

CS 4102 Unit B Exam #2**Name** Winston Zhang

You MUST write your e-mail ID on **EACH** page, and put your name on the top of this page, too.

You MUST write each answer on the same page as the question in the space provided for an answer. (This is necessary because we scan exams and grade them using GradeScope.)

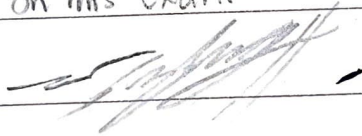
If you are still writing when "pens down" is called, your exam will be ripped up and not graded.

There are 6 pages to this exam, including this page. There are a total of 100 points on the exam. Once the exam starts, please make sure you have all the pages. Questions are worth different amounts, so be sure to look over all the questions and plan your time accordingly.

This exam is **CLOSED** text book, closed-notes, closed-cell phone, closed-smart watch, closed-computer, closed-neighbor, etc. It is an honor violation to communicate information about the exam to anyone who might not have yet taken it. Please write and sign the honor pledge below.

On my honor as a student, I have neither given nor received

unauthorized aid on this exam



© 2022 Thomas B. Horton and John R. Hott

The instructors of this course hold copyright in this exam. Per UVA Policy PROV-005, students may not copy, reproduce, display, or distribute any part of this exam to other individuals without explicit written consent of the instructors. Students may neither exchange nor distribute any part of this exam for commercial purposes, for compensation, or for any other purpose other than study by students enrolled in the class during the same term. Improper distribution of such materials by students disrupts the University's learning environment and is therefore a violation of the Standards of Conduct, and could subject a student to disciplinary action.

Page 2: Problem Solving with BFS or DFS

1. [12 points] A graph is two-colorable if each vertex can be assigned one of two colors, say orange or blue, such that for every edge in the graph (v_i, v_j) the vertices v_i and v_j do not have the same color.

For example, a tree is two-colorable because the nodes in alternating levels of the tree can be different colors. An undirected graph with three vertices and edges that make it look like a triangle is not two-colorable, but four vertices and with edges that make it look like a square is two-colorable.

Write an algorithm that takes an connected undirected graph $G = (V, E)$ and returns a list of edges E' that can be removed from G to make it two-colorable, i.e. the graph $G = (V, E - E')$ is two-colorable. If G is two-colorable to begin with, the list you return will be empty.

Note: we'd like you to modify BFS or DFS-visit to solve this, and it is sufficient to only describe how you'd modify one of those algorithms and how it's called in order to solve this problem. Also, we suggest you label each vertex either *orange* or *blue* as you discover it and use this information.

We can use an approach similar to BFS, initializing the graph with white nodes as well as an empty ArrayList to track which edges to remove.

Starting at start edge S , color it orange and then add it to the queue.

Then add each of its neighbors to the queue, coloring them blue. When all neighbors have been enqueued and colored, dequeue S .

Repeat this process of visiting and enqueueing nodes, then enqueueing all of their ^{unvisited} neighbors and coloring them with the opposite color. If a node is colored, then found to be adjacent to a node of the same color, add the edge connecting the two to the array list.



By the time this loop is over, all nodes will have been visited, and all invalid edges will have been found and added to the ArrayList.

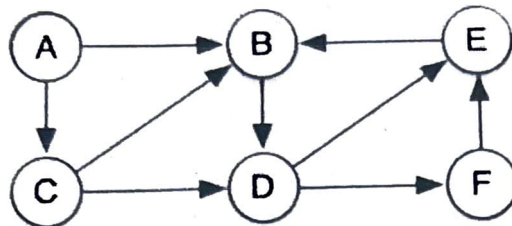
We can then finish by returning the ArrayList. If no offending edges are found, the algorithm returns an empty ArrayList.

2. [5 points] Give the Θ order-class of the worst-case time-complexity of your algorithm above.

$\Theta(V+E)$

Page 3: DFS and Topological Sort

The following questions require you to do a DFS-visit() starting at vertex A on this graph. To help us grade this more easily, when there is an equally valid choice of which vertex to visit next, always choose based on increasing alphabetical order (e.g., if B and C are both valid choices for the next visit, choose B first).



3. [12 points] List discovery and finish times (beginning at $t = 1$) for each vertex in the table shown below.

Vertex	A	B	C	D	E	F
Discovery	1	2	10	3	4	6
Finish	12	9	11	8	5	7

4. [6 points] Using your answer above, give the specific Topological Ordering that would be produced by the DFS-based algorithm we studied in class. If there is no valid topological sorting for this digraph, write "None".

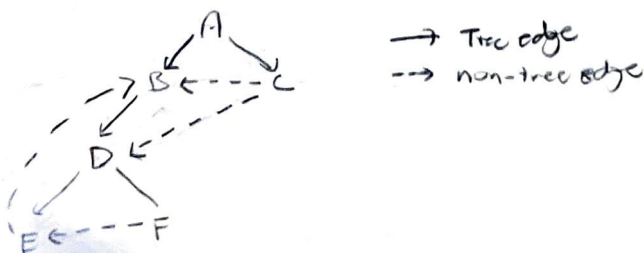
A, C, B, D, F, E

5. [2 points] List all back edges, or write "None" if there are none:

EB

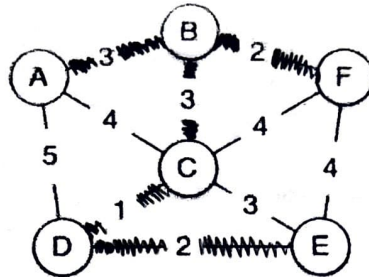
6. [2 points] List all cross or descendent edges, or write "None" if there are none:

CB, CD, FE



Page 4: MST Algorithms

7. [10 points] Run Prim's algorithm on the graph below, **starting the algorithm from vertex A**. List the order in which the edges are added to the MST, referring to the edges by the vertices they connect, e.g. AC.



Your answer (the 5 edges in the order chosen):

AB	BF	BC	CD	DE
----	----	----	----	----

8. [6 points] The run-time for Kruskal's MST algorithm using the disjoint-set data structure depends on *find()* (also known as *find-set()*) and *union()*. Answer the following questions based on the array/list implementation we studied where vertices are identified by integers.

- A. If none of the improvements to the basic *find-union* algorithms are used, what is the Big-Theta time-complexity for *union()* in the worst-case?



$\Theta(n)$

- B. If *find()* uses path-compression and *union()* uses unit-by-rank, what is the run-time for Kruskal's MST algorithm?

$\Theta(E \log V)$

Page 5: BFS and DFS, plus SCCs

9. [18 points] True/False questions on BFS/DFS. For each of the statements below, fill in the circle *completely* if you think the statement is True. If you think it's False, leave the circle unfilled.

- 
- 
- ☒ If graph G is **undirected**, then a call to BFS starting at vertex s will always visit the same number of nodes as a call to DFS-visit starting at the same vertex s .
 - ☐ If graph G is **directed**, then a call to BFS starting at vertex s will always visit the same number of nodes as a call to DFS-visit starting at the same vertex s .
 - ☒ (We can think of BFS and DFS-visit as each creating a search tree and assigning vertices to a given level in their tree.) When called on the same undirected graph G starting at vertex s , then BFS will always put every node v on a level in the BFS tree that is the same level that DFS puts v in the DFS tree **or** at a different level that is **closer** to s .
 - ☐ The reason that BFS uses a queue to store vertices is so that it can efficiently access the current vertex's neighbors by removing them from the front of the queue.
 - ☒ When BFS executes, if the queue is ever size $V - 1$, then the height of the BFS tree must be 1.
 - ☒ When running DFS-visit() on a **connected directed** graph G , if you find an edge (v, w) when visiting v that connects to a vertex w that's colored *gray*, this indicates a cycle in G .

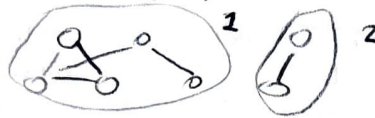
10. [6 points] Questions about the Strongly Connected Components algorithm we studied. For each of the statements below, fill in the circle *completely* if you think the statement is True. If you think it's False, leave the circle unfilled.

- ☒ After the algorithm we studied finds all strongly connected components in a digraph G , it is possible that two edges (a, x) and (y, b) exist in G such a and b are in one strongly connected component and x and y are in a different strongly connected component.
- ☐ The Strongly Connected Components algorithm we studied uses a modified version of DFS-Sweep that calls DFS-visit on the vertices in the transpose of G in the order of the **increasing finish times** found running DFS-Sweep on G .

Page 6: SCCs, Kruskal's, Prim's, and Dijkstra's Algorithms

11. [6 points] The approach we studied to find the connected components of an **undirected** graph used DFS-sweep to repeatedly call DFS-visit on an un-visited node, and labeled all the vertices visited by each DFS-visit with the same component number. In one or two sentences at most, explain why this strategy does not work to find the SCCs for a **directed** graph.

In an undirected graph, any node that can be visited from the node that DFS-visit starts on is part of the same SCC, since direction is not considered.



Once we consider direction, we cannot do this anymore, because a reachable node is not necessarily in the SCC, which is why we must run DFS-sweep on the transpose



12. [15 points] True/False questions. For each of the statements below, fill in the circle *completely* if you think the statement is True. If you think it's False, leave the circle unfilled.

- ☐ It is possible that a connected graph G will force Kruskal's MST algorithm to consider every edge in E in order to find the MST for G .
- ☒ Using an *indirect heap* makes *decreaseKey()* on a min-heap faster, but *insert()* becomes asymptotically slower because the indices in the indirect heap must be updated while percolating a value up towards the root of the heap.
- ☒ When we use *indirect heaps* to make *decreaseKey()* on a min-heap faster, extra space is required and the size of that space is the same Big-Theta class as the heap's size.
- ☒ The primary difference between Prim's MST algorithm and Dijkstra's Shortest Path Algorithm is that they store different values about vertices in the priority queue they use.
- ☐ In our proof showing the correctness of *Dijkstra's Shortest Path algorithm*, when proving the inductive step we showed it was possible for an optimal solution to include a different edge than what Dijkstra's algorithm chose, but only if there was more than one path with the same minimum cost to the vertex.