UNIVERSITY OF TORONTO
Department of Computer Science

Feb 28, 2014 Midterm Exam

CSC373H1S

Robert Robere

Duration - 1 hour

No Aids Allowed.

PLEASE COMPLETE THE SECTION BELOW AND THE SECTION BEHIND THIS PAGE:

First Name: _____

Last Name: _____

Exam Instructions

- **Check that your exam book has 8 pages** (including this cover page and 0 blank pages at the end). The last 0 pages are for rough work only, *they will* **not** *be marked*. Please bring any discrepancy to the attention of an invigilator.
- There are 5 questions worth a total of 60 points. Answer all questions on the question booklet.
- For Questions 4 and 5, if you do not know how to answer any part of the question then you can leave that part 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 $O(n \log n)$. 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 draw a picture of your favorite animal (real or fake) 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.

Question #	Score/Points
1	/ 12
2	/ 4
3	/ 4
4	/ 20
5	/ 20
Total	/ 60

1. **Problem Definitions** [12]

   Define the inputs and outputs of each of the following problems. **Scheme: 1.5 marks for each correct input and output**.

   a. **Single-Source Shortest Paths** [3]

   Input:

   An edge-weighted graph $G = (V, E)$, and a single node $s \in V$.

   Output:

   The length of the shortest path from s to every other vertex in V.

   b. **Matrix Chain Multiplication** [3]

   Input:

   A list of n matrices $M_1, M_2, \ldots, M_n$, where matrix M_i has dimension $(i, i+1)$.

   Output:

   The minimum number of arithmetic operations needed to multiply out the sequence $M_1 \cdot M_2 \cdots M_n$ using the naive matrix multiplication algorithm.

   c. **Weighted Interval Scheduling** [3]

   Input:

   A list $\mathcal{I}$ of n intervals on the real line, and a weight function $w : \mathcal{I} \to \mathbb{R}$.

   Output:

   A subset $S \subseteq \mathcal{I}$ of intervals such that every pair of distinct intervals $I, J \in \mathcal{I}$ do not intersect and the sum of the weights of the intervals in S is maximized.

   d. **Interval Colouring** [3]

   Input:

   A list $\mathcal{I}$ of n intervals on the real line.

   Output:

   A coloring $f : \mathcal{I} \to \{1, 2, \ldots, k\}$ of the intervals in $\mathcal{I}$ such that if two distinct intervals $I, J \in \mathcal{I}$ intersect then $f(I) \neq f(J)$, and the number of distinct colours k is minimized.

2. **Algorithm Definitions** [4]

   a. State the names of two algorithms that solve the minimum-weight spanning tree problem. **Scheme: 1 mark each** [2]

   Prim's algorithm and Kruskal's algorithm.

   b. What distinguishes each of these algorithms from one another? **Scheme: 1 mark for adaptive/non-adaptive, and 1 mark for vertices/edges.** [2]

   Prim's algorithm is an *adaptive* greedy algorithm over the *vertices* of the graph. It builds up a contiguous minimum spanning tree by beginning from some arbitrarily chosen root vertex s, and then adaptively re-ordering the other vertices in the graph according to the minimum distance to a vertex in the tree. Kruskal's algorithm is a *non-adaptive* greedy algorithm over the *edges* of the graph. At the beginning of execution, it permutes the edges of the graph into non-decreasing order by weight, and then iteratively proceeds through the edges, adding them to a tree whenever they do not create a cycle.

3. **Algorithmic Paradigms** [4]

   As stated in class and in the lecture notes, what are the two distinguishing features (or "hallmarks") of greedy algorithms? **Scheme: 2 marks per feature.**

   Irrevocable Decisions When a greedy algorithm makes a decision about any of the items in the input, it never goes back and changes its decision.

   Greedy Property Greedy algorithms implicitly assume that making a greedy choice at each step (i.e. finding a *local* optimum) will eventually lead to an optimal solution (a *global* optimum).

4. **Show Dog! (Bad Dog! Smelly Dog!)** [20]

Dogs – they're great. You love your dog so much that if you were ever separated, you would simply fall apart. Since your dog is simply the best dog, you have decided to enter him in the yearly UltraDog Special (the most prestigious of all dog shows, featuring the top dogs from every corner of the globe). Since the meet is only n days away, you have decided to put your dog on a rigourous training regimen. However, you don't want to over-train your dog — a tired dog is not a good dog. Especially not a good show dog.

So, starting from now (Day 1) until the UDS, you can choose each day to either train your dog or let your dog rest (but not both). However, the ultimate benefit that your dog receives from either training or taking a rest will depend *only* on the day chosen. Formally, on the ith day ($1 \leq i \leq n$), training your dog will give it a benefit of t_i, while letting your dog rest will give a benefit of r_i. Resting takes only a day (leaving your dog ready and able to train — or rest — again tomorrow). On the other hand, training takes *two* days — in other words, *you can not train or rest on the day after you choose to train your dog*.

To formalize this, define a *training regimen* to be a map $f : \{1, 2, \ldots, n\} \to \{T, R, *\}$, where for each i, $f(i) = T$ if you decide to train your dog on day i, $f(i) = *$ if you are continuing the training from day $i - 1$, and $f(i) = R$ if you decide to let your dog rest on day i.

The *benefit* of a training regimen is the sum of the benefits of the training days and the rest days:

$$B = \sum_{i \mid f(i)=T} t_i + \sum_{i \mid f(i)=R} r_i.$$

Note that if you train on day i (and thus lose day $i + 1$) you *only* get the benefit t_i, not t_i and t_{i+1}. Moreover, due to losing a day after training, you are not allowed to train your dog on day n.

Input: Two lists of integers (positive or negative) $T = \{t_1, t_2, \ldots, t_{n-1}\}$, $R = \{r_1, r_2 \ldots, r_n\}$, where t_i represents the benefit to your dog if you train on day i, while r_i represents the benefit to your dog if you rest on day i.

Output: The maximum achievable benefit of any training regimen over the given training and resting days. (Note that you do not need to reconstruct the best regimen — you only need to return its benefit.)

Give a dynamic programming algorithm for this problem. To receive full marks it must run in $O(n)$ time.

a. Give a high-level description of each cell in the recurrence used by your algorithm. **Scheme: 4 marks for the correct definition.** [4]

Let $x \in \{r, t\}$ (standing for "resting" and "training", respectively).

$M[i, x]$:= the maximum benefit of a training regimen on the first i days if you perform x on day i.

b. Formally define the recurrence relation for the recurrence you gave in Part (a). **Scheme: 2 marks for base cases, 4 marks for correct recursive case.** [6]

Base cases: For each $x \in \{r, t\}$, $M[0, x] = 0$. $M[1, r] = r_1, M[1, t] = t_1$. If $i \geq 2$ then

$$M[i, x] = x_i + \max\{M[i - 1, r], M[i - 2, t]\}$$

c. Give a dynamic programming algorithm implementing the recurrence relation. Be sure to give a brief, high-level description of your algorithm, a pseudocode implementation, and an argument about the running time. See Algorithm 1 for the algorithm definition. **Scheme: 3 marks for a correct algorithm, 2 marks for running time with argument.** [5]

Our algorithm implements the recursive definition presented in part (b), which computes for each day i the optimal training regimen up to that day.

The algorithm clearly runs in $O(n)$ time due to the two single-nested for-loops.

Algorithm 1: Dogs Training

Let M be a $n \times 2$ array;
$M[0, r] = M[0, t] = 0$;
$M[1, r] = r_1$;
$M[1, t] = t_1$;
/* Fill the array */
for $i = 2, 3, \ldots, n-1$ **do**
  $tmp = \max\{M[i-1, r], M[i-2, t]\}$;
  $M[i, r] = tmp + r_i$;
  $M[i, t] = tmp + t_i$;
end
$M[n, r] = \max\{M[n-1, r], M[n-2, t]\} + r_n$;
return $\max\{M[n, r], M[n-1, t]\}$

d. Prove that your recurrence relation from Part (b) is correct. **Scheme: 5 marks for a correct proof, partial marks removed for lack of precision, errors.** [5]

The base cases are clearly correct, so we continue by induction: suppose that $M[j, x]$ stores the maximum benefit of any training regimen ending on day j, where we perform x on day j, for $j < i$. We prove that the definitions of $M[i, r]$ and $M[i, t]$ are correct. Consider an optimal training regimen O ending on day i, where we choose to rest (or train) on day i. The training regimen O therefore consists of a sequence of training/resting choices

$$O = \{x_1, x_2, \cdots, x_{i-1}, x\}.$$

Let $O' = O \setminus \{x\}$. Then O' is a training regimen either ending on day $i-1$, where we rest on day $i-1$, or ending on day $i-2$ where we train on day $i-2$. By the inductive hypothesis, then, the benefit from O' is exactly $\max\{M[i-1, r], M[i-2, t]\}$. It follows that the value of O is $\max\{M[i-1, r], M[i-2, t]\} + x_i$, where $x \in \{t, r\}$ is choice of training or resting on day i. If we rest on day i this proves the correctness of $M[i, r]$, and if we train on day i this proves the correctness of $M[i, t]$.

5. **Extreme Snowbearding** [20]

You are an extreme snowbearder — a very particular breed of daredevil. Extreme snowbearding is a dangerous sport in which competitors freeze large chunks of ice to their face and then sculpt the ice into fabulous and garish beards using various saws, picks, knives, and wedges. The sculpted beards are then rated by a panel of m judges. Each judge gives an (integer) score between -10 to 10; a score of -10 means that your beard is utterly boorish, obnoxious, and unrefined; a score of 10 is only given if several of the judges are driven immediately and irrevocably insane with joy upon the very sight of your carefully chiseled ice-hair.

Modern snowbearding competitions go like this: you are given a fixed list of n beards that you will be expected to perform for the panel of judges. You must choose in advance which of the n beards you will present, and which you will not. (For example, you may choose to present every beard, or you may choose to present no beard at all.) In round j, if you decide to present beard j then your final score will be the sum of each individual judge's score for that beard. If you decide not to present beard j, then you will receive a score of 0. Your final score after all n rounds is simply the sum of your scores from each round.

Due to the (sadly) small size of the extreme snowbearding community, you know exactly how each of the m judges are going to score each of the possible beards that you can present at this year's SuperBeard Standoff. This information is recorded in an $m \times n$ table T, where for all $1 \leq i \leq m$ and all $1 \leq j \leq n$ we define

$$T[i, j] := \text{the score given by Judge } i \text{ if you present Beard } j.$$

(Thus, for all i and j, $T[i, j]$ is an integer between -10 and 10).

For example, your input table could be the following (in the case of $m = 2$ judges and $n = 3$ beards):

	Beard 1	Beard 2	Beard 3
Judge 1	-4	2	1
Judge 2	10	1	-5

In this case, your best choice would be to present Beards 1 and 2 — presenting Beard 1 gives a score of $-4 + 10 = 6$, and presenting Beard 2 gives a score of $2 + 1 = 3$ — for a total score of 9. We formally define the problem below:

Input: Two positive integers m, n and an $m \times n$ table $T[i, j]$ as defined above.

Output: The maximum possible score obtainable from the judges at this year's SuperBeard Standoff.

a. Give a greedy algorithm solving the Snowbearding Problem. To receive full marks it must run in $O(mn)$ time. Be sure to give a brief, high-level description of your algorithm, a pseudocode implementation, and an argument about the running time. **Scheme: 3 marks for high level description, 5 marks for the algorithm, and 2 marks for the running time with argument.** [10]

Our greedy algorithm will iterate through the beards in order, and sum up the scores from each judge. If a beard will return a positive score from all of the judges then we choose to present it. Otherwise, we reject it.

This algorithm clearly runs in $O(nm)$ time: the for loop iterates n times and and an $O(m)$ time computation is needed for each sum.

b. **Scheme: 10 marks for a correct proof, marks removed for lack of precision, errors.** [10]

Prove that your greedy algorithm is optimal.

Suppose that the greedy algorithm does not output an optimal solution. Let $O \subseteq \{1, 2, \ldots, m\}$ be an optimal selection of judges, and let $G \subseteq \{1, 2, \ldots, m\}$ be the selection of judges made by the greedy algorithm. For any $i \in \{1, 2, \ldots, m\}$ let s_i be the total score obtained by presenting the ith beard. If $s_i > 0$ then $i \in G$, and if $i \notin O$ then we could add it to O and increase the value, which contradicts that

Algorithm 2: Greedy Beards

```
val = 0;
for j = 1, 2, ..., n do
    tmp = sum_{i=1}^{m} T[i, j];
    if tmp > 0 then
        val = val + tmp;
    end
end
return val
```

O is optimal. If $s_i < 0$ then $i \notin G$, and so if $i \in O$ then we could remove it from O and increase the value, again contradicting that O is optimal. Thus if G and O ever differ on a beard it follows that O is not optimal. This means that G and O cannot differ on any beard, and so G must be an optimal solution.