84/100

Data Structures and Algorithms II
Fall 2020
Final Exam

Name: Allister Liu

Email: liu14 @ cooper. edu.

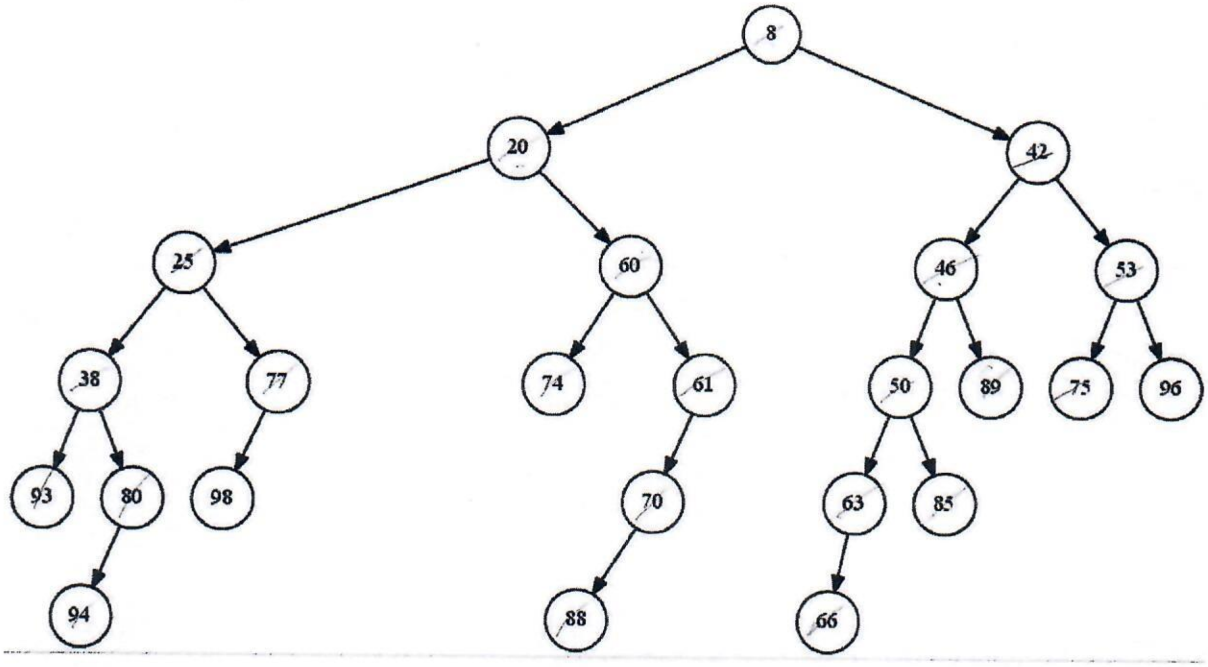
This is a take-home examination. You may print this out and write on the pages; or you may write your answers on separate pages; or you may type your answers within this document. This is an open-book, open-notes, open-Internet test. You may use calculators, computers, or other electronic equipment. The important rule is: You must answer all the questions on your own. Do not discuss the questions with other students, and do not post questions online asking for help.

When you are done, scan or photograph your answers (if you don't type them) and email me the final answers (to carl.sable@cooper.edu). Of course, you may use whatever scrap paper if you like, but do not submit that.

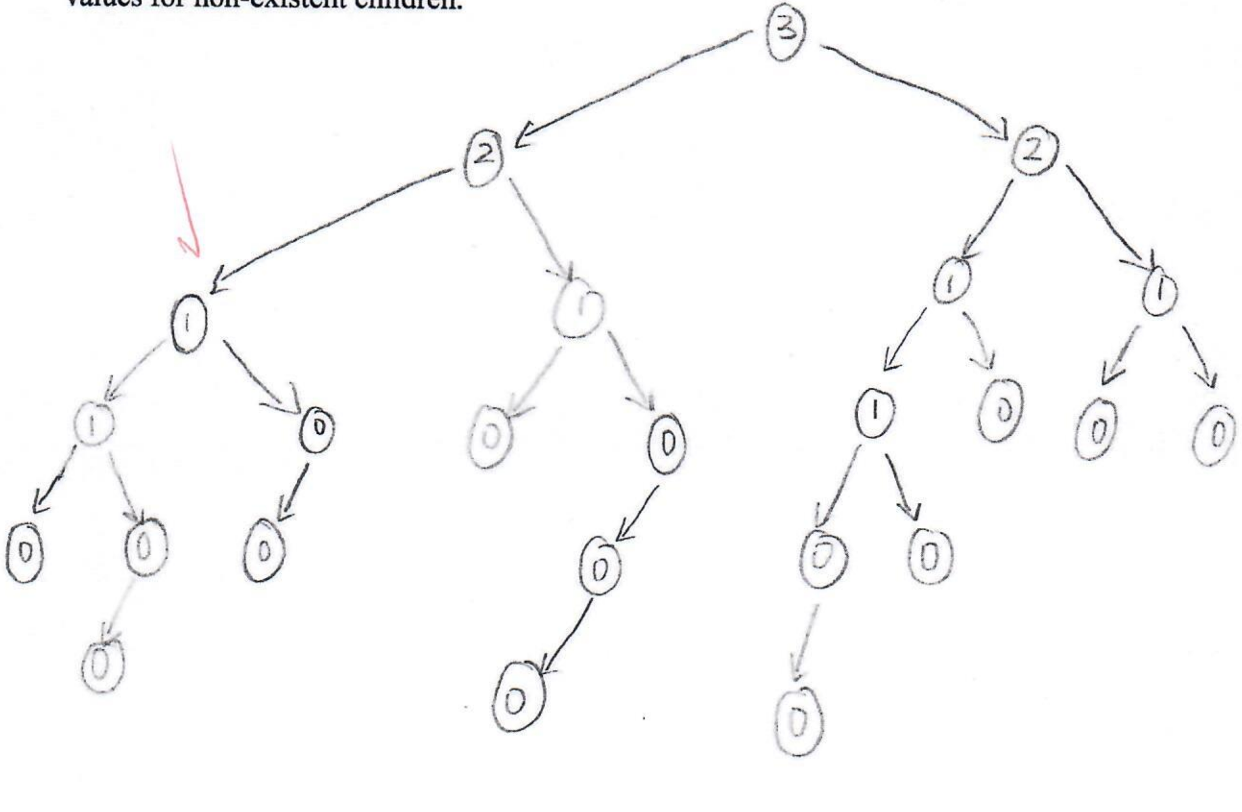
You have 24 hours to complete the test, starting when I send it.

Part I:
Priority Queues and Heaps

(1) Consider the following leftist heap:

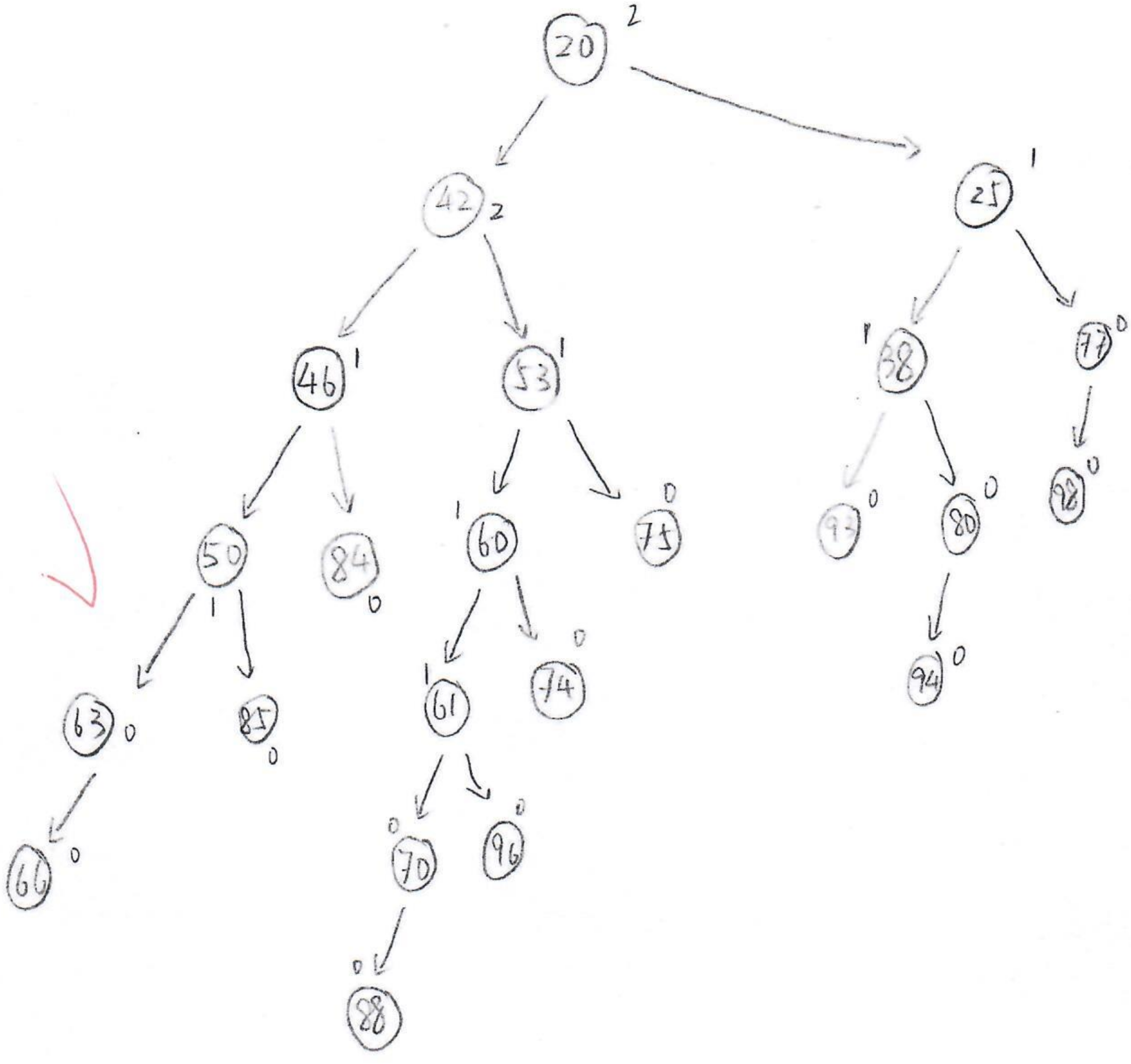


(a) Label each node in the given leftist heap with its null path length. Use the conventions discussed in class (which match those of our textbook). You do not need to indicate the -1 values for non-existent children.



12/12

(b) Show the leftist heap that results when a deleteMin is applied to the given leftist heap.



(c) Label each node in the leftist heap from part (b) with its null path length. Use the conventions discussed in class (which match those of our textbook). You do not need to indicate the -1 values for non-existent children.

12/12

(2) Briefly answer the following questions related to priority queues.

(a) Assume that 20 consecutive deleteMin operations are applied to a binomial queue initially containing 200 nodes (before the deleteMin operations are applied), spread across multiple binomial trees. Is this enough information to know how many binomial trees will remain after the 20 deleteMin operations, and what their sizes (in terms of heights) will be? If your answer is no, explain why not. If you answer is yes, state the number of binomial trees, and specify the height of each remaining binomial tree.

Yes. After 20 deleteMin, 200-20=180 nodes

$\therefore\ 2^7+2^5+2^4+2^2=180$

$\therefore$ 4 binomial trees, each with height 7, 5, 4, 2.

✓

(b) We have learned that Huffman coding can rely on a priority queue at each greedy step to select which two tries to combine. However, priority queues are typically not really important, practically speaking, for this particular algorithm. Why not?

✓ Because it takes linear time to read and find the frequency of the distinct characters in the file. Therefore, it'll probably spend most of the runtime dealing with disk I/O rather than the algorithm.

(c) Why would a binomial queue be more efficient than a binary heap for some applications that require a priority queue? Why would a binary heap be more efficient than a binomial queue for some other applications that require a priority queue?

✓ For merging insertion & deleteMin, binomial queue takes $O(\log N)$ for worst case

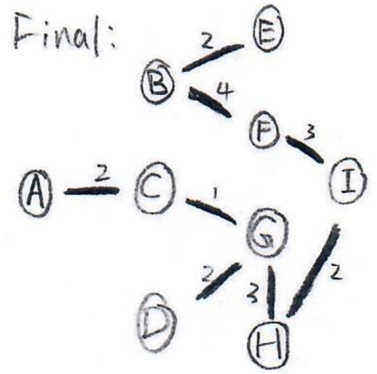
✓ If the data is provided at once, building takes linear $O(N)$ time / for worse case.

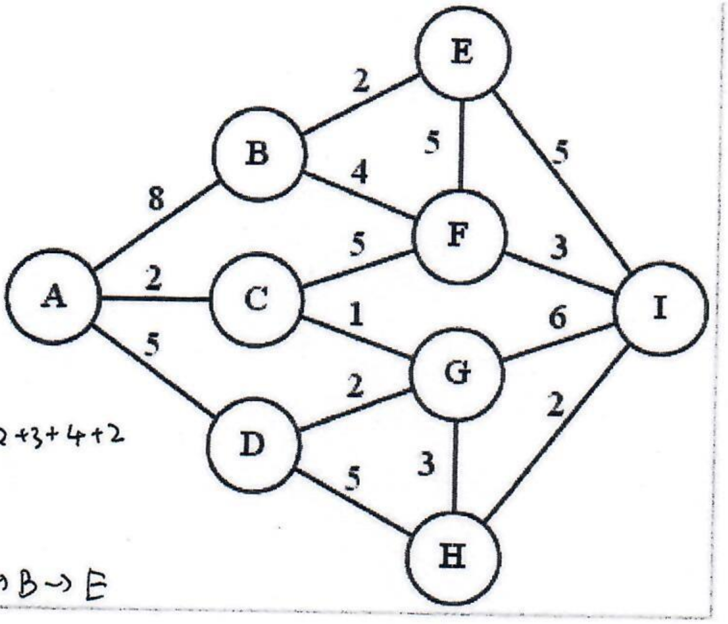
with binary heap

Part II:
Graph Algorithms

(3) Consider the following weighted, undirected graph:

Final:





∴ total cost = 2+1+2+3+2+3+4+2

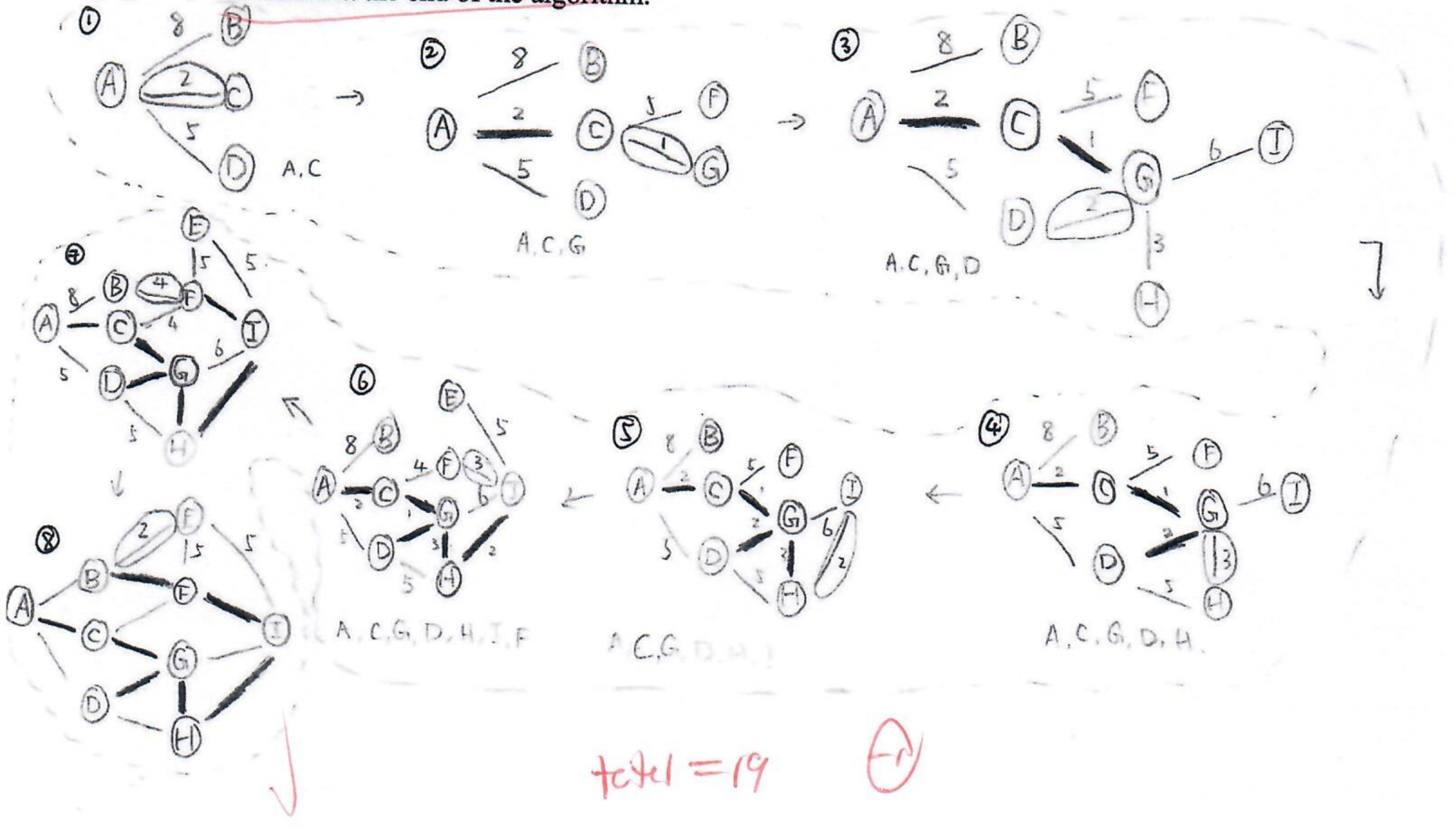
= 19

A → C → G → D

↘ H → I → F → B → E

You are going to use Prim's algorithm to determine a minimum spanning tree of the graph.

(a) Starting at node A, apply Prim's algorithm. Show what the spanning tree looks like *after each step* of the algorithm (the final step should result in a minimum spanning tree for the graph). If there are any steps where you have more than one choice of what to do, choose between the possibilities arbitrarily. Also indicate the total cost of the minimum spanning tree obtained at the end of the algorithm.



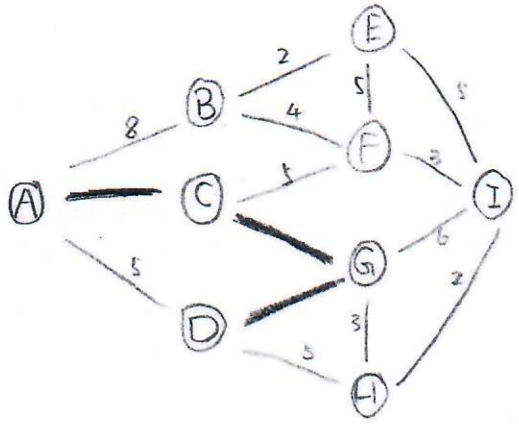
total =19

-1

10/12

(b) In class, we looked at pseudo-code for Prim's algorithm that was extremely similar to Dijkstra's algorithm. Assume this is implemented using a priority queue (e.g., a binary heap). Consider the stage of the algorithm right after the fourth iteration of the main "while" loop (meaning that A and three other vertices have been added to the spanning tree so far). For each node that is still in the priority queue, list the node's name, it's p-value and its d-value.

After 4th iteration.



Node	d-value	p-value
B	8	A
E	∞	null.
F	5	C
H	3	G
I	~~5~~ 6	G

-1

(4) Briefly answer the following questions related to graphs and graph algorithms:

(a) Name one advantage of an adjacency list representation of a graph compared to an adjacency matrix representation of a graph. Also name one advantage of an adjacency matrix representation of a graph over an adjacency list representation of a graph.

An adjacency list takes much less space than an adjacency matrix, especially for representation of a graph. sparse

An adjacency matrix only takes constant time O(1) to determine the existance of edge.

(b) Assume that you are dealing with an instance of a variation of the maximum bipartite matching problem in which one subset of the vertices represents job applicants, and the other subset of vertices represents jobs. Each links represents a possible matching of an applicant with a particular job they are capable of doing. Only one applicant can be hired for each job. However, unlike a typical instance of the maximum bipartite matching problem, each applicant can be hired for up to two different jobs. You want to determine the maximum number of jobs that can be fulfilled. Is this variation of the problem still reducible to the maximum-flow problem? Explain your answer.

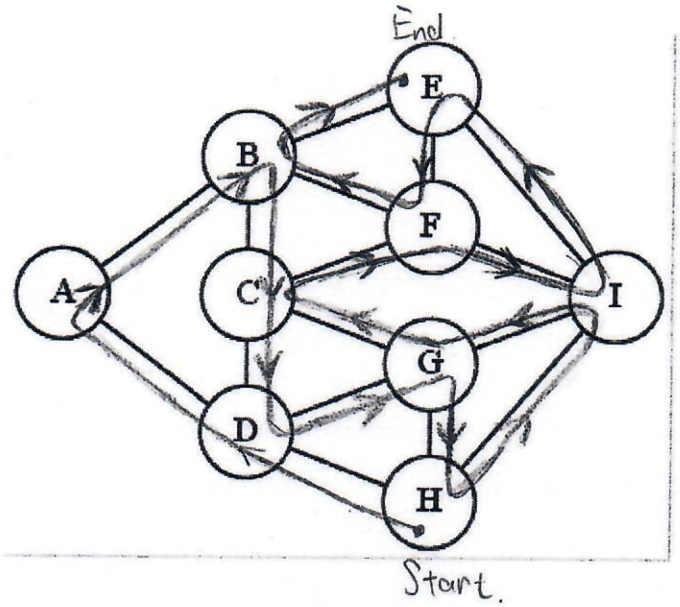
No, it's not reducible. Because there's no augmented path avaliable passing through every vertices.

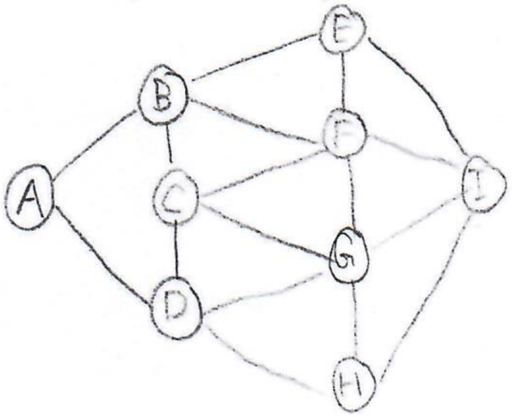
-4

Yes, similar reduction, but edges from source to applicants have capacity 2

8/12

4 (c) Consider the graph below. Is an Euler tour possible? If so, indicate one (e.g., by copying the graph and highlighting the appropriate edges, or by indicating sequence of vertices followed). If not, briefly explain why not. Is an Euler circuit possible? If so, indicate one. If not, briefly explain why not.





Yes. Euler tour is possible:

H → D → A → B → C → D → G

I ← F ← C ← G ← I ← H ↵

↳ E → F → B → E

No. Euler circuit is not possible, ~~we~~ because all the vertices have to have even degrees, but H and ~~F~~ have odd degrees.

E

Part III:
Algorithm Strategies

(5) 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 and attempted to create my own ASCII art. Here is my depiction of the Peter Cooper statue that resides in front of the Cooper Union Foundation Building:

```
=====================================================
\  |||||||||||||||||||||||||||||||||||||||||||||||| /
 \  (  )       (  )      (  )      (  )      (  )  /
  \                                               /
   ===============================================
   =                                             =
   ===============================================
  /==\                                         /==\
  o==o                                         o==o
  |  |                                         |  |
  |  |            %%%%%                        |  |
  |  |           %     %                       |  |
  |  |          %% = = %%                      |  |
  |  |          %%  &  %%                      |  |
  |  |          %%  :  %%                      |  |
  |  |         %%% === %%%                     |  |
  |  |          %%%%%%%%%                 ==   |  |
  |  |            %%%%%                   //   |  |
  |  |       ===        ===\__          %%%%   |  |
  |  |     /   | \     /  |   \_______%%=\| %% |  |
  |  |    |    |  |   |   |=\         %\ ==  % |  |
  |  |    |    |  |   |   |  \________%%\|| %% |  |
  |  |    |    |__|___|   |             %%%%   |  |
  |  |    |           \_  |             //     |  |
  |  |    | ________|||   |            ||      |  |
  |  |   (o)              |  (o)       ||      |  |
  |  |  /||||||=====   ///=====|       ||      |  |
  |  |  /||||||=====    //=====|       //      |  |
  |  |  /|||||||====     /====||      ||       |  |
  |  | //| |____===________=== |      ||       |  |
  |  | //| |     ===      ===| |      ||       |  |
  |  | //| |     ===      ===| |      ||       |  |
  |  | //| |     ===      ===| |      //       |  |
  |  |  /||||||||===||||||===|||     ||        |  |
  |  |  /||||||||===||||||===|||     ||        |  |
  |  |  /||||||||\_/||||||\__\||     ||        |  |
  |  | /================================\      |  |
  |  | ==================================      |  |
  |  | |||                            |||      |  |
  |  | |||                            |||      |  |
  |  | |||                            |||      |  |
  |  | |||       &:|:=   (oo&:=       |||      |  |
  |  | |||                            |||      |  |
  |  | |||                            |||      |  |
  |  | |||                            |||      |  |
  |  | |||                            |||      |  |
  |  | |||                            |||      |  |
  | %%/==================================\%%%%%%% |
 ==================================================
 /                                                 \
/___________________________________________________\
```

I saved this image as a text file called "PC_Statue.txt" and wrote a simple C++ program to perform a statistical analysis. It turns out the file contains a total of 2,606 characters, including 13 distinct characters (I am not including EOF, and you can ignore EOF for the rest of the question). The 13 distinct characters, in increasing ASCII order, are: newline, space, '%', '&', '(', ')', '/', ':', '=', '\', '_', 'o', and '|'. The frequencies of each of these characters are as follows:

Character	Count
newline	51
space	1529
%	66
&	3
(	8
)	7
/	40
:	4
=	382
\	21
_	94
o	8
|	393
TOTAL:	2606

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

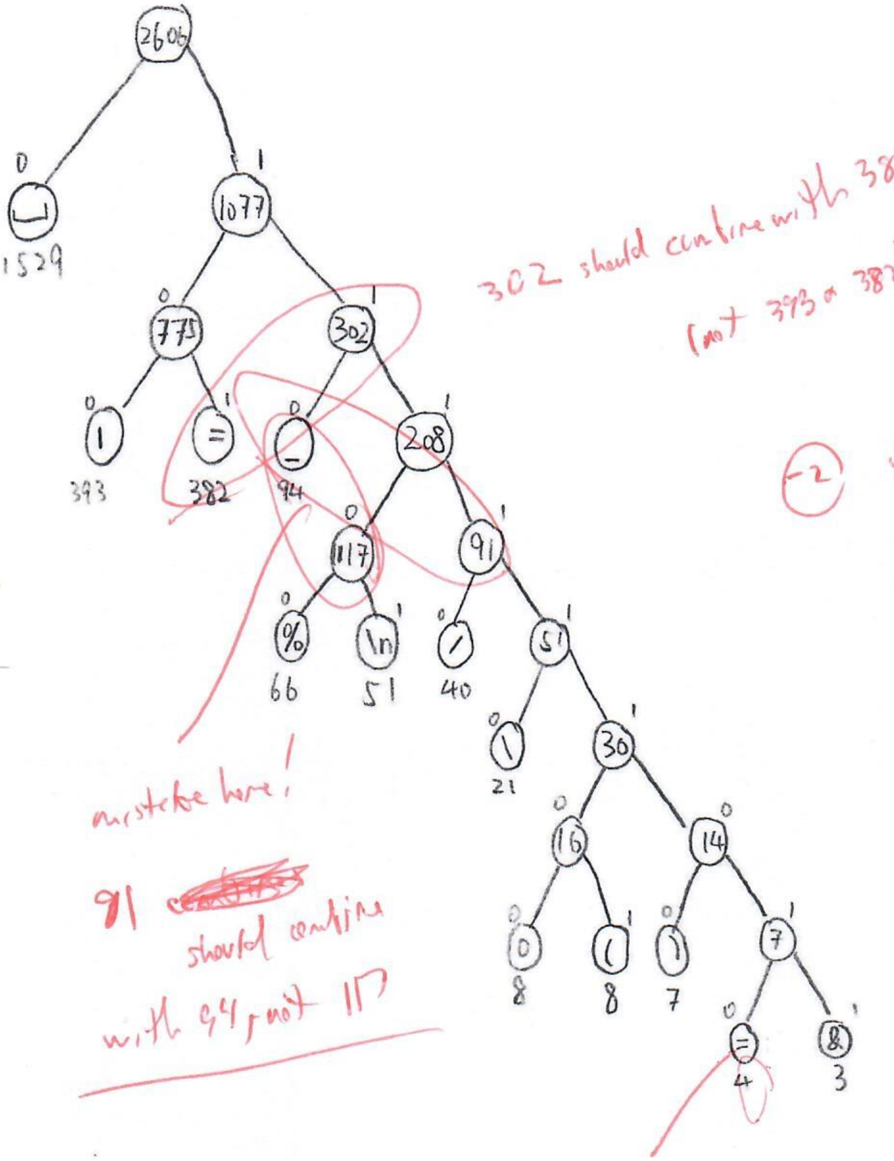
2,606 bytes.

(b) 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 bits as possible given this constraint? (In other words, assume for this part that you are required to use the same number of bits for each instance of every character.)

$$\frac{4 \times 2606}{8} = 1303 \text{ bytes.}$$

Now suppose you want to encode PC_Statue.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.

(c) 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.)



302 should combine with 382
(not 393 or 382)

(-2) two mistakes

mistake here!

91 ~~[illegible]~~ should combine
with 94, not 117

: (looks like =)

10/12

(d) According to your trie from part (c), fill in the chart below.

Character	Count	Code	# of bits in code	Total bits per character
newline	51	11101	5	255
space	1529	0	1	1529
%	66	11100	5	330
&	3	111111111	9	27
(	8	11111101	8	64
)	7	11111110	8	56
/	40	11110	5	200
:	4	111111110	9	36.
=	382	101	3	1146
\	21	111110	6	126
_	94	110	3	282
o	8	11111100	8	64
|	393	100	3	1179
Total bits per file:				5294
Total bytes per file (rounded up if needed):				662

✓ Looks correct based on your non-optimal trie. Optimal trie leads to 5180 bits = 648 bytes

(e) In the Huffman-encoded file, using your encoding that is indicated in part (d), what bit pattern would specifically represent the string "(oo&:="?

11111101 11111100 11111100 111111111 111111110 101.

(o o & : =

✓

8 / 12

(6) Briefly answer the following questions related to algorithm strategies.

(a) Consider a bottom-up dynamic programming algorithm that fills in an N by M matrix, where every cell stores an integer (assume that none of the integers grow particularly large). Further suppose that the value in each cell is typically determined by applying simple arithmetic that involves every cell in the same row to its left, and every cell in the same column above it. What is the time complexity of the algorithm? What is the space complexity of the algorithm?

Time complexity : ~~$O(N \cdot M)$~~ -2 $\Theta(NM[N+M]) = \Theta(N^2M+NM^2)$

Space complexity : $O(N \cdot M)$ ✓

(b) In class, we defined the knapsack problem, which asks you to select a subset of items from N items; each item i has value v_i and weight w_i, and both are positive integers. There is also a limit, W, to the total weight of items you can choose (a weight constraint). The decision version of the knapsack problem asks whether it is possible to choose items with a total value of at least some given value k. The optimization version asks you to maximize the value of your items. Consider the following backtracking algorithm to apply to the optimization version of the knapsack problem. Consider every possible first item to add to the backpack one at a time. For each, consider every possible second item (avoiding duplicates, which are not allowed). For each set of two, consider every possible third item; etc. Whenever you cross the weight limit, W, you backtrack to the last position where you had other choices and them move forward again. Whenever you discover a new set of items that fits in the backpack, with a total value greater than you have seen before, you remember it. Is this approach guaranteed to lead to the optimal solution? Would the worst-case running time be exponential or polynomial?

Yes. Exponential ✓

(c) Now consider the following greedy approach to solve the knapsack problem. First, add the largest item that fits in the backpack (i.e., that has a weight less than or equal to W). Then, add the largest remaining item that fits in the backpack (so the total weight is still less than or equal to W). When no item fits in the backpack, you stop. Is this approach guaranteed to lead to the optimal solution? Would the worst-case running time be exponential or polynomial?

No. ✓ ~~Exponential~~

polynomial

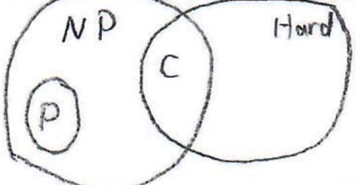
-2

Part IV: Theoretical Computer Science

8|12

(7) This problem will concern four complexity classes that have been discussed in class; specifically, the class P, the class NP, the class NP-complete, and the class NP-hard. Suppose that the following information is known about problems P_1, P_2, P_3, and P_4:

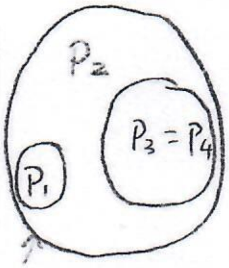
- P_1 is in P
- P_2 is in NP
- P_3 is in NP-complete
- P_4 is in NP-hard



I have used this setup before, but I believe the questions asked below are new. For each of the following questions, *assume that P ≠ NP*, unless something stated in the question asks you to consider the opposite possibility.

(a) Fill in each slot the of the following table, in which rows represent the four problems above, and columns represent the four complexity classes. Write YES if the problem is definitely in the complexity class; write NO if the problem is definitely not in the complexity class; and write MAYBE if the problem may or may not be in the complexity class. Remember that you should be *assuming that P ≠ NP*, although we do not know this for certain. I already filled in four YES values, matching the information given above.

	P	NP	NP-complete	NP-hard
P_1	YES	YES	NO	NO
P_2	MAYBE	YES	MAYBE	MAYBE
P_3	NO	YES	YES	YES.
P_4	NO	MAYBE	MAYBE	YES



(b) If you found a polynomial-time reduction from P_4 to P_2, would that change the table from part (a)? If so, how?

Yes, it would change to

(P_4 to NO YES YES YES)

	P	NP	C	H
P_1	Y	Y	N	N
P_2	N ~~M~~	Y	Y ~~M~~	~~M~~ Y
P_3	N	Y	Y	Y
P_4	N	Y	Y	Y

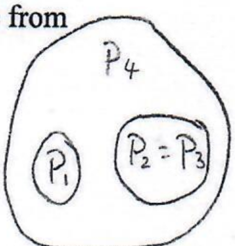
-1

(c) If you found a polynomial-time reduction from P_2 to P_4, would that change the table from part (a)? If so, how? Yes, it would change the table.

	P	NP	C	H
P_1	Y	~~N~~	~~N~~	Y
P_2	N	~~Y~~	~~Y~~	Y
P_3	N	~~Y~~	~~Y~~	Y
P_4	M	~~M~~	~~M~~	M

-2 No, it would not,

All NP problems can reduce to any NP-hard problem



(d) Name every possible polynomial-time reduction, involving two of the four problems from P_1, P_2, P_3, and P_4, that would lead you to conclude that P = NP.

~~$P_2 \rightarrow P_1$~~ (-1, not this one)

P_2 can already be in P

$P_3 \rightarrow P_1$

$P_4 \rightarrow P_1$

12/12

(8) Briefly answer the following questions related to theoretical computer science.

3 (a) Gödel proved that for any formal mathematical system that is expressive enough to state all desired facts about natural numbers, the system must be either inconsistent or incomplete. What would it mean for such a system to be inconsistent? What would it mean for such a system to be incomplete?

Inconsistant system: A system that contradicts itself. (both a statement and its negation can be derived in the system).

Incomplete system: A system that not every true statement can be expressed by the language can be theoretically proven within the language

12 (b) What is intended to be represented on the input tape of a universal Turing machine?

A description (or encoding) of a Turing machine, along with the input to this Turing machine

28 (c) We learned that Richard Karp published "Reducibility Among Combinatorial Problems" in 1972, proving that 21 problems were NP-complete. Did his proofs resemble the proof used by Stephen Cook about a year earlier, which proved that the Boolean Satisfiability Problem is NP-complete? Explain your answer. (Limit your answer to one or two sentences.)

No, Karp's methodology was nothing like Cook's, Karp started with Cook's result and used a series of reductions to form a hierarchy of problems that are NP-complete.

37 (d) Why might it be useful to apply an ε-approximation algorithm to an optimization problem?

Because for many real-life applications that deal with optimization problems, we don't need an algorithm that is guaranteed to find the best solution. An ε- approximation algorithm is guaranteed to find a solution whose cost is "close" to the best possible solution.

A polynomial-time algorithm