Algorithms 4133

HW1

Samuel Lin 010880917

In this Homework Assignment, we were given a mostly-implemented program which tested the functionality of our to-be-implemented linked list and queue structures. The program also included a test where it utilized our linked list and queue structures to create a graph modeling the UARK campus map and found a path from one part of campus to another. Since I am an undergraduate student, I only had to implement a singly linked list and a normal queue.

A linked list is a linear data structure that is comprised of nodes which are linked together using pointers. Each node consists of a value, a next pointer which points to the next node in the list, and for a doubly linked list, a previous pointer which points to the previous node in the list. The head node is the beginning of the linked list and the tail node is the last node in the linked list. The linked list functions I had to implement were the destructor, insert, find, remove, and size.

The insert function first checks for the value in the list, if found, returns the pointer to that node which has the value, if not, it will insert the value at the end of the list and set the root pointer to be the new created node.

The find function iterates through the list to see if any values match. If a value is not found, then find will return a null pointer. If a value is found, a pointer to the node with the matching value is returned.

The remove function iterates through the linked list to check if any values match. If there is a matching value, the node is removed from the list and the previous node’s next pointer is set to the deleted node’s next pointer. The function returns the root pointer

The size function iterates through the linked list and for every node that is not null, increments a counter by 1. At the end of the list, it returns the counter value.

A queue is another linear data structure that operates as first in first out order. The node structure was also used here to implement the queue data structure. Each node in this queue data structure also consists of a value, a pointer to the next node, and a pointer to the previous node. When a value is inserted into a queue, it’s inserted at the end. When a value is popped from the queue, it’s taken from the head. The queue functions I had to implement were the queue destructor, empty, pop, and push.

The queue destructor iterates through the queue and deletes each pointer, freeing the memory for every node in the queue. It then sets the head and tail pointers to null.

The empty function checks if the head and tail pointers of the queue are null, if yes then it returns true, if not, then it returns false, its purpose is to check if the queue is empty or not.

The pop function is used to take a value from the queue. It returns the value from the head node of the queue, then deletes the head node, setting the next node as the new head.

The push function is used to add a value to the queue. It creates a new node and inserts the value into the new node, sets the tail pointer to point to this node, and the next pointer to be null. It doesn’t return anything.

These are the results after running the main function through terminal by executing the make file.

This program was implemented and tested on an m1 MacBook Air.

