ECE 250 - Project 4
Graphs
Design Document

Jamie Yen, UW UserID: j6yen

Dec. 06th, 2022

Overview of Classes

Class 1: Graph

Description:

This class helps in storing cooperation records using linked lists which each node represents an individual's own cooperation records and are recorded in it's own vector. Operations of manipulating these records, such as insert and deletes were provided. Further more, these informations will be extracted to help building connected graphs using Kruskal's algorithm.

Member variables(all private):

1. Researcher type pointer dataSet: points to the head of the researcher linked lists
2. int maxVertex: the largest vertex in the graph
3. int edgesNum: number of the total edges in the graph
4. int vertexNum: number of the total vertices in the graph

Member functions(all public):

1. **mst**: i. call extract() ii. call sort iii. union vertices to creat mst starting from the largest edges iv. go through the vertices parent list to count how many vertices are in the same set as vertex a
2. **load**: i. the edge is a self loop? return ii.vertex existed? add the record to it's collabRecord vector iii. vertex not exist in the graph before? insert the vertex in the vertices linked list in descending order and insert the first collabRecord, record # of new vertices that were added
3. **insert**: i. invalid input? call exception ii. implemented using similar strategy as the load function above, except that it has outputs after one inseartion is done
4. **print**: i. invalid input? call exception ii. traverse through the linked list to find the vertex iii. print all the other name in collabrecord vector
5. **deletes**: i. invalid input? call exception ii. vertex exist in graph? iii. delete the edges containing the vertex iv. delete the researcher node from the linked list
6. **size**: call the vertexNum in the graph and print it
7. **findParent**: helper function for mst i. at index n it's parent is itself? return n ii. call findParent at parent[n] recursivly until its it's is the parent of itself
8. **unionSets**: i. call findParent to check if a and b vertex have the same parent thus are from the same set ii. union the sets a and b are in
9. **sort**: sort the vector edges in descending order so that we can start selecting edges from the largest weight
10. **extract**: extract all the edges from dataSet to vector edges

Constructor: not used

Desctructor: delete the dataSet linked list to ensure no memory leaks

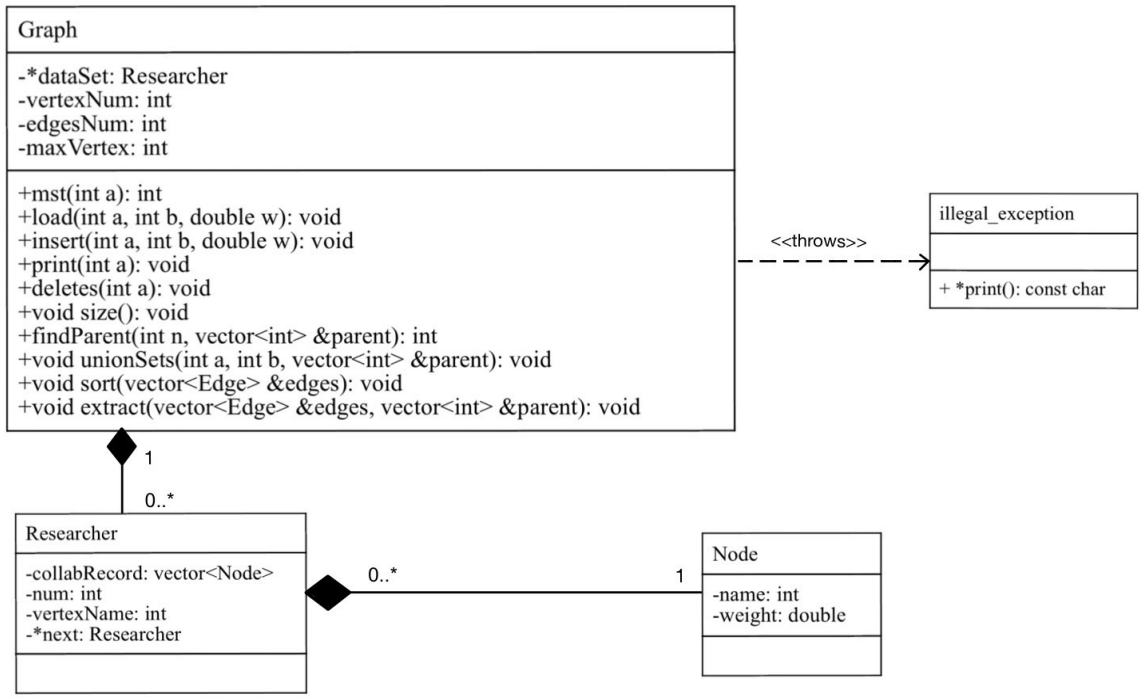
Class 2: Researcher

Description:
This researcher class is a class used as entries for the linked list to record all the vertices's data
Member variables(all privates):
1. Node type vector collabRecord: record each researcher's own cooperation datas
2. int num: record the number of collaboration this researcher has with other researchers
3. int vertexName: record the name of the researcher
4. Researcher type pointer next: points to the next researcher
Constructor: initiailze then name of the researcher and also storing it's first collaboration with other researcher
Destructor: not used since the Graph class will be taking care of deallocation

Class 3: Node
Description:
This node class is a class used as entries for recording each researacher's own data.
Member variables(all privates):
1. int name: record the name the re
2. double weight: record how much this cooperation has impacted the second researcher

Class 4: illegal_exception
Description:
An exception class that inherits the std::exception class to handle input words that contain any characters other than those of the lower-case english alphabet.
Member function(public):
constant char type pointer print: return the string illegal argument

UML Class Diagram



Test Cases

1. insertion test: I will call load and call size after it, I should see the number of vertices to be 23133. I will also insert some edges that starts from different value of vertices to test if my linked list is implemented properly so that there's no issue from inserting at front, middle and the end. I should see no segmentation fault.
2. deletion test: I will insert some edges and then delete all the nodes in the graph, when I call size and mst, I should see the size to be 0 and no mst output. After that I will insert some edges again and they would operate properly.
3. single vertex test: I will insert two nodes, a and b that are only connected to each other, then delete node a, when I call size, I should see the size to go down by one instead of 2 since vertex b still remains in the graph. When I call mst on b, I should see the output to be 1 since b does not have edges connected to it and is a subtree of 1.
4. mst test: I will insert three edges that a->b, b->c, b->d, b->c and e->f. When I call mst on a, I should see the output to be 4 and when I call mst on f, I should see output to be 2. When I call mst on z, which isn't in the graph, I should see the output to be 0.

Performance

-print

The expected runtime for print is O(degree(a)), to satisfy this, I stored all the edges that are adjacent to the same vertex in the same vector, thus when print gets called, I only need to find the collaboration records vector for a, and then print all of them afterward.

-size

The expected runtime for size is O(1), so I make sure to keep track of the number of vertices that are currently in my graph while doing insertion and deletion operations. Therefore I just need to call the size variable in my class for this function.

-mst

The Kruskal's algorithm were used to implement my mst function, pseudo code it is below:

	time complexity
extract all the edges from my dataSet	O (E)
sort them in descending order by weight	θ (ElgE) = θ (Elg (V*V)) = θ (ElgV)
for (all the edges from big to small)	O (E)
if (the two vertices of the edge are in disjoint sets)	
add the edge to the mst	
ans += 1; we got one more vertex in the graph	
num of edges in the mst become numEdge +1	
when numEdge = Num of vertices -1 the loop terminates	
check the parent of all the vertices	θ (23133)
if they share the same parent with a	
count++	
print (count)	

Thus the runtime for my mst function is O (E)+θ (ElgV)+O (E)+ θ (23133), and with that an overall runtime of O(|E|log(|V|)) is achieved