National University of Singapore

School of Computing

IT5003 - Data Structures and Algorithms

Final Assessment

(S2 AY2025/26)

Time Allowed: 2 hours

INSTRUCTIONS TO CANDIDATES:

1. Do **NOT** open this assessment paper until you are told to do so.
2. This assessment paper comprises FOURTEEN (14) printed pages, including this page.
3. This is an **Open Book Assessment**.
   You cannot use any electronic device except one calculator.
4. You can use either pen or pencil. Just make sure that you write **legibly**!
5. Important tips: Pace yourself! Do **not** spend too much time on one (hard) question.
   Read all the questions first! Some (sub-)questions might be easier than they appear.
   The first four questions have Generative AI answers that you can use/fix/improve (or rewrite).
6. If you choose to write your own answer and for the last two questions,
   you can use **pseudo-code** in your answer but beware of penalty marks for **ambiguous answer**.
   You can use **standard, non-modified** classic algorithm in your answer by just mentioning its name, e.g. run Inorder Traversal on BST T, DFS on graph G, Dijkstra's on graph G', etc.
7. The total marks is 50. All the best :)

A Application Questions ($5 \times 10 = 50$ marks)

A.1 Find It (10 marks)

Given two positive integer lists $nums1$ (length n, $1 \leq n$) and $nums2$ (length m, $1 \leq m$), both *sorted in non-decreasing order*, return the *minimum integer* common to both lists. All integers are $\leq 2^{31} - 1$. If no common integer exists, return -1.

Example 1: If $nums1 = [1, \underline{2}, 3, \underline{7}]$, $nums2 = [\underline{2}, 4, \underline{7}]$, then the common integers are $\underline{2}$ and $\underline{7}$. We return 2, the minimum of the two.

Example 2: If $nums1 = [1, 3, 5, 7]$, $nums2 = [2, 4, 6, 8]$, then we return -1 because there is no common element.

Solve this problem in Python and analyze its time complexity in terms of n and/or m.

A.2 Special Sublist (10 marks)

You are given a list of integers L containing n positive integers, each at most 10^9, and two positive integers d and m ($1 \leq d \leq m \leq n$). Your task is to return the maximum sum among all sublists of L with length m that contain at least d distinct integers. If no such sublist exists, return 0.

Example 1: Given $L = [4, 1, 7, 7, 7, 3, 6]$, $d = 3$, and $m = 4$, there are two valid sublists: $[4, 1, 7, 7]$ (3 distinct integers, sum 19) and $[7, 7, 3, 6]$ (3 distinct integers, sum 23). The maximum sum is 23.

Example 2: Given the same L and m as in Example 1, but with $d = 2$, the maximum sum is 24 (from the sublist $[7, 7, 7, 3]$).

Example 3: Given the same L and m as in Example 1, but with $d = 4$, the maximum sum is 0 as no such sublist exists.

Solve this problem in Python and analyze its time complexity in terms of n and/or m.

A.3 Special Binary Tree (10 marks)

You are given a binary tree T (not a BST) where each vertex has a specific negative integer value. The root has value -1 (negative one). For any vertex with value x (a negative value), if it has a left/right child, its value is $2x/2x - 1$, respectively. Basically, as seen in Figure 1, the value of any vertex (the number inside the circle) follows the standard 1-based Binary Heap indexing, but using negative values. Note that T is not necessarily a complete binary tree; some vertices can be missing. For instance, in Figure 1, we don't have vertices with values $-8, -13, -14, -16, -17, -19, -21$, etc. (this missing vertices list is not exhaustive).

You are given the root of T. Each vertex has three attributes: val (a negative integer), $left$ (the reference to the left subtree), and $right$ (the reference to the right subtree). T contains n vertices and has a maximum height of 20. You are then given m queries in list Q ($m \leq n$), each consisting of a negative integer not smaller than $-2^{21} + 1$, not necessarily distinct. For each query, determine if a vertex with that value exists in T. Report the total count of successful queries.

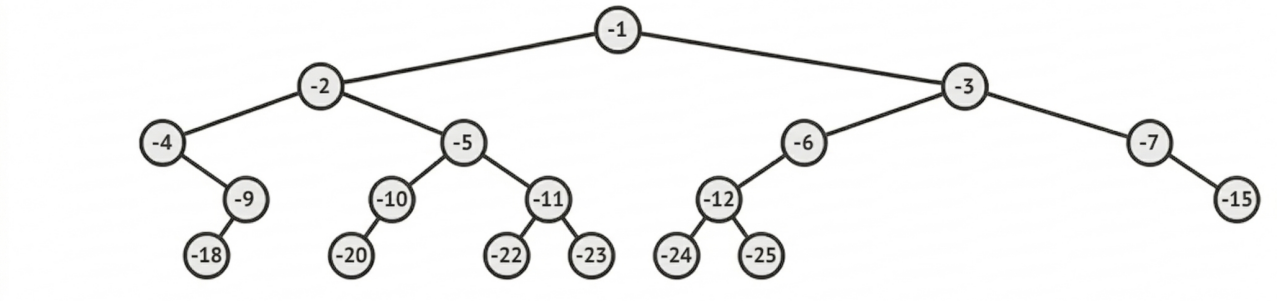


Figure 1: An Example Binary Tree T

For example, if T with $n = 18$ vertices is shown in Figure 1, and the $m = 8$ queries are $Q = [-7, -13, -21, -8, -2, -9, -19, -2]$, then the answer is 4, as only the four values $[-7, -2, -9, -2]$ are found in Figure 1.

Solve this problem in Python and analyze its time complexity in terms of n and/or m.

A.4 Life is not Simple (10 marks)

You are given an undirected, weighted simple graph with n $(2 \leq n)$ vertices labeled 0 to $n-1$ as a list of m $(0 \leq m \leq n \cdot (n-1)/2)$ undirected edges in a list EL, where $EL[i] = [a_i, b_i, w_i]$ represents a bidirectional edge between vertex a_i and b_i with weight w_i $(0 \leq a_i, b_i \leq n-1, 1 \leq w_i \leq 10^3)$. The graph is not necessarily connected, but it is guaranteed vertex $n-1$ is reachable from vertex 0.

The score of a path is the *minimum* weight of any edge along that path.

In class, we learn about *simple* path. But for this task, your path do not have to be simple; edges and vertices can be visited multiple times.

Return the minimum possible score of a path between vertex 0 and $n-1$.

Solve this problem in Python and analyze its time complexity in terms of n and/or m.

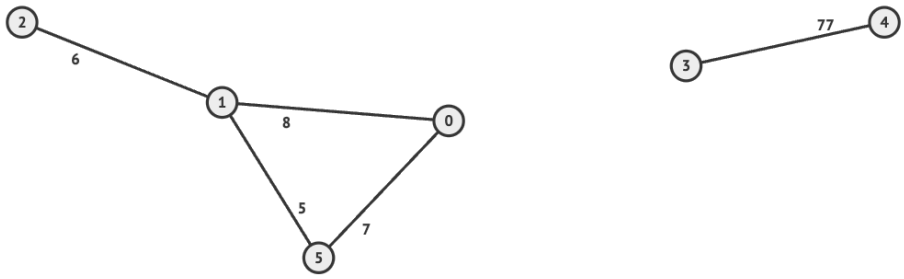


Figure 2: Graph For Example 1

Example 1: If $n = 6$, $EL = [[0, 1, 8], [1, 2, 6], [5, 1, 5], [5, 0, 7], [4, 3, 77]]$ (see Figure 2), then the output is 5. The still simple path from vertex 0 to vertex $n - 1 = 6 - 1 = 5$ with the minimum score is: $0 \to 1 \to 5$. The score of this path is $min([8, 5]) = 5$.

Example 2: If $n = 6$, $EL = [[0, 1, 8], [1, 2, 6], [5, 1, 5], [5, 0, 7], [4, 3, 77], [3, 2, 4]]$ (see Figure 3), then the output is 4. The not simple path from vertex 0 to vertex 5 with the minimum score is now: $0 \to 1 \to 2 \to 3 \to 2 \to 1 \to 5$. The score of this 'detour' path is $min([8, 6, 4, 4, 6, 5]) = 4$.

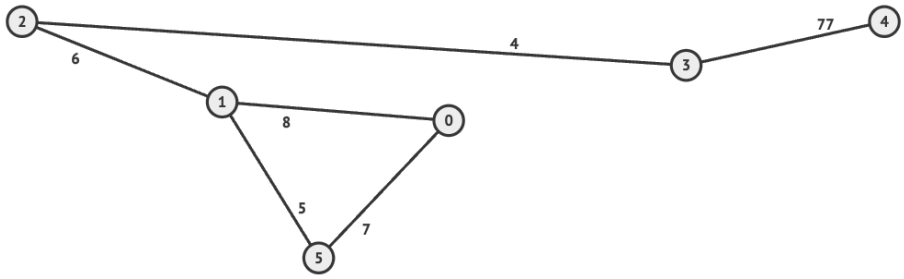


Figure 3: Graph For Example 2

A.5 An SSSP Variant (10 marks)

You are given a directed, weighted simple graph with n ($2 \leq n$) vertices labeled 0 to $n - 1$, and m ($0 \leq m \leq n \cdot (n - 1)/2$) directed edges stored in a list EL, where $EL[i] = [a_i, b_i, w_i]$ represents a directed edge from vertex a_i to vertex b_i with weight w_i ($0 \leq a_i, b_i \leq n - 1, 1 \leq w_i \leq 10^3$).

Each vertex x has a button that can be used at most once: when you arrive at vertex x, if its button has not been pressed, you may press it to reverse *one of the incoming edges* $y \to x$ (with weight w) into $x \to y$, and immediately traverse the reversed edge with weight $7 \cdot w$.

Return the shortest path weight from source vertex $s = 0$ to target vertex $t = n - 1$. If t is unreachable, return -1.

Example 1: If $n = 2$, $EL = [[1, 0, 5]]$, then we return 35.
The only way to go from $s = 0$ to $t = 1$ is to use the button at vertex 0, reverse $1 \to 0$ (weight 5) into $0 \to 1$, and take the path $0 \to 1$ with weight $7 \cdot 5 = 35$.

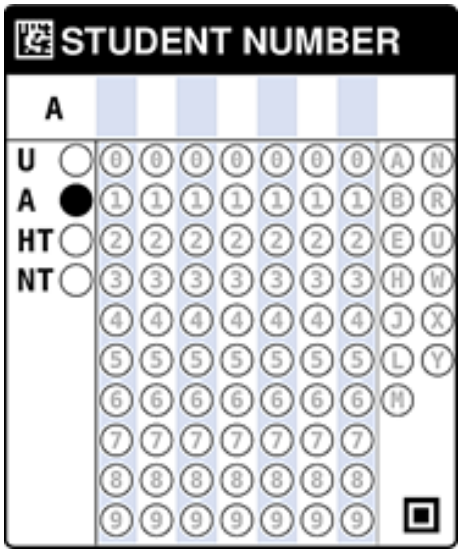
Example 2: If $n = 3$, $EL = [[0, 1, 10], [1, 2, 3], [2, 0, 2]]$, then we return 13.
There are two ways to go from $s = 0$ to $t = n - 1 = 3 - 1 = 2$.
If we use the button at vertex 0 to reverse $2 \to 0$ (with weight 2) into $0 \to 2$, and take the path $0 \to 2$, we will arrive at $t = 2$ with weight $7 \cdot 2 = 14$.
If we instead take the path $0 \to 1 \to 2$, we will arrive at $t = 2$ with weight $10 + 3 = 13$.

Example 3: If $n = 3$, $EL = [[1, 0, 10], [1, 2, 5], [2, 0, 2]]$, then we return 14.
There are two ways to go from $s = 0$ to $t = n - 1 = 3 - 1 = 2$.
If we use the button at vertex 0 to reverse the wrong edge $1 \to 0$ (weight 10) into $0 \to 1$, and take the path $0 \to 1 \to 2$, we will arrive at $t = 2$ with weight $7 \cdot 10 + 5 = 75$.
If we use the button at vertex 0 to reverse the correct edge $2 \to 0$ (weight 2) into $0 \to 2$, and take the path $0 \to 2$, we will arrive at $t = 2$ with weight $7 \cdot 2 = 14$.

This is a starred task ('you are out of messages until 7pm').
Design a correct $O((n + m) \log n)$ algorithm to solve this task for 10^* marks.
(or leave it blank for 2 marks).
An incorrect or significantly inefficient solution will receive no marks.

The Answer Sheet for Semester 2 AY2025/26

Write your Student Number in the box below using **(2B) pencil**. **Do NOT write your name**.



This portion is for examiner's use only

Section	Maximum Marks	Your Marks	Grading Remarks
A.1	10		
A.2	10		
A.3	10		
A.4	10		
A.5	10		
Total	50		

Box A.1. Find It

```
class SolutionGrasp:
    def findIt(self, nums1: List[int], nums2: List[int]) -> int:
        res = float('inf')
        n, m = len(nums1), len(nums2)
        for target in nums1:
            low, high = 0, m - 1
            found = False
            while low <= high:
                mid = (low + high) // 2
                if nums2[mid] == target:
                    found = True
                    break
                elif nums2[mid] < target:
                    low = mid + 1
                else:
                    high = mid - 1
            if found:
                res = min(res, target)
        return res if res != float('inf') else -1

s = SolutionGrasp()
print(s.findIt([1, 2, 3, 7], [2, 4, 7])) # should be 2?
print(s.findIt([1, 3, 5, 7], [2, 4, 6, 8])) # should be -1?
```

My analysis for SolutionGrasp is:

Ignoring time complexity, it gives correct output for all valid inputs: ◯ YES | ◯ NO, I will elaborate

Ignoring correctness, it runs in: ◯ $O(n+m)$ | ◯ $O(n \cdot m)$ | ◯ $O(m \log n)$ | ◯ Other, I will elaborate

It can crash on at least one valid corner case input: ◯ YES, I will elaborate | ◯ NO

I can improve upon the Generated AI code: ◯ YES, I will elaborate | ◯ NO, it is the best possible

Elaboration of my analysis above:

Only if you need to improve upon the Generated AI code.

My faster+correct code runs in: ◯ $O(n)$ | ◯ $O(n+m)$ | ◯ $O(n \cdot m)$ | ◯ Other, as written above.

Box A.2. Special Sublist

```
class SolutionTwin:
    def specialSublist(self, L: List[int], d: int, m: int) -> int:
        n = len(L)
        if m > n:
            return 0

        max_sum = 0

        for i in range(n - m + 1):
            subarray = L[i:i+m]

            distinct_elements = set(subarray)
            if len(distinct_elements) >= d:
                current_sum = sum(subarray)
                if current_sum > max_sum:
                    max_sum = current_sum

        return max_sum

s = SolutionTwin()
print(s.specialSublist([4, 1, 7, 7, 7, 3, 6], 3, 4)) # should be 23?
print(s.specialSublist([4, 1, 7, 7, 7, 3, 6], 2, 4)) # should be 24?
print(s.specialSublist([4, 1, 7, 7, 7, 3, 6], 4, 4)) # should be 0?
```

My analysis of SolutionTwin is:

Ignoring time complexity, it gives correct output for all valid inputs: ◯ YES | ◯ NO, I will elaborate

Ignoring correctness, it runs in: ◯ $O(n)$ | ◯ $O(n+m)$ | ◯ $O(n \cdot m)$ | ◯ Other, I will elaborate

It can crash on at least one valid corner case input: ◯ YES, I will elaborate | ◯ NO

I can improve upon the Generated AI code: ◯ YES, I will elaborate | ◯ NO, it is the best possible

Elaboration of my analysis above:

Only if you need to improve upon the Generated AI code.

My faster+correct code runs in: ◯ $O(\log n)$ | ◯ $O(n)$ | ◯ $O(n+m)$ | ◯ Other, as written above.

Box A.3. Special Binary Tree

```
# Definition for a binary tree node.
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class SolutionShallow:
    def countTotal(self, root: Optional[TreeNode], queries):
        values = []

        def postorder(node):
            if node:
                postorder(node.left)
                postorder(node.right)
                values.append(node.val)

        postorder(root)
        values.sort()

        def binary_search(arr, target):
            low = 0
            high = len(arr) - 1
            while low <= high:
                mid = (low + high) // 2
                if arr[mid] == target:
                    return True
                elif arr[mid] < target:
                    low = mid + 1
                else:
                    high = mid - 1
            return False

        count = 0
        for q in queries:
            if binary_search(values, q):
                count += 1

        return count

s = SolutionShallow()
T = TreeNode(-1, # details not shown, but T is as in Figure 1
# ...
print(s.countTotal(T, [-7, -13, -21, -8, -2, -9, -19, -2])) # should be 4?
```

My analysis for SolutionShallow is:

Ignoring time complexity, it gives correct output for all valid inputs: ◯ YES | ◯ NO, I will elaborate

Ignoring correctness, it runs in: ◯ $O(n)$ | ◯ $O(n \log n)$ | ◯ $O(n^2)$ | ◯ Other, I will elaborate

It can crash on at least one valid corner case input: ◯ YES, I will elaborate | ◯ NO

I can improve upon the Generated AI code: ◯ YES, I will elaborate | ◯ NO, it is the best possible

Elaboration of my analysis above:

Only if you need to improve upon the Generated AI code.

My faster+correct code runs in: ◯ $O(\log n)$ | ◯ $O(n)$ | ◯ $O(n \log n)$ | ◯ Other, as written above.

Box A.4. Life is not Simple

```
class SolutionShannon:
    def notSimple(self, n: int, EL: List[List[int]]) -> int:
        adj = [[] for _ in range(n)]
        for u, v, w in EL:
            adj[u].append((v, w))
            adj[v].append((u, w))

        min_score = float('inf')
        visited = [False] * n
        queue = [0]
        visited[0] = True

        idx = 0
        while idx < len(queue):
            u = queue[idx]
            idx += 1
            for v, w in adj[u]:
                if w < min_score:
                    min_score = w
                if not visited[v]:
                    visited[v] = True
                    queue.append(v)

        return min_score

class SolutionTalk:
    def notSimple(self, n: int, EL: List[List[int]]) -> int:
        adj = defaultdict(list)
        for u, v, w in EL:
            adj[u].append((v, w))
            adj[v].append((u, w))

        visited = [False] * n
        ans = [inf]

        def dfs(u):
            visited[u] = True
            for v, w in adj[u]:
                if not visited[v]:
                    ans[0] = min(ans[0], w)
                    dfs(v)
            visited[u] = False # backtrack (undo)

        dfs(0)
        return ans[0]
```

My analysis for SolutionShannon is:

Ignoring time complexity, it gives correct output for all valid inputs: ◯ YES | ◯ NO, I will elaborate

Ignoring correctness, it runs in: ◯ $O(n+m)$ | ◯ $O(n \log m)$ | ◯ $O(n \cdot m)$ | ◯ Other, I will elaborate

It can crash on at least one valid corner case input: ◯ YES, I will elaborate | ◯ NO

My analysis for SolutionTalk is:

Ignoring time complexity, it gives correct output for all valid inputs: ◯ YES | ◯ NO, I will elaborate

Ignoring correctness, it runs in: ◯ $O(n+m)$ | ◯ $O(n \log m)$ | ◯ $O(n \cdot m)$ | ◯ Other, I will elaborate

It can crash on at least one valid corner case input: ◯ YES, I will elaborate | ◯ NO

Thus, I will use: ◯ Either | ◯ SolutionShannon | ◯ SolutionTalk | ◯ My better solution below

Elaboration of my analysis above:

Only if you need to improve upon the Generated AI code.

My faster+correct code runs in: ◯ $O(1)$ | ◯ $O(n+m)$ | ◯ $O(n \log m)$ | ◯ Other, as written above.

Box A.5. An SSSP Variant (This is a STARRED task without Generative AI assistance)

◯ I give up, just give me the free 2 marks

◯ I know how to solve this, I will elaborate below (pseudocode is allowed and encouraged):

My solution runs in $O($_____$)$.

– END OF PAPER; All the Best –