CIS 477 (Fall 2022) Disclosure Sheet

Name: Kevin Lopez

HW # 1

Yes No Did you consult with anyone on parts of this assignment, including other students, TAs, or the instructor?

Yes No Did you consult an outside source (such as an Internet forum or a book other than the course textbook) on parts of this assignment?

If you answered Yes to one or more questions, please give the details here:

By submitting this sheet through my Blackboard account, I assert that the information on this sheet is true.

This disclosure sheet was based on one originally designed by Profs. Royer and Older.

Kevin Lopez

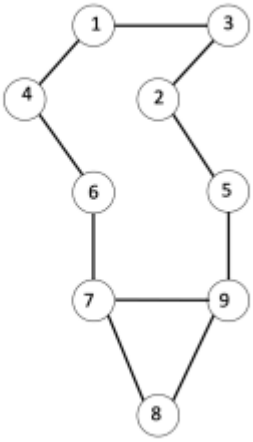
CIS477

10/30/22

HW4

1) Problem 1: Suppose you are given the undirected graph shown below, and you begin at node 1. (a) In what order does DFS visit the nodes? Draw the DFS tree, including backedges. (b) In what order does BFS visit the nodes? Assume that when there is a choice of which node to visit next, the DFS/BFS search will always select the node with the lowest number. (If you are using Latex, see https://en.wikibooks.org/wiki/LaTeX/Importing_ Graphics for an explanation of how to

include graphics with your document. You can hand-draw your solutions and import them into



the file.)

a. DFS:
    i. 1
    ii. 3 2 5 9 7 8
    iii. Recurse back to 1
    iv. 4 6
b. BFS:
    i. 1
    ii. 3 4
    iii. 2 6
    iv. 5 7
    v. 8 9

2) Suppose you have a graph describing roads (edges) between locations in a city (nodes). Assume that all edges are directed. You need to get from one location to another, but along the way, need to drop off a package at a third location. Present an efficient algorithm to find the shortest path from a starting location u to a destination v location, such that the path includes location x.
    a. To do this algorithm, You must first do a search to find the third location for the package(x) and then find the location (v) starting from u. You can do breathfirst search to find the shortest path to the location and then another breathfirst search to make it to the final destination.
        i. newQueue = Queue.Push(node u,[path1,path2])
        ii. Explored = set()
        iii. While (queue is not empty):
            1. topqueue,path1,path2 = newQueue.pop(node u)
            2. If topQueue is not in explored:
                a. Explored.append(topquue)
                b. Queue.push(Topquue.succesor())
                c. Add node to path1
                d. If topqueue == node(x):
                    i. Add to path2

e. If topqueue == node(v):
    i. Add to path2
    ii. Return [path1] + [path2]

3) DFS and BFS are about finding paths from a starting node s to all other nodes. Suppose that in a directed graph, we instead want to find paths (in the case of BFS, shortest paths) to a target node t from all other nodes. How would you modify DFS/BFS to do this? (A few sentences of explanation suffices.)
    a. Instead of adding the explored nodes into a set and not checking them again, you create a list of nodes that lead to a node(t). This way there is a list of nodes and since it is directed, it only goes to it once and restarts from the starting node back down all other nodes. You can also let it explore other paths simultaneously to make it faster
4) You are laying optical fiber cables to bring internet to a new area of the country. The cables can only be placed between certain pairs of cities. In other words, the cities and possible cable locations can be described as an undirected graph, where each node represents a city and the edges represent the possible locations for placing cable. Each edge e is associated with a distance de that tells you the distance between the corresponding cities in miles.

    You want to connect City S to City T through a series of cables (where S and T are specified by the user); however, each cable that you have is exactly L miles long, and you cannot connect two cables outside of cities. So if there is an edge from City S to City A that is L miles long, and an edge from City A to City T that is L−1 miles long, then you could lay cable from S to A to T. However, if the edge from S to A was L + 1 miles long, you would have to find a different route.

    Create an algorithm to determine whether you can connect City S to City T, given the structure of the graph and your limited-length cables. Your algorithm should run in time O(|E|), where |E| is the number of edges. Show that your algorithm meets this running time.
    a. To do this algorithm, you must do an iteration of depth first search while backward checking to see if the node behind is at most L miles long. This runs in O(|E|) time because it excludes the full total of edges to be explored and will only check the nodes that fit the criteria
        i. stack = stack()
        ii. stack.push(startingNode,miles=0,path=[])
        iii. while stack
            1. currentNode,miles1,path1 =stack
            2. if startingNode == endingNode:
                a. return path
            3. if miles1 < miles:
                a. stack.push(currentNode.Succesor)
            4. else: continue
5) Extra Credit: This problem is a continuation of Problem 4. A vendor is offering to sell you longer cables. Create an algorithm to determine the minimum length required that will allow you to connect City S to City T. Your algorithm should run in time O(|V | + |E|) log |V |) time, where |V | is 2 the number of vertices. Show that your algorithm meets this running time

a. For this we can use a cost effective algorithm that tracks the total length of the cables. It then explores all the cheapest options in terms of cables. Then in breadfirst search we check each successor node and explore the cheapest one.
    i. queue = queue()
    ii. queue.push(startingNode,miles=0,path=[],pastMiles=0)
    iii. while stack
        1. currentNode,miles1,path1,pastMiles =stack
        2. if startingNode == endingNode:
            a. return path
        3. if miles1 +miles < pastMiles:
            a. stack.push(currentNode.Succesor)
            b. pastMiles = miles1+miles
        4. else: continue
    iv.