NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
FINAL ASSESSMENT – Marking Scheme
AY2024/25 Semester 1

CS2040 – Data Structures and Algorithms

27 November 2024 Time allowed: 120 minutes

INSTRUCTIONS TO CANDIDATES

1. Do **NOT** open the question paper until you are told to do so.
2. This question paper contains **THREE (3)** sections with sub-questions. Each section has a different length and different number of sub-questions. It comprises **FORTEEN-plus-TWO (16)** printed pages, including this page (and 2 scratch pages).
3. Answer all questions in this paper itself. Answer the questions in Examplify, **NOT on this paper**.
4. This is an **Open Book Quiz**. You can check the lecture notes, tutorial files, problem set files, CP4 book, or any other books that you think will be useful. But remember that the more time that you spend flipping through your files implies that you have less time to actually answer the questions. No code editors or IDEs are allowed.
5. When this Final Assessment starts, **please enter your answers into the Exemplify software.**
6. The total marks for this paper is **100**.

STUDENT NUMBER: | A | | | | | | | | |

TUTORIAL GROUP

For examiners' use only		
Question	***Max***	***Marks***
Q1–10	40	
Q11–16	18	
Q17–18	5+5	
Q19–21	9+9+9	
Q22	5	
Total	**100**	

Marking/Grading Scheme for Q16, Q19, Q20 and Q21

(All other questions are MCQ/MRQ.)

Q16 Marking/Grading Scheme

0 marks:
No answer.

1 mark:
Answer contains incorrect claims. The answer contains a quite significant misunderstanding of how Kahn's algorithm works and/or what a topological sort is and/or what SCCs are.

2 marks:
Answer contains (mostly) correct claims, but is incomplete. Some understanding is there and some partial arguments are correct, e.g., iff **N** == **V**, then the graph has **V** SCCs, i.e., it is an acyclic graph (DAG). The case of 1 SCC versus 2 or more SCCs is not discussed or the wrong conclusions are drawn (e.g., incorrectly claiming that **N** == 0 means exactly 1 SCC).

3 marks:
Answer is for the most part correct, but the argument is missing some parts, e.g., the 3 cases (or at least the 2 cases that conflict when **N** == 0).

4 marks:
The answer is for the most part, or fully, correct, with, e.g., a simple counter example.

Comments

Common misconceptions:

- Some students wrote that we could run other algorithms, such as Kosaraju's. First, not enough information is given (no **G**) to run Kosaraju's, and second, even if we could, that is not the question. The question did not ask about the exact number of SCCs. Basically, it asked, with Kahn's, can we distinguish whether the graph falls into one of 3 categories: (A) 1 SCC or (B) V SCCs or (C) 2...V-1 SCCs. It turns out only (B) can be determined accurately with Kahn's.
- N cannot be larger than V with Kahn's. Some students have stated that if N > V then there are multiple cycles. However, only vertices with in-degree 0 can appear in A, and no duplicates will appear, therefore N <= V.
- Some students wrote that if N == 0, then there is exactly 1 SCC (one big cycle). However, this is not necessarily true. There could be multiple smaller cycles.

- Some students wrote that if N == V then there is 1 large cycle, i.e., 1 SCC. That is not the case. If there exists 1 large cycle then N == 0 (no nodes have in-degree 0).
- Some students thought that Kahn's will output all the nodes into A[], even if there are cycles that involve multiple vertices. That is not the case. Kahn's will not process and hence not put a vertex into A[] if its indegree is > 0.
- Some students looked at the justification in the inverse direction, i.e., given a certain number of SCCs, what would **V**, **N** and **A** be. However when we make logical deductions, X implies Y is not equal to Y implies X.

Q19 Marking/Grading Scheme

Incorrect algorithm: 0m

Inefficient algorithm: max 4m

Partially correct algorithm: max 6-8m

Correct algorithm: 9m

Some marks can be deducted for other minor mistakes.

Comments

The most common answer given was to augment the DSLNodes with the index or rank of the nodes in the bottom level. This is technically a correct algorithm, due to a flaw in the question. However, students should realise that this is not a very feasible idea due to the inefficiency of maintaining the ranks of every node. The grader decided that due to the flaw of the question and that this solution if correctly described still shows understanding of the question and Skip List structure, students should still get max 6 marks instead of a lower mark for inefficiency.

The most common mistake thus also occurs for this given solution, where students incorrectly check the rank of the current node instead of the rank of the right node when deciding whether to move to the right or bottom of the current node. This results in a completely incorrect algorithm. Another somewhat common mistake is to blindly copy the select algorithm for BSTs.

A good trend observed is that despite it being a more difficult and non-standard problem, most students attempted the problem, giving at least the more inefficient linear time algorithm.

In general, algorithms were mostly quite clearly written. However, students should take note to use terms that are not ambiguous for that context. E.g., Augmenting DSLNode with size without explaining what size represents does not really make sense.

Q20 Marking/Grading Scheme

a) **3 marks:** Correctly models the question as a graph problem, specifying the vertices, edges, and edge weights.
2 marks: Graph model is mostly correct but misses minor details.
1 mark: Attempts to model but contains major errors or lacks clarity.

b) **6 marks:** Correct algorithm with all necessary details such as appropriate transformations and priority queue modeling.
5 marks: Suboptimal but correct algorithm.
2–5 marks: Partially correct algorithm, missing critical details like edge transformations, incorrect relaxation logic, or improper priority queue usage.
1 mark: Attempts the question but proposes an inappropriate approach, such as finding the maximum spanning tree or solving the problem as a maximin/minimax path.

Common mistakes:

- **Forgetting to use a max heap:** A max heap is essential when modeling this as a maximization problem, as it ensures prioritizing nodes with higher probabilities during the relaxation process.
- **Confusing with maximum spanning tree (MST):** Modeling this as an MST rather than a single source "longest" path problem is a common error.
    - For example, for a graph with three vertices A, B, and C where p(A, B) = 0.9, p(B, C) = 0.6, and p(A, C) = 0.55, the maximum probability path from A to C is the direct edge A–C with a probability of 0.55. However, the MST might produce the path A - B - C with a probability of 0.54.
- **Modeling as a maximin/minimax path problem:** Similar issues arise when solving for a maximin/minimax path, which does not guarantee the correct result.
- **Using probabilities as edge weights without appropriate handling:** When probabilities are used directly as edge weights, multiplication should be used during relaxation. If addition is preferred, a log transformation of edge weights is necessary to convert the multiplication of probabilities into addition.
- **Incorrect initialization of the priority queue (PQ) in Dijkstra's:** Examples include initializing the estimated probability to all nodes as +infinity in a maximization problem or initializing the probability of going from the source to itself as 0 when probabilities are directly used as edge weights.
- **Reformulating the problem as minimizing failure probability:** Some attempts incorrectly transform the problem into finding the minimum probability of not

being able to go home rather than maximizing success probability. This approach is incorrect as the two are not equivalent.

- For example, consider a graph with three vertices A, B, and C where p(A, B) = 0.9, p(B, C) = 0.8, and p(A, C) = 0.75. The maximum probability path from A to C is the direct edge A-C with a probability of 0.75. However, when the edge weights are transformed into failing probabilities 1 - p, they become 0.1 for A – B, 0.2 for B – C and 0.25 for A – C. The path A – B – C is chosen based on minimizing failure instead of the correct maximum success path A - C.

Q21 Marking/Grading Scheme

Answers are broadly categorized into 3 categories and further deducted based on any mistakes.

Maximum Marks	Commonly presented answers
2	Algorithms that are largely incomplete / completely incorrect / have huge oversights / Run slower than $O(Q^2)$ • Using Direct Addressing Table when question specifies **real numbers** • Algorithms that use duplicate copies of numbers in the data structure. Add(k, x): k is potentially huge. Keeping multiple nodes with duplicate values, in an AVL tree for example, getting minimum and maximum values will take O(logN) time where N is the total number of copies of all numbers (which is potentially unboundedly massive)
4	• Algorithms that run in $O(Q^2)$, i.e brute force approach. Common answer is to maintain a HashMap to track the frequencies of numbers. Can be maintained in O(1) time. But ComputeRange() will take O(Q) time to iterate through all keys to get max/min
9	• Algorithms that run in O(QlogQ)

Common mistakes:

- Using BinarySearchTree instead of Balanced BinarySearchTree (not penalized)
- Edge case for ComputeRange() when only having 1 number should return 0 since maximum element == minimum element, many students returned the number itself (not penalized)
- No guarantee that there is at least 1 number at all times! Only guaranteed there is at least 1 number when ComputeRange() is called. Thus calling successor/predecessor/max/min on an empty tree to update min/max assumed to fail unless explicitly handled. Better solution is to just call maxElement() and minElement() when ComputeRange() is called.
- Very common misconception that PriorityQueue / MaxHeap / MinHeap have access to all elements in sorted ordering. E.g Taking the “last element” of a maxheap does not give you the minimum element!

- Very common mistake for answers that use HashMap + MaxHeap + MinHeap (2nd accepted answer below) is to not do lazy deletion: After Remove(k,x), if x now has 0 frequency, and we immediately try to remove it from maxheap/minheap this is O(Q) time, not O(logQ). Searching for arbitrary element in PQ/Heap already incurs O(Q) time, deleting it then costs another O(logQ) time. Remove is only O(logQ) for the head/root of the PQ/Heap. Thus it is more efficient to defer the removal till the ComputeRange() query where deletions are only done to the head/root.