ECE250 Project 4 Design Document- Graphs

Devon Miller-Junk 20763552 November 22, 2019

Project 4: Graphs

For the design of project 4, my plan was to create a quad tree that can perform search and insert operations in O(nlgn) if it is balanced. The clear, and traversal operations should take O(n) time.

**Note:** To compile and run my program please use the following

g++ -std=c++11 -o undirectedGraphtest undirectedGraphtest.cpp

./undirectedGraph <inputFile >outputFile

1. **The Templated Max/Min Heap Class**

The Templated Max/Min Heap class implements a max/min heap (you choose which one in the constructor). The Max/Min heap contains a pointer to an array objects (type of object determined by the template), a Boolean value representing if the heap is a max or min heap, a size of the heap array and the number of elements in the heap array. It also contains two private functions. Comparison takes two indexes as integers and returns true/false depending on whether they should be swapped or not. The swap functions take to indexes and swaps the two values at those indexes, returning null. Both functions take constant time.

The Heap class also contains two constructors. Both take a boolean value to indicate whether to make a max or min heap and an integer size of the array to create. Then, one of the constructors takes an array of objects that are turned into a heap using heapify. Both constructors take linear time since each value in the array must be initialized to a null value, and for the second constructor it takes linear time to heapify an array. The destructor of the heap class deletes the array of objects that make up the heap.

The Heap class also contains several other functions used for modifying it. The first two are heapify and upsort. They both accept an index as a parameter and return void. They take time proportional to the height of the tree (logn) and re-arrange the order of the nodes to keep the heap properties. The search function takes a string as a parameter and returns the index of the vertex matching the given string, returning the index if found. This takes linear time. The length function returns the size of the array, the isEmpty function returns true if the array is empty, and the root Value function returns a double representing the double value of the min/max elements in the heap. The insert function takes a pointer to an object and inserts that object into the heap. It takes log(n) time to then preserve the heap characteristics. The remove functions take in index as a parameter and returns the value that was removed at that index. It takes log(n) time to remove since it again must traverse the height of the heap. The print function outputs to cout the entire heap in order, taking linear time.

* 1. **The Heap Node Struct**

The Heap Node Struct is a struct used to hold the vertices inside of the heap. It overloads the correct operators to allow for comparisons between different vertices in the correct fashion in the creation of the heap. The Heap node struct contains a pointer to a vertex, which is the vertex to be put in the heap. The heap node also contains two constructors, one that accepts no parameters and simply creates an empty vertex, and one that accepts a pointer to a vertex, and points its own vertex to the same vertex. The Heap node also overloads the equality, greater, less than, and equals operators. Each operator accepts a heap node as a parameter (except equals which accepts a pointer to a heap node). The equality, greater than, and less than operators then return a Boolean value depending on if the parameter and current Node have equal, greater, or less distance as a comparison between them. The equals operator assigns the vertex of the current node to that of the parameter and returns void. The last two functions of the heap node struct are the print function and the get Value function. The print function calls and returns the print function on the vertex (returning a string), and the getValue function returns the vertex of that node. All functions take constant time.

1. **The Templated Linked List Class**

The linked list class is a templated class that links together a bunch of templated nodes. The linked list itself contains a pointer to the starting and ending node, as well as the number of nodes in the list.

There is a constructor that initializes the size to 0 and starting and ending pointers to the null pointer. The destructor removes elements from the linked list until none remain (takes linear time). The linked list equals operator was overloaded, and it accepts a list as the input parameter. It takes linear time and replaces the list with the input parameter. The push function appends the input object to the end of the linked list (constant time) and returns void. The pop function removes the last element of the linked list and returns void (constant time), the remove function accepts a node and removes it from the linked list, to then return void (constant time). The get start and get back functions return a pointer to the first and last node in the linked list. The get size function returns the size of the linked list. The print function takes linear time and prints the names of the nodes in order to standard out. The search function returns a pointer to the node whose name matches the string input parameter. It takes linear time.

* 1. **The Node Struct**

The Node struct contains a pointer to a value and a pointer to the next and previous node structs. It is used in the linked list to link connecting adjacent nodes together. The Node struct contains a constructor that initializes the pointers to the null pointer and the value to the default value. The destructor simply calls the destructor on the value. Two operators were also overloaded. The equality operator takes a string as a parameter and returns true if the name of the value is equal to the input parameter. The equals operator accepts a pointer to an object as a parameter and assigns the current value of the node equal to the input parameter (void is returned).

1. **The Graph Class**

The Graph class contains a linked list of vertices and an integer representing the number of edges. It also contains two private functions. Reset Paths returns void and resets parent, unlocated, and distance values of each vertex in the nodes linked list (linear time on the number of vertices) and returns void. The relax function takes a source vertex, and destination vertex, and a weight (a double). It returns true if the current distance of the destination is greater than that of the source plus the weight, and alters the parent, unlocated, and distance values.

The graph constructor and destructor construct and destruct the linked list of vectors. The search function takes a string as a parameter and return a pointer to the vertex that has that name (linear time on the number of vertices). The insert function takes a vertex as an input and returns true if that vertex was inserted. This takes linear time since it must first search to see if no other vertices match the input. The set path function inserts or changes the edge from the first-string input to the second-string input to the double input. Returns true if the operation occurred (linear time on the number of nodes). The Dijkstra function takes a vertex and finds the single source shortest path to each other vertex in the graph. Since a min heap is used to do this, the entire function takes time proportional to the number of edges (E) multiplied by the time it takes to insert each node into the min heap (lgV) plus the number of vertices time the time to extract the min vertex (or VlgV). Thus, the total running time is VlgV + ElgV or Θ(ElgV). The get distance and get path functions simply return the distance and path of the given vertex (returned as a string or double) that was generated by running the Dijkstra shortest path algorithm. The print function prints each node in the graph (linear time) and the getSize/getNumEdges functions returns the number of nodes/edges in the graph correspondingly.

* 1. **The Vertex Struct**

The Vertex struct is used to hold the information about the cities in the graph. Each vertex holds a Boolean and double (unlocated and distance), that when combined with the parent value (a pointer to a vertex) are used in the shortest path algorithms. Each vertex also contains a string name, and a linked list of edges called paths. The vertex constructor takes no inputs and constant time. The destructor takes linear time depending on the length of the linked list of edges. The vertex struct also overrides the equality operator, taking a string as input and returns true if the input equals the name of the vertex. The equals operator was overloaded and takes a vertex as an input, and void is returned. The print function takes no parameters and returns either NULL or the name of the vertex.

* 1. **The Edge Struct**

The edge struct contains a length value as a double and a string that is its destination. The equality operator is overloaded and takes a string as an input and outputs true if the destination of the edge equals the name. The equals operator is also overloaded, takes an edge as a parameter and returns void. The print function returns the destination of the edge.

1. **Utility File**

I used a utility file to hold a few functions that are useful for string parsing and input handling. These functions just modify the strings received from the input file. The remove new line function takes a string and returns the same string with all new line characters removed. The get command function takes a string and returns the substring that exists before the first space in that string. The get inputs function takes a string and returns the substring that exists after the first space in that string. The separate by commas function takes a string and an integer size as parameters. It returns the input separated by commas into an array with length of the size parameter.

1. **References**

Tahvildari, L. (2019, Fall). *ECE 250 Lectures*. *ECE 250 Lectures*. Waterloo.

Ward, P. (2018, Fall). *ECE 150 Lectures*. *ECE 150 Lectures*. Waterloo.