Algorithm PA3

Chien-I, Tseng

B11901072

1 Problem Statement

This program solves the Minimum Feedback Arc Set problem of directed/undirected weighted graph.
To the case of the undirected graph, I use the concept of MST to solve it. To the case of the directed graph, I didn't finish the algorithm.

2 Algorithm Design

Maximum Spanning Tree

Like the MST in the textbook, I use Kruskal's algorithm to find the Spanning tree. But in the step of sorting the edge, I change the sorting process in decreasing order.

Space Complexity and Time Complexity

1. Space Complexity : O(V + E)
MST set and Vertex vector : O(V)
Edge vector : O(E)
2. Time Complexity : O(E log E)
sort : O(E log E)
find and union : almost O(1)

3 Data Structure

In ./src path, there are 2 code file: main.cpp and cyclebreak.cpp.
In this project, I don't want to split the data definition in .h file

Class Definition

Written in the file ./src/cyclebreak.h

Edge

```cpp
//In class Edge, there are three variant: source, target, weight
class Edge {
  public:
    int source;
    int target;
    int weight;
    Edge(int s = 0, int t = 0, int w = 0) : source(s), target(t), weight(w) {}
    //define operator to compare the weight of 2 different edges
    bool operator<(const Edge& other) const {
      if (weight != other.weight) return weight > other.weight;
      if (source != other.source) return source < other.source;
      return target < other.target;
    }
};
```

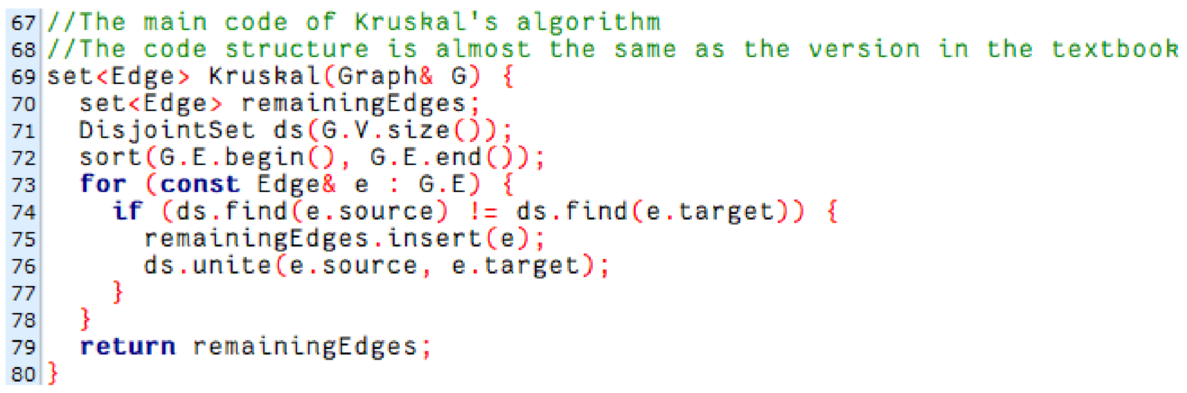
DisjointSet

```cpp
//class DisjointSet is created since Kruskal's algorithm needs it
class DisjointSet {
  private:
    vector<int> parent;
    vector<int> rank;
  public:
    DisjointSet(int size) {
      parent.resize(size);
      rank.resize(size, 0);
      for (int i = 0; i < size; i++) {
        parent[i] = i;
      }
    }
    //define a findset function to check whether they're in same set
    int find(int x) {
      if (parent[x] != x) {
        parent[x] = find(parent[x]);
      }
      return parent[x];
    }
    //define a union function to unite 2 vertices
    void unite(int x, int y) {
      int px = find(x);
      int py = find(y);
        if (px == py) return;
        if (rank[px] < rank[py]) {
          parent[px] = py;
        }else if (rank[px] > rank[py]) {
          parent[py] = px;
        }else {
          parent[py] = px;
          rank[px]++;
        }
    }
};
```

Graph

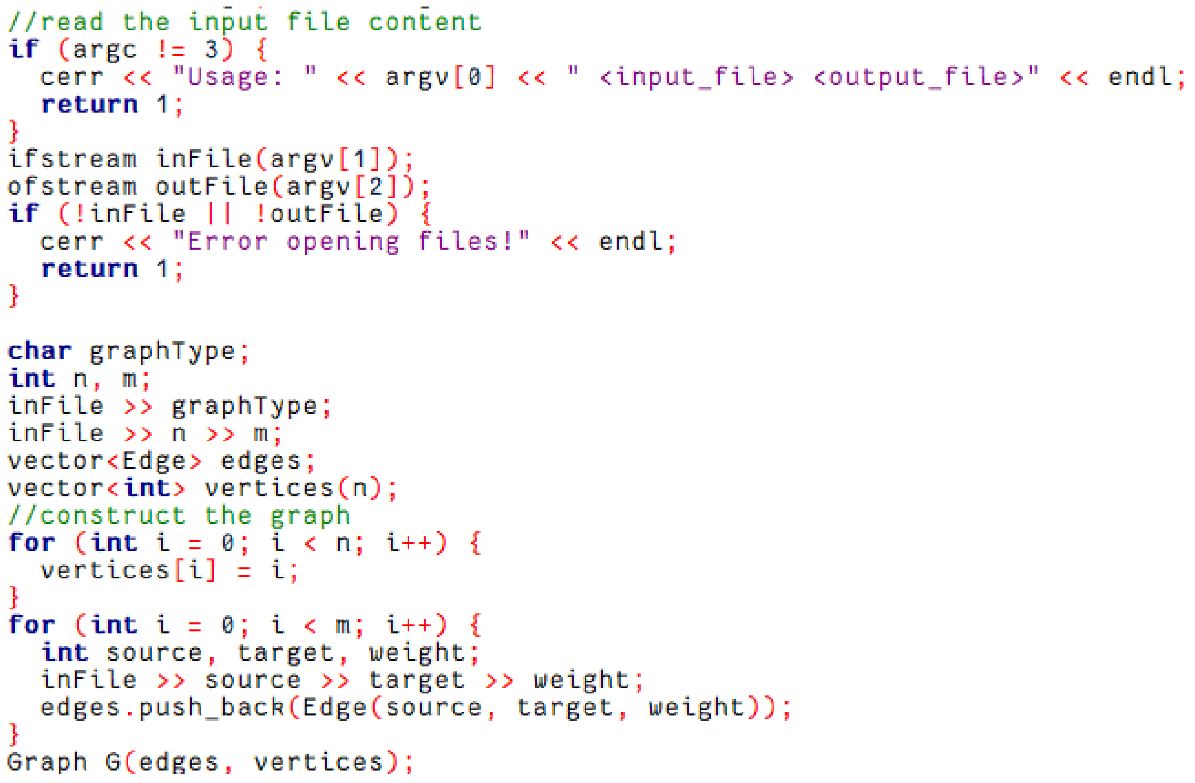
```cpp
//In class Graph, it has 2 vectors: the Edge vector E and the Vertex vector V
class Graph {
  public:
    vector<Edge> E;
    vector<int> V;
    Graph(vector<Edge>& edges, vector<int>& vertices) : E(edges), V(vertices) {}
};
```

Main Kruskal’s Algorithm



```
//The main code of Kruskal's algorithm
//The code structure is almost the same as the version in the textbook
set<Edge> Kruskal(Graph& G) {
  set<Edge> remainingEdges;
  DisjointSet ds(G.V.size());
  sort(G.E.begin(), G.E.end());
  for (const Edge& e : G.E) {
    if (ds.find(e.source) != ds.find(e.target)) {
      remainingEdges.insert(e);
      ds.unite(e.source, e.target);
    }
  }
  return remainingEdges;
}
```

Main Input and Output Process



```
//read the input file content
if (argc != 3) {
  cerr << "Usage: " << argv[0] << " <input_file> <output_file>" << endl;
  return 1;
}
ifstream inFile(argv[1]);
ofstream outFile(argv[2]);
if (!inFile || !outFile) {
  cerr << "Error opening files!" << endl;
  return 1;
}

char graphType;
int n, m;
inFile >> graphType;
inFile >> n >> m;
vector<Edge> edges;
vector<int> vertices(n);
//construct the graph
for (int i = 0; i < n; i++) {
  vertices[i] = i;
}
for (int i = 0; i < m; i++) {
  int source, target, weight;
  inFile >> source >> target >> weight;
  edges.push_back(Edge(source, target, weight));
}
Graph G(edges, vertices);
```

```
  if (graphType == 'u') {
    //for undirected graph, I choose Maximum Spanning Tree to finds the remained edge after cyclebreaking
    //To find a Maximum ST, i choose Kruskal's algorithm with sorting in decreasing order
    //the correctness is still the same as the Minimum ST version.
    set<Edge> remainingEdges = Kruskal(G);
    int totalWeight = 0;
    vector<Edge> removedEdges;
    for (const Edge& e : G.E) {
      bool found = false;
      for (const Edge& re : remainingEdges) {
        if ((e.source == re.source && e.target == re.target) || (e.source == re.target && e.target == re.source)) {
          found = true;
          break;
        }
      }
      if (!found) {
        totalWeight += e.weight;
        removedEdges.push_back(e);
      }
    }
    outFile << totalWeight << endl;
    for (const Edge& e : removedEdges) {
      outFile << e.source << " " << e.target << " " << e.weight << endl;
    }
  } else {  // graphType == 'd'
    //Here I still dont have good algorithm to solve the directed weighted Minimum Feedback Arc Set problem
    //To avoid showing error and the entire program shutdown, I output 0 instead writing nothing
    outFile << 0 <<endl;
```

4 AI asistance

The input and output part is written in AI. Though other part isn't directly generated in AI, but the part of concept and the idea of MST is inspired by AI and schoolmates. The code is followed in the textbook. In the process of coding in VS Code, copilot AI helps me, but it's not helpful though.