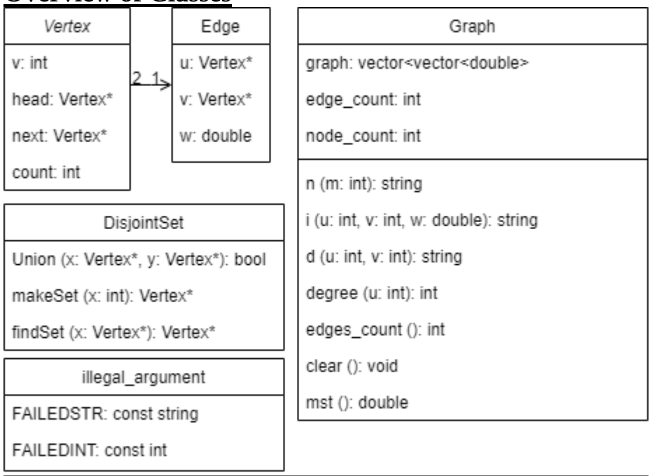
ECE250 – Project 4
MST Using Kruskal's Algorithm
Design Document

Qinying Wu, q227wu

Mar 30th, 2020

1. Overview of Classes



Class Name	Description	Member Variables and Functions
Vertex	Vertices objects in the graph	***v: int*** the integer representation of vertex v ***head: Vertex**** the vertex representing the disjoint set that the current vertex v belongs to ***next: Vertex**** the next vertex linked to the current vertex to represent a disjoint set ***count: int*** if the current vertex is the representative vertex of the disjoint set, count stores the total number of vertex in the disjoint set, otherwise count=1
Edge	Vertices connections in the graph	***w: double*** the weight of the edge ***u: Vertex**** one of the vertices connected by the edge ***v: Vertex**** the other vertex of the edge
DisjointSet	Disjoint set objects to represent forest of nodes	***Union (x: Vertex*, y: Vertex*): bool*** performs a union operation on the vertex disjoint sets. Parameters x and y are vertices in the graph. Returns true if disjoint set of x is different than that of y after union. Returns false if x and y are in the same disjoint set ***makeSet (x: int): Vertex**** make a node in the graph into a vertex object. Parameter x is the integer representation of the node in the graph. Returns the newly created vertex pointer object ***findSet (x: Vertex*): Vertex**** find the disjoint set of a given vertex. Parameter x is the vertex object. Returns the head representative vertex of the disjoint set (linked list)
Graph	The graph object	***graph: vector<vector<double>>*** the adjacency matrix with entries as weight of the edge between two vertices in the graph ***edge_count: int*** the total count of edges in the graph ***node_count: int*** the total count of nodes in the graph
		n (m: int): string initialization of the graph. Parameter m is the number of vertices in the graph. Returns success if successful in creating the graph. If m<0, throws an illegal_argument exception ***i (u: int, v: int, w: double): string*** insert an edge between two vertices. Parameters u and v are integer representations of the vertex objects, w is the weight of the edge. Returns success if success in insertion or updating the weight of an existing edge. Throws an illegal_argument exception if u and/or v are outside the valid range of vertices ***d (u: int, v: int): string*** deletes the edge between two vertices. Parameters u and v are integer representation of arbitrary vertices. Throws an illegal_argument exception if either u and/or v out of range. Returns "failure" if edge weight is 0 (edge does not exist). Returns "success" if the edge between vertices u and v is deleted successfully ***degree (u: int): int*** returns the number of edges connected to a vertex if the vertex is in the graph. Throws an illegal_argument exception if the vertex is out of range of the graph. Parameter u is an integer representation of an arbitrary vertex. ***edges_count (): int*** returns the total count of edges in the graph in string ***clear (): void*** removes all the edge from the graph ***mst (): double*** calculates the minimum spanning tree of the graph. Returns the minimum weight of the mst if the graph is connected, otherwise return -1 to indicate a not connected graph
illegal_argument	Exception class	***FAILEDSTR: const string*** indicative value for string function exception ("failure") ***FAILEDINT: const int*** indicative value for integer function exception (-1)

2. Constructors and Destructor Decisions

Class Name	Constructor	Destructor
Vertex	One integer parameter – vt is assigned to v to represent the current vertex. Member variables head and next are each assigned to nullptr. Member variable count is assigned to 1 since each vertex is in its own disjoint set upon initialization	Member variables head and next are each dereferenced to nullptr
Edge	Three parameters – vertex objects u and v are assigned to member variables u and v, respectively. Double w is assigned to the member variable w	Vertex u and v are dereferenced to nullptr
DisjointSet **illegal_argument**	Does not pass in any parameters, simply an initialization of objects	Does not pass in any parameters, simply destructs the object
Graph	Two integer parameters – e is assigned to edge_count, v is assigned to node_count	Clears the graph adjacency matrix

3. Asymptotic Upper Bounds

Union	O(min(findSet(x).size, findSet(y).size)) first check the disjoint set size of each x and y, and add the smaller set to the larger set
makeSet **findSet** **edges_count** **i, d**	O(1) makeSet – converts one integer representation of the vertex into a pointer vertex object and returns it findSet, edges_count – accessing the corresponding member variable in their respective ADTs is constant time i, d – updates the entry in the adjacency matrix by directly accessing the given matrix row and column indexes
degree	O(V) traverse through the row of the corresponding square adjacency matrix index. The number of entries visited is equivalent to the number of vertices in the graph represented by V here
n	$O(V^2)$ V is the total count of vertices in the graph. The graph is initialized with V rows and V columns which is equivalent to the square of V
clear	$O(V^2)$ traverse through every entry of the adjacency matrix. Since it is a square matrix with same row and column counts equal to the number of vertices in the graph, the time complexity would thus be the square of the vertex count
mst	O(V+E+Elog(E)) Make each vertex in the graph into a disjoint set requires $V^* T_{makeSet}$ =O(V) time. Make each edge in the graph into an Edge object requires E*(O(1))=O(E) time. Sort the edges in ascending order using std::sort() requires O(Elog(E)) time. Finally traverse through the sorted list of edge objects to find the mst is E^*T_{Union}=O(E).

4. Test Cases

a) i u;v;w, d u;v, degree u
    a. u=v
    b. u!=v
        i. both u and v in range
        ii. either u or v, or both u and v out of range
    c. w<=0 and w>0 (for i)
    d. after calling the clear function
b) edges_count
    a. before/after inserting a valid/invalid edge
    b. before/after deleting a valid/invalid edge
    c. before/after calling the clear function
c) mst
    a. disconnected graph
    b. connected graph
    c. after calling the clear function
d) Union (u,v)
    a. u.DSet=v.DSet
    b. u.DSet!=v.DSet
e) findSet (u)
    a. u.DSet contains itself only
    b. u.DSet contains other nodes
        i. u is the representative node of the set
        ii. u is not the representative node of the set
	Example: (expected output)
	n 4 (success)
	d 0;2 (failure)
	d 1;1 (failure)
	degree 2 (failure)
	edge_count (edge count is 0)
	mst (not connected)
	clear (success)
	mst (not connected)
	i 3;3;2 (failure)
	i 91;2;33 (failure)
	i 2;3;1 (success)
	i 0;1;1 (success)
	i 0;2;2 (success)
	i 1;2;1 (success)
	i 1;3;44 (success)
	i 3;1;2 (success)
	degree 1 (degree of 1 is 3)
	degree 0 (degree of 0 is 2)
	degree 2 (degree of 2 is 3)
	degree 4 (failure)
	degree 3(degree of 3 is 2)
	mst (3)
	edge_count (edge count is 5)
	clear (success)
	edge_count (edge count is 0)