CSE 431/531 (Fall 2021) Final Exam (Version A) Solutions

Student Name : _____ Instructor : Shi Li

Student Username : _____ Student ID : _____

Problem	1	2	3	4	5	6	7	8	Total
Maximum Score	36	6	14	12	12	12	10	10	110
Your Score									

The total score of all problems is 112. If you get more than 100 points, your score will be truncated at 100. So, the final score you can get is at most 100.

Common Notations

- $\mathbb{R}$: set of real numbers.

- $\mathbb{Z}$: set of integers.

- $\mathbb{R}_{\geq 0}, \mathbb{R}_{>0}, \mathbb{Z}_{\geq 0}, \mathbb{Z}_{>0}$: sets of non-negative real numbers, positive real numbers, non-negative integers and positive integers respectively.

- $[n] = \{1, 2, 3, \cdots, n\}$ for a positive integer n.

Remarks for Problems 4 and 6

- It is highly recommended that you follow the framework suggested in the problem description, as it will save you a big amount of time. In particular, we are not asking you to optimize the running time: It suffices that your algorithm runs in polynomial time.

- The purpose of examples for the two problems is to help you understand the problems. There may be multiple optimum solutions for the instances. It is possible and OK that your algorithm finds a different optimum solution. So, do not use the given optimum solution to guide your algorithm design.

Problem 1 (36 Points). Indicate if each of the following statements is true or false. A true/false answer for each statement is sufficient; you do not need to give proofs/counterexamples for your answers.

(1.1) $e^n = O(2^n)$. True or False? False

(1.2) Let $f, g, h : \mathbb{Z}_{\geq 0} \to \mathbb{R}$ be three asymptotically positive functions such that $f(n) = O(h(n))$ and $g(n) = O(h(n))$. Then $10f(n) + g(n) = O(h(n))$. True or False? True

(1.3) If the recurrence of a running time T is $T(n) = 3T(n/2) + O(n)$, then solving the recurrence using the master theorem gives us $T(n) = O(n \log n)$. True or False? False

(1.4) A graph $G = (V, E)$ is bipartite if and only if it does not contain a triangle. (A triangle is a sub-graph with 3 distinct vertices $u, v, w \in V$ and 3 edges $(u, v), (v, w)$ and (u, w).) True or False? False

(1.5) A directed graph $G = (V, E)$ can be topologically sorted if and only if it does not contain a directed cycle. True or False? True

(1.6) Let $G = (V, E)$ be a connected graph with $s \in V$. Let u and v be two vertices in V such that the shortest path from s to u is shorter than that to v (assuming all edges in E have length 1). Then it is guaranteed that u will be visited before v in the Breadth-First Search of G starting from s. True or False? True

(1.7) Consider an instance for the interval scheduling problem. It is safe to include the shortest job in the final schedule. True or False? False

(1.8) Consider the offline caching problem. The First-In-First-Out (FIFO) algorithm will always give the optimum solution. True or False? False

(1.9) Consider the Huffman-Code instance with 5 letters a, b, c, d, e, and their frequencies are respectively $12, 2, 5, 6, 3$. Then the following mapping gives an optimum prefix encoding of the 5 letters : a $\rightarrow$ "0", b $\rightarrow$ "100", c $\rightarrow$ "101", d $\rightarrow$ "110", e $\rightarrow$ "111". True or False? False

(1.10) There exists an $O(n)$-time comparison-based sorting algorithm (n is the size of the array to be sorted). True or False? False

(1.11) Merge-sort can be implemented as an in-place sorting algorithm. True or False? False

(1.12) Since the number of inversions in an array of n integers can be as large as $\binom{n}{2} = \frac{n(n-1)}{2}$, any algorithm that counts the number of inversions must run in time $\Omega(n^2)$ in the worst case. True or False? False

(1.13) Let $G = (V, E)$ be a connected graph with edge weights $w : E \rightarrow \mathbb{R}_{>0}$. Let e^* be the heaviest non-bridge edge in E (assuming e^* exists). Then, there is a minimum spanning tree T of G that does not contain e^*. True or False? True

(1.14) Let $G = (V, E)$ be a connected graph with edge weights $w : E \rightarrow \mathbb{R}_{>0}$. Let T be a minimum spanning tree of G w.r.t the weights w. Then, after we square all the edge weights (that is, change the weight of each $e \in E$ to $(w_e)^2$), T must still be a minimum spanning tree of G. True or False? True

(1.15) Let $G = (V, E)$ be a directed graph with edge weights $w : E \rightarrow \mathbb{R}$ (weights can be positive, 0 or negative). It is guaranteed that G does not contain negative cycles. Then Dijkstra's algorithm can solve the single-source shortest paths problem on G. True or False? False

(1.16) Assume P $\neq$ NP and let $X \in$ NP. Then X does not admit a polynomial time algorithm. True or False? True

(1.17) Assume P $\neq$ NP. Then the circuit-satisfiability problem is not in P. True or False? True

(1.18) It is possible that P $=$ NP but P $\neq$ Co-NP. True or False? False

Problem 2 (4 Points). Using the definition of the O-notation to prove $10(n + 10)^2 = O(n^2)$.

Proof. For every $n \geq 10$, we have $10(n + 10)^2 \leq 10(n + n)^2 = 40n^2$. So, $10(n + 10)^2 = O(n^2)$. $\square$

Problem 3 (14 Points). We are given a directed graph $G = (V, E)$ with $|V| = n$ and $|E| = m$, using the linked-list representation. You need to design an $O(n + m)$-time algorithm to decide between the following three cases:

- there is no topological-ordering for G, in which case your algorithm should output "none",
- there is a unique topological-ordering for G, in which case your algorithm should output "unique", and
- there are at least two different topological orderings for G, in which case your algorithm should output "multiple".
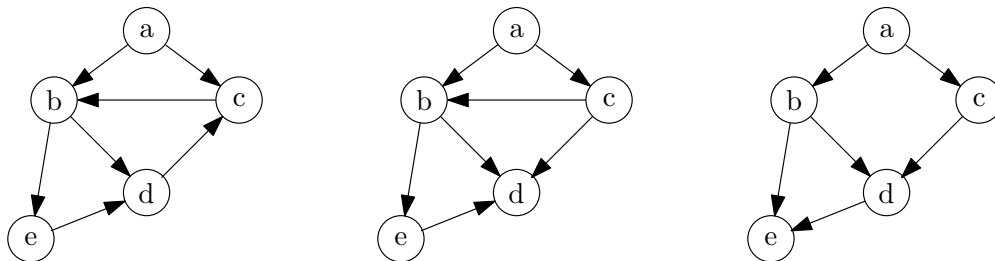

Figure 1: Example input graphs for Problem 3.

For example, consider the three graphs in Figure 1. The outputs for the left-side, middle and right-side graphs are respectively "none", "unique" and "multiple": There is no topological ordering for the left-side graph, there is a unique topological ordering (a, c, b, e, d) for the middle graph, and there are two different topological orderings (a, b, c, d, e) and (a, c, b, d, e) for the right-side graph.

Giving a pseudo-code for your algorithm is sufficient, if the correctness and running time can be easily seen. You can use the topological sort algorithm learned in the class, but you need to describe the main steps of the algorithm, instead of simply saying "using the algorithm learned in the class".

1: initialize the d array over V, so that $d[v]$ is the in-degree of v in G
2: $top \leftarrow 0$, **for** every $v \in V$ **do**: **if** $d[v] = 0$ **then** $top \leftarrow top + 1$, $stack[top] \leftarrow v$
3: $unique \leftarrow true, sorted \leftarrow 0$
4: **while** $sorted < n$ **do**
5:     **if** $top = 0$ **then return** "none"
6:     **if** $top \geq 2$ **then** $unique \leftarrow false$
7:     $u \leftarrow stack[top], top \leftarrow top - 1, sorted \leftarrow sorted + 1$
8:     **for** every outgoing edge (u, v) of u **do**
9:         $d[v] \leftarrow d[v] - 1$
10:         **if** $d[v] = 0$ **then** $top \leftarrow top + 1, stack[top] \leftarrow v$
11: **if** $unique = true$ **then return** "unique" **else return** "multiple"

Problem 4 (12 Points). We are given infinite number of boxes of size $B \in \mathbb{Z}_{>0}$, and n items of sizes $s_1, s_2, \cdots, s_n \in \mathbb{Z}_{>0}$. Assume $s_i \leq B$ for every $i \in [n]$. We need to use the fewest boxes to pack the items, subject to the following two conditions:

- We can pack at most 2 items in any box.

- The total size of the items packed into a box can not exceed B.

For example, suppose $B = 100, n = 7$ and $(s_1, s_2, \cdots, s_7) = (50, 60, 70, 20, 30, 90, 100)$. Then, we can pack the 7 items using 5 boxes, which is the smallest possible, as follows:

boxes	1	2	3	4	5
items put into boxes	1, 5	2, 4	3	6	7

4

Design a greedy algorithm that outputs the minimum number of boxes need to pack all items, by following the steps below:

(4a) Assume $n \geq 1$. Design a simple strategy that chooses 1 or 2 item(s) that can be put into a box.

(4b) Prove that there is an optimum solution, in which the item(s) you choose in (4a) are packed in one box.

(4c) After you irrevocably decided to put the item(s) in a box, what is the residual instance that you need to solve?

(4a) First, we choose the smallest item i. Then we choose the largest item $j \neq i$ such that $s_i + s_j \leq B$. If such a j exists, we put i and j in one box. Otherwise we simply put i into one box.

(4b) If the item j does not exist, then now two items can be put into a box together. So the optimum solution puts the n items in n separate boxes and our decision is clearly safe.

Now assume the item j exists. Let S be any optimum solution. If i and j are put in the same box in solution S, then we are done. Otherwise, we consider two cases:

- If i is put in a box by itself in the solution S, then we move the item j to the box containing i. This gives us a different optimum solution S'. In S', i and j are put together.
- If i and k are put in a box in the solution S for some $k \notin \{i, j\}$. Then by our choice of j we have $s_k \leq s_j$. Then swapping item k and item j in the solution S gives us a new optimum solution S', in which i and j are put in the same box. S' is a valid solution as replacing j with k will not violate the capacity requirement, and putting j with k in the same box does not violate the capacity requirement either.

So, we are done in this case.

(4c) We remove the chosen item(s) from the set and the residual instance is defined by the remaining items. Rename the items if necessary.

Problem 5 (12 Points). Given an array A of n distinct numbers, we say that some index $i \in [n]$ is a local minimum of A, if $A[i] < A[i-1]$ and $A[i] < A[i+1]$ (we assume that $A[0] = A[n+1] = \infty$).

Suppose the array A is already stored in memory. Give an $O(\log n)$-time divide-and-conquer algorithm to find a local minimum of A. A pseudo-code for your algorithm is sufficient if its correctness and running time can be easily seen.

For example, if $n = 10$ and $A = (60, 30, 20, 10, 80, 50, 40, 70, 100, 90)$, then the indices $4, 7$ and 10 are all local minimums. It suffices for your algorithm to return one local optimum. So, your algorithm can return $4, 7$ or 10 for this instance.

1: $l \leftarrow 1, r \leftarrow n$
2: **while** $\ell < r$ **do**
3:     $m \leftarrow \lfloor \frac{\ell+r}{2} \rfloor$
4:     **if** $A[m] < A[m+1]$ **then** $r \leftarrow m$ **else** $\ell \leftarrow m+1$
5: **return** ℓ

Problem 6 (12 Points). Given an array A of n non-negative integers, we need to find the maximum-length increasing subsequence of A. That is, we want to find a maximum-length sequence $(i_1, i_2, \cdots, i_t)$ of integers such that $1 \le i_1 < i_2 < i_3 < \cdots < i_t \le n$ and $A[i_1] < A[i_2] < A[i_3] < \cdots < A[i_t]$. For simplicity, you only need to return the length of the longest increasing subsequence of A, not the actual subsequence.

For example, if $n = 8$ and $A = (50, 60, 30, 80, 40, 20, 60, 70)$, then the longest increasing subsequence of A is $(A[3] = 30, A[5] = 40, A[7] = 60, A[8] = 70)$, which has length 4.

Design a polynomial time dynamic programming algorithm to solve the problem, by following the steps below:

(6a) Define the cells of the dynamic programming.

(6b) Show how to compute the cells using recursion.

(6c) Specify the order in which you compute the cells.

(6a) For every $i \in [n]$, let $f[i]$ be the length of the longest increasing subsequence of A that ends at i.

(6b) For every $i \in [n]$, we have

$$f[i] = \max_{j<i:A[j]<A[i]} f[j] + 1,$$

where we assume if there is no $j < i$ with $A[j] < A[i]$, then $\max_{j<i:A[j]<A[i]} f[j] = 0$ (and thus $f[i] = 1$).

(6c) We compute $f[i]$ for every i from 1 to n in the order. The output of the whole algorithm is $\max_{i \in [n]} f[i]$.

Problem 7 (10 Points). Use Prim's algorithm to solve the following minimum spanning tree (MST) instance.
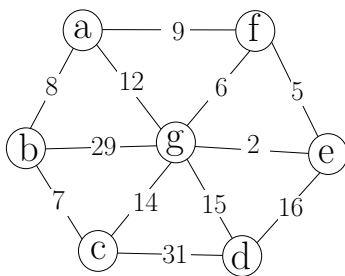
You need to use vertex "a" as the root vertex (i.e, the first vertex added to S). You can use the following table to describe the execution of the algorithm. For vertices in $v \in S$, $d(v)$ and $\pi(v)$ are irrelevant and thus you can let them be "/". For a vertex $v \notin S$ with $d(v) = \infty$, $\pi(v)$ is also "/". The second and third columns correspond to iteration i indicate the edge added to the MST and the vertex added to S in iteration i respectively.

i	edge added to MST in iteration i	vertex added to S in iteration i	b d	π	c d	π	d d	π	e d	π	f d	π	g d	π
0	/	a	8	a	∞	/	∞	/	∞	/	9	a	12	a
1	(a, b)	b	/	/	7	b	∞	/	∞	/	9	a	12	a
2	(b, c)	c	/	/	/	/	31	c	∞	/	9	a	12	a
3	(a, f)	f	/	/	/	/	31	c	5	f	/	/	6	f
4	(e, f)	e	/	/	/	/	16	e	/	/	/	/	2	e
5	(e, g)	g	/	/	/	/	15	g	/	/	/	/	/	/
6	(d, g)	d	/	/	/	/	/	/	/	/	/	/	/	/

Problem 8 (10 Points). For each of the following problems, state (i) whether the problem is known to be in NP, and (ii) whether the problem is known to be in Co-NP. If your answer is yes, you should briefly describe the certifier and the certificate used in the proof.

(8a) Given a graph $G = (V, E)$, two distinct vertices $s, t \in V$, the problem asks if there is a Hamiltonian path from s to t in G.
Problem known to be in NP? Yes   Problem known to be in Co-NP? No

(8b) Given a string A and an integer k, the problem asks whether there exists a palindrome subsequence of A of length at least k.
Problem known to be in NP? Yes   Problem known to be in Co-NP? Yes

(8c) A boolean formula is said to be a contradiction if it evaluates to "false" for every assignment of "true/false" values to variables. For example, $(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$ is a contradiction. (In the formula, "$\vee$", "$\wedge$" and "$\neg$" stand for "or", "and" and "not" respectively.) Given a boolean formula with n variables (n is not fixed), the problem asks whether it is a contradiction.
Problem known to be in NP? No   Problem known to be in Co-NP? Yes

(8a) For NP: The certificate is a permutation of V that forms a Hamiltonian path in G. Given G and the permutation, the certifier checks if the permutation really forms a Hamiltonian cycle in G.

(8b) For NP: This is no need to define a certificate. The certifier finds the longest palindrome subsequence of A using dynamic programming, and checks if its length is at least k.

For Co-NP: This is no need to define a certificate. The certifier finds the longest palindrome subsequence of A using dynamic programming, and checks if its length is less than k.

(8c) For Co-NP: The certificate is an assignment of true/false values to the n variables, for which the formula evaluates to true. Given the formula and the assignment, the certifier checks if the formula evaluates to true for the assignment.