Data Structures and Algorithms II
Fall 2019
Final Exam

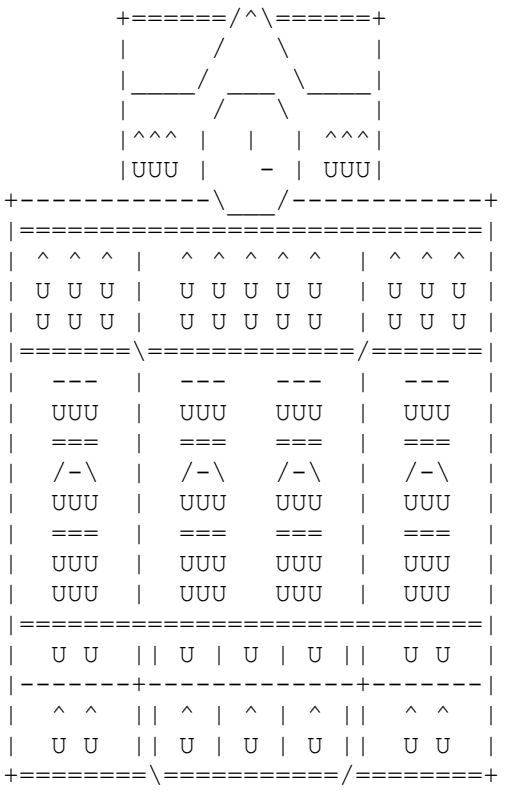
Name:

Email:

Write all of your answers directly within this test booklet. Use the backs of pages if necessary. This is an open-book, open-notes test. You should not use any electronic devices. You may use whatever scrap paper you like, but do not hand it in.

Please also write your name on the top of each page.

(1) ASCII art refers to the drawing of pictures composed of ASCII characters. Despite having no praiseworthy artistic abilities according to any reasonable standard, I went all out for this test - the last time was four years ago - and attempted to create my own ASCII art. (For those of you participating in the Cooper Union book club this year, I would say this is the opposite of sfumato.) Here is my depiction of the Cooper Union Foundation Building:



```
        +======/^\======+
        |      /   \      |
        |_____/  ___ \_____|
        |      /   \      |
        |^^^ |  |  | ^^^|
        |UUU |   - | UUU|
+------------\___/------------+
|============================|
| ^ ^ ^ |  ^ ^ ^ ^ ^  | ^ ^ ^ |
| U U U |  U U U U U  | U U U |
| U U U |  U U U U U  | U U U |
|=======\============/=======|
|  ---  |  ---   ---  |  ---  |
|  UUU  |  UUU   UUU  |  UUU  |
|  ===  |  ===   ===  |  ===  |
|  /-\  |  /-\   /-\  |  /-\  |
|  UUU  |  UUU   UUU  |  UUU  |
|  ===  |  ===   ===  |  ===  |
|  UUU  |  UUU   UUU  |  UUU  |
|  UUU  |  UUU   UUU  |  UUU  |
|============================|
|  U U  || U | U | U ||  U U  |
|-------+------------+-------|
|  ^ ^  || ^ | ^ | ^ ||  ^ ^  |
|  U U  || U | U | U ||  U U  |
+========\==========/========+
```

I saved this image as a text file called "Foundation.txt" and wrote a simple C++ program to perform a statistical analysis. It turns out the file contains a total of 790 characters, including eleven distinct characters (I am not including EOF, and you can ignore EOF for the rest of the question). The eleven distinct characters, in increasing ASCII order, are: newline, space, '+', '-', '/', '=', 'U', '\', '^', '_', and '|'. The frequencies of each of these characters are as follows:

Character	Count
newline	26
space	298
+	8
-	68
/	11
=	148
U	90
\	11
^	25
_	14
|	91

Suppose you want to encode Foundation.txt using Huffman coding. Assume that you will provide the mapping necessary to decode the compressed file as a separate file; you do not have to consider that for the various parts of this question.

(a) Draw a picture of the trie that results from the algorithm. (There is more than one possible answer; just show one trie that might result from the algorithm.)

(b) The original file, Foundation.txt, uses ASCII encoding. How many bytes does it occupy?

(c) How many bytes would be necessary if a compressed version of the file is required to use the same number of bits for every character, but as few as possible given this constraint? (In other words, assume for this part that you are required to use the same number of bits for every instance of every character.)

(d) According to your trie from part (a), fill in the chart below. (Do not use any electronic device. If you make simple arithmetic mistakes, you will get partial credit.)

Character	Count	Code	# of bits in code	Total bits per character
newline	26			
space	298			
+	8			
-	68			
/	11			
=	148			
U	90			
\	11			
^	25			
_	14			
|	91			
Total bits per file:				
Total bytes per file (rounded up if needed):				

(2) Consider the following NP-complete problems, both of which were mentioned in class:

(a) *The partition problem*: Given a set of N positive integers, determine if it be split into two independent subsets (i.e., every integer should be in exactly one of the two subsets) such that the sums of the numbers in each of the two subsets are the same.

(b) *The knapsack problem*: Given a set of N items where each $item_i$ has value v_i and weight w_i (where both v_i and w_i are positive integers), and also given an integer weight limit W, and an integer x, determine if it is possible to choose a subset of items subject to the weight constraint (i.e., the total weight is less than or equal to W) with a total value of at least x.

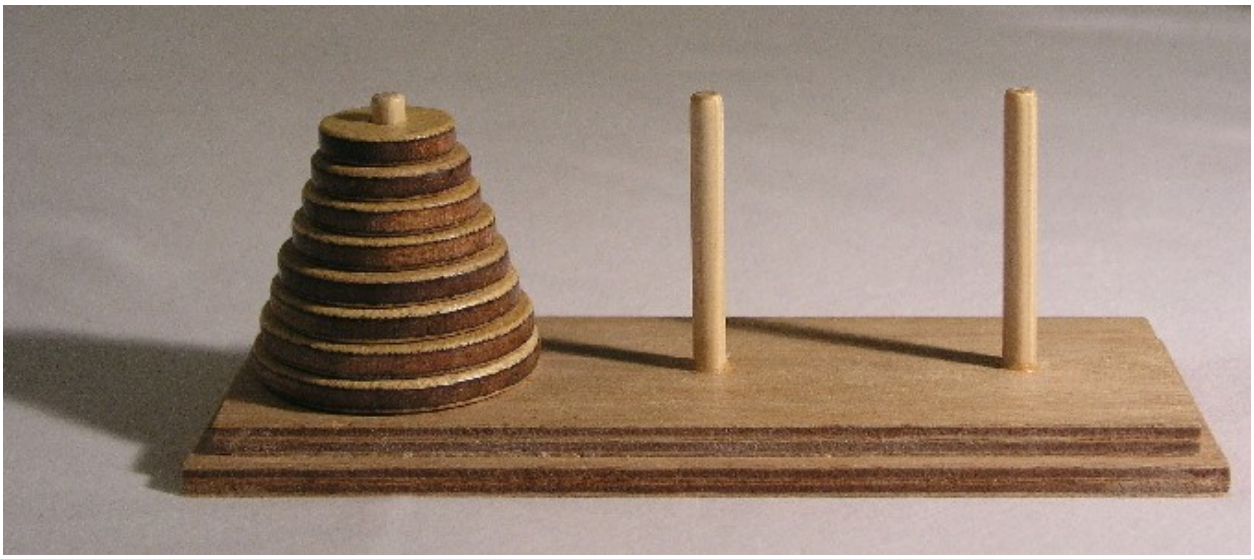
On past tests, I have asked questions involving one or the other of these problems, but never this.

Assume that you know that the partition problem is NP-complete, but you do not yet know that the knapsack problem is NP-complete. *Given that the partition problem is NP-complete, prove that knapsack problem is NP-complete.*

HINT: The proof involves a reasonably simple reduction, but not so simple that I consider it a fair question on the test without a hint. Consider instances of the knapsack problem involving items with weights equal to their values.

If you can't figure either it out, state what you are trying to do for partial credit. Also remember that to prove that a problem is NP-complete, you need to show two things.

(3) Below is a picture on a Tower of Hanoi puzzle, taken from Wikipedia; this particular puzzle has 8 disks. I had one just like it when I was a kid. Recall that when solving a Tower of Hanoi puzzle, only one disk can be moved at a time, each move involves lifting the top disk from one stick and dropping it onto another stick, and a larger disk can never be placed on top of a smaller disk. The goal is to move the pile from the source stick to a destination stick.



(a) We discussed the optimal solution to this puzzle in class during our unit on algorithm strategies. Which algorithm strategy is this an example of, and why is it considered an example of that strategy?

(b) Let T(N) represent the minimum number of steps necessary for solving a Tower of Hanoi puzzle with N disks. Show the recurrence relation describing T(N) for $N > 1$.

(c) How many steps are necessary for solving the puzzle that is shown, containing 8 disks?

(d) If the first move is move 1, the second move is move 2, etc. what is the move number in which the largest disk moves from the source stick to the destination stick.

(4) In class, we learned about the following types of algorithm strategies: greedy algorithms, divide-and-conquer algorithms, dynamic programming, randomized algorithms, exhaustive search, and backtracking algorithms.

You are going to consider applying different types of algorithms to a variation of the traveling salesman problem (TSP). In this variation of TSP, you are given a *complete directed graph* with N vertices, where *all edge costs are positive integers*. (Complete means that directed edges exist in both directions for every pair of edges; the costs, however, vary.) The problem attempts to find the shortest cycle from a specified starting node, s, that visits every other node exactly once and then ends back at s. You may consider algorithms that are not guaranteed to find the optimal solution, but all of these algorithms should find a valid solution, meaning that each will find a cycle that starts and ends at s and visits every other vertex exactly once).

For each of the following strategies, answer three questions:

i. What type of algorithm strategy is it?
ii. Is it guaranteed to find the optimal (i.e., shortest) cycle?
iii. Is the running time of the algorithm polynomial or exponential with respect to N, the number of vertices in the graph?

(a) Loop through every permutation of the N nodes that starts with s and follow the path indicated by that order of vertices, then move back to s at the end. Keep track of the shortest cycle that you find.

(b) Start at s and repeatedly do the following: From the current node, follow the shortest edge that leads to a node which has not yet been visited. After every node has been visited exactly once, return to s.

(c) Start at s and perform a depth-first-search through the graph as follows: Recursively visit every neighbor of s, one at a time. For each node visited, recursively visit each of its neighbors. Assume that the parameters to the recursive function include an indication of the path so far (from s to the current node) and the path cost so far. Whenever you visit a node, if it has previously been visited on the current path, just return. Whenever you reach depth N-1 in the search (meaning that every node has been visited once), consider that path followed by a move back to s. If this path cost is less than that of the best cycle found so far (which might be stored in a global variable), update the best cycle and its cost. Finally, as the search proceeds, whenever the cost of the path so far is great than that of the best cycle found so far, the recursive function returns without searching further along that path.

(d) Start at s and perform a breadth-first-search through the graph as follows: Enqueue/push s onto an initially empty queue. As long as the queue is not empty, repeat the following. Dequeue/pop an item from the queue. Assume that each item indicates the current node, the path so far, and the path cost so far. If there are still unvisited nodes, Enqueue/push every unvisited neighbor (along with the updated path and path cost) onto the queue. If there are no unvisited nodes, consider the path so far followed by a move back to s. If this path cost is less than that of the best cycle found so far, update the best cycle and its cost.

(e) In addition to the graph, also accept a positive integer, k, that is guaranteed to have an upper bound given by a polynomial function applied to N. Then repeat the following k times: Start at s, and randomly choose one neighbor at a time (only choosing between neighbors that have not yet been visited on the current path). After every vertex has been visited once, return to s. For each cycle found, compare the path cost to that of the best cycle found so far, and if the new cycle is better, update the best cycle and its cost.

(5) The traveling salesman problem, mentioned in question 4, is an optimization problem. The related decision problem is called the Hamiltonian cycle problem. One way to express the Hamiltonian cycle problem is as follows: Given a directed graph, G, and a starting vertex, s, determine if there is any cycle that starts at s, visits every other vertex exactly once, and then ends at s. (Note that unlike in question 4, G is not necessarily a complete graph.)

The Hamiltonian cycle problem is known to be NP-complete, but for this problem, *assume that you do not yet know that the Hamiltonian cycle problem is NP-complete*. However, *assume that you do know that the Hamiltonian cycle problem is in NP*, because it is obvious that a potential solution can be verified in polynomial time.

Additionally, *assume that you do know that the partition problem, mentioned in question 2, is NP-complete* (you can refer back to question 2 for a description of the problem).

For each of the following parts of this question, you will further assume that you have made some specific discovery (e.g., that one problem can be reduced to another). *For each part of the question, state an interesting conclusion that you can draw based on the stated discovery*. (For example, if you can conclude that the Hamiltonian cycle is NP-complete, that would be an interesting conclusion, since you are assuming that you didn't know that yet.) If there is no interesting conclusion that you can draw, just write "nothing".

Keep in mind that *you do not know if P = NP* (if you could draw a conclusion about this one way or another, that would most certainly be an interesting conclusion).

Answer each part of this question independently.

To be a bit helpful, here is a summary of the assumptions:

- You don't yet know that the Hamiltonian cycle problem is NP-complete.
- You know that the Hamiltonian cycle problem is in NP.
- You know that the partition problem is NP-complete.
- You don't know (nobody knows) if P = NP.

(a) You discover a way to map any instance of the partition problem (i.e., a specified set of N integers that serves as the input to the partition problem) to a graph, G, and starting vertex, s, such that there is a Hamiltonian cycle in G starting and ending at s if and only if the specified set of N integers can be partitioned into two independent subsets with equal sums.

(b) You discover a way to map any instance of the Hamiltonian cycle problem (i.e., a graph, G, and a starting vertex, s, that serve as the input to the Hamiltonian cycle problem) to a set of N positive integers, such that there is a way to partition the set of N integers into two independent subsets with equal sums if and only if there is a Hamiltonian cycle in G starting and ending at s.

(c) You discover a polynomial-time algorithm (i.e., something that could theoretically run on a regular Turing machine) that solves the partition problem.

(d) You discover a polynomial-time solution that could be theoretically formulated for a nondeterministic Turing machine that solves the partition problem.

(e) You prove that there is no polynomial-time algorithm that solves the Hamiltonian cycle problem.