Name Mark Floryan

Quiz - Module 4: Graphs (BFS / DFS)

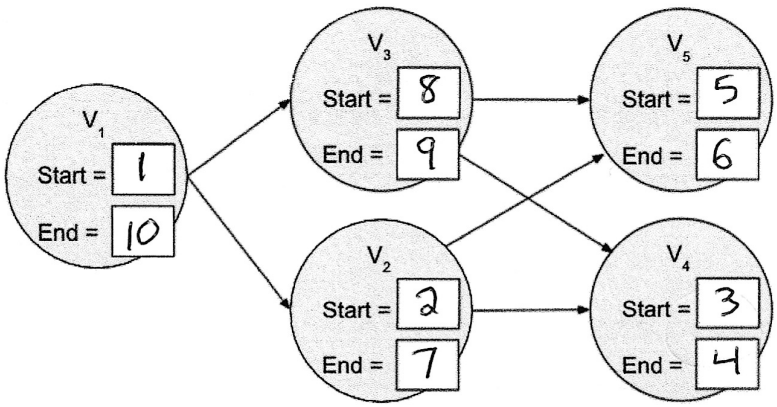
1. [8 points] Answer the following True/False questions regarding *topics from module 4.*

When running *DFS*, a node has color *white* while we are searching what will become its DFS subtree.	**True**	**False**
The runtime of *DFS-Sweep* is $\Theta((V+E)*V)$ because you need to run a recursive *DFS* up to $\|V\|$ times.	**True**	**False**
With *BFS* it is possible that the internal *queue* contains $\|V\|-1$ nodes on it at the same time.	**True**	**False**
DFS, run from one single arbitrary start node, is guaranteed to visit every node on any arbitrary **undirected** graph. (Slides called this function *DFS-visit* or *DFS-recurse*.)	**True**	**False**
BFS, run from one single arbitrary start node, is guaranteed to visit every node on a connected, **undirected** graph.	**True**	**False**
DFS finds a back-edge in a directed graph if and only if an outgoing edge while traversing leads to a *gray node*.	**True**	**False**
Our algorithm for finding SCCs in a digraph G works by calling *DFS-sweep* on the transpose graph where the calls it makes to the recursive *DFS-visit* function are made on vertices in the order of decreasing finish time from a topological sort on G.	**True**	**False**
The *Strongly Connected Components* of a directed graph are themselves a *tree* (if each *SCC* itself is treated as one node).	**True**	**False**

2. [1 points] Let's say want to use BFS to find the shortest path from s to v (i.e. smallest number of edges) in an unweighted and undirected graph and return a list containing the nodes on that path (or an empty list if no path exists). Which one of the statements is true about using BFS to do this?

   - If we find v, we still need to continue searching because we may later find a path to v thats better than the one we just found.
   - When we find v, the path from s to v will be the nodes currently in the queue.
   - When we find v, we can get the path if we have kept a record of which node u was being processed when its neighbor v was added to the queue.
   - The path will be the nodes on the stack, and we can find them by backing up through the active recursive calls until we get back to the call on the start node s.

3. [5 points] Run DFS on the following graph. List start and finish times (beginning at $t = 1$) in the boxes in each node. Use V_1 as your start node. *When multiple nodes can be searched, always search neighboring nodes in increasing order (e.g., if V_2 and V_3 are both adjacent to the current node, search V_2 first).*



4. [3 points] Using your answer above, give the specific *Topological Ordering* that would be produced by the *DFS-based* algorithm we discussed in class.

V_1 V_3 V_2 V_5 V_4

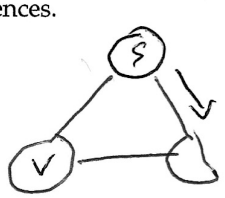
For these questions, consider a graph $G = (V, E)$ on which we run both *BFS* and *DFS* (separately). These produce a BFS-Tree T_B and a DFS-Tree T_D respectively. Also consider some arbitrary node $v \in V$. The following questions ask about the relative depth of v in T_B and T_D

5. [2 points] Is it possible for the depth of v in T_B and T_D to be *equal*? If so, draw a small example graph. If not, briefly explain why in one or two sentences.

yes, if G is a tree (one example)

6. [2 points] Is it possible for the depth of v to be **less** in T_B than in T_D? If so, draw a small example graph. If not, briefly explain why in one or two sentences.

yes



DFS goes away from s first

7. [2 points] Is it possible for the depth of v to be **more** in T_B than in T_D? If so, draw a small example graph. If not, briefly explain why in one or two sentences.

No, because depth of v in T_D is optimal

Name Mark Floryan

Quiz - Module 5: Kruskal's and Find-Union

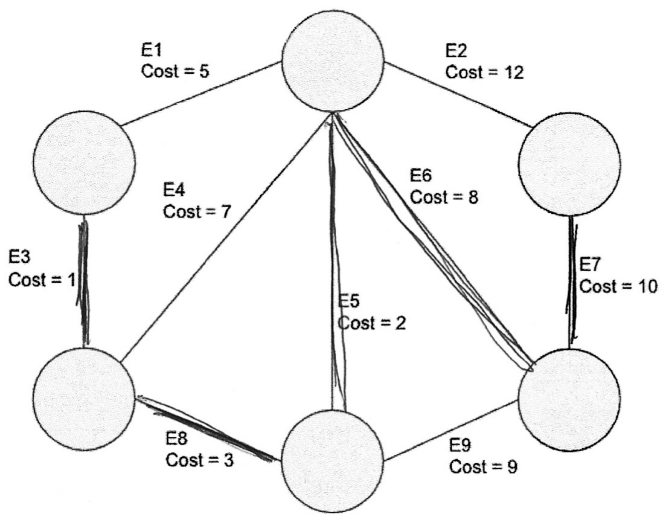
1. [6 points] Answer the following True/False questions regarding *Minimum Spanning Trees and Kruskal's Algorithm*

If an undirected graph has at least one cycle, then it must have more than one *spanning tree.*	**True**	**False**
When implementing *union(i,j)* in an array-based find-union, it might make future operations faster if i takes the label of j (rather than the other way around).	**True**	**False**
If a graph G has exactly two edges that both have the same minimum weight, then G must have at least two *MSTs*.	**True**	**False**
Kruskal's algorithm cannot end its loop before processing every edge. The algorithm must pop and consider every edge of a graph before terminating.	**True**	**False**
The Find-Union's *union(i,j)* method runs in constant time if the labels of the two sets are provided as i and j.	**True**	**False**
The Find-Union's *find()* method is best-case constant time.	**True**	**False**

2. [4 points] Answer the following short questions which ask about runtimes, sizes of graphs, etc.

Suppose an undirected graph $G = (V, E)$ is **not connected**, but has three connected components, how many edges does any *spanning forest* of G contain (in terms of $\|V\|$ and/or $\|E\|$)? *Note: Because G is not connected and has three components, there cannot be one single spanning tree but there can be three disconnected spanning trees.*	$\|V\| - 3$
Suppose I have a **complete**, undirected graph $G = (V, E)$ with exactly four nodes. How many unique *spanning trees* does G have (we want the exact number)?	16
What is the runtime of Kruskal's algorithm if find() and union() are $\Theta(1)$ time	$E \cdot \log V$
Suppose we find a way to *implement all priority queue operations in constant time.* What would Kruskal's overall runtime become in this scenario?	E

3. [3 points] Run Kruskal's algorithm on the given graph. List the order in which the edges are added to the MST. List an edge by its provided label.



E3 E5 E8
E6 E7

Consider the following recursive implementation of *union(i,j)* on an array-based find-union data structure. *Assume we have a rank array that stores the rank of each element i*

```
int[] sets; //The internal array
int[] rank; //Ranks. Initially all 0

/* Union sets that i and j belong to */
void union(int i, int j){
        int x = find(i); int y = find(j);                    //line 1

        if(rank[x] < rank[y]) sets[y] = x;                   //line 2
        else{
                sets[x] = y;                                 //line 3
                if(rank[x] == rank[y]) rank[x]++;            //line 4
        }
}
```

4. [2 points] This program does not work as intended. However, just two characters can be changed to make it correct. In the code above, circle the two characters that need to be changed to fix this method.

5. [2 points] What would you change each of the two characters to in order to fix the bugs?

< => >

x => y