ECE250-Project 5

Bingjian Du, b23du

Apr 19th , 2020

1. **Overview of Classes**

**Class**:

edge

**Description**:

this is an edge with vertices and weight.

**Member variables**:

Double weight;

node \*city1;

node \*city2;

**Member functions:**

~edge();default destructor

edge();initialize an edge with weight -1

edge(double w,node\* city1, node\* city2); initialize an edge with necessary info

functions for accessing private variables: node\* get\_city1(), node\* get\_city2(), double get\_weight()

**Class**:

node

**Description**:

this is an vertice with its name, distance to the potenitial start point, list of its adjacent city, and parent

**Member variables**:

string name;

double distance; //distance to the city we want, initialized as -1

vector<node\*> adjacent; //direct connected cities

node \*parent;

**Member functions:**

node()=default;

~node()=default;

node(string name);

node(double distance);

int get\_vertex();//accessing private variables

string get\_name();//accessing private variables

void set\_distance(double distance);

void set\_parent(node\* parent);

node \*get\_parent();

vector<node\*> get\_adjacent();

double get\_distance();

void add\_adjacent(node \*p);

void reset();//clean distance and parents

bool operator<(const node &a); //overload operators

friend node operator+(const node &c1, const node &c2);

**Class:**

min\_q

**Description:**

This is a minimum queue used to sort the distances and apply Dijkstra’s algorithm

**Member variables:**

vector<node \*> vertices;

**Member functions:**

min\_q(vector<node \*> vertices);

~min\_q()=default;

void heapify(int i);

void build();

void del(int i);

vector<node \*> get\_q();

node \*extractMin();

**Class:**

graph

**Description:**

This is the graph storing edge info and giving responds to the input command

**Member variables:**

vector<vector<edge\*>> matrix;

vector<node \*> cities;

int node\_count;

int edge\_count;

**Member functions:**

graph(); constructor

~graph(); destructor

void insert(string name);

void setd(string name1,string name2, double distance);

bool search(string name);//delete

void degree(string name);

void graph\_nodes();

void graph\_edges();

double distance(string name1,string name2);

void shortest\_d(string name1,string name2);

void print\_path(string name1,string name2);

node\* find\_city1(string name); //return obj

int find\_city2(string name); //return index

void clear();

void init\_vertex();

1. **Class diagrams**

|  |  |  |  |
| --- | --- | --- | --- |
| edge | node | min\_q | graph |
| double weight;  node \*city1;  node \*city2; | string name;  double distance; //distance to the city we want, initialized as -1  vector<node\*> adjacent; //direct connected cities  node \*parent; | vector<node \*> vertices; | vector<vector<edge\*>> matrix; //adjacent matrix  vector<node \*> cities; //list of cities by the order of insertion  int node\_count;  int edge\_count; |
| edge();  edge(double w,node\* city1, node\* city2);  node\* get\_city1();//accessing private variables  node\* get\_city2();  double get\_weight();  ~edge()=default; | node()=default;  ~node()=default;  node(string name);  node(double distance);  int get\_vertex();//accessing private variables  string get\_name();//accessing private variables  void set\_distance(double distance);  void set\_parent(node\* parent);  node \*get\_parent();  vector<node\*> get\_adjacent();  double get\_distance();  void add\_adjacent(node \*p);  void reset();//clean distance and parents  bool operator<(const node &a); //overload operators  friend node operator+(const node &c1, const node &c2); | min\_q(vector<node \*> vertices); ~min\_q()=default;  void heapify(int i); //heapify a point with index  void build(); //for initialization  void del(int i); //delete a point  vector<node \*> get\_q();  node \*extractMin(); //extract a point from the top of queue and return its pointer | graph();  ~graph()=default;  void insert(string name);  void setd(string name1,string name2, double distance);  bool search(string name);//delete  void degree(string name);  void graph\_nodes();  void graph\_edges();  double distance(string name1,string name2);  void shortest\_d(string name1,string name2);  void print\_path(string name1,string name2);  node\* find\_city1(string name); //return obj  int find\_city2(string name); //return index  void clear();  void init\_vertex(); |

1. **Constructors/Destructor**

**Class edge**: edge() initialize an edge with weight -1

edge(double w, node\* city1, node\* city2) initialize an edge with necessary info

**Class node**: node(string name) initialize a node with name and default distance (set as -1 representing infinity)

< +operators are overloaded for the ease of manipulation of nodes.

**Class min\_q**: initialize a vector of pointers to nodes as a queue.

**Class Graph**: set node\_count and edge\_count to zero.

1. **Test Cases**

There are 2 cases I tested in addition to the example tests.

Test1: the size is 1, it should be successful but insert, delete should be failure.

Test2: insert a connected tree, calculate the mst. Delete some edges make it not connected. Add new edges with new weights and calculate the mst again.

Test3: update the weight of edge.

1. **Performance**

**Analysis of applying Kruskal’s algorithm:**

the algorithm is applied in the member function named mst() under class graph. The number of edges is E, the number of vertices is N. Sorting the edges takes O(ElgE) by std::sort(). Makeset() takes O(n). The for loop takes O(E)\*(Findset() and merge()). Findset() which returns the head of a node takes O(1). In my implementation of disjoint-set, each node’s head is updated at most lgN times, the total time spent in merge() during the for loop is O(NlgN). The total time for applying Kruskal’s algorithm is O(NlgN+E+ElgE). Since |E|>=|N|-1, we can restate it as O(ElgN).

However, the edges are initially stored in a 2d vector (adjacency matrix). Declaring an empty vector of edge takes O(1). Pushing all the existed edges to the vector takes O(N^2). The total time complexity of the function mst() is O(N^2).