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

2. 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();

3. 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.
Class set: the destructor is modified to manually delete all the elements in the disjoint set to ensure there is no memory leak.
Other constructors and destructors are kept as default.

4. 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.

5. 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).