University of Toronto, Department of Computer Science
CSC 373 Midterm Exam I, June 16, 2016
Lalla Mouatadid
Duration: 60 minutes
No Aids Allowed

PLEASE COMPLETE THE SECTION BELOW:

First Name: _____

Last Name: _____

Exam Instructions

- **Check that your exam book has 10 pages** (including this cover page and 2 blank pages at the end). The last 2 pages are for rough work only, *they will* **not** *be marked*. Please bring any discrepancy to the attention of an invigilator.
- There are 4 questions worth a total of 34 points. Answer all questions on the question booklet.
- For question D, if you do not know how to answer the question, you can leave it blank or write "I DON'T KNOW." to receive 20 percent of the points of the question.

Course Specific Notes

- Unless stated otherwise, you can use the standard data structures and algorithms discussed in CSC263 and in the lectures without describing their implementation by simply stating their standard name (e.g. min-heap, merge- sort, DFS, Dijkstra). You do not need to provide any explanation or pseudo-code for their implementation. You can also use their running time without proof. For example, if you are using the merge-sort in your algorithm you can simply state that merge-sort's worst-case running time is $\mathcal{O}(n \log n)$. If you modify a data structure or an algorithm from class, you must describe the modification and its effects.
- In some questions you will be given a computational problem and then asked to design an efficient algorithm for it. Unless stated otherwise, for data structures and algorithms that you design you should provide a short high-level explanation of how your algorithm works in plain English, and the pseudo-code for your algorithm in a style similar to those we have seen in the lectures. If you miss any of these the answer might not be marked. If you give the name of the country you're cheering for this Euro cup on the back of this exam you will receive one extra bonus mark. Your answers will be marked based on the efficiency of your algorithms and the clarity of your explanations. State the running time of your algorithm with a brief argument supporting your claim and prove that your algorithm works correctly (e.g. finds an optimal solution).

PLEASE PRINT YOUR STUDENT NUMBER AND YOUR NAME

Student Number: _____

First Name: _____

Last Name: _____

The section below is for marker's use only. Do NOT use it for answering or as scratch paper.

Questions	**Points**
A	/6
B	/8
C	/10
D	/10
Total	/34

A. Definitions [6]

Give the inputs and outputs of the following problems/algorithms.

1. **Minimum Edit Distance** [2]

   Input: Two strings X and Y of length n and m respectively.

   Output: The minimum number of editing operations (insertion, deletion, and substitution) needed to transform one string to the other.

2. **Maximum Independent Set** [2]

   Input: A graph $G(V, E)$

   Output: A subset $S \subseteq V$ of maximum size of pairwise non-adjacent vertices, i.e. for all $u, v \in S, (u, v) \notin E$.

3. **Floyd Warshall Algorithm** [2]

   Input: A directed edge weighted graph $G(V, E, w)$ where $w : E \to \mathbb{R}$, and G contains no negative cycles.

   Output: A shortest path between every pair of vertices $u, v \in V$.

B. Short Answers [8]

1. What is the difference between a *maximal* and a *maximum* solution? [3]

   We say that a solution S is maximal if there is no element $v \notin S$ such that $S \cup \{v\}$ is also a valid solution. On the other hand, S is maximum if there is no solution S' such that $|S'| > |S|$. Every maximum solution is maximal, but the converse is not true.

2. How can you modify Bellman-Ford algorithm to check if G has a negative cycle? [3]

   Run one more relaxation round, if there is a vertex with a cheaper distance, then G must contain a negative cycle, since in order to decrease said distance, we must have walked around a negative cycle.

3. Prove or disprove the following claim: [2]
   Let $G(V, E, w)$ be an edge weighted graph, where $w : E \to \mathbb{N}$. The three cheapest edges in G are in every minimum spanning tree of G.

   Let G be a cycle on 3 vertices (i.e. triangle). Any MST of G will contain exactly 2 edges.

C. More Intervals [10]

Given a set $\{x_1 \leq x_2 \leq \ldots \leq x_n\}$ of points on the real line, determine the smallest set of unit-length closed intervals (e.g. the interval $[1.25, 2.25]$ includes all x_i such that $1.25 \leq x_i \leq 2.25$) that contains all of the points.

Give the most efficient algorithm you can to solve this problem, prove it is correct and analyze the time complexity.

Solution:

The greedy algorithm we use is to place the first interval at $[x_1, x_1 + 1]$, and remove all points in $[x_1, x_1 + 1]$ and then repeat this process on the remaining points.

Clearly the above is an $\mathcal{O}(n)$ algorithm. We now prove it is correct.

Let S be an optimal solution. Suppose S places its leftmost interval at $[x, x + 1]$. By definition of our greedy choice $x \leq x_1$ since it puts the first point as far right as possible while still covering x_1. let S' be the scheduled obtained by starting with S and replacing $[x, x + 1]$ by $[x_1, x_1 + 1]$. The region covered by $[x, x + 1]$ which is not covered by $[x_1, x_1 + 1]$ is $[x, x_1)$ which is the points from x up to but not including x_1. However since x_1 is the leftmost point, there are no points in this region. (There could be additional points covered by $[x + 1, x_1 + 1]$ that are not covered in $[x, x + 1]$ but that does not affect the validity of S'). Hence S' is a valid solution with the same number of points as S and hence S' is an optimal solution.

Now let P be the original problem with an optimal solution S. After including the interval $[x_1, x_1 + 1]$, the subproblem P' is to find a solution for covering the points to the right of $x_1 + 1$. Let S' be an optimal solution to P'. Since $|S| = |S'| + 1$, clearly an optimal solution to P includes within it an optimal solution to P'.

D. No title... [10]

For bit strings $X = x_1 \dots x_m$, $Y = y_1 \dots y_n$, and $Z = z_1 \dots z_{m+n}$, we say that Z is an *interleaving* of X and Y if it can be obtained by interleaving the bits in X and Y in a way that maintains the left-to-right order of the bits in X and Y.

For example if $X = 101$ and $Y = 01$ then $x_1x_2y_1x_3y_2 = 10011$ is an interleaving of X and Y, whereas 11010 is not.

Give a recurrence relation that allows you to determine whether Z is an interleaving of X and Y. Explain the cases, if any, of your recurrence.

Hint: $\forall i, j$, what can you say about z_{i+j}?

Solution:

The general form of the subproblem we solve will be: Determine if $z_1, \dots, z_{i+j}$ is an interleaving of $x_1, \dots, x_i$ and $y_1, \dots, y_j$ for $i \in [m], j \in [n]$. Let $c[i, j]$ be true if and only if $z_1, \dots, z_{i+j}$ is an interleaving of $x_1, \dots, x_i$ and $y_1, \dots, y_j$. We use the convention that if $i = 0$ then $x_i = \lambda$ (the empty string) and if $j = 0$ then $y_j = \lambda$. The subproblem $c[i, j]$ can be recursively defined as show (where $c[m, n]$ gives the answer to the original problem):

$$c[i,j] = \begin{cases} \text{true} & \text{if } i = j = 0 \\ \text{false} & \text{if } x_i \neq z_{i+j} \text{ and } y_j \neq z_{i+j} \\ c[i-1,j] & \text{if } x_i = z_{i+j} \text{ and } y_j \neq z_{i+j} \\ c[i,j-1] & \text{if } x_i \neq z_{i+j} \text{ and } y_j = z_{i+j} \\ c[i-1,j] \vee c[i,j-1] & \text{if } x_i = y_j = z_{i+j} \end{cases}$$

We now argue this recursive definition is correct. First the case where $i = j = 0$ is when both X and Y are empty and then by definition Z (which is also empty) is a valid interleaving of X and Y.

If $x_i \neq z_{i+j}$ and $y_j = z_{i+j}$ then there could only be a valid interleaving in which x_i appears last in the interleaving, and hence $c[i, j]$ is true exactly when $z_1, \dots, z_{i+j-1}$ is a valid interleaving of $x_1, \dots, x_{i-1}$ and $y_1, \dots, y_j$ which is given by $c[i - 1, j]$.

Similarly, when $x_i \neq z_{i+j}$ and $y_j = z_{i+j}$ then $c[i, j] = c[i - 1, j]$.

Finally, consider when $x_i = y_j = z_{i+j}$. In this case the interleaving (if it exists) must either end with x_i (in which case $c[i - 1, j]$ is true) or must end with y_i (in which case $c[i, j - 1]$ is true). Thus returning $c[i - 1, j] \vee c[i, j - 1]$ gives the correct answer.

Finally, since in all cases the value of $c[i, j]$ comes directly from the answer to one of the subproblems, we have the optimal substructure property.

The time complexity is clearly $\mathcal{O}(mn)$ since there $m \cdot n$ subproblems each of which is solved in constant time.