CSE 310 Data Structures & Algorithms
Spring 2024

Instructor: Yiran "Lawrence" Luo

Final Exam (Version A), Apr 30th, 2024
Arizona State University

- The exam is closed-book, but you may use one double-sided A4/US letter cheat sheet during the exam.
- One paper is intentionally left empty at the end. That is your scratch paper.
- Be meticulous. Read the questions carefully. Mind the traps.
- The maximal possible score is 100 points, with 1 bonus which counts as Extra Credits.
    - We will apply a curve to the average of 80% if the raw average is below 78%.
- Time limit: 75 minutes.

The Scorebook (reserved for the graders only)

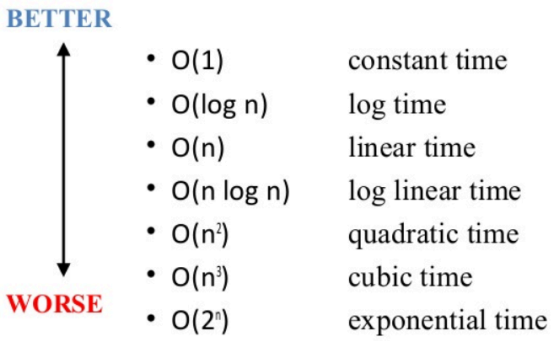
Part	Score
I (Multiple Choices)	/ 20
II (True or False)	/ 10
III	/ 30
IV	/ 20
V	/ 20
Total	/ 100
Bonus	/ 1 (counted towards Extra Credits)

Your name (print):_____

Your ASU ID:_____

Helper Formulas & References

Big-O rankings



The Master Theorem for recurrence analysis (a, b, d are all positive constants)

If $T(n) \leq aT\left(\frac{n}{b}\right) + O(n^d)$

then

$$T(n) = \begin{cases} O(n^d \log n) & \text{if } a = b^d \quad \text{(Case 1)} \\ O(n^d) & \text{if } a < b^d \quad \text{(Case 2)} \\ O(n^{\log_b a}) & \text{if } a > b^d \quad \text{(Case 3)} \end{cases}$$

Big-O Time Complexities of Graph-based Algorithms

Given a satisfactory graph of G(E, V) per algorithm (E - set of edges, V - set of nodes/vertices):

DFS/BFS/TopoSort: O(|E| + |V|)
Prim's/Kruskal's/Dijkstra's: O(|E| * log(|V|))

P.S. Kruskal's algorithm for MST is in fact O(|E| * log(|E|) but we have already proved in class that it is equivalent with O(|E| * log(|V|)) .

Big-O Time Complexities of Dynamic Programming Algorithms per Case

Finding the minimum number of coins that sum up to *a total change of S*: **O(S)**
Finding the length of the LCS between *two sequences of length* L_X *and* L_Y: **O($L_X L_Y$)**
Finding the minimum number of scalar multiplications of *a k-matrix chain product*: **O(k^3)**

Part I. Multiple-choice. There is only one correct option for each question. (2.5 * 8 pts)

1. (DP, Concept) According to the descriptions below, which of the following PC components **best resembles** the fundamental logic behind Dynamic Programming?

    a. The Arithmetic Logic Unit (ALU) in your CPU that performs bitwise calculations superbly fast.
    b. The CPU Cache that temporarily reserves calculated data for later CPU operations.
    c. The Hard Drive that permanently stores and sorts your finished data into organized grids.
    d. The Power Management Chip that arranges the power consumption for different components (e.g. motherboard, RGB lights) by monitoring their running cost.

2. (DFS/BFS) Recall we have talked about how we implement Depth-First-Search with a stack, and Breadth-First-Search with a queue. Let's assume both the DFS stack and the BFS queue store Node struct objects per unit.

    Given a special-case graph $G = (V, E)$ of **n nodes but 0 edges**, what is the necessary capacity of the stack, a.k.a. the space complexity, in order to perform DFS on this particular G to check out all the nodes? Choose the ***tightest*** upper bound in terms of Big-O-of-n.

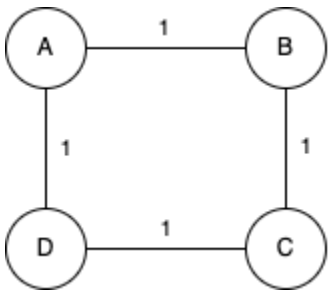
    a. O(log(n))
    b. O(1) pushing and popping only up to 1 Node at a time since there is no edge.
    c. O(n log(n))
    d. O(n)

3. (Topological Sort) Running the DFS-based Topological Sorting algorithm on a 3-Node Directed Acyclic Graph $G = (V, E)$ yields this output sequence : < A B C > . Which of the following statements is TRUE regarding the relation between Node A and Node B in G?

    a. Node B must be one of the predecessors of Node A.
    b. Node B must be one of the successors of Node A.
    c. Both a. and b. are true.
    d. Neither a. nor b. is true. E.g. A->C<-B and you start with Node C.

4. (MST) In the Undirected Weighted Graph on the right, how many possible Minimum Spanning Trees are there?

    a. 1
    b. 4
    c. 8
    d. 16



5. (MST, Dijkstra's) We are given a specific connected undirected weighted graph $G' = (V, E)$ (i.e. all nodes are reachable from others via edges) where **all edges carry the same positive weight value**. Which of the following statements is TRUE?

    a. Even without any tie-breaking mechanism, Prim's and Kruskal's will still yield the same MST out of G'.
    b. A Single Source Shortest Path subgraph from running Dijkstra's on G' with any node as the source is effectively one MST of G'.
    c. Kruskal's algorithm in this scenario can be done in O(|E|) since it doesn't need to sort the edges any more.
    a. Prim's algorithm in this scenario can be done in O(|E|) since you can simply extend to any unincorporated node at each iteration.

6. (Dijkstra's) In Project 2, you are tasked to perform multiple times of Dijkstra's algorithm from different source nodes. Given a special connected undirected weighted graph of **n nodes and (n-1) edges** (i.e. all nodes are reachable from others via edges), what is the Big-O-of-n time cost to obtain all its possible Single Source Shortest Paths subgraphs for all n source nodes? Choose the ***tightest*** upper bound.

    a. O(n log(n))
    b. O(n) Since the graph contains exactly (n-1) edges, all SSSP solutions are essentially the graph by itself as the graph is a spanning tree.
    c. $O(n^2 \log(n))$
    d. $O(n^2)$

(DP in 1D, Greedy, Coin Change Problem) Below is the context for Question 7 and 8.
You have made 25 pounds of handmade candy today. However, you only have package bags in volumes of 1, 5, and 7 pounds in the shop. To save packaging cost for resale, you want to use as few packages as possible while no package bag is half-filled.

7. If you use the Greedy algorithm, how many packages will you end up using?

    a. 4
    b. 5
    c. 6
    d. 7, 3 7-lb's and 4 1-lb's

8. If you use the DP algorithm, how many packages will you end up using instead?

    a. 4
    b. 5, 5 5-lb's or 1-5-5-7-7.
    c. 6
    d. 7

Part II. True or False. You only need to answer in T or F before each statement. (2 * 5 pts)

[ ] If Node X of an arbitrary DAG G=(V,E) appears last in one of the topological orders, Node X will never appear first in any of all possible topological orders of graph G.

False. One counterexample is DAGs with no edges.

[ ] The very reason Prim's and Kruskal's have the same time complexity in Big-O is because both algorithms try to incorporate *as few edges as possible* into the MST.

False. All MSTs of the same graph have |V| - 1 edges regardless.

[ ] Dijkstra's algorithm applies to weighted graphs that have *up to one* negative weighted edge.

False. Dijkstra's cannot handle any negative weight.

[ ] Dynamic Programming algorithms in general save time by using no extra space.

False. DP trades efficiency by using extra space to save the pre-calculated.

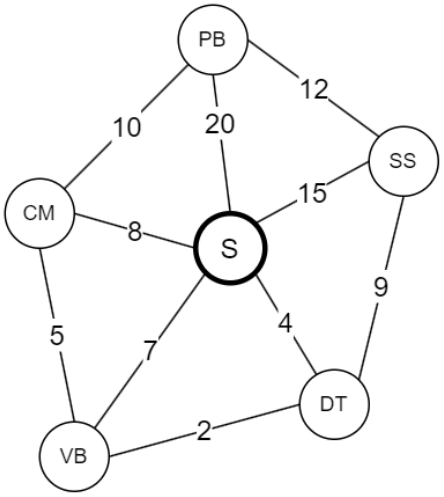
[ ] We may simplify the auxiliary 2d DP array m[n, n] that keeps track of the number of scalar matrix multiplications into a 1d array, because all we do in DP is combine those neighboring two matrices who share the largest dimensions as early as possible.

False. Because we need to consider both the individual costs to reach two intermediate products as well as the merging cost to multiply two intermediate products.

PART III. Checkpoint Strategy with Dijkstra's (30 pts)

Suppose you are playing GTA Online with a customized timed mission. You are tasked to travel to five checkpoints scattered all over the State of San Andreas to collect five craters. Once you collect the crater at a checkpoint, you will be teleported back to the starting point. Inspired by what you have learned in CSE 310, you come up with a good strategy to beat this mission as fast as possible using Dijkstra's Algorithm.

For your convenience, the map is remodeled into a weighted undirected graph as below. The starting point is at node S, and all the other nodes represent the five checkpoints. The weights of the edges represent the time cost in between two checkpoints.



1. Using the Dijkstra's Single Source Shortest Path algorithm, what is the final iteration of the table of records? Fill in the blanks down below **(2 * 10 pts)**.

Node	S	VB	DT	SS	PB	CM
Distance-from-S	0	6	4	13	18	8
Previous Node	NIL	DT	S	DT	CM	S

2. What is the order of nodes being incorporated into the Shortest Paths subgraph, aka the 'Considered' set of nodes? The first node to be incorporated has been given. **(2 * 5 pts)**

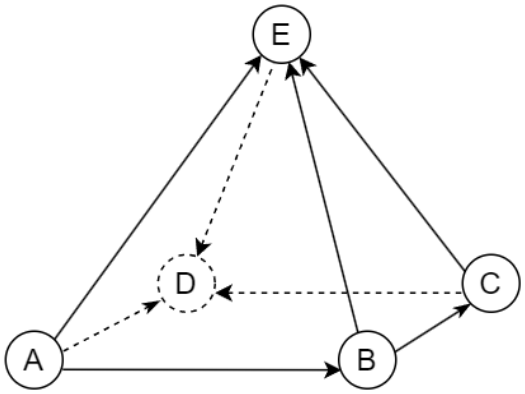
Answer: {S, DT, VB, CM, SS, PB (the order of the Distance-from-S in 1.) }

Part IV. Topological Ordering, but in 3D (20 pts)

In practice, a graph may not need to be rendered in 2D. We have the following two directed graphs where their vertices/nodes and edges form up 3D pyramids (the obstructed edges are shown in dashed lines).

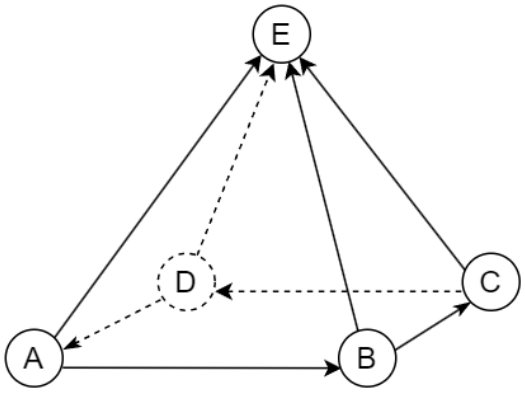
Do such graphs still have topological orders? If yes, use the DFS-based approach and show us one of the possible topological orders of the nodes; if no, explain why in at most 2 full sentences in English. **(2 pts for correct yes-or-no short answer, 8 pts for correct topological order example / explanation**.)

1. **(10 pts)**



Yes. **A B C E D** is the one and only one possible solution.

2. **(10 pts)**



No, because the bottom surface is a cycle A->B->C->D->A.

Part V. The Maximum Length of Hidden Message, LCS Style. (20 pts)

Special Agent 310 has sent you two sequences of letters via a secret telegram. As his handler, you have already known there is a hidden message encoded as the Longest Common Subsequence (LCS) between the two sequences. As a crucial step, you want to figure out **the longest possible length of the hidden message** using the Dynamic Programming-based LCS-LENGTH algorithm introduced in our class.

The two letter sequences that you have received from Agent 310 are:

X = < O N S N> and
Y = < S O N N E > .

Show us the finalized Table c that keeps track of the LCS length between X_i (the first i-element prefix of X) and Y_j (the first j-element prefix of Y) as part of the DP algorithm. Hint: You may want to fill in the table row by row, or column by column. The 0-row and the 0-column have been initialized for your convenience.

c[i, j]	j = 0	1	2	3	4	5
i = 0	0	0	0	0	0	0
1	0	**0**	**1**	**1**	**1**	**1**
2	0	**0**	**1**	**2**	**2**	**2**
3	0	**1**	**1**	**2**	**2**	**2**
4	0	**1**	**1**	**2**	**3**	**3**

For the record, the LCS between X and Y is <O N N>.

Bonus Challenge - The Other LCS, Longest Common SUBSTRING (1 pt)

While a subsequence is achieved by removing arbitrary items from the original sequence, a **substring** is a consecutive subpart of the original. E.g. The substrings of 'abc' are '', 'a', 'b', 'c', 'ab', 'bc', and 'abc' . Noticeably, 'ac' is a subsequence of 'abc' but is NOT a substring of 'abc'.

Given the same inputs from Part V, the two letter sequences:

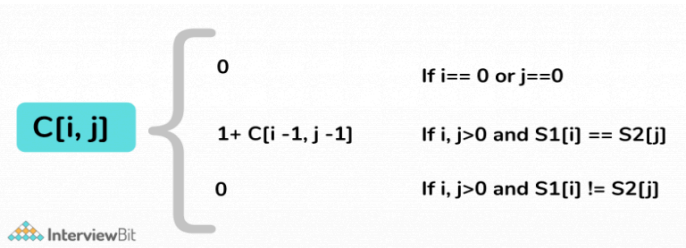
X = < O N S N> and
Y = < S O N N E > .

We may still use the same auxiliary Table c structure that records the lengths. But this time, **what kind of substructure length** are you supposed to keep track of between X_i (the first i-elements of X) and Y_j (the first j-elements of Y), in order to find **the length of the Longest Common SUBSTRING** between X and Y? And where is your eventual return answer located in the new Table c?

As your answer, show us the new final state of the Table c for **tracking the length of Longest Common SUBSTRING instead,** and justify your solution in one or two sentences in the space below if necessary.

c[i, j]	j = 0	1	2	3	4	5
i = 0	0	0	0	0	0	0
1	0	0	1	0	0	0
2	0	0	0	2	1	0
3	0	1	0	0	0	0
4	0	0	0	1	1	0

We choose to keep track of the Longest Common Suffix between X_i and Y_j. Such that if the last elements x_i and y_j do not match, we reset the counter to 0.



The length of the Longest Common Substring is at the maximum value of the entire Table c, which is at c[2, 3] = 2 and the actual string is 'ON'. The intuitive way to understand the solution is that 'once you break the combo, you start over counting the hits from 0'.

0.5 will be graded if the student shows the length of Longest Common Substring is 2. Another 0.5 will be granted if the student shows the answer is located at c[2,3] in 'ON'.