COMP 3711 – Design and Analysis of Algorithms
2023 Fall Semester – Written Assignment 3
Distributed: October 27, 2023
Due: November 13, 2023, 23:59

Your solution should contain

(i) your name, (ii) your student ID #, and (iii) your email address

at the top of its first page.

Some Notes:

- Please write clearly and briefly. In particular, your solutions should be written or printed on *clean* white paper with no watermarks, i.e., student society paper is not allowed.
- Please also follow the guidelines on doing your own work and avoiding plagiarism as described on the class home page. ***You must acknowledge individuals who assisted you, or sources where you found solutions.*** Failure to do so will be considered plagiarism.
- The term *Documented Pseudocode* means that your pseudocode must contain documentation, i.e., comments, inside the pseudocode, briefly explaining what each part does.
- Many questions ask you to explain things, e.g., what an algorithm is doing, why it is correct, etc. To receive full points, the explanation must also be *understandable* as well as correct.
- Please make a *copy* of your assignment before submitting it. If we can't find your submission, we will ask you to resubmit the copy.
- Submit a SOFTCOPY of your assignment to Canvas by the deadline. If your submission is a scan of a handwritten solution, make sure that it is of high enough resolution to be easily read. At least 300dpi and possibly denser.

1. (20 points) Consider the following (unrealistic) scenario. You are running a political campaign and want to collect petition signatures in the park. Your research team tells you that everyone who goes to the park visits for one of n known time intervals during the day (if they come for a particular interval, they stay for the entire time interval; the different time intervals might overlap).

   If you go to the park at any time during one of the intervals you will get the signatures of everyone there for that interval. You want to figure out a minimum sized set of times that you have to go to the park to get everyone's signature.

   In computer science such a situation is called a *covering problem* and is modelled formally as described below.

   - A real *interval* is $I = [s, f]$ where $s \leq f$
   - A real *point* x *covers* interval $I = [s, f]$ if $x \in I$, i.e., $s \leq x \leq f$.
   - Let $\mathcal{I} = \{I_1, \ldots, I_n\}$ be a set of n intervals and $A = \{a_1, a_2, \ldots a_k\}$ a set of points.
     Then "A covers $\mathcal{I}$" if every interval in $\mathcal{I}$ is covered by at least one point in A. More formally, if, for every $I_j \in \mathcal{I}$ there exists $a_i \in A$ such that $a_i \in I_j$.
   - $|A|$, the size of A, is just the number of points in A.
     A is a *minimal cover* of $\mathcal{I}$ if A is a smallest cover of $\mathcal{I}$.
     That is, for every cover A' of $\mathcal{I}$, $|A| \leq |A'|$.

   The input to this problem is the $2n$ numbers $s_1, f_1, s_2, f_2, \ldots, s_n, f_n$ with $s_j \leq f_j$, that define $\mathcal{I}$; $I_j = [s_j, f_j]$.

   The problem is to construct a minimal cover for $\mathcal{I}$ in $O(n \log n)$ worst-case time. Your output should be a set A which is a minimal cover.

   (a) Prove that there exists a minimal cover A such that every point in A is one of the f_j. That is, $A \subseteq \{f_1, \ldots, f_n\}$.

   (b) Give documented pseudocode for your algorithm.
   In addition, below your algorithm explain in words/symbols what your algorithm is doing

   (c) Prove correctness of your algorithm. Your proof must be mathematically formal and understandable.
   Note: Break your proof up into clear logical pieces and skip space between the pieces. Explicitly state what each part is assuming and proving.
   For examples of such proofs please see the lecture notes and tutorials.

   As seen in class, greedy algorithms tend to be simple. Their proofs of correctness are more complicated.

   (d) Explain why your algorithm runs in $O(n \log n)$ worst case time.

2. (20 points) The table below lists a 10 letters (a to j) and their frequencies in a document. Apply Huffman coding algorithm taught in class to construct an optimal codebook. Your solution should contain three parts:

   (a) A full Huffman tree.

   (b) The final codebook that contains the binary codewords representing a to j (sorted in the alphabetical order on a to j).

   (c) The codebook from part (b) but now sorted by the increasing lengths of the codewords.

i	1	2	3	4	5	6	7	8	9	10
a_i	a	b	c	d	e	f	g	h	i	j
$f(a_i)$	21	35	28	16	28	50	27	44	26	17

   Rules:

   - In part (a) you should explicitly label each edge in the tree with a '0' or '1'. You should also label each leaf with its corresponding character a_i and $f(a_i)$ value.
   - Part (b) should be a table with 2 columns. The first column should be the letters a to j. The second column should be the codeword associated with each character.
   - In part (c) you should break ties using the alphabetical order of the characters. For example, if a, d and e all have codewords of length 5, then write the codeword for a on top of the codeword for d on top of the codeword for e.

3. (20 points) Run the stable marriage algorithm as described in class on the following set of people and jobs. Let the people make proposals. Each person has a priority list of jobs, and there is also a priority list of the people for each job. As in the lecture notes, the priority decreases from left to right in the two tables below.

People	Jobs			
p_1	j_4	j_1	j_2	j_3
p_2	j_2	j_3	j_1	j_4
p_3	j_4	j_2	j_3	j_1
p_4	j_2	j_3	j_4	j_1

Jobs	People			
j_1	p_4	p_1	p_2	p_3
j_2	p_4	p_1	p_3	p_2
j_3	p_3	p_1	p_2	p_4
j_4	p_1	p_3	p_4	p_2

   Show the intermediate results of your run as in the lecture notes.

4. (20 points) Let T be a set of n tasks. Each task $t \in T$ is associated with a value $v(t)$, a length $\ell(t)$, a release time $r(t)$, and a deadline $d(t)$. The values, lengths, release times, and deadlines are positive integers for all tasks in T. We assume that $\ell(t) \leq d(t) - r(t)$ for every task $t \in T$, and that the release times and deadlines of the n tasks in T are distinct.

   A schedule of all n tasks in T is *legal* if no task t starts before its release time $r(t)$ and the processor runs at most one task at any time. If a task t finishes by its deadline $d(t)$, you collect its value $v(t)$. However, there is no guarantee that all n tasks finish before their deadlines in a legal schedule; if a task t does not finish by its deadline $d(t)$, you do not collect its value $v(t)$.

   Design a dynamic programming algorithm to determine the maximum value that you can collect by running a legal schedule of the n tasks in T on a single processor. Define and explain your notations. Define and explain your recurrence and boundary conditions. Write your algorithm in pseudocode. Derive the running time of your algorithm.

5. (20 points) Consider a two-dimensional array $A[1..n, 1..n]$ of distinct integers. We want to find the *longest increasing path* in A. A sequence of entries $A[i_1, j_1], A[i_2, j_2], \ldots, A[i_k, j_k], A[i_{k+1}, j_{k+1}], \ldots$ is a path in A if and only if every two consecutive entries share a common index and the other indices differ by 1, that is, for all k,

   - either $i_k = i_{k+1}$ and $j_{k+1} \in \{j_k - 1, j_k + 1\}$, or
   - $j_k = j_{k+1}$ and $i_{k+1} \in \{i_k - 1, i_k + 1\}$.

   A path in A is increasing $A[i_1, j_1], A[i_2, j_2], \ldots, A[i_k, j_k], A[i_{k+1}, j_{k+1}], \ldots$ if and only if $A[i_k, j_k] < A[i_{i+1}, j_{k+1}]$. The length of a path is the number of entries in it.

   Design a dynamic programming algorithm to find the longest increasing path in A. Your algorithm needs to output the maximum length as well as the indices of the array entries in the path. Note that there is no restriction on where the longest increasing path may start or end. Define and explain your notations. Define and explain your recurrence and boundary conditions. Write your algorithm in pseudo-code. Derive the running time of your algorithm.