CS4102 - ALGORITHMS - SPRING 2022

DUE FRIDAY, MARCH 18, 2022 AT 11:30 PM VIA GRADESCOPE

B2 BASIC (SOLUTIONS)

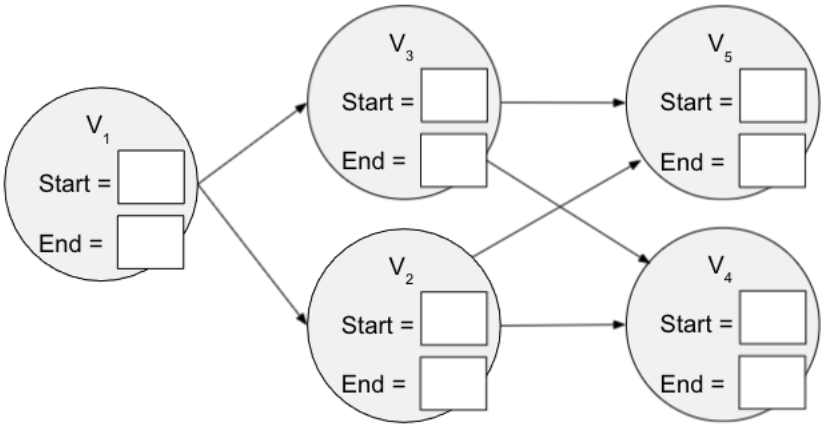
Collaboration Policy: You are encouraged to collaborate with up to 3 other students, but all work submitted must be your own *independently* written solution. List the computing ids of all of your collaborators in the `collabs` command at the top of the tex file. Do not share written notes, documents (including Google docs, Overleaf docs, discussion notes, PDFs), or code. Do not seek published or online solutions for any assignments. If you use any published or online resources (which may not include solutions) when completing this assignment, be sure to cite by naming the book etc. or listing a website's URL. Do not submit a solution that you are unable to explain orally to a member of the course staff. Any solutions that share similar text/code will be considered in breach of this policy. Please refer to the syllabus for a complete description of the collaboration policy.

Collaborators: list collaborators's computing IDs

Sources: Cormen, et al, Introduction to Algorithms. *(add others here)*

PROBLEM 1 *DFS and Topological Sort*

1. Run DFS on the following graph. List start and finish times (beginning at $t = 1$) for each node in the table shown below the image of the graph. Note: For this problem, by "start" we mean the discovery time, and by "end" we mean finish times.) Use V_1 as your start node. *To help us grade this more easily, 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).*



Vertex	V_1	V_2	V_3	V_4	V_5
Start	?	?	?	?	?
End	?	?	?	?	?

Solution:

Vertex	V_1	V_2	V_3	V_4	V_5
Start	1	2	8	3	5
End	10	7	9	4	6

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

Solution: Sorted by reverse finish time: V_1, V_3, V_2, V_5, V_4

PROBLEM 2 *True or False. (You don't have to explain this in your submission, but you should understand the reason behind your answer.)*

A. If you use DFS to find a topological sorting on a directed graph, the last vertex discovered in the search could legally be the last vertex in the sorted ordering of the vertices.*Update: Assume your first call to DFS-visit() visits every node in the digraph!*
Solution: True (but see below about the update). If vertex v is the last discovered, then there can't be any edges from v to an undiscovered vertex (or DFS would have visited that one, discovering it after v). Therefore v could be last in a topologically sorted list of vertices without violating the definition of topological sorting. *About the update:* If the first call to DFS-visit() did not reach every vertex, then this is **not true**. In that case, the last vertex discovered by the first call could legally be last in some overall topological ordering, but not in the ordering that our DFS-sweep() strategy would find. (Recall there can be and often is more than one valid topological ordering.)

B. For the disjoint set data structure we studied, if we had a $\Theta(\log n)$ implementation of *find-set()*, then the order class for the time-complexity of *union(i,j)* would be improved (i.e., better than the result we learned).
Solution: True. Before doing *union()*, we have to call *find-set()* twice, which is we learned *union()* is $\Theta(n)$. After those calls, *union()* just does an assignment, so its time-complexity will be the same as find-set's.

C. Both path-compression and union-by-rank try to improve the cost of future calls to *find-set()* by making the trees representing a set shorter without changing the set membership for the items in that set.
Solution: True. This is a succinct description of the overall goal of both of those improvements!

PROBLEM 3 *Kruskal's Runtime*

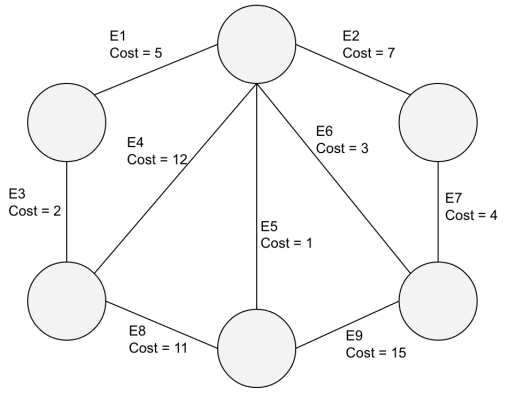
What is the runtime of Kruskal's algorithm if find() and union() are $\Theta(1)$ time?
Solution: $\Theta(E \log V)$

PROBLEM 4 *Strongly Connected Components*

Your friend Kai wants to find a digraph's SCCs by initially creating G_T and running DFS on that. In other words, he believes he might be able to *first* do something with the the transpose graph as the first step for finding the SCCs. (The algorithm we gave you first did something with G and not with G_T.)
Do you think it's possible for Kai to make this approach work? If not, describe a counter-example or explain why this will fail.
If it is possible, explain the steps Kai's algorithm would have to do to complete the algorithm, and briefly say why this approach can lead to a correct solution.
Solution: Yes, it is possible because a graph G and its transpose G_T have the same SCCs.
Kai would have to follow the same steps as the algorithm taught in class, but reverse the role of G and G_T. After using DFS to find the finish times in G_T, he'd call DFS-visit() on vertices in G in the order of their finish time as found in G_T.

PROBLEM 5 *Executing Kruskal's MST Algorithm*

Run Kruskal's algorithm on the graph below. List the order in which the edges are added to the MST, referring to the edges by their provided labels.
(*Consider how your answer would change if E1 had weight 12. However, you don't need to provide an answer to us for this part.*)



Your answer (list of edges in order):
Solution: List of edges: E5, E3, E6, E7, E1
If E1 had weight 12, it would not have been the last edge added. We would have considered E8 before E1 and chosen E8 as the final edge.

PROBLEM 6 *Difference between Prim's MST and Dijkstra's SP*

In a few sentences, summarize the relatively small differences in the code for Prim's MST algorithm and Dijkstra's SP algorithm.
Solution: In short, Prim's uses the weight of a *single edge* from the tree to a newly discovered node w, while Dijkstra's uses the *total distance from the start node to* w. These values are used when a node is first discovered and the cost is stored the PriorityQueue, and when a new connection is found that is better than the stored value, and DecreaseKey is called to update the cost, and is used by ExtractMin to return the next best value.

PROBLEM 7 *True or False. (You don't have to explain this in your submission, but you should understand the reason behind your answer.)*

A. An *indirect heap* makes *find()* and *decreaseKey()* faster (among others), but *insert()* becomes asymptotically slower because the indices in the indirect heap must be updated while percolating value up towards the root of the heap.
Solution: False. We do have to swap values in the indirect heap, but this is constant time and so the time-complexity of insert() will not change. (Note: with this question we wanted you to think about heap operations on their own. The question is not about the overall complexity of Dijkstra's or Prim's.)

B. If all edges in an undirected connected graph have the same edge-weight value k, you can use either BFS or Dijkstra's algorithm to find the shortest path from s to any other node t, but one will be more efficient than the other.
Solution: True, both will find the shortest path. BFS will be $\Theta(V + E)$, which is faster than Dijkstra's worst-case $\Theta(E \log V)$

C. In the proof for the correctness of Dijkstra's algorithm, we learned that the proof fails if edges can have weight 0 because this would mean that another edge could have been chosen to another fringe vertex that has a smaller distance than the fringe vertex chosen by Dijkstra's.

Solution: False. The proof only fails if edge weights can be negative. If an edge weight is zero, then the δ described in the proof could be 0, which could mean there is an equally good choice that Dijkstra's could have chosen but did not, but its the same distance and not better so the proof holds.

PROBLEM 8 *Gradescope Submission*

Submit a version of this `.tex` file to Gradescope with your solutions added, along with the compiled PDF. You should only submit your `.pdf` and `.tex` files.