CSE 431/531 (Spring 2022) Makeup Final Exam

Student Name : _____ Instructor : Shi Li

UBITName : _____ Student ID : _____

Problem	1	2	3	4	5	6	7	8	9	Total
Maximum Score	36	5	10	10	10	10	10	10	10	100
Your Score										

The sum of the maximum scores over all problems is 111. If you get more than 100 points, then your final score will be truncated to 100.

Remarks The purpose of examples given in the problem statements is to help you understand the problems. There may be multiple (optimum) solutions for the instances. It is possible and acceptable that your algorithm outputs a different one. So, do not use the given solution to guide your algorithm design.

It is recommended that you write your solutions in the exam paper, in the space after each problem. There should be enough space for the solutions. If not, you can write down your solutions in the blue book and mention this in the exam paper.

Some Standard Notations

- $\mathbb{Z}_{\geq 0}$: set of non-negative integers.
- $\mathbb{R}, \mathbb{R}_{\geq 0}, \mathbb{R}_{>0}$: sets of real numbers, non-negative real numbers, and positive real numbers.
- $[n]$ for an integer $n \geq 0$: the set $\{1, 2, 3, \cdots, n\}$.

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) $\lceil n \log_{1.1}(n^2) + n \rceil = O(n \log_2 n)$. True or False?

(1.2) Let $f, g : \mathbb{Z}_{\geq 0} \to \mathbb{R}$ be two asymptotically positive functions. Then $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$. True or False?

(1.3) If the recurrence of a running time T is $T(n) = 4T(n/2) + O(n^2)$, then solving the recurrence using the master theorem gives $T(n) = O(n^2)$. True or False?

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

(1.5) Consider the graph $G = (V, E)$ with $V = \{a, b, c, d\}$ and $E = (a, b), (a, c), (b, d)$, and the depth-first-search (DFS) algorithm over G with a being the starting vertex. Assume b is the first vertex visited after a. Then c is visited before d in the DFS algorithm. True or False?

(1.6) Consider an instance of the interval scheduling problem. It is safe to schedule the shortest job. True or False?

(1.7) Consider an instance of the Huffman codes problem. It is safe to pair the two least frequent letters and make them brothers in the tree. True or False?

(1.8) Consider the min-heap data structure, it takes $O(1)$-time to insert an element, remove an element, and extract the minimum element from/to the heap. True or False?

(1.9) Any comparison-based sorting-algorithm needs to take $\Omega(n \log n)$-time in the worst case. (n is the size of the array to be sorted). True or False?

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

(1.11) Consider an instance of the weighted interval scheduling problem. It is safe to schedule the job with the smallest length-to-weight ratio. True or False?

(1.12) The running time of the dynamic programming algorithm for the longest common subsequence problem is $O(n^2)$. True or False?

(1.13) Let $G = (V, E)$ be a connected graph with edge weights $w : E \to \mathbb{R}_{>0}$, and assume all edge weights are distinct. Let C be a simple cycle in G, and e^* be the lightest edge on C. Then, there is a minimum spanning tree T of G that contains e^*. True or False?

(1.14) Let $G = (V, E)$ be a directed graph with edge weights $w : E \to \mathbb{R}$ (weights can be positive, 0 or negative). Let $n = |V|, m = |E|$ and $s \in V$. It is guaranteed that G does not contain negative cycles. Then we can use Dijkstra's algorithm to compute the shortest paths from s to all vertices in V. True or False?

(1.15) Let $G = (V, E)$ be a directed graph with edge weights $w : E \to \mathbb{R}$ (weights can be positive, 0 or negative). Let $n = |V|$ and $m = |E|$. It is guaranteed that G does not contain negative cycles. We need to compute the length of the shortest paths between all pairs of vertices. Then, Floyd-Warshall's algorithm can solve the problem and it runs in $O(n^3)$ time. True or False?

(1.16) If $P = NP$, then the Hamiltonian cycle problem can be solved in polynomial time. True or False?

(1.17) Let X be a NP-complete problem. Then $P = NP$ if and only if X has a polynomial time algorithm. True or False?

(1.18) $P \cap NP = \emptyset$, i.e, a problem can no be in both P and NP. True or False?

Problem 2 (5 Points). Use the definition of the O-notation to prove $5\left\lceil n + \sqrt{n+1} \right\rceil = O(n)$.

For every $n \geq 1$, we have $5\left\lceil n + \sqrt{n+1} \right\rceil \leq 5(n + n + 1) \leq 5(n + n + n) = 15n$. Therefore $5\left\lceil n + \sqrt{n+1} \right\rceil = O(n)$.

Problem 3 (10 points). Solve the following optimum binary search tree instance. We have 5 elements e_1, e_2, e_3, e_4 and e_5 with $e_1 < e_2 < e_3 < e_4 < e_5$ and their frequencies are

- $f_1 = 80, f_2 = 20, f_3 = 5, f_4 = 70$ and $f_5 = 15$.

Recall that the goal is to find a binary search tree for the 5 elements so as to minimize $\sum_{i=1}^{5} \text{depth}(e_i) f_i$, where $\text{depth}(e_i)$ is the depth of the element e_i in the tree. You need to

- Fill the empty cells in Table 1.
- Draw the optimum binary search tree and give its cost.

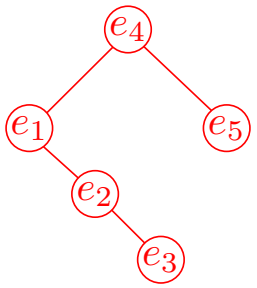
You can write down your steps on the computation of the cells, but they will not be graded.

$opt[i,j]$ i \ j	1	2	3	4	5
1	80	120	135	300	340
2		20	30	125	155
3			5	80	110
4				70	100
5					15

$\pi[i,j]$ i \ j	1	2	3	4	5
1	1	1	1	1	4
2		2	2	4	4
3			3	4	4
4				4	4
5					5

Table 1: *opt* and π tables for the optimum binary search tree instance. For cleanness of the table, we assume $opt[i,j] = 0$ if $j < i$ and there are not shown in the table.

The cost of the optimum binary search tree is 340.
Draw the optimum binary search tree below:



Problem 4 (10 Points). You need to find the minimum spanning tree of the graph G in Figure 1, by filling the table in Figure 2. If an edge is not added to the MST, you do not need to mention that. So in the table, you only specify the edges that are added to the MST, in the order they are added, as well as how adding each edge changes the partition of vertices.

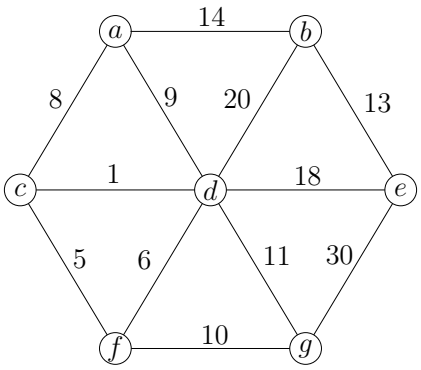


Figure 1: Instance for Kruskal's Algorithm in Problem 4.

i	i-th edge added to MST	partition of vertices after adding i-th edge
0	none	$\{a\},\{b\},\{c\},\{d\},\{e\},\{f\},\{g\}$
1	(c,d)	$\{a\},\{b\},\{c,d\},\{e\},\{f\},\{g\}$
2	(c,f)	$\{a\},\{b\},\{c,d,f\},\{e\},\{g\}$
3	(a,c)	$\{a,c,d,f\},\{b\},\{e\},\{g\}$
4	(f,g)	$\{a,c,d,f,g\},\{b\},\{e\}$
5	(b,e)	$\{a,c,d,f,g\},\{b,e\}$
6	(a,b)	$\{a,b,c,d,e,f,g\}$

The total weight of the MST is 51.

Figure 2: Table for Problem 4.

Problem 5 (10 Points). Recall that a cycle in an (undirected) graph $G = (V, E)$ is a sequence of $t \geq 3$ different vertices $v_1, v_2, \cdots, v_t$ such that $(v_i, v_{i+1}) \in E$ for every $i = 1, 2, \cdots, t-1$ and $(v_t, v_1) \in E$.

Given a graph $G = (V, E)$ with $|V| = n$ and $|E| = m$ (which is not necessarily connected), design an $O(n + m)$-time algorithm to decide if G contains a cycle or not; if it contains a cycle, output one. (You can output arbitrary one if there are different cycles). You can assume the graph is given using the linked-list representation. You need to give the pseudo-code for solving the problem, and briefly mention why the running time of the algorithm is $O(n+m)$. If the correctness of the algorithm is easy to see, there is no need to prove it.

Example. For the graph in Figure 3(a), there are three cycles and you need to output "yes" and *any one of the three cycles*: $(1, 2, 8, 3)$, $(3, 7, 6, 8)$, and $(1, 2, 8, 6, 7, 3)$. Also, the representation of a cycle is not unique. For example, you can also output $(2, 1, 3, 8)$ for the first cycle. For the graph in Figure 3(b), you should output "no".

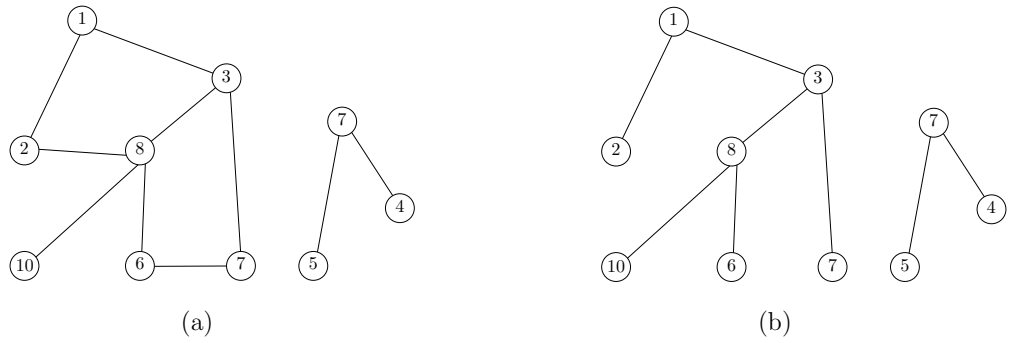


Figure 3: An Example Instance for Problem 5.

Remark. If you simply output "yes/no" without giving the actual cycle (for a yes-instance), you can still get up to 8 points.

We use DFS to detect if a cycle exists. The idea is that if we find one edge (v, u) that is not in the tree, when we run DFS(v), then we know u is an ancestor of v, and the edge (v, u) and the path in the DFS tree from v to u form a cycle.

Algorithm 1 DetectCycle

```
1: for every v ∈ V do visited[v] ← false
2: for every vertex v ∈ V do
3:     if visited[v] = false then DFS(v)
4: print("no")
```

Algorithm 2 DFS(v)

```
1: visited[v] ← true
2: for every neighbor u of v do
3:     if visited[u] = false then
4:         par[u] ← v, DFS(u)
5:     else if u ≠ par[v] then
6:         print("yes"), print(v), w ← v
7:         while w ≠ u do
8:             w ← par[w], print(w)
9:         exit the whole algorithm.
```

The running time of the algorithm is dominated by the DFS algorithm, and thus it is $O(n+m)$.

Problem 6 (10 Points). An independent set of a graph $G = (V, E)$ is a set $U \subseteq V$ of vertices such that there are no edges between any two vertices in U. The maximum independent set problem asks for the independent set of G with the maximum size. The problem is hard on general graphs. Here we want to solve the problem on trees in polynomial time : given a tree $T = (V, E)$, find the maximum independent set of the tree. For example, maximum independent set of the tree in Figure 4 has size 7, denoted by the solid circles.

You can solve the problem using the following steps:

(6a) Design a safe strategy in which you choose some vertex v in the tree, and include v in the independent set.

(6b) Prove that the strategy you designed in step (6a) is safe.

(6c) Suppose you decided to put a vertex v in the independent set. Then, how do you reduce the remaining task into one or many sub-instances of the maximum independent set on trees problem?

If you do not follow the steps, and instead, you give a pseudo-code for the algorithm, then the pseudo-code will cover (6a) and (6c), but not (6b). You still need to prove the correctness by showing your strategy is safe.

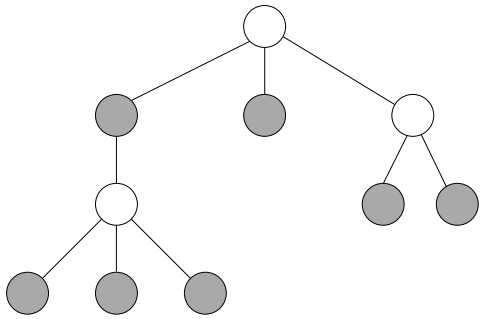


Figure 4: The solid circles denote the maximum independent set of the tree, which has size 7.

(6a) We take any leaf v of the tree and add it to the independent set.

(6b) We prove that it is safe to include v in the independent set. Let S be a maximum independent set. If $v \in S$, then we are done. Otherwise, $v \notin S$ and let u be the unique neighbor of v in the tree. Then $S \setminus \{u\} \cup \{v\}$ must be another maximum independent set. Thus, in any case there is an maximum independent set that contains v.

(6c) After we include v in the independent set, we remove v and its neighbor from the tree. This breaks the tree into many sub-trees, and the remaining goal is to solve the maximum independent set instance in each of the sub-trees.

Problem 7 (10 Points). Suppose you are given n pictures of human faces, numbered from 1 to n. There is a face comparison program called same, that, given two different indices i and j from $1, 2, \cdots, n$, returns whether face i and face j are the same, i.e, are of the same person. A majority face is a face that appears more than $n/2$ times in the n pictures.

The problem is to find a picture with the majority face, or declare there is no majority face, by calling the program same. Your algorithm can only call same $O(n \log n)$ times. You need to argue why your algorithm satisfies the requirement. If the correctness of the algorithm is easy to see, then a pseudo-code is sufficient for the correctness part.

Remark. The algorithm same can only return whether two faces i and j are the same or not. If they are not the same, it can *not* tell you whether "face $i <$ face j" or "face $i >$ face j".

Example. Suppose $n = 5$ and the function calls to same and their returned values are as follows: same$(1, 2)$ = false, same$(1, 3)$ = false, same$(2, 3)$ = true, same$(3, 4)$ = false and same$(3, 5)$ = true. Then your algorithm can return 2 (3 or 5) with confidence, since it knows that faces 2, 3 and 5 are the same.

Hint. Use divide-and-conquer and the following fact: If an element t is the majority in an array A, and we break A into two sub-arrays AL an AR, then at least one of the events happen: (a) t is the majority element in AL, and (b) t is the majority element in AR.

Algorithm 3 check-majority(l, r, t)

```
count ← 0
for i ← l to r do
    if same(i, t) then count ← count + 1
return true if count > (r − l + 1)/2 and false otherwise
```

Algorithm 4 majority(l, r)

```
if l = r then return l
m ← ⌊(l+r)/2⌋
a ← majority(l, m)
b ← majority(m + 1, r)
if a ≠ ⊥ and check-majority(l, r, a) then return a
if b ≠ ⊥ and check-majority(l, r, b) then return b
return ⊥
```

In the main algorithm, we call majority$(1, n)$. It will return the majority face or $\perp$, which means that there is no majority face.

The recurrence for the number of times we call the *same* procedure is $T(n) = 2T(n/2) + O(n)$. Thus the algorithm calls *same* $O(n \log n)$ times.

Problem 8 (10 Points). Consider the following job-selection problem. Suppose you can undertake one job in each of the following n weeks. The set of possible jobs is divided into those that are low-stress and that are high-stress. The basic question, each week, is whether to take on a low-stress job or a high-stress job.

If you select a low-stress job in week i, then you get a revenue of $l_i > 0$ dollars; if you select a high-stress job, you get a revenue of $h_i > 0$ dollars. The catch, however, is that in order for you to take on a high-stresss job in week i, it is required that you do no job (of either type) in week $i-1$. On the other hand, it's OK if you take a low-stress job in week i even if you have done a job (of either type) in week $i-1$.

So, given a sequence of n weeks, a plan is specified by a choice of "low", "high", or "none" for each of the n weeks, with the property that if "high" is chosen for week $i > 1$, then "none" has to be chosen for week $i-1$. (You can choose "high" for week 1.) The value of the plan is determined in the natural way: for each i, you add l_i to the value if you choose "low" in week i, and you add h_i to the value if you choose "high" in week i. (You add 0 if you choose "none".)

For example, suppose $n = 5$ and the h_i and l_i values are given in the following table:

i	1	2	3	4	5
h_i	50	20	20	10	60
l_i	30	15	10	5	50

Then the maximum revenue you can achieve is 135, achieved by the plan "high, low, low, none, high". In this plan, the revenue you get in the each of the 5 days are 50, 15, 10, 0 and 60.

Design an efficient (i.e, polynomial-time) dynamic-programming algorithm to solve the problem. For convenience you only need to output the maximum revenue you can achieve. You can follow the steps below:

(8a) Define the cells of the dynamic programming.

(8b) Show how to compute the cells using recursion.

(8c) Specify the order in which you compute the cells.

(8a) For every $i \in [n]$,

- let $f[i, \text{high}]$ denote the maximum revenue I can obtain in the first i days, with the constraint that in day i, I choose the high-stress job,
- let $f[i, \text{low}]$ denote the maximum revenue I can obtain in the first i days, with the constraint that in day i, I choose the low-stress job, and
- let $f[i, \text{none}]$ denote the maximum revenue I can obtain in the first i days, with the constraint that in day i, I choose no jobs.

The final answer is $\max\{f[n, \text{none}], f[n, \text{high}], f[n, \text{low}]\}$.

(8b)

$$f[i, \text{none}] = \begin{cases} 0 & \text{if } i = 1 \\ \max\{f[i-1, \text{none}], f[i-1, \text{low}], f[i-1, \text{high}]\} & \text{if } i > 1 \end{cases}$$

$$f[i, \text{low}] = f[i, \text{none}] + l_i \qquad \forall i \in [n]$$

$$f[i, \text{high}] = \begin{cases} h_1 & \text{if } i = 1 \\ h_i + f[i-1, \text{none}] & \text{if } i > 1 \end{cases}$$

(8c) For every i from 1 to n, we compute $f[i, \text{none}], f[i, \text{low}], f[i, \text{high}]$ in that order.

Problem 9 (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. For the problem (9c), you need describe the certifier and certificate for the proof of the problem being in NP and/or Co-NP, if you answered yes. (For (9a) and (9b), you only give "yes/no" answers.)

(9a) Given a graph $G = (V, E)$, the problem asks whether G contains a Hamiltonian cycle or not.

Problem known to be in NP? yes Problem known to be in Co-NP? no

(9b) Given a graph $G = (V, E)$ (which might be connected or not connected) and two vertices $s, t \in V$, the problem asks whether there is a path connecting s and t in G or not.

Problem known to be in NP? yes Problem known to be in Co-NP? yes

(9c) A boolean formula is said to be a *contradiction* if it evaluates to 0 for every assignment of 0/1 values to the 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 a fixed constant), the problem asks whether it is a contradiction.

Problem known to be in NP? no Problem known to be in Co-NP? yes

Certifier for NP, if your answer for NP is yes: N/A

Certificate for NP, if your answer for NP is yes: N/A

Certifier for Co-NP, if your answer for Co-NP is yes:
The certifier takes the boolean formula, and an assignment of boolean values to the variables, and check if the formula evaluates to true on this assignment.

Certificate for Co-NP, if your answer for Co-NP is yes:
The certificate is the boolean assignment for which the formula evaluates to true.