(CS1.301) Algorithm Analysis and Design (Monsoon 2024)

End of Semester Examination

Alloted time: 180 minutes

Total marks: 90

Please read the instructions VERY carefully.

- There are a total of 9 questions with varying credit, spanning pages 2 to 7. Please ensure all pages are included in your copy. The total credit for each question is indicated at the top-right corner, and the credit for each sub-part (if applicable) is clearly mentioned at the beginning for clarity.
- Discussions amongst the students are not allowed. No electronic devices (including smart watches) nor notes/books of any kind are allowed. Any dishonesty shall be penalized heavily.
- Any theorem/lemma/claim/fact that was proved in the class can be used without proof in the exam, only by explicitly writing its statement, and a clear remark that it was chosen from the class notes.
- The questions have been designed to be unambiguous, and queries will not be addressed during the examination. If you encounter any ambiguity, please note it in your answer script and proceed accordingly. Answers based on a misinterpretation of the questions may not be awarded credit.
- Be clear in your arguments. Partial marking is available for every question but vague arguments shall not be given any credit.
- You do not have to write a pseudocode unless explicitly asked.
- Analysis of running times, and proofs of correctness need to be done unless explicitly asked not to.

Question 1 [5 marks]

Let $T = (V, E)$ be a spanning tree on n vertices. Let U be the subset of vertices V, whose degree is at least 3. That is, $U = \{u \in V \mid \deg(u) \geqslant 3\}$. Then show that $|U| \leqslant \frac{|V|}{2}$.

Question 2 [5 marks]

Let $G = (V, E)$ be a directed graph. Let $G_{rev} = (V, E_{rev})$ be a graph obtained from G by reversing the edge directions.

$$E_{rev} = \{(v, u) \mid (u, v) \in E\}.$$

Show that G and G_{rev} have the same strongly connected components.

Question 3 [8 marks]

Given a sorted array of distinct integers $A[1, n]$, we want to find out whether there is an index i for which $A[i] = i$. Give an algorithm that runs in time $O(\log n)$.

Question 4 [8 marks]

Let $G = (V, E)$ be a connected graph with n vertices, m edges, and positive edge costs that you may assume are all distinct. Let $T = (V, E')$ be a spanning tree of G; we define the bottleneck edge of T to be the edge of T with the greatest cost. A spanning tree T of G is a minimum-bottleneck spanning tree if there is no spanning tree T' of G with a cheaper bottleneck edge.

(a) [5 marks] Show that every minimum spanning tree of G is always a minimum-bottleneck tree of G.

(b) [3 marks] Is every minimum-bottleneck tree of G a minimum spanning tree of G? Prove or give a counterexample.

Question 5 [12 marks]

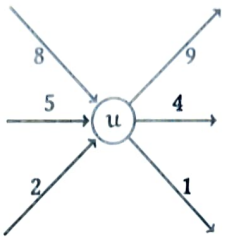
Algorithm 1: Monotonically Increasing Dijkstra's Algorithm

Input: Graph $G = (V, E)$ with edge weights $w(u, v) \geqslant 0$, source vertex $s \in V$
Output: Monotonically increasing shortest path distances from s to all other vertices in V

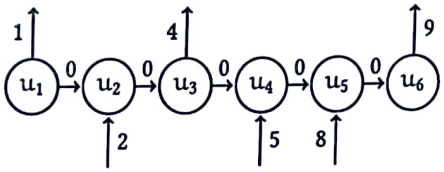
```
Function MonotonicIncreasingDijkstra(G, s):
    foreach v ∈ V do
        dist[v] ← ∞;
        prev[v] ← null;
        lastEdgeWeight[v] ← −∞;
    end
    dist[s] ← 0;
    lastEdgeWeight[s] ← 0;
    Initialize priority queue Q with all vertices, keyed by dist[v];
    while Q is not empty do
        u ← Extract-Min(Q);
        foreach v ∈ Neighbors(u) do
            if dist[u] + w(u, v) < dist[v] and w(u, v) > lastEdgeWeight[u] then
                dist[v] ← dist[u] + w(u, v);
                prev[v] ← u;
                lastEdgeWeight[v] ← w(u, v);
                Decrease-Key(Q, v, dist[v]);
            end
        end
    end
```

A path is monotonically increasing if the weight of every edge (considered in sequence) on that path is strictly increasing. Given an edge-weighted directed graph $G = (V, E)$, your task is to find a monotonically increasing shortest path from a vertex s to every other vertex (if it exists).

(a) [6 marks] Person A suggests Algorithm 1 (as described above). Do you think this algorithm is correct and gives us the correct answer? Argue formally/mathematically.



(a) Vertex u with its incoming and outgoing edges in G



(b) The copies of vertex u in G′ are denoted as $u_1, u_2, \ldots, u_6$, with new zero-weight edges added between consecutive copies. The edges incident on u in G are distributed among these new vertices in G′.

Figure 1: Example of construction of G′, vertex by vertex

(b) [6 marks] Person B, without knowledge of Person A's suggestion, proposes the following procedure for transforming a graph $G = (V, E)$ into a new graph $G' = (V', E')$:

- Create vertex copies: For each vertex $u \in V$ with degree d_u, create d_u copies of u in V', named $u_1, u_2, \ldots, u_{d_u}$.
- Sort edges: List the edges incident to u as $e_1, e_2, \ldots, e_{d_u}$, sorted by increasing weights (assume distinct edge weights).
- Add zero-weight edges: Add edges with weight 0 between consecutive copies of u:

$$(u_1, u_2), (u_2, u_3), \ldots, (u_{d_u-1}, u_{d_u}).$$

- Map original edges: For each edge e_j (where $1 \leqslant j \leqslant d_u$), connect it to the corresponding copy u_j in G′:
  - If e_j is an incoming edge to u in G, make it incoming to u_j in G′.
  - If e_j is an outgoing edge from u in G, make it outgoing from u_j in G′.

This process is repeated for all vertices $u \in V$.

Key observation: Edges in G appear in G′ as connections between specific vertex copies (e.g., (u_k, v_ℓ)).

Person B argues that finding the shortest path in G′ will correspond to a *monotonically increasing shortest path* in G (in terms of edge weights).

The task: Determine whether Person B's claim is mathematically correct.

Remark: To efficiently handle the zero-weight edges in G', we will use an adjacency matrix and a separate matrix for edge weights. In this question, zero weight edge does not mean the absence of an edge in the adjaceny matrix.

Question 6 [12 marks]

Suppose we are given a set U of objects labeled $p_1, p_2, \ldots, p_n$. For each pair p_i and p_j, we have a numerical distance $d(p_i, p_j)$. We futher have the property that for all $1 \leqslant i \leqslant n$, $d(p_i, p_i) = 0$ and for all $1 \leqslant i \neq j \leqslant n$, $d(p_j, p_i) = d(p_i, p_j) > 0$.

For a given parameter k as input, k-clustering of U is a partition of U into k nonempty sets $C_1, C_2, \ldots, C_k$. Spacing of a k-clustering is defined to be the minimum distance between any pair of points lying in different clusters. That is,

$$\text{Spacing}(C_1, C_2, \ldots, C_k) = \min_{1 \leqslant u \neq v \leqslant k} \{\min\{d(p, p') \mid p \in C_u \text{ and } p' \in C_v\}\}.$$

Given that we want points in different clusters to be far apart from one another, a natural goal is to seek the k-clustering with the maximum possible spacing. In other words, we want to find the partition of U into k non-empty sets that maximizes the following expression.

$$\max_{U = C_1 \sqcup C_2 \sqcup \ldots \sqcup C_k} \{\text{Spacing}(C_1, C_2, \ldots, C_k)\}$$

Can Kruskal's algorithm be used by stopping it just before it adds its last $k-1$ edges, to efficiently find a k-clustering that has maximum spacing? Argue.

Question 7 [12 marks]

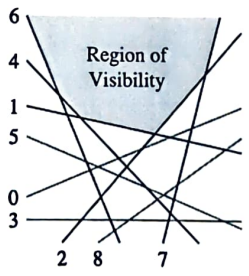


Figure 2: Example: Lines 0,3,5,8 are hidden but all other lines are visible.

Consider n lines $L_0, L_2, \ldots, L_{n-1}$ in the 2D plane as follows. Each line L_i is specified by the equation $y = m_i x + c_i$. Assume the input is given as $(m_0, c_0), (m_1, c_1), \ldots, (m_{n-1}, c_{n-1})$.

We want to compute the set of lines which will be visible when one sees from $y = +\infty$. Let us elaborate. At an x-coordinate x_0, that line will be visible which has maximum the y coordinate among all lines. That is, the line which maximizes $m_i x_0 + c_i$. See Figure 2 above for more details.

A line is called hidden if it is not visible at any x-coordinate. **Design an $O(n \log n)$ time algorithm to find the set of visible (which are not hidden) lines.** Assume that no three lines intersect at same point, and no two lines are parallel.
Hint: Given n lines, split the set of lines into two sets, considering the median (with respect to slope) line as separator. Obtain the visible lines (and the boundary of region of visibility) of both the sets of lines and merge them in $O(n)$ time. Please explain the merge procedure in a short and crisp way.

Question 8 [14 marks]

Damerau-Levenshtein distance between two strings (over any alphabet) is defined as the minimum number of valid operations required to convert one string into another, where the valid operations are

- deletion of a character,
- insertion of a character,
- substituting one character with another, and
- swapping of two adjacent characters.

There is no restriction on the order in which these operations are performed. Note that this is different from the problem that we discussed in the class.
For example, the two strings **from** and **north** have distance 4, as shown below.

from $\leftrightarrow$ **form** $\leftrightarrow$ **norm** $\leftrightarrow$ **norh** $\leftrightarrow$ **north**

Yet another example is the following.

gifts	$\mapsto$	pgifts	(insertion of 'p')
pgifts	$\mapsto$	prgifts	(insertion of 'r')
prgifts	$\mapsto$	proifts	(substitution of 'g' to 'o')
proifts	$\mapsto$	profits	(transposition of 'if' to 'fi')
profits	$\mapsto$	profit	(deletion of 's')

We want to design a dynamic programming algorithm to compute the Damerau-Levenshtein distance between two given strings. Let the two given strings be $\mathbf{a} = a_1 a_2 \dots a_n$ and $\mathbf{b} = b_1 b_2 \dots b_m$, where each a_i and b_j belong to an alphabet Σ.
Let $d(i,j)$ denote the distance between the substrings $a_1 a_2 \dots a_i$ and $b_1 b_2 \dots b_j$, for $0 \leqslant i \leqslant n$ and $0 \leqslant j \leqslant m$ ($i = 0$ or $j = 0$ just means empty substring). We will build an $(n+1) \times (m+1)$ table D whose (i,j) entry is supposed to be $d(i,j)$ for $0 \leqslant i \leqslant n$ and $0 \leqslant j \leqslant m$. Finally, $D(n,m)$ will be our answer.

Set $D(0,i) = i$ and $D(j,0) = j$ for each i and j. To fill the rest of the table we use the following equation:

$$D(i,j) = \min \begin{cases} 0 & \text{if } i = j = 0, \\ D(i-1,j)+1 & \text{if } i > 0, \\ D(i,j-1)+1 & \text{if } j > 0, \\ D(i-1,j-1) & \text{if } i,j > 0 \text{ and } a_i = b_j, \\ D(i-1,j-1)+1 & \text{if } i,j > 0 \text{ and } a_i \neq b_j, \\ D(i-2,j-2)+1_{(a_i \neq b_j)} & \text{if } i,j > 1 \text{ and } a_i = b_{j-1} \text{ and } a_{i-1} = b_j, \end{cases}$$

where $1_{(a_i \neq b_j)}$ is the indicator function which is equal to 0 when $a_i = b_j$ and equal to 1 otherwise. Each recursive call matches one of the cases covered by the Damerau–Levenshtein distance:

- $D(i-1,j)+1$ corresponds to a deletion (from **a** to **b**),
- $D(i,j-1)+1$ corresponds to an insertion (from **a** to **b**),
- $D(i-1,j-1)$ corresponds to a match of the symbols a_i and b_j,
- $D(i-1,j-1)+1$ corresponds to a mismatch of the symbols a_i and b_j,
- $D(i-2,j-2)+1_{(a_i \neq b_j)}$ corresponds to a transposition between two successive symbols when $a_i = b_{j-1}$ and $a_{i-1} = b_j$.

(a) [4 marks] What is the time complexity and space complexity of the afore mentioned procedure.

(b) [6 marks] Construct the matrix D for the words **a** = Sunday and **b** = Saturday and compute the Damerau–Levenshtein distance between these words.

		S	a	t	u	r	d	a	y
	0	1	2	3	4	5	6	7	8
S	1								
u	2								
n	3								
d	4								
a	5								
y	6								

(c) [4 marks] Argue about the correctness or incorrectness of the provided algorithm.

Question 9 **[14 marks]**

Consider the problem of TA (teaching assistants) allocation to courses. A TA may be suitable for a subset of courses and not suitable for other courses. Suppose we are given the suitability information by a $m \times n$ matrix with $\{0,1\}$ entries, where m is the total number of TAs and n is the number of courses. We are also given the required number of TAs for each course, say, $r_1, r_2, \ldots, r_n$. Naturally, a TA can only be assigned to a course for which they are suitable. And a TA can be assigned to at most one course. We want to find out whether it's possible to meet the TA requirement for every course.

For example, suppose for course X the suitable TAs are $\{1, 2, 3\}$. And for course Y , the suitable TAs are $\{2, 3\}$. Suppose X and Y both require two TAs each. Then it is not possible to meet the requirement for every course.

We plan to solve this problem using the network flow algorithm as a subroutine. For any given input for TA allocation (suitability matrix and r_i's), we want to first build an appropriate flow network. We should design the network in a way that the value of the maximum flow should tell us whether it's possible to meet the TA requirement for every course.

Building the flow network: We create one vertex for every TA and one vertex for every course. We add a (directed) edge from a course to a TA if and only if that TA is suitable for that course. Additionally, we also create a source vertex s and a sink vertex t.

(a) [2 marks] Where will you put the outgoing edges from s and what will be their capacities?

(b) [2 marks] Where will you put the incoming edges to t and what will be their capacities?

(c) [2 marks] What should be the capacity of the edges which are going from a course to a TA?

(d) [2 marks] If it is possible to meet the TA requirement for every course, then what will be the maximum flow in your network?

(e) [2 marks] If it is not possible to meet the TA requirement for every course, then what can you say about the maximum flow in your network?

(f) [4 marks] Prove that when it is not possible to meet the TA requirement for every course, then there must be a subset C of courses such that the total number of TAs who are suitable for some course in C is less than the required numbe$\sum_{i \in C} r_i$. You can directly use the max flow min cut theorem.

Hint: Say U is a minimum s-t cut (U contains s). Consider the set of courses in U.

End of questions