HW2 CS 180

ASHER CHRISTIAN 006-150-286

1. Exercise 10, P 110

given an undirected graph $G = (V, E)$ and nodes $v, w \in G$. Compute the number of shortest $v - w$ paths in G in $O(m+n)$ time for n nodes and m edges. I propose a modified version of BFS that searches from v until finding w and then counts all connections to w in the last layer. The pseudocode is as follows

- `Initialize array Discovered of length` n `setting Discovered[v] = true and Discovered[n] - false for all other n`
- `Initialize a pointer to a linked list Current to consist of the element v`
- `Initialize a pointer to a linked list Next to be empty`
- `Initialize count = 0`
- `While count = 0 and Current nonempty`
  - `For each node u` $\in$ `Current`
    - `Consider each edge` (u, l) `incident to` u
    - `If l = w`
      - `increment count by one and continue onto the next node u`
    - `If Discovered[l] = false then`
      - `Set Discovered[l] = true`
      - `add l to Next`
  - `Clear current and switch Current and Next pointers`
- `If count > 0 return count else return "no connections"`

By the nature of the BFS algorithm, this algorithm is guaranteed to find the shortest path from v to w in $O(m+n)$ time. The modification is that after finding the shortest path it checks all elements that are $n-1$ steps away from v to see if they also reach w and adding those to the count. This is not adding any operation that would not be completed in a worst case BFS search so it does not increase the time complexity.

2. Exercise 4, P 108

To accurately label the butterflies take a similar approach to the bipartite coloring problem. In this case create a graph with an edge between every two specimins that have a judgement and apply the following algorithm

- `Initialize a Discovered array and set Discovered[v] = False for each node`
- `initialize an array with labels`
- `While there is a an undiscovered node`
  - `Pick an arbitrary node v and label it` A
  - `create a queue and add v to it`
  - `While the queue is not empty`
    - `Pick the first element of the queue`
    - `For each adjacent element`

Date: 22.01.25.

        - `If the element is not discovered: Set it to discovered, label it A or B depending on the relation of the edge and the label of the preceding edge, and add the element to the queue`
        - `Else if the element is discovered: Test to see if the coloring proposed by their relation is equivalent to the coloring that the node has. If there is a contradiction abort the program and return the statement.`
- `If the program completes without error return true`

This program checks every node and does a constant time operation on them to check/set/verify coloring. So this program runs in the same time as traditional breadth first search in $O(m+n)$. Assume for contradiction that it outputs the wrong analysis. Two cases.

- Program output true - real output false
  Then there is one set of nodes (u, v) such that their labels are not matching. However, every node was discovered in the course of the algorithm and every edge checked, therefore the edge u, v was checked. If it was the first time u or v would be checked it would have the correct coloring. If it was a different time then the mismatch would be caught and returned.
- Program output false - real output true
  Then there was a node pair (u, v) such that the program ended early upon visiting v from u, but this can not be the case because every color is determined by the relation leading back to the starting node.

3. Exercise 9, P 110

I propose an algorithm to solve this problem. The algorithm uses breadth first search starting at s
The version of breadth first search stores the nodes at each layer. Consider the algorithm that stops after reaching t from s. There must be $\lfloor \frac{n}{2} \rfloor$ layers at this point and t must be placed in the $\lfloor \frac{n}{2} \rfloor + 1$ layer. In each of the previous layers there must be at least one node. There must be a layer with exactly one node. For assume for contradiction taht every layer contained 2 or more nodes. Then there would be at least n nodes in the $\frac{n}{2}$ layers. But no layercan include the start node or the end node so there are 1 or 2 nodes too many. Thus we can search through every layer until we find one with only 1 node and do this in $O(n)$ timebecause there are at most n nodes to check. And return a node in a layer by itself. If this node is deleted by theorem from previous chapter every edge connecting nodes in a BFS Tree connects nodes to either the same or adjacent layers so severing a standalone layer would sever all connections between the nodes before and the nodes after that layer. This algorithm operates in $O(n+m)$ time because it is the same as BFS with an added step at the end taking $O(n)$ time.

4. Exercise 11, P 111

- `Create a new graph G and an array for each node that stores the nodes it has connected with.`
- `For each triple` (C_i, C_j, t_k) `create a node` (C_i, t_k) `and` (C_j, t_k) `and connect the two in both directions`
- `Check to see if either` C_i `or` C_j `have previous nodes if they do connect their latest nodes to their new nodes in the forward time direction`
- `Add the created nodes to the list of nodes for` C_i `and` C_j `respectively.`

- store the first node such that C_a is at time greater than the starting time
- Once all nodes are created run BFS on the first node as mentioned before and terminate if find C_b at a time before time y
- If BFS terminates by not finding a node matching the above description then C_b does not get infected before time y

This algorithm works for if it finds a path then by the specifications there must be a sequence of times and respective computers that the virus traveled from C_a to C_n in forward time. This is because every edge on the graph shows either constant time or forward time travel so every path is reachable by the virus. This is because every edge in the graph is a true path the virus could have taken. Suppose that there was a path but the algorithm did not find it, this is not possible since consider this supposed path It must leave C_a at some point to travel to C_i this must be recorded in the graph by a finite number of time steps staying at C_a and then a constant time step from C_a to C_i. Inductively this argument holds for all future C_i until C_b so the supposed path is represented by the graph and thus BFS would have found it. The algorithm generates a graph in $O(m)$ time doing constant operations per triple. then since the nodes and edges is a linear function on m the BFS runs in $O(m)$ time as well and so the overall algorithm is $O(m)$.

5. Exercise 12, P 112

- Initialize a directed graph G with a node P_i^b and P_i^d for each $i \in \{1, .., n\}$
- For each P_i add the edge $P_i^b \to P_i^d$
- For each relation:
  - If P_i died before P_j born add an edge $(P_i^d \to P_j^b$
  - If P_i and P_j were alive at the same time add the edge $P_i^b \to P_j^d$ and the edge $P_j^b \to P_i^d$
- Run toposort algorithm on this graph - If no toposort is available report that no dates exist
- In the topological graph assign each node a number in ascending order corresponding to a date and the dates align with the proposed birth and death dates of each person

I propose first that this algorithm works. In the final output every birth comes before every death and if someone died before another was born then by the guarantee of the topo sort algorithm they died before the birth because the graph had a directed edge signaling that. If two people were alive at the same time, this algorithm guarantees that both people were born before the other died which guarantees that they were alive at the same time. So any ordering given satisfies the conditions. Assume the algorithm said no ordering exists when one does. Then the algorithm must have found a cycle because it did not create a topo sorting but a cycle implies that someone was born before they were born or died before they died which could not have happened.

6. Exercise 6 in HW

- Create an undirected graph G and populate it with nodes labeled $0, 1, ..., N-1$.
- create an edge between each adjacent number e.g. $1 \to 2$, $j \to j+1$
- Create an empty hash map H mapping a number to a list of nodes
- For each element in the array
  - Check H for other nodes with the same value and create an edge for each if the nodes are not exactly one away from the current node

  - `Add the array index to the list of nodes associated with the element value`
- `Run BFS starting at the node` 0 `and stopping when reaching the node` $N-1$ `recording the BFS tree`
- `Return the shortest path from` 0 `to` $N-1$

I claim that this algorithm returns shortest possible path. In fact for any valid array index path every jump taken is represented by an edge in the graph and every edge in the graph is a valid move. Therefore the shortest path through edges in the graph is the same as the shortest path using array indices. Additionally BFS is guaranteed to return the shortest series of edges and thus the shortest array indices.