ECE250-Project 4

Bingjian Du, b23du

Mar 30th , 2020

1. **Overview of Classes**

**Class**:

edge

**Description**:

this is an edge with vertices and weight.

**Member variables**:

Double weight;

int,x,y; x,y is the index of the vertices

**Member functions:**

~edge();default destructor

edge();initialize an edge with weight 0, index -1

edge(double w,int u, int v); initialize an edge with necessary info

functions for accessing private variables: int get\_x(), int get\_y()

**Class**:

node

**Description**:

this is an vertice with its index and next node. It is the basic unit of a linkedlist.

**Member variables**:

int vertex;

node \*next;

**Member functions:**

node();default constructor

~node();default destructor

node(int vertex); initialize a node with an index

functions for accessing private variables: int get\_index(); node \*get\_next();

**Class:**

nodelist

**Description:**

This is a linked list, representing a set of elements

**Member variables:**

node \*head;

node \*tail;

**Member functions:**

nodelist();default constructor

~nodelist();default destructor

nodelist(node\*a);initialize a linked list with a node as its head and tail

functions for accessing private variables: node \*get\_head(); node \*get\_tail();

**Class:**

set

**Description:**

This is a universe set with several set represented as nodelist

**Member variables:**

vector<nodelist> theSet;

**Member functions:**

set();default constructor

~set();default destrucor

void makeset (int num); make a universe set with the number of vertices

node \*findset (int num); pass the index to the function and return its head to check if two vertices are in the same set.

void merge(int a, int b); merge two vertices into the same set by setting them with same head and tail.

**Class:**

graph

**Description:**

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

**Member variables:**

vector<vector<edge>> matrix; adjacency matrix

int size; the number of vertices

class illegal\_commal{};handling failure

int edge\_count; the total number of edges

**Member functions:**

graph(); constructor

~graph(); destructor

void set\_size(int m); set the size of the table

void print\_ecount(); print the number of edges

void insert(int u, int v, double w); connect index u and index v

void del(int u, int v);delete the edge between index u and index v

void degree(int u); return the degree of index u

void clear(); remove all the edges

void mst(); create an mst and print the total weight

1. **Class diagrams**

|  |  |
| --- | --- |
| edge | node |
| double weight;  int x,y; | int vertex;  node \*next; |
| edge();  edge(double w,int u, int v);  int get\_x();//accessing private variables  int get\_y();  double get\_weight();  ~edge()=default; | node()=default;  ~node()=default;  node(int vertex);  int get\_vertex();//accessing private variables  node \*get\_next();  void set\_next(node \*a);  bool operator==(const node &a);  bool operator!=(const node &a); |

|  |  |
| --- | --- |
| nodelist | set |
| node \*head;  node \*tail; | vector<nodelist> theSet; |
| nodelist()=default;  nodelist(node \*a);  ~nodelist()=default;//accessing private variables  node \*get\_head();  node \*get\_tail();  void set\_head(node \*a);  void set\_tail(node \*a); | set()=default;  ~set()=default;  void makeset(int num); //consistent with lecture notes  node \*findset(int num);  void merge(int a, int b); |
| graph |
| vector<vector<edge>> matrix;  int size;  class illegal\_argument{};  int edge\_count; |
| void set\_size(int m); //function name consistent with commands  int get\_size();  void print\_ecount();  graph();  ~graph();  void insert(int u,int v,double w);  void del(int u,int v);//delete  void degree(int u);  void clear();  void mst(); |

1. Constructors/Destructor

Class edge: edge() initialize an edge with weight 0, index -1

edge(double w,int u, int v) initialize an edge with necessary info

Class node: node(int vertex); initialize a node with an index.

== != operators are overloaded for the ease of comparison of nodes.

Other constructors and destructors are kept as default.

1. Test Cases

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

Test1: the size is 1, it should be

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.

1. Performance

It is uniform hashing which means in each slot the number of elements is O(1), so the expected average runtime of insert, search and delete should be constant O(1).