**CS 590 Final Exam**

**Fall 2021**

This exam has a total of 9 problems from topics covered throughout the semester. This test must be taken by an individual and must not be shared or discussed with anyone. Students are not allowed to use any resources such as class notes, textbooks, or internets. The internet is only allowed when you are attaching any paper work to the file or submitting the exam.

The test is designed to answer in this file. You must answer in this document. For any needs, you may work on a separate paper and insert below appropriate questions as images. **No additional files than this test will not be considered for grading.**

For a full credit consideration, you must show all work and provide explanations. Partial credits are considered on the quality of answers. Please read the questions carefully.

The exam starts at 6:30 PM on Wednesday the 8th of December 2021 and must be submitted by 9:00 PM on the same day. Submit the test as a single file as ***LASTFIRSTname\_FinalExam\_F2021\_A.docx***. 5 points late submission penalty will be applied for every 5 minutes late submission after 9 pm. **Neither late submission after 9:15 pm nor resubmission with corrections will not be considered.**

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | /5 | 6 | /10 |
| 2 | /5 | 7 | /15 |
| 3 | /15 | 8 | /15 |
| 4 | /10 | 9 | /15 |
| 5 | /10 | Total | /100 |

Statement of Academic Honesty:

* I will not use any non-instructor approved electronic devices and sources to assist me on an exam.
* I have not received, I have not given, nor will I give or receive, any assistance to another student taking this exam, including discussing the exam with students in this course.
* I understand that acts of academic dishonesty may be penalized to the full extent allowed by the University Code, including receiving a failing grade for the course. I recognize that I am responsible for understanding the provisions of University Conduct Code as they relate to this academic exercise.

Name: Nicholai L’Esperance

Date: 12/8/2012

Initial: NRL

In the following multiple-choice format questions, you can highlight the answer.

1. What is the running time of insertion sort when here n is the number of the items to be sorted? (5 pts)
2. Let an undirected graph with positive edge weights. Dijkstra’s single-source shortest path algorithm can be implemented using the binary heap data structure with time complexity: (5 pts)
3. Give asymptotic upper and lower bounds for in each of the following recurrences. Assume that is constant for sufficiently small Make your bounds as tight as possible, and justify your answers. you can either type answers using Microsoft Equation (Insert>Equation, a short key control and =) or insert the photo of work. (15 pts)
   1. (6 pts)

Using masters theorem, a=8, b=2, k=3.5, p=0.

log28 = 3 < k 🡪 MT3 🡪 ϴ( )

f(n) dominates the subproblems.

* 1. (9 pts)

Using master’s theorem: a=3, b=3, k=1, p=-1 (

log33 = 1 = k 🡪 MT2 🡪 p=-1 🡪 case b 🡪 ϴ(

f(n) grows at a similar rate as the subproblems (neither dominates the other)

1. We can sort a given set of *n* numbers by first building a binary search tree containing these numbers (using *TREE\_INSERT* repeatedly to insert the numbers one by one) and then printing the numbers by an in-order tree walk. What are the worst-case and best-case running times for this sorting algorithm? (10 pts)

Best case is: nlgn

Worst case is: n2

1. How can we determine if a directed graph contains a cycle? What is the running time of the algorithm that can be used to detect cycles? (10 pts)
   1. In a given directed graph, determine how many cycles the graph contains and provide the paths. (5 pts)

Cycles: In a directed graph, there is not a minimum number of vertices (undirected minimum=3). Requirement: only first and last vertices may be equal.  
I spot five cycles (that can be represented by different permutations of the following):

A -> A

A -> B -> C -> A

A -> B -> D -> A

B -> C -> B

B -> D -> B

* 1. Explain how we can detect the cycle if we use Depth First Search. Do not provide the pseudo code but the steps. Make a complex analysis. (5 pts)

**Step 1)** run the DFS on the graph, recording start and finish times.

**Step 2a)** compute the graph Transpose; that is, reverse the directions of all the edges

**Step 2b)** run the DFS on the Transpose of the graph. However, consider nodes in reverse order of the finishing times in step 1.

**Analyze Results)** Using the results from the DFS on the Transpose, we can determine all of the strongly connected components. The existence of SCCs (that is, SCCs with >1 vertex) indicates that there is a cycle, as the layman’s definition of an SCC is that there is a path between any node within an SCC to another node in the SCC and back. There are two points I want to elaborate on:

1) How do we determine if there are cycles? We know there are cycles when the second DFS on the transpose graph results in \*any\* tree edges. Any single tree edge in the second DFS run indicates there is an SCC with more than one vertex, which means that there \*must\* be a cycle. In this algorithm, every vertex connected by a tree edge in the second DFS run is part of the same SCC. An equivalent method, rather than identifying edges types as tree edges, is to look at discovery->completion times. If there are any overlapping vertex time intervals, it means there is a tree edge, therefore an SCC with >1 vertex, and therefore a cycle. That is:

u.d < v.d < v.f < u.f

2) How does this work? Intuitively, the first DFS run gives us information about all of the vertices that are reachable from a given “start” vertex. By reversing the edges and running the DFS in the other direction, we can identify which vertices are connected back to the start vertex in the other direction. It is important to note that we consider vertexes in reverse order of the finish times of step 1, however. By changing the order of vertexes we consider, we are essentially forcing the second run of the DFS to discover “back” edges on the first run, which is what gives us our cycle!

1. Given an unweighted directed graph , describe an algorithm to find if there is a path between two vertices . What is the running time of this algorithm? (10 pts)
   1. Which algorithm can be used to solve the problem? Justify your answer. (4 pts)

A breadth-first search can be used to solve this problem. In this particular case, I choose a BFS over the DFS, as I don’t care about start/finish times. Additionally, the BFS may not visit all the nodes in the graph, so it may be (marginally) quicker than DFS. BFS also uses a queue rather than recursion.

* 1. Describe the procedure of algorithm and provide its running time. (6 pts)

BFS requires a start vertex, which we have in this problem statement (). The method for the BFS is a running FIFO queue, that is filled with vertices as they are discovered. Visited vertices are kept track of, so they are not visited multiple times. Starting with the start vertex:

1. Add all adjacent vertices of into the FIFO queue.
2. Initialize a set of visited vertices. At the start, it should only have .
3. While the FIFO queue is not empty:
   1. get the first vertex from the queue
   2. add this vertex to the set of visited vertices
   3. for every adjacent vertex that has not already been visited, add them to the FIFO queue.
   4. \*if we cared about distance, increment a counter; every item added to the queue is one greater than the vertex that added it. The starting vertex is 0.

For this particular problem, we can modify the BFS to end the while loop early if the finish node ( is discovered, saving a marginal amount of time. If there is a path between , then will end up in our queue/visited. The run time of this algorithm is O(V+E).

1. Create a Red-Black Tree by inserting following sequence number (8, 29, 30, 7, 5, 16, 13). Write the steps and draw the final tree. For the node indication, have notations as b and r for black and red, respectively. Alternatively, you can color the letter in red or black. For rotations, use L for left and R for right. In the table blow, the problem has the first 5th steps for an example. (15 pts)
   1. Fill out the table. Or if you prefer to show steps as you draw the RBT, you are welcome to do so. You can insert more rows as needed. (10 pts)

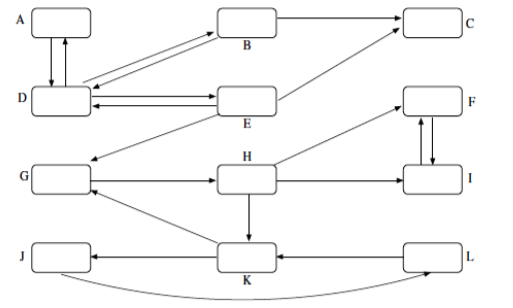
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Step | Insert Number | Location | Rotation (L or R) | Color (R or B) |
| 1 | 8 |  |  | B |
| 2 | 29 | Right below 8 |  | R |
| 3 | 30 | Right below 29 |  | R |
| 4 |  |  | R w.r.t. 29 |  |
| 5 |  |  |  | 8 to R, 29 to B |
| 6 | 7 | Left below 8 |  | R |
| 7 |  |  | R w.r.t 8 |  |
| 8 |  |  |  | 8 to B, 7 to B |
| 9 | 5 | Left below 7 |  | R |
| 10 | 16 | Left below 29 |  | R |
| 11 | 13 | Left below 16 |  | R |
| 12 |  |  |  | 16 to B, 30 to B, 29 to R |

* 1. The RBT up to step 5 is provided. Finish the RBT by editing the diagram below or simply attach the final tree after drawing it. (5 pts)

A picture containing text, clock

Description automatically generated

1. You are given the following graph :



Perform DFS on the given graph. You will visit the nodes in alphabetically increasing order starting with K. Give the discovery *u.d* and finishing times *u.f* for every vertex *u* in the graph. Classify all edges (tree, back, forward, and cross edges). (15 pts)

I assume by increasing, K will discover J before G! This assumption means H also discovers I instead of F. When I need to start from a new vertex after finishing K, I will start with the next largest alphabetical letter, E. I apologize for the confusion, but alphabetically increasing order to me is vague; do I follow paths always increasing alphabetically from the current vertex, or the lowest alphabetical vertex adjacent to the current vertex? I do not have time to email to clarify, so I am explaining my approach.

|  |  |  |
| --- | --- | --- |
| **Node** | **Discovery** | **Finish** |
| A | 21 | 22 |
| B | 17 | 20 |
| C | 18 | 19 |
| D | 16 | 23 |
| E | 15 | 24 |
| F | 9 | 10 |
| G | 6 | 13 |
| H | 7 | 12 |
| I | 8 | 11 |
| J | 2 | 5 |
| K | 1 | 14 |
| L | 3 | 4 |

|  |  |  |
| --- | --- | --- |
| **i** | **j** | **Edge Type** |
| K | J | Tree |
| K | G | Tree |
| J | L | Tree |
| L | K | Back |
| G | H | Tree |
| H | K | Back |
| H | I | Tree |
| H | F | Forward |
| I | F | Tree |
| F | I | Back |
| E | G | Cross |
| E | C | Forward |
| E | D | Tree |
| D | E | Back |
| D | A | Tree |
| D | B | Tree |
| A | D | Back |
| B | D | Back |
| B | C | Tree |

I also diagrammed my trace. Here, green edges represent “tree” edges. All other edges are labeled in black as either “back”, “forward”, or “cross” edges.

Diagram

Description automatically generated

1. Sort the following array using merge sort: [17,42,4,12,25,48,7]. Show the instance of the array after the completion of each recursive call to the merge method. What is the running time of merge sort? (15pts)

Merge sort works by recursively breaking arrays apart into smaller pieces, and then interleaving them back together in sorted order, one step at a time. I will use the Word highlighter to show how the arrays are broken apart.

Initial) [4, 7, 12, 17, 25, 42, 48]

* + 1. [17, 42, 4, 12] | [25, 48, 7]
    2. [17, 42] | [4, 12] | [25, 48] | [7]
    3. [17] | [42] | [4] | [12] | [25] | [48] | [7]
    4. [17, 42] | [4, 12] | [25, 48] | [7]
    5. [4, 12, 17, 42] | [7, 25, 48]
    6. [4, 7, 12, 17, 25, 42, 48]

Here, the first three steps are all Divide steps, and the final three steps are Conquer steps.  
  
The run time of merge sort is ϴ(