Ryan Pascal
rdp27@zips.uakron.edu
2932732

Project 1

Data structures 001

Dianne Foreback

Name

Ryan Pascal

rdp27@zips.uakron.edu

2932732

Manifest of files

1. components.cpp

This file is where all the logic for the project exists. All the code for the project is in this file.

2. Project1.pdf
3. Data.dat

   Example data file

Project Summary

The purpose of this project to read in a file to create an adjacency list for the user to manipulate. The adjacency list represents a graph consisting of a set of vertices and a set of edges. A vector of lists is used where each element in the vector is a node on the graph and each element in that elements list is a connection that node has. Once the adjacency list has been built the user will be prompted to input two list identifies for an attempt at merging those lists. If a common vertex exists among the two inputted identifiers the lists will be merged.

The implement of this project is done the following way. Once a proper file name has been entered each line will be looped over one at a time. The line is then split so that it can be looped by each number. As the elements are being looped they will be inserted into the list at the proper position. After each line has been processed and the adjacency list was built the user will be prompted to merge two lists. Once two lists have been identified by the user the lists will be compared to determine if they have a common vertex. To achieve this task linearly a map was used. The two lists are looped at the same time and each element is placed into the map. As the lists are iterated if we come across a value that is already in the map we know there is a common vertex. Finally, the smaller list is merged into larger list and the smaller list is removed from the adjacency list. The user will continue to be prompted to merge lists until they enter a -1.

Questions

Ryan Pascal
rdp27@zips.uakron.edu
2932732

a. Per number 4 in the Coding Requirements, give the worst-case runtime to determine where a node should be inserted into an adjacency list and explain/justify your analysis.

The worst-case runtime to determine where a node should be inserted is O(n). This is the case because to determine the location to insert the list will need to be looped until a value is found that is bigger than the value we are testing. Then the element can be inserted into the list at the position of the value found. If no element is found, we only looped the length of the list and insert into the end of the list.

b. Per number 4 in the Coding Requirements, give the worst-case runtime to build the entire adjacency list and explain/justify your analysis of this runtime.

The worst-case runtime to build the entire adjacency list is $O(n^3)$. This is the case because we need to loop over both the lines of the input file, the elements in each line, and finally we need to loop to find the position for inserting into the list.

c. What if vector of vector is used for the adjacency list instead of a vector of lists, how would the runtimes for parts a and b change? Explain/justify your analysis.

If we used a vector of vector for the adjacency list the runtimes for part a and b would increase by a factor of n. To insert into a list it is constant time, however inserting into a vector is O(n) (excluding the end of the vector which is constant). Due to this it will take the same amount of time to find the position where the new element needs to be inserted, which this process is O(n), but to insert at this position it will take O(n). So, for a the worst-case runtime would still be $O(\mathrm{n})$. For b the worst-case runtime would be $O(n^4)$.