VE281 Project Five Report

Liu Yihao 515370910207

1 Appendix

1.1 The project files

1.1.1 main.cpp

```cpp
//
// Created by liu on 17-11-25.
//

#include <list>
#include <vector>
#include <queue>
#include <set>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <memory>
#include <algorithm>

class union_set {
    std::list<union_set *> children;
    union_set *parent = nullptr;

    union_set *find_ancestor() {
        if (parent) return parent->find_ancestor();
        return this;
    }

public:
    static bool merge(union_set *a, union_set *b) {
        auto _a = a->find_ancestor();
        auto _b = b->find_ancestor();
        if (_a == _b) return false;
        _a->children.emplace_back(_b);
        _b->parent = _a;
        return true;
    }
};

```

```
36  struct node {
37      std::vector<std::pair<node *, int>> neighbor;
38      bool visited = false;
39      size_t id;
40      int cost = -1;
41      node *predecessor = nullptr;
42      size_t degree = 0;
43      union_set set;
44
45      explicit node(size_t id) : id(id) { }
46
47      struct comp {
48          bool operator()(const node *a, const node *b) {
49              if (a->cost == b->cost) return a->id < b->id;
50              return a->cost < b->cost;
51          };
52      };
53  };
54
55  struct edge {
56      node *a, *b;
57      size_t weight;
58  };
59
60  struct graph {
61      std::vector<std::unique_ptr<node> > nodes;
62  };
63
64  void print_path(node *n) {
65      if (!n) return;
66      print_path(n->predecessor);
67      std::cout << n->id << " ";
68  }
69
70  int main(int argc, char *argv[]) {
71      std::fstream fin;
72      std::istringstream ss;
73      if (argc > 1) {
74          fin.open(argv[1]);
75          if (fin.is_open()) {
76              std::cin.rdbuf(fin.rdbuf());
77          }
78      }
79      size_t N, srcId, destId;
80      std::cin >> N >> srcId >> destId;
81
82      graph g{};
83      g.nodes.reserve(N);
84      for (int i = 0; i < N; ++i) {
```

```
85          g.nodes.emplace_back(std::make_unique<node>(i));
86      }
87
88      std::vector<edge> edgeQueue;
89
90      while (!std::cin.eof()) {
91          size_t src, dest, weight;
92          std::string str;
93          std::getline(std::cin, str);
94          if (str.empty()) continue;
95          ss.clear();
96          ss.str(str);
97          ss >> src >> dest >> weight;
98          auto srcNode = g.nodes[src].get();
99          auto destNode = g.nodes[dest].get();
100         srcNode->neighbor.emplace_back(destNode, weight);
101         ++destNode->degree;
102         edgeQueue.emplace_back(edge{srcNode, destNode, weight});
103     }
104
105     std::set<node *, node::comp> set;
106     std::queue<node *> queue;
107
108     for (auto &item : g.nodes) {
109         std::sort(item->neighbor.begin(), item->neighbor.end(),
110                   [](const std::pair<node *, int> &a, std::pair<node *, int> &b)
                      ↪ {
111                       return a.second < b.second;
112                   }
113         );
114         if (item->degree == 0) {
115             queue.push(item.get());
116         }
117     }
118
119
120     auto srcNode = g.nodes[srcId].get();
121     auto destNode = g.nodes[destId].get();
122
123     srcNode->cost = 0;
124     set.emplace(srcNode);
125     while (!destNode->visited && !set.empty()) {
126         auto now = *(set.begin());
127         set.erase(set.begin());
128         now->visited = true;
129         for (auto &item : now->neighbor) {
130             auto neighbor = item.first;
131             if (!neighbor->visited) {
132                 auto newCost = now->cost + item.second;
```

```
                if (neighbor->cost < 0) {
                    neighbor->cost = newCost;
                    neighbor->predecessor = now;
                    set.emplace(neighbor);
                } else if (neighbor->cost > newCost) {
                    set.erase(neighbor);
                    neighbor->cost = newCost;
                    neighbor->predecessor = now;
                    set.emplace(neighbor);
                }
            }
        }
    }

    if (destNode->visited && destNode->predecessor) {
        std::cout << "Shortest path length is " << destNode->cost << std::endl;
    } else {
        std::cout << "No path exists!" << std::endl;
    }

    std::list<node *> list;
    while (!queue.empty()) {
        auto now = queue.front();
        queue.pop();
        list.emplace_back(now);
        for (auto &item : now->neighbor) {
            auto neighbor = item.first;
            if (neighbor->degree > 0) {
                --neighbor->degree;
            }
            if (neighbor->degree == 0) {
                queue.push(neighbor);
            }
        }
    }

    if (list.size() == N) {
        std::cout << "The graph is a DAG" << std::endl;
    } else {
        std::cout << "The graph is not a DAG" << std::endl;
    }

    std::sort(edgeQueue.begin(), edgeQueue.end(), [](const edge &a, const edge
    ↪ &b) {
        return a.weight < b.weight;
    });

    size_t mst = 0;
    size_t node_count = 1;
```

```
    for (auto &now : edgeQueue) {
        if (union_set::merge(&(now.a->set), &(now.b->set))) {
            mst += now.weight;
            if (++node_count == N) break;
        }
    }

    if (node_count == N) {
        std::cout << "The total weight of MST is " << mst << std::endl;
    } else {
        std::cout << "No MST exists!" << std::endl;
    }

    return 0;
}
```

1.1.2 Makefile

```
all: main.cpp
    g++ -std=c++14 -O3 -o main main.cpp
```