CS 2123 Data Structures

Programming Project 4

In this project, we will be implementing the Graph data structure using an adjacency matrix. We will then use that data structure to compute a shortest path between two vertices of an unweighted graph using Breadth First Search (BFS). BFS uses a basic Queue, and we will use your queue implementation Project 2 as well.

**Files provided in the project:**

1) **Graph.h and Graph.**c. Graph.h file contains the prototypes of all of the functions we will be using for an adjacency matrix implementation of a Graph. You must provide the implementation of the following functions in Graph.c

1. **Graph newGraph(int n).** This function takes as input the number of vertices of the graph, and mallocs a new graph, allocates an nxn array for the adjacency matrix of the graph, initializes each value of the matrix to be 0 (no edges yet), and then returns the address of the graph.
2. **void freeGraph(Graph g).** Free the adjacency matrix and then the graph itself.
3. **void addEdge(Graph g, Edge e).** Given a graph and an edge, set the corresponding entry in the adjacency matrix to be 1.
4. **Edge firstAdjacent(Graph g, Vertex v).** Given a graph and a vertex v, return the first edge of the graph that has v as the “fromVertex”. If no such edge exists return an edge with both the fromVertex and toVertex set to be -1.
5. **Edge nextAdjacent(Graph g, Edge e).** Given a graph g and an edge e, find the next edge in g after e that has the same “fromVertex” as e. If no such edge exists return an edge with both the fromVertex and toVertex set to be -1.
6. **void shortestPath(Graph g, Vertex start, Vertex destination).** Compute a shortest path in g starting at “start” and ending at “destination” using Breadth First Search. If a shortest path exists, then print the sequence of vertices along the path that you computed in the following format: “Shortest path from 0 to 6: 0 -> 3 -> 6”. If no path exists, then print a message such as: “There is no path from 6 to 3.”. I’ll provide an output file for the input file later this week.

**2) Queue.h and Queue.c**. BFS uses a queue data structure to perform the search. Included is the Queue.h header file. This version of Queue.h is almost identical to the file from Project 2, only we are using typedef to make an Element be an int (Vertex). Also the “isEmpty()” function has been changed to “isEmptyQueue()” to avoid conflicting with the Stack function. You can use your Queue.c file from Project 2 for this project with very minor changes.

3) **Stack.h and Stack.c.** It is helpful to use a stack to compute the actual path after the BFS traversal has completed. This version of Stack.h is almost identical to that of Project 1, except it typedefs an Element to be an int (Vertex). Also the “isEmpty()” function has been changed to “isEmptyStack()” to avoid conflicting with the Queue function. You can use your Stack.c file prom Project 1 for this project with minor changes.

4) **p4Input.txt**. The project will contain the information for a single graph, and then will ask you to compute several shortest paths for different start/destination pairs for the same graph. The first line contains the number of vertices in the graph, and the second line contains the number of (directed) edges in the graph. The next set of lines contains two integers representing the “fromVertex” and the “toVertex” for each edge. For example, the line “3 5” implies that we should add an edge from vertex 3 to vertex 5 in the graph. Next is a single integer which represents the number of shortest paths you will need to compute. The remaining lines tells you which shortest paths to compute. For example, “ShortestPath: 0 8” implies that you should compute a shortest path starting at vertex 0 and ending at vertex 8.

5) **abc123Project4.c**. Rename this file to your abc123. This file should parse the input from p4p2Input.txt, create the graph, and then call the shortestPath function for each of the requested pairs.

5) **Makefile**. Update the makefile to reflect your abc123. Compile using **make**. Execute the program using **./p4**

**Extra Credit**

For extra credit, you can create a new file **abc123P4ExtraCredit.c** that solves the internship question we discussed in class. This file should have its own main() function. Prompt the user to input two integers “a” and “b” such that b > a. We want to convert “a” into “b” using the following operations: +2, -1, or \*3. For example, if we are currently at 4, then the next step could be either 6 (using the +2 operation), 3 (using the -1 operation), or 12 (using the \*3 operation). We want to compute a minimum number of operations to convert “a” into “b”.

Compute an optimal solution for this problem and output the sequence of numbers in your solution.

**Submitting:**

Zip up your project folder in a file name abc123Project3.zip (with your own abc123) and submit on the dropbox on Blackboard by the due date.