ShanghaiTech University

CS101 Algorithms and Data Structures

Fall 2024

Final Exam

Instructors: Dengji Zhao, Yuyao Zhang, Xin Liu, Hao Geng

Time: Jan 8th 8:00-10:00

INSTRUCTIONS

Please read and follow the following instructions:

- You have 120 minutes to answer the questions.
- You are not allowed to bring any papers, books, or electronic devices including regular calculators.
- You are not allowed to discuss or share anything with others during the exam.
- You should write the answer to every problem in the dedicated box **clearly**.
- You should write **your name and your student ID** as indicated on the top of **each page** of the exam sheet.
- If you need more space, write "Continued on Page #" and continue your solution on the referenced scratch page at the end of the exam sheet.

Name	
Student ID	
Exam Classroom Number	
Seat Number	
(please copy this and sign)	All the work on this exam is my own.

THIS PAGE INTENTIONALLY LEFT BLANK.

DO NOT WRITE ANY ANSWER IN THIS PAGE!

1. (18 points) Single Choice

Each question has **exactly one** correct answer. Fill your answers **in the box below**.

Notice: Fulfill your answer or we may take it unspecified.

(a)	◯ A	◯ B	◯ C	◯ D	(b)	◯ A	◯ B	◯ C	◯ D
(c)	◯ A	◯ B	◯ C	◯ D	(d)	◯ A	◯ B	◯ C	◯ D
(e)	◯ A	◯ B	◯ C	◯ D	(f)	◯ A	◯ B	◯ C	◯ D

(a) (3') Consider a disjoint set of size 6 - with path compression but **no** union by rank.

The disjoint set is initialized by `parent[i] = i` for $i \in [1, 6]$, and has 2 operations:

- `find(int i)`: Return the root of the tree that contains `i`, while do path compression.
- `union(int i, int j)`: `parent[find(i)] = find(j)`.

Now given a sequence of operations, `union(1, 2)`, `union(3, 4)`, `union(5, 4)`, `union(4, 2)`, `union(6, 5)`. What is the height of the disjoint-set tree that contains `2`?

A. 1

B. 2

C. 3

D. 4

(b) (3') Given an undirected graph $G = (V, E)$ whose adjacency matrix is A ($\forall v_i, v_j \in V$, $A_{ij} = 1$ if $(v_i, v_j) \in E$ and 0 otherwise). Which of the following statements is false?

A. $A = A^T$ i.e. A is symmetric.

B. $\forall\, v_i \in V$, $(A^2)_{ii} = \deg(v_i)$.

C. $tr(A^2) = 2|E|$.

D. $\sum_{v_i, v_j \in V} A_{ij} = |E|$.

(c) (3') Which of the following statements about topological sort is correct?

1. Topological sort can be applied to any connected graph.
2. Topological sort is used to find (one of the) shortest paths in a weighted graph.
3. Topological sort is a linear ordering of vertices in a directed acyclic graph, where for every directed edge $(u \to v)$ from vertex u to vertex v, u comes before v in the ordering.
4. Topological sort can result in multiple valid orderings for the same graph.

A. Only 1 and 2

B. Only 3 and 4

C. Only 1, 2 and 4

D. All of the above

(d) (3') To find the minimum number of moves for the N-puzzle problem, the fastest one of the following algorithms is

	3	2
8	7	1
4	5	6

$\Longrightarrow$

1	2	3
4	5	6
7	8	

A. BFS

B. DFS

C. Dijkstra

D. A*

(e) (3') Given n points $\{p_i = (x_i, y_i)\}$ in a plane, we can obtain a weighted complete graph $G = (V, E)$, where the vertices correspond to points in the plane and the weights of the edges $e = (v_i, v_j)$ are equal to the distance between pairs of points p_i and p_j i.e. $w(e) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. Which of the following statements is true?
Hint: $|V| = n, |E| = \frac{n(n-1)}{2}$.

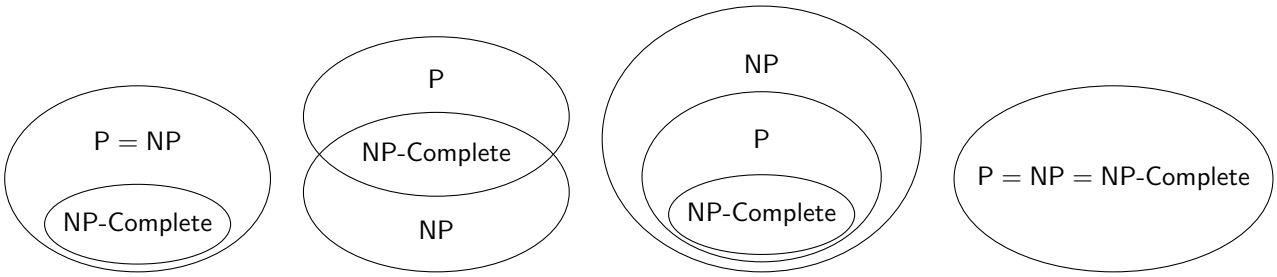
A. Using Kruskal's algorithm can find the MST more efficiently, which takes at most $\Theta(n^2 \log n)$ time.

B. Using Prim's algorithm can find the MST more efficiently, which takes at most $\Theta(n^2 \log n)$ time.

C. We can use L-∞ Distance i.e. $\max(|x_i - x_j|, |y_i - y_j|)$ as a consistent heuristic function to do A^* graph search on this graph.

D. We can use Manhattan Distance i.e. $|x_i - x_j| + |y_i - y_j|$ as a consistent heuristic function to do A^* graph search on this graph.

(f) (3') Suppose zsc2003 has found an algorithm that correctly solves the Subset-Sum problem in polynomial time. Which one of the following Venn diagrams can correctly represent the relationship among P, NP, and NP-Complete under this circumstance? (Suppose all problems involved both have yes-instances and no-instances.)



A. B. C. D.

2. (20 points) Multiple Choices

Each question has **one or more** correct answer(s). Select all the correct answer(s). For each question, you will get 0 points if you select one or more wrong answers, but you will get 2 points if you select a non-empty subset of the correct answers. Fill your answers **in the box below**.

Notice: Fulfill your answer or we may take it unspecified.

(a)	◯ A	◯ B	◯ C	◯ D	(b)	◯ A	◯ B	◯ C	◯ D
(c)	◯ A	◯ B	◯ C	◯ D	(d)	◯ A	◯ B	◯ C	◯ D

(a) (5') Which of the following statements about the minimum spanning tree is correct?

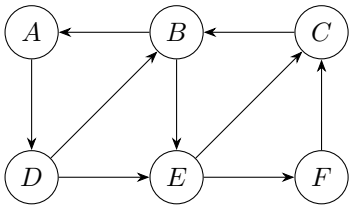
A. If a connected graph has $|V|$ nodes and $|E|$ edges, all edges have non-negative weights, then we can use the Dijkstra algorithm with a binary heap in $O(|E| \log |V|)$ time to get the minimum spanning tree.

B. If we do Prim's algorithm from one vertex and get a unique MST, we can also get the shortest path from the vertex to other vertices.

C. We can get the maximum spanning tree of a graph by negating the edge weights and running any minimum spanning tree algorithm on it.

D. After we pick up all the edges in the shortest path i.e. $d(u) + w(u, v) = d(v)$, they will form a tree if the graph is connected and undirected.

(b) (5') Suppose we have the following graph, and we perform a Depth-First Search (DFS) or Breadth-First Search (BFS) starting from a specified vertex. Below are some possible orders in which the vertices might be visited during the traversal. Which of these orders could actually occur, based on the rules of DFS or BFS?



A. DFS starting from A, order: ADEFBC

B. DFS starting from D, order: DBAECF

C. BFS starting from B, order: BEAFCD

D. BFS starting from E, order: EFCBAD

(c) (5') For a simple positive-weighted directed graph, if the heuristic $h(\cdot)$ in A* algorithm is consistent, which of the following statements is/are TRUE? Here are some notations:

- $w(u, v)$ is weight of edge $u \to v$.
- $d(u, v)$ is the actual length of shorted path from u to v.
- $p(u) = d(s, u) + h(u)$ is the priority in GRAPH-SEARCH, where s is the source vertex.

A. $h(u) \leq w(u, v) + h(v)$ for any edge $u \to v$.

B. $h(u) \leq d(u, v) + h(v)$ for any two vertices u, v such that there is a path from u to v.

C. $p(u)$ is non-decreasing during the iteration of u in GRAPH-SEARCH.

D. $h(\cdot)$ is admissible.

(d) (5') Which of the following statements must be true?

A. Consider a problem where the input is an integer $x \geq 0$. If the time complexity of this problem is $O(x)$, then it's in P.

B. Consider two problems X and Y. Suppose that $X \leq_p Y$, then X is in $\mathsf{NP\text{-}Complete}$ only if Y is in $\mathsf{NP\text{-}Complete}$.

C. Consider two problems X and Y. Suppose that $X \leq_p Y$, then X is in P only if Y is in P.

D. If $X \in \mathsf{NP\text{-}Complete}, Y \in \mathsf{NP}, X \leq_p Y$, then $Y \notin \mathsf{P}$ can prove $\mathsf{P} \neq \mathsf{NP}$.

3. (12 points) Adjacency matrix

One basic usage of an adjacency matrix is to represent a graph. However, it also has other applications in graph theory. In this question, we only consider **simple undirected unweighted graph**. We use G to denote a graph, and A to denote its adjacency matrix.

By using dynamic programming, we can know that $(\mathbf{A^n})_{\mathbf{ij}}$ **is equal to the number of different paths from** $\mathbf{v_i}$ **to** $\mathbf{v_j}$ **with length n**. You can use this conclusion in this question without any proof.

(a) **Basic properties**

i. (2') The i_{th} diagonal element of A^2 represents _____.

(b) **Counting triangles**

Let's start with a simple task! You are required to compute the number of triangles in a graph. In a graph, a triangle is defined as a cycle with **exactly** 3 vertices in it.

i. (3') Since there is no self-loop in this graph, a path with length 3 from a vertex v_i to itself must be a contour on a triangle which includes v_i, and a triangle will include **exactly** _____ different vertices. So the total number of triangles in this graph is $\frac{1}{3} \cdot \sum_{i=1}^{|V|} \frac{t_i}{2}$, where $t_i =$_____.

(c) **Counting pentagons**

Now we want to calculate the number of pentagons in a graph. In a graph, a pentagon is defined as a cycle with **exactly** 5 vertices in it.

i. (2') For a graph with no triangle in it, write a simple expression to calculate the number of pentagons in that graph. You can use A to represent the adjacency matrix of the given graph. (Hint: $tr(M) = \sum_{i=1}^{k} M_{ii}$, where M is an $k \times k$ matrix)

ii. (3') Prove the correctness of your solution.

(d) **Not always easy...**

For a graph with triangles in it, it is not so easy to calculate the number of all pentagons. But it's relatively easy to calculate the number of pentagons which includes some certain nodes.

i. (2') Suppose you are given a graph G with n vertices and its adjacency matrix A. Given a node $v_i \in G$ which is not included in any triangle, what is the number of pentagons that includes v_i?

A. $\frac{(A^5)_{ii}}{2}$ B. $\frac{(A^5)_{ii} - \sum_{k=1}^{n} A_{ik} \cdot (A^3)_{kk}}{2}$ C. $\frac{(A^5)_{ii} - \sum_{k=1}^{n} (A^2)_{kk} \cdot (A^3)_{kk}}{2}$

4. (11 points) Boruvka's Algorithm

Let $G = (V, E)$ be a weighted connected graph. Let $w(u, v) \in \mathbb{R}$ be the weight of the edge $(u, v) \in E$. The goal of this problem is to invent a new algorithm to find one MST of a graph.

(a) Review:

i. (2') The time complexity of finding one MST of a connected graph in general is _____ for Kruskal's algorithm (optimized by disjoint sets, taking $\alpha(n)$ as a constant) and _____ for Prim's algorithm (optimized by binary heaps).

A. $\Theta(|V|^2)$ B. $\Theta(|E|)$ C. $\Theta(|V| \log |E|)$ D. $\Theta(|E| \log |E|)$

ii. (0') Recall **Cut property**: Given $G = (V, E)$ and $S \subset V$ is a non-empty proper subset of V. The minimum weighted edge in $\{(u, v) | u \in S, v \in V - S\}$ must be part of at least one MST. Prim's algorithm uses the property.

(b) Inspiration: Consider how Prim's find one MST of a graph. We only find one edge at a time, what if we find multiple edges at a time?

i. (1') If the graph has been divided into m disjoint connected components, design a method to find $\Theta(m)$ edges in one MST (No need to justify your answer in this subproblem.)

ii. (2') Prove the correctness of the algorithm. You may use the Cut property to show that the selected edges belong to at least one Minimum Spanning Tree (MST). Additionally, denote x as the number of edges selected by the algorithm. Verify that x satisfies the inequality: $am \leq x \leq bm$ where a and b are constants, and m is the number of connected components in the graph so that $x = \Theta(m)$.

(c) Details: After we find edges in MST, we should merge the connected components.

i. (1') Which data structure is best for efficiently merging the connected components?

i. _____

A. Binary Heap B. Disjoint sets C. Binary Search Tree D. Hash Table

ii. (1') What is the time complexity of all m connected components into one component?

ii. _____

A. $\Theta(|V|)$ B. $\Theta(|E|)$ C. $\Theta(m)$ D. $\Theta(m \log |V|)$

(d) Integration: From all those above, we can finally design a new algorithm.

i. (3') If we use the trivial method to find edges each time i.e. determine the edges by traversing all of them. Describe the whole algorithm and analyze the time complexity of them (Make sure your upper bound is tight otherwise you may lose your points).

ii. (1') Determine the time complexity of your algorithm (with analysis).

5. (10 points) Minimal dot product of two arrays

Given two arrays a and b of length n, we define the dot product of the two arrays as $\sum_{i=1}^{n} a_i b_i$. Now you are allowed to reorder the elements of both arrays and your task is to minimize the dot product of two arrays after reordering. Mathematically, a reorder of an array is defined by a permutation i.e. a bijective mapping $p : [n] \to [n]$ and the dot product after reordering is $\sum_{i=1}^{n} a_i b_{p_i}$.

For example, if $a = \langle 1, 2, 3 \rangle, b = \langle 4, -2, -1 \rangle$, then the minimal dot product is

$$1 \times 4 + 2 \times (-1) + 3 \times (-2) = -4$$

Please design a **greedy algorithm** as efficient as you can that returns the minimal dot product.

(a) (3') Describe your algorithm in **natural language** or **pseudocode**.

(b) (1') The time complexity of this algorithm is $O($_____$)$ (give the most precise upper bound).

(c) (6') Prove the correctness of your greedy algorithm by **Exchange Arguments**.

i. (1pt) **Definition of a solution**

ii. (1pt) **Definition of optimality**

iii. (2pt) **How to change any optimal solution X^* to greedy solution X iteratively**

iv. (2pt) **Why such change doesn't make X^* worse**

6. (9 points) Conclusion about the variation of shortest path problems

We have a directed weighted graph $G = (V, E)$ that $|V| = n$ and $|E| = \Theta(n)$ (it is a sparse graph).

For each problem, select the **most precise option** that describes the best algorithm to solve this problem.

A. Can be solved in $O(n)$ time.

B. Can be solved in $O(n \log n)$ time.

C. Can be solved in $O(n^2)$ time.

D. Can be solved in $O(n^2 \log n)$ time.

E. Can be solved in $O(n^3)$ time.

F. Can be solved, but no polynomial solution so far.

G. Can not be solved, because it may not exist.

(a) (1') Find the shortest path between two given vertices. Edge weights are nonnegative.

◯ A ◯ B ◯ C ◯ D ◯ E ◯ F ◯ G

(b) (1') Find the shortest path between two given vertices. Edge weights could be negative, but there are no negative cycles.

◯ A ◯ B ◯ C ◯ D ◯ E ◯ F ◯ G

(c) (1') Find the shortest path between two given vertices. Edge weights could be negative, and there could exist negative cycles.

◯ A ◯ B ◯ C ◯ D ◯ E ◯ F ◯ G

(d) (1') Find the shortest path between two given vertices. Edge weights are nonnegative and **identical**.

◯ A ◯ B ◯ C ◯ D ◯ E ◯ F ◯ G

(e) (1') Find the shortest path between two given vertices. The graph is a **DAG**.

◯ A ◯ B ◯ C ◯ D ◯ E ◯ F ◯ G

(f) (1') Find the shortest path between **every pair** of vertices. Edge weights are nonnegative.

◯ A ◯ B ◯ C ◯ D ◯ E ◯ F ◯ G

(g) (1') Find the shortest **simple path** between two given vertices. Edge weights are nonnegative.

◯ A ◯ B ◯ C ◯ D ◯ E ◯ F ◯ G

(h) (1') Find the shortest **simple path** between two given vertices. Edge weights could be negative, but there are no negative cycles.

◯ A ◯ B ◯ C ◯ D ◯ E ◯ F ◯ G

(i) (1') Find the shortest **simple path** between two given vertices. Edge weights could be negative, and there could exist negative cycles.

◯ A ◯ B ◯ C ◯ D ◯ E ◯ F ◯ G

7. (10 points) Gokemon Po

Bob is playing "Gokemon Po", an augmented reality game where he needs to catch n monsters located at specific places in his town. The monsters must be caught in a specific sequence: Bob can only catch monster m_i after catching all previous monsters m_j where $j < i$.

To catch a monster m_i, Bob has two options:

- Buy the monster in the game for c_i dollars.
- Catch it for free at its location.

If Bob is not already at the monster's location, he must pay a ride-share service to transport him there. The minimum cost to travel from the location of monster m_i to m_j is given by $s(i, j)$.

The task is to develop an $O(n^2)$-time algorithm to find the minimum total cost Bob needs to spend to catch all n monsters, starting from the location of the first monster, m_1.

(a) (2') How will you define the sub-problems?

(b) (5') Describe and justify the Bellman Equation for computing the solution to sub-problems.

(c) (2') What is the solution (minimum total cost) in terms of your sub-problems?

(d) (1') What is the runtime complexity of your algorithm? (answer in $\Theta(\cdot)$ and in the most simplified form)

8. (10 points) A world with SAT

The goal of this problem is to decide which type of k-SAT problem is in P or NP-Complete.

Recall a **SAT formula** is a logical formula in conjunctive normal form (CNF). Specifically, a **k-SAT formula** ϕ is a conjunction (AND) of clauses $C_1, C_2, \ldots, C_m$, and each clause C_i is a disjunction (OR) of k literals (variables or their negations) i.e. from $X \cup \{\neg x_1, \neg x_2, \ldots, \neg x_n\}$.

k-SAT: Given a set of Boolean variables $X = \{x_1, x_2, \ldots, x_n\}$ and a **k-SAT formula** ϕ on X, determine whether there exists at least one truth assignment τ that makes the formula ϕ evaluate to true. The yes-instance of k-SAT is:

$$\text{k-SAT} = \left\{ \langle\phi\rangle \middle| \begin{array}{l} \phi \text{ is a } k\text{-SAT formula with variables } X = x_1, x_2, \ldots, x_n \\ \text{and clauses } C_1, C_2, \ldots, C_m \text{ such that there exists a truth} \\ \text{assignment } \tau : X \to \{\text{true}, \text{false}\} \text{ such that } \tau(\phi) = \text{true}. \end{array} \right\}$$

Notation: The clause of the original question is denoted as C_i, and the jth literal in the ith clause can be denoted as $l_{i,j}$. Solve the problems below:

(a) (2') Prove $\forall k \geq 2$, k-SAT is in NP by giving out **polynomial-size certificate** and **polynomial-time certifier**.

(b) (2') If (k+1)-SAT $\in$ NP-Complete($k \geq 3$), show that k-SAT $\in$ NP-Complete by giving your polynomial-time many-to-one reduction f.
Hint: Consider partitioning the (k+1)-SAT clauses into parts and introduce new variables to connect them.

(c) (2') Prove that x is a *yes*-instance of (k+1)-SAT $\Leftrightarrow$ $f(x)$ is a *yes*-instance of k-SAT.

Hint for (d) and (e): 2-SAT $\in$ P, while 3-SAT $\in$ NP-Complete.

(d) (2') Briefly show whether your reduction will hold when $k = 2$ and why.

(e) (2') Here is a polynomial-time many-to-one reduction from k-SAT to (k+1)-SAT .

Given a k-SAT formula, for every k-SAT clause $C_i = l_{i,1} \vee \ldots \vee l_{i,k}$, we transform it into two (k+1)-SAT clauses with a new variable x since the following boolean formula holds:

$$(l_{i,1} \vee \ldots \vee l_{i,k}) \Leftrightarrow (l_{i,1} \vee \ldots \vee l_{i,k} \vee x) \wedge (l_{i,1} \vee \ldots \vee l_{i,k} \vee \neg x)$$

The reduction is for any k-SAT yes-instance $C_1 \wedge C_2 \wedge \ldots \wedge C_n$, it can transformed into a (k+1)-SAT yes-instance $D_1 \wedge E_1 \wedge \ldots \wedge D_n \wedge E_n$ with a new variable x, where $D_i = (C_i \vee x) = (l_{i,1} \vee \ldots \vee l_{i,k} \vee x), E_i = (C_i \vee \neg x) = (l_{i,1} \vee \ldots \vee l_{i,k} \vee \neg x)$.

Briefly explain whether it will hold when $k = 2$ and why.

THIS PAGE INTENTIONALLY LEFT BLANK.

Label it clearly when you need to continue your solution on the scratch page.

THIS PAGE INTENTIONALLY LEFT BLANK.

THIS PAGE INTENTIONALLY LEFT BLANK.

THIS PAGE INTENTIONALLY LEFT BLANK.

THIS PAGE INTENTIONALLY LEFT BLANK.

THIS PAGE INTENTIONALLY LEFT BLANK.