Sum Yi Li
(UID. 505146702)

Discussion 1A
10:00 am - 11:50 am
Goldstein Orpaz

(1)

- Given an arbitrary graph G that may or may not be a DAG.
- While we have not visited all the nodes in the graph G or we have not found a cycle in the graph G.
- Try to find a node v with no incoming edges
- If we find a node v with no incoming edges, then we will find a topological order
    - Order the node v first
    - Delete v from G
    - Recursively compute a topological ordering of G - {v} and append this order after v
    - Output the topological order of the recordered list

- Graph G is DAG

- If we fail to find a node v with no incoming edges, which means every node has at least one incoming edge, this shows that Graph G has a cycle
    - We are currently at an arbitrary node b
    - We will repeatedly follow an edge and pick the first adjacency edge among all the incoming edges
    - Record down every new visited node
    - Repeat this until we re-visit node b again
    - Output all the visited nodes between the two follow up visits of node b, this is our cycle for G. Graph G is not a DAG

① cont.

The algorithm takes $O(n)$ to search for a cycle because we are following the edges of the current node we are at until we reach back to the original node we start off

②

- We are using the algorithm to build an undirected and not connected graph because we will perform breadth first search and each component requires a starting node.
- All our specimen are represented as vertex while any edge between two arbitrary vertex V_x and V_y are represented as the judgements between the two specimens.

- For every component B of the graph G
  - Appoint a starting node n and give a letter label A
  - Make a note that node n as being visited
  - Create a output list for the breadth first search and record down node n. The output list is represented as Z
  - For the first layer of the BFS graph, label it with the starting node as $L(0) = n$
  - Let y be the index of traversing between different layers such that $y = 0, 1, 2, 3, 4, \ldots$
    - For every node x within layer y which is $L(y)$
      - Examine every edge between node x and an arbitrary node w, which are edges that are incident to node w
      - If node w has not been visited which is has not been labeled yet
        - Label node W as being visited
        - If the judgement comes out as the same, then it means node w has the same label as node x

- else, the judgement come out as different, then it means node v has the opposite label of node x
- Endif
- Insert node v to the output list Z. Also, append node v to the next layer which is $L(y+1)$

Endif
Endfor
Endfor
Endfor

refer to the starting statement from the previous page

- We are gonna check whether the m judgements are consistent by running the following pseudocode.

```
- Examine every single edge between node x and node w.
    - If the judgement comes out to be the same
        - If node x and node w don't have the same label
            - Output inconsistency message
        - Endif
    - Else the judgement comes out to be different
        - If node x and node w have the same label
            - Output inconsistency message
        - Endif
    - Endif
- Endfor
```

③ - According to the problem statement, we know that the distance b between node S and t is greater than $n/2$ which means $b > n/2$

- We will use BFS algorithm to follow the path from the starting node S to the ending node t
- Let assume we will find node t at layer Y
- Let further assume that within layer $Y_1, Y_2, Y_3, \ldots, Y_{b-1}$, there exists a single node.
- We want to eliminate the possibility that each layer Y_i could have at least size two to show there is at least one layer that has only one node.
- If the size is two for each layer, the distance will be included at least $2(\frac{n}{2}) = n$ nodes
- But, we only have n nodes and both node S and t are not within $Y_1, Y_2, Y_3, \ldots, Y_{b-1}$ layers
- Therefore, there is a layer Y_i such that it only contains one node which is node V.
- The problem statement says that deleting node v from graph G will destroy all the paths from node S to node t
- Let set up a set of nodes that does not include node t within $Y_1, Y_2, \ldots, Y_{i-1}$ layers
This means that our set A is the union between node S and all the layers $Y_1, Y_2, \ldots, Y_{i-1}$

- Any edge that are not included in set A should be included in the layer Y_i
- In order to move to node t, we will somehow get out from set A.
- This means that node v is the only node at layer Y_i.

④ - We will create a directed graph as our algorithm solution.

- We first examine all the possible sorted tripes.
- We create an array and each of the slot will point to a separate linked list
- Each of the linked list is related to computer C_a
- Throughout the scan, for every triple (C_i, C_j, t_K) we have examined, we gonna separate the triple into 2 separate nodes which are (C_i, t_K) & (C_j, t_K)
- We will connect the **two** nodes with edges that could go into either direction
- We will create a list for C_i and a list for C_j.
- The list for C_i will be constructed as linking all the nodes that has C_i but different communication time, such as $(C_i, 1)$ & $(C_i, 2)$
- The list for C_j will be constructed as linking all the nodes that has C_j but different communication time.
- If we have already passed the first triple that include C_i, we will draw a directed edge from (C_i, t_a) to (C_i, t_b), t_a just refer to any previously last one node within the list of C_i.
- We will repeat the last bullet point for the list of C_j.
- Both C_i and C_j let us to keep track of each triple's nodes and edges by making the time constant.

- In order to determine whether a virus introduced at computer C_a at time x have infected computer C_b by time y, we need to examine the list of C_a
- We will traverse the list of C_a up to the point when we reach the last node in the list.
- Let assume that the first node is (C_a, x) and the last node is (C_a, x') such that $x' \leq x$
- Starting from (C_a, x'), we will use BFS and trace all the possible reachable nodes from there
- Throughout the journey, if there is a node (C_b, y') that is reachable such that $y' \leq y$
- Then, we will conclude that computer C_b might have been infected by virus by time y.
- If there isn't a node (C_b, y') that is reachable, then C_b might have not been infected by the virus.

⑤

- For this algorithm, we will represented the algorithm by a directed graph A.
- People from the village will be denoted by P_x
- Every single villager will have their own birth dates and death dates. They will be denoted by B_x and D_x
- The edges in the graph will be represented as links that connect the current fact with the previous fact.
- We will link up the birth dates and the death dates of each person. It is denoted by the set (B_x, D_x)
- According to the problem statement, we will create an edge for (D_i, B_j) under the fact that "person P_i died before person P_j was born for some i and j"
- According to the problem statement, we will create an edge for between two sets (B_i, D_j) and (B_j, D_i) under the fact that "the life spans of P_i and P_j overlapped at least partially for some i and j"
- However, there are two possible situations that could have happened for the graph: whether the graph has a cycle or not

- First, if the graph has a cycle, this means that each event has to be in the order that correctly place before the next event.
- However, a cycle also means that there isn't an event which could be considered as the starting node. So, there does not exist an beginning or an end
- Therefore, if our graph exists with cycles, the facts are not internally consistent
- Second, if the graph does not have a cycle, then we can represent the order of each person's birth date and death date in a topolcgical order
- Therefore, if our graph does not exist with cycles, the facts are internally consistent.

6a)

- According to the problem statement, the array is sorted, therefore we could use binary search.
- We are looking for the first occurence of array $G[x] = x+1$ which is our missing number at the lowest index
- We will recursively search the first half of array G when $G[\frac{n}{2}] = \frac{n}{2}+1$ and $x \leq \frac{n}{2}$
- We will recursively search the second half of array G
- In both cases, the time analysis is

$$O(n) = O(\frac{n}{2}) + \Theta(1)$$

- So, the worst case running time is

$$\boxed{O(\log n)}$$

6b)

- According to the problem statement, the array is not sorted, we need to check every slot of the array G for the missing element
- The algorithm works as follows:
- We will create a helper array W with index from 1 to n+1 & each slot of the array will be initialize to zero.
- For x from 1 to n, we scan through both array G & W based on $W[G[x]] = 1$
- Then, we scan through array W to search for the slot that contains a zero, the slot index will indicate our missing element.

Time Analysis:

Construct array W $\rightarrow O(n)$

Scan array G $\rightarrow O(n)$

Scan array W $\rightarrow O(n)$ +

$$3\,O(n) \approx \boxed{O(n)}$$