Homework 2

Krish Patel
CS 180

Q1)
We have to add on top of the algorithm given in 3.6, an algorithm that uses BFS or DFS to check whether the graph contains a cycle or not. For this, we could use an algorithm involving recursive Depth first search(DFS), where we start at the root node, and traverse through the entire graph using it. As we do this, we mark a node as visited every time we encounter a node, and then look for nodes other than that the parent. If we encounter a new node other than the parent that was marked as visited, then there exists a cycle in the graph. It would exit as soon as it found a cycle and if it traverses through the entire graph without finding a cycle, it would output that it is a DAG and move onto the topological ordering algorithm. The time complexity of DFS is O(m+n), and running the topological ordering algorithm after would give a time complexity of O(2m + 2n), which is equal to O(m+n)

Q2)
We have a list of n specimens of butterflies, that can be classified into two different species: A and B. Instead of direct classification, we have information regarding two specimens in a pair. There can be three classifications, “same”, “different”, and “ambiguous”. For this algorithm, we would be using a undirected graph
How does the algorithm look like?

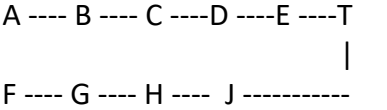
- First create a graph with n nodes, each representing a unique butterfly specimen.
- We would need two types of edges, one labelled as same, the other labeled as different. This is similar to assigning different edges with different colors.
- If two butterflies are the same, then label the edge as same, or else label it as different. We do the same, and if its ambiguous, we don’t need to label the edge.
- Then we traverse the entire graph using a BFS algorithm.
- We do this by taking a pair that is different, and label them as different. We run two different BFS algorithms on these different nodes. If from both ends, we label the node differently using each algorithm
- As BFS progresses, label nodes as A or B based on the judgments and BFS path.
- If both BFS traversals contradict either each other or the given judgments at any node, declare judgments as inconsistent.
  If traversal completes without contradictions, the judgments are consistent.

This algorithm has the lowest complexity when it comes to identifying any inconsistencies in the data.
This would have a complexity of (M+N) including the complexity of constructing the graph
Q3)
Question specifics: There are two major nodes, s and t, the number of levels or this distance between these two nodes are greater than n/2, i.e a majority of the nodes lie between both the points(and this path makes the shortest distance between the points s and t. To solve this question, we could designated levels(where each level represents the distance form the root node s to the current node at that level. We need to prove that there are chokepoints in the

graph, where deleting one node would lead to a disconnected graph, where s and t lie on different ends of the disconnected graph.

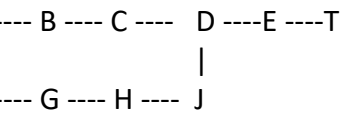
Intuitively, when looking at a graph where the distance between two nodes is greater than n/2, there are at least n/2 nodes lying in between the points s and t. According to the question, we need to prove that there are node/s where there is no alternate. To prove this, lets take the worse case scenario:

S ---- A ---- B ---- C ----D ----E ----T. (shortest path, not considering all the routes
According to the question, the minimum number of nodes in the setup must be less than twice the distance, which is less than 12. Assuming there are 11 nodes(the maximum), to prove by contradiction we would need a scenario where there is another alternative path from S to T which has a length greater than n/2



As we can see in the above scenario, creating an alternative path of length n/2 is not possible, and the above path is invalid as the new path length is less than n/2

Other scenario is



However, as we can see with the graph above, deleting either A, D or E would cease all paths from S to T.

Now, to find what node is critical, we create an algorithm
Steps:

- First we start from node S, and we designate levels based on the path length from S. Using this, there would be at least n/2 levels, and the destination node is T
- Then based on this we would designate levels and nodes to each of the levels. According to the proof above, if any level has one node only, that is a critical node and all paths that start from S have to use that route to reach to T

Q4)
For this question, we have a network of computers communicating, represented in the format (Ci,Cj, tk), where Ci and Cj communicated with each other at time tk. We have ben given a sequence of communications. The total number of such triples are m, and the number of computers in the network is n.

Solution: for this we could create a directed graph, where each of the nodes represent the computer pairs with the time they communicated at, etc. The levels associated with the nodes are related to the time, and if tk > tl, then the node containing tk would be on a higher level than the tl one.

How does the algorithm look like:

- Firstly, split the nodes into subnodes which holds the computer and the time it communicated. A triple would split into a pair, and we could create a two way edge between both these pairs which lie on the same level. For each triple (Ci, Cj, tk) in the trace data, create two subnodes: (Ci, tk) and (Cj, tk).
- When a computer communicates with another, it could be of two types- one that has previously communicated with a computer(already communicated in previous intervals) or newly encountered.
- We then construct a directed graph using these subnodes and their edges. This graph will capture a sequence of communications and the relationships between computers at different times and represent the network connections
- Given the computer Ca and the computer Cb, we find the node for Ca after the time interval it had been affected
- Perform a directed graph traversal (e.g., Depth-First Search or Breadth-First Search) starting from the subnode (Ca, x). During the traversal, keep track of the set of infected subnodes and label them as infected. Continue the traversal until you reach Cb or until no more nodes are left to discover.
- If you encounter the node with Cb through this list, then label it as possibly affected, else label it as uninfected

Proof that it works: Using a directed graph, we can travel linearly in time and can traverse the sequence of computers that could have been in indirect contact with the infected computer.

Q5)

There are a set of P people deceased in the past, and we know about the relationships between two individuals(whether one was born before the other or if their lives overlapped at some point of time. For this we could use a directed graph, where each person represent a node. Considering the layers are based on time, no cycles should exist in the graph, which would represent an inconsistency. We start building a graph based on the following rules:

We have information regarding P people who lives and were born in the past 200 years, all of which are deceased. We know about the relationships between two individuals(whether one was born before the other or if their lives overlapped at some point of the time) To the solve this, we could use a directed graph where each person represents a node. Considering that the layers are based on time, no cycles should exist in the graph

- Construct a directed graph G with n nodes (each for one person) and the edges for relative life. Initialize the birth and and death arrays of size n to -1.
- Construct the graph based on the following facts. Add edge from Vi to Pj for the fact "Vi died before Pj was born". Ensure no contradictory edges for overlapping life spans of Vi and Pj.
- Use DFS(Depth first search to detect cycles from one of the source nodes. If found, return inconsistent data. Sort graph nodes topologically. Assign birth and death years based on predecessors.

- If valid birth and death years assigned and there are no cycles found, return the nodes. If not, report inconsistent data.

Why It Works:

- The problem is modeled as a directed graph where the direction of edges represents time (i.e., if Vi lived before Pj).
- Cycles would mean time inconsistencies (e.g., A lived before B, B lived before C, but C lived before A).
- Topological sorting orders nodes such that every directed edge (u, v) from node u to node v has u come before v, which helps in determining possible birth and death years without contradiction.
- If the sort is successful and birth/death years can be assigned without contradiction, the data is consistent. Otherwise, it isn't.

Q6)

We could make use of an algorithm that would make it a tree with different levels, and we need to find the best way to reach the end of the array. Then once the graph is constructed, we could run a BSF algorithm to return the shortest path with the jumps we can make.

- Steps for the algorithm
  We could do this by taking the first element of the array and labelling it as the root of the tree. Then, based on the rules of jumping, we could label the levels of the tree, where adjacent numbers and same values are on the next levels.
- We look the elements that are adjacent to the current nodes at that level. We do a search in the array looking for similar values in the rest of the array. This would also lie on the next layers with layers, with the node having directed edges towards the one in the next level. We could run this as a recursive function to find nodes.
- Once we use a node in the graph and it has been discovered(through a jump), we label it as discovered. When making a tree, we will disregard all values and indexes that have already been discovered. This is because the shortest path to that particular value and index is the only concern.
- Once we reach the last index, we return the path from the source node to the end.