ban_Vertex into black list.
9: **Store** corresponding in/out vertex and edge weight.
10: **Ban** corresponding edge weight.
11: **end if**

Algorithm 7 Remove Vertex from Black List

1: **Input:** Unban_Vertex
2: `unban_vertex_index` ← `verList.locateEle(Unban_Vertex)`
3: **if** `unban_vertex_index = -1` **then**
4: **Print** message.
5: **Return**
6: **else**
7: **if** Unban_Vertex is not in black list **then**
8: **Print** message.
9: **Return**
10: **else**
11: **Recover** graph.
12: **Remove** Unban_Vertex out of black list.
13: **end if**
14: **end if**

Algorithm 8 Limit Max Transit Vertex Number

```
1: Input: Max_Trans_Num
2: if 0 <= Max_Trans_Num <= numVer-2 then
3:     Print message.
4:     Set max_transition_vertex_num in graph.
5: else
6:     if Max_Trans_Num = -1 then
7:         Print message.
8:         Unlimit max_transition_vertex_num in graph.
9:     else
10:        Print message.
11:    end if
12: end if
```

2 实现难点

2.1 找出前 K 条最短路径 (主要实现函数)

Algorithm 9 Find Top K Shortest Paths —— Main Function

1: **Input:** `Start_Vertex`, `End_Vertex`, `Top_K_Shortest_Paths`
2: **Initialize**
3: **Dijkstra** to find first shortest path.(实现难点)
4: **Put** first shortest path(if existed) into top k shortest path.
5: **Generate** candidate paths.(实现难点)
6: **while** counter < limitation **and** candidate paths is not empty **do**
7: **Remove** same path in candidate paths.
8: **Select** the shortest path in candidate paths.
9: **Remove** the select one in candidate paths.
10: **if** satisfy maxTrans **then**
11: **Insert** new shortest path into top k shortest path.
12: **end if**
13: **Generate** new candidate paths based on the select one above.
14: **Counter** ++.
15: **end while**
16: **Print** final outcome.
17: **Delete** memory.

2.2 有限制的 Dijkstra 算法 (实现难点)

Algorithm 10 Dijkstra with limitation —— 实现难点

Input: st, dist, preNode, isInS, Transit_Vertex_Counter
Get last shortest path and corresponding veclist.
Initialize basic var and array in Dijkstra.
while 还未遍历完 **do**
    for vertex in graph **do**
        the same things.
        Skip if the vertex is in black list.
        the same things.
    end for
    the same things.
    for vertex in graph **do**
        Skip if the vertex is in black list.
        the same things.
        Refresh transition vertex counter.
        Check whether can pass maxTrans.
        Refresh only when passing maxTrans.
    end for
end while

2.3 生成候选路径 (实现难点)

Algorithm 11 Generate Candidate Paths —— 实现难点

1: **Input:** Start_Vertex, End_Vertex, Top_K_Shortest_Paths, Candidates_Paths
2: **Get** last shortest path and corresponding veclist.
3: **Initialize** remove edges set and add edges used to be deleted.
4: **for** each vertex in last shortest path(except for end vertex) **do**
5: **Initialize** empty root path.
6: **Add** vertices to root path.
7: **Get** deviation vertex and remove edge.
8: **Remove** edges with same prefix.
9: **Dijkstra**(base on deviation vertex) to find another new path.
10: **Combine** new path with root path
11: **Add** new generate path to candidates paths based on weight.
12: **Delete** `memory`.
13: **end for**
14: **Recover** `graph`.

3 使用的数据结构

3.1 向量表

3.1.1 用于记录前 K 条最短路径的向量表

```
VecList<PathInformation*>* top_k_shortest_path;
```

3.1.2 利用 Dijkstra 算法找出的第一条最短路径

```
VecList<T>* first_shortest_path;
```

3.1.3 用于记录生成的所有候选路径的向量表

```
VecList<PathInformation*>* candidatePaths;
```

3.2 结构体

3.2.1 用于存放路径信息的结构体

```
struct PathInformation{
    // 路径总权值长度
    int path_length;
    // 路径上经过的顶点
    VecList<T>* path_go_through_vertices;
};
```

3.2.2 生成的所有候选路径中最短的路径

```
PathInformation* new_shortest_path
```

4 引用网站

Yen's Algorithm - 博客园 (主要参考)

前 K 条最短路径 - CSDN

前 K 条最短路径 - 博客园

Yen's Algorithm - Burning Bright