Homework #2 Solutions

(Due 11:59 PM Oct. 20)

1. **Exercise 3, Page 107 (20pts)**

- The following algorithm is based on the topological ordering algorithm discussed in the textbook.
- Algorithm (12pts):
    - Create a hashmap of all n vertices in the directed graph. This structure will keep track of the indegree of each vertex.
    - Initialize the hashmap by looping through every edge in the graph:
        - for every edge, increment the counter of the indegree in the hashmap of the node the edge points to.
    - Check every vertice in the hashmap, if it has an indegree of 0, add it to a set S with all vertices that have an indegree of 0.
    - While there exists a vertice with an indegree of 0:
        - Arbitrarily pick a vertex v from the set of vertices that have indegree of 0.
        - Remove that vertex from the set, and place it next in the topological ordering.
        - For every edge e originating from v, decrement the indegree of the vertex that e points to. If the indegree of the vertex e points to become 0, add it to the set of vertices with an indegree of 0.
    - If at this point all vertices have been put into the topological ordering, return that ordering since the given graph is a DAG.
    - If at any point there are no more vertices with in degree of 0, AND not all vertices have been put into the topological ordering:
        - Pick an arbitrary vertex v that has positive indegree, that has not been put into the topological ordering.
        - Find an edge that points to v, and "walk backwards" along that edge to the source of that edge.
        - Continue walking backwards along edges in a similar manner until you arrive back at a node that was visited before
        - Return this cycle since the graph is not a DAG.
- Proof of correctness (4pts):
    - See 3.18 from the textbook to show that if G has a topological ordering, G is a DAG.
    - See 3.20 from the textbook to show that if G is a DAG it must have a topological ordering. Thus the contrapositive is that if G does not have a topological ordering, then it is not a DAG and thus must have a cycle.

- Time complexity (4pts):
    - Using this “edge-centered accounting method” this algorithm for topological sorting can be shown to run in O(V+E) time, as shown in class and the textbook.
    - The worst case runtime for finding a cycle in a directed graph that is not a DAG is O(V), as in the worst case the longest cycle involves visiting every node once.
    - Thus the entire algorithm has a runtime of O(V+E)

Note: There exists a version of topological sort with DFS. That is acceptable for full credit with accompanying proof and time complexity analysis.

2. **Exercise 4, Page 107 (20pts)**

- Algorithm (12pts):
    - Create a graph where the butterflies are nodes, and the same/different judgments are the edges that connect the nodes.
    - Pick an arbitrary vertex v that represents a butterfly and perform BFS from v only using “different” edges.
        - Once all “different” judgments have been exhausted, connect nodes with “same” judgments from the leaves of the current tree on the same level as itself.
        - Continue running BFS from any newly accessible “different” edges.
        - Repeat until all nodes are explored.
    - Repeat the processes until all “different” edges have been processed to account for possibility G is not fully connected.
    - Assign all nodes on that tree a label based on the parity of the level it's on.
    - Assign all remaining nodes that were not explored by any BFS tree of “different” edges labels with the same parity.
    - For every “same” judgment:
        - If any “same” judgment involves connecting nodes from levels that have different parity, all the judgments are not consistent.
        - if any vertex was not labeled, label it with the same label of the other vertex in its corresponding “same” judgment.
    - For every “different” judgment:
        - If any “different” judgment connects any nodes from levels that have the same parity, then the judgments are not consistent.
    - Otherwise, all the judgments are consistent.
- Proof of correctness (4pts):
    - The only way a set of judgments can be inconsistent is if and only if there is a cycle that contains an odd number of “different” judgments.
    - Assume that the algorithm labeled a set of judgments that is inconsistent, as consistent:
        - This means that by using the parity of the labels given by the algorithm, the algorithm returned a valid bipartite partitioning of the nodes of a graph which is not bipartite, a contradiction.
    - Assume that the algorithm labeled a set of judgments that is consistent, as inconsistent:
        - That means that the BFS tree of the “different” judgments detected a cycle containing an odd number of “different” edges. A bipartite graph cannot have such a cycle, a contradiction.
- Time Complexity (4pts):
    - This algorithm uses a modified version of BFS, which runs in O(V+E).

- The algorithm also in the worst case sweeps through all judgments twice, thus incurring O(E) runtime, thus the whole algorithm has O(V+E) runtime.

3. **Exercise 9, Page 110 (10pts)**

- Algorithm (6pts)
    - Perform BFS and create a BFS tree with vertex S as the root node.
    - For every level in the resulting BFS tree
        - If at this level there is only one vertex, return that vertex.
- Proof of correctness (2pts):
    - Consider the BFS tree of a graph with n vertices that contains vertices S and T such that the distance between S and T is greater than n/2.
    - Thus the number of levels the BFS tree has must be greater than n/2.
    - Assume such a node v that cuts all paths from S and T does not exist.
    - Therefore there must exist 2 entirely unique paths of length greater than n/2 between S and T
    - However, since this is a BFS tree all nodes only appear in the BFS tree once, and since it took more than n/2 edges to construct the original path, there is not enough nodes to construct any other entirely unique path from S to T whose distance is greater than n/2, thus a contradiction.
    - Therefore there must always exist at least one node that is on a level all by itself.
    - That node with the unique distance must be the result, because the path from S to T must pass through that node.
- Time complexity(2pts):
    - The algorithm uses BFS, which runs in O(V+E) time.
    - In the worst case, searching through all levels of a BFS tree will take O(V) time.
    - Thus the whole algorithm will have O(V+E) runtime.

4. **Exercise 11, Page 111 (10pts)**

- Algorithm (6pts):
    - Maintain a hashmap H that maps each computer to a value indicating they are healthy.
    - Disregard all triples regarding communications before input time x.
    - For all remaining triples:
        - Check if a triple is the only one that occurs at this time by looking at the next triple:
            - If the triple is the only one that occurs at that time check if at least one of the two computers is infected. If so, mark the other as infected.
            - If none or both computers are infected, do nothing.
        - If there is more than one triple at this time, gather all triples at this time and create a graph by using the computers as nodes, and by using the triple that records two computers communicating with each other as edges.
        - Explore the created graph with BFS.
            - While exploring the triples at the same time with BFS, if even one computer in a connected component is infected, mark this entire component as infected.
        - After iterating through all triples, traverse all triples at this time again and add all computers in all the infected components to the set of infected computers.
    - After all remaining triples up have been analyzed up to time y:
        - If hashmap indicates the computer in question c_b was infected, return true. Otherwise return false.
- Proof of correctness (2pts):
    - If a computer in reality was infected earlier than the algorithm returned, then there must exist a series of triples that indicated that earlier in time. Since all triples are evaluated in sorted time order, this is a contradiction.
    - If a computer in reality was infected after then the algorithm returned, then the algorithm at some point encountered a series of triples that made c_a infect c_b. This is a contradiction from the assumption that the c_b was infected at a later time.
- Time complexity (2pts):
    - This algorithm first creates a hashmap structure for all m computers, thus incurring O(m) runtime.
    - Although the algorithm runs BFS multiple times, the algorithm processes each triple it encounters once, incurring O(n) runtime.
    - Thus the entire algorithm has a O(n+m) runtime.

5. **Exercise 12, Page 112 (20pts)**

- Algorithm (12pts):
    - Created a directed graph according to the following rules:
        - each person p_i will have two nodes associated with them, p_{ib} and p_{id}, representing when they were born and when they died. Create a directed edge pointing from p_{ib} to p_{id}.
        - For each fact stating p_i died before p_j was born, create a directed edge from p_{id} to p_{jb}
        - For each fact stating p_i was alive at the same time as p_j, create a directed edge from p_{ib} to p_{jd}, and another edge from p_{jb} to p_{id}.
    - Perform a topological sort of the created graph.
    - If a topological ordering is not possible, then return the facts are not consistent.
    - Otherwise the facts are consistent, and assign birth/death dates in ascending order based on the topological ordering.
- Proof of correctness (4pts):
    - Facts are inconsistent only if there is a cycle of facts including at least one of the type "p_i was born before p_j".
    - A cycle purely consisting of the facts "p_i was born before p_j" will trivially create a cycle in the directed graph the algorithm creates.
    - A cycle that contains one or more facts of the type "p_i was alive at the same time as p_j" will also create a cycle. This is because assuming that the given fact is true either:
        - p_i must have been born before p_j died must be true OR
        - p_j must have been born before p_i died must be true.
    - Thus there must exist an edge that connects the cycle of the fact of the type "p_i was alive at the same time as p_j" in the set of facts that creates a cycle.
    - Assume that the algorithm labeled a set of facts incorrectly.
        - This is a contradiction because a topological ordering can only be created if and only if the underlying graph is a DAG.
- Time complexity (4pts):
    - To create the directed graph, the algorithm creates either 1 or 2 edges per fact, thus incurring O(E) runtime, where E is the number of facts. There are V people the facts are about, our algorithm creates 2V nodes, and for each node we create an edge, thus we have O(V) time complexity for creating processing people.
    - This algorithm runs a modified topological sort of the graph with O(E) edges and O(V) nodes, thus the whole algorithm has time complexity O(V+E).

6. **Given an array arr[] of size N, the task is to find the minimum number of jumps to reach the last index of the array starting from index 0. In one jump you can move from current index i to index j, if arr[i] = arr[j] and i != j or you can jump to (i + 1) or (i – 1). (20pts)**

- Algorithm (12pts):
    - Convert the input array into a graph as follows:
        - Every element in the array becomes a node. Each node stores its own value, as well as its index in the array.
        - Create an edge for each node to the node that represents its neighbor in the array.
        - Create an edge for each node to every other node that shares its same value.
    - Run BFS on the resulting graph with the node at index 0 as the root.
    - Output the level of the node whose index is the last element of the array.
- Proof of correctness (4pts):
    - The graph constructed by definition represents all possible jumps possible from an element in the array.
    - Assume that there is a quicker route to jump to the end of the array.
    - That means that the level that the last node of the array appeared in the BFS tree is not equal to its tree distance from the root, a contradiction of the BFS property.
- Time complexity (4pts):
    - In constructing the graph that represents the possible jumps, in the worst case the resulting graph will be fully connected, and this step will take O(N^2) time.
    - Running BFS will then also be bounded by O(N^2). Thus this algorithm will run in O(N^2) time.

Note: There also exists an O(N) solution that uses hashmaps mapping nodes and their values with their indices that they appear in the array. That is acceptable for full credit with accompanying proof and time complexity analysis