ASSIGNMENT 4

DUE: Tuesday October 26, 11:59 PM. DO NOT COPY. ACKNOWLEDGE YOUR SOURCES.

Please read http://www.student.cs.uwaterloo.ca/~cs341 for general instructions and policies.

Note: “Giving” an algorithm means doing the four parts (i)–(iv) as described on the course web page. This goes without saying for all future assignments and exams.

1. **[10 marks] Dynamic programming for cutting plywood sheets.**

   Suppose you have an $n \times m$ rectangle and you want to cut it into smaller rectangles to sell. You are given an array $P[1..n, 1..m]$ where $P[i, j]$ gives the price for an $i \times j$ rectangle, $1 \leq i \leq n, 1 \leq j \leq m$. (Note that all rectangles have integer side lengths.) Each cut of one rectangle must be a complete horizontal cut from side to side or a complete vertical cut from top to bottom.

   Give a dynamic programming algorithm to find the maximum price you can get by cutting an $n \times m$ rectangle into smaller rectangles. For example, you can get $P[n, m]$ by selling the rectangle uncut, but you might get a better price by cutting into smaller pieces.

   Your solution must include the standard four parts (i)–(iv). Here are further instructions about each part:

   (i) Describe your algorithm using these steps:

   (a) Describe the subproblems you will solve.

   (b) Give a recursive formula for solving a subproblem, and give the base case(s).

   (c) Describe the order in which subproblems will be solved.

   (ii) Give pseudocode for your algorithm.

   (iii) Argue that your algorithm is correct. (Show/state that: the base case is correct; the recursive formula is correct; and the order of solving subproblems guarantees that the solutions to smaller subproblems are available when you solve a subproblem. This can be very brief.)

   (iv) Analyze the runtime, stating the number of subproblems, the time to solve each subproblem and the overall runtime (as functions of $n$ and $m$). Use a word RAM model, i.e., assume that each entry in $P[i, j]$ fits in one word of $O(\log n + \log m)$ bits, and, for the run time, count the number of word operations, i.e., assume that each word operation takes $O(1)$ time.

2. **[10 marks] Dynamic Programming for room assignment during a pandemic.**

   Suppose you have $n$ people lined up waiting for consultations. Person $i$ needs $q_i$ minutes for their consultation, where $q_i$ is a positive integer. There are $t$ rooms in which the consultations can be held and each room can be used for a maximum of $L$ minutes. The goal is to assign the people to the rooms so that the first $n_1$ go to room 1, the next $n_2$ go to room 2, \ldots, the last $n_t$ go to room $t$. Because you want to minimize contact during the pandemic, you want to leave gaps between successive people using the same room. For example, if $L = 10$ and room 1 will be used for $i = 1, 2, 3$ with $q_1 = 4, q_2 = 1, q_3 = 2$ then since $q_1 + q_2 + q_3 = 7$ there are 3 spare minutes, and you would have a gap of 1.5 minutes between person 1 and person 2, and a gap of 1.5 minutes between person 2 and person 3.
The objective function is to maximize the minimum gap between any two people. In other words, let $G$ be the smallest gap between any pair of people that are scheduled consecutively in the same room. The goal is to choose $n_1, n_2, \ldots, n_t$ so that $G$ is maximized.

Give a dynamic programming algorithm to find this maximum $G$. The runtime must be $O(n^3)$ or better. If it is not possible to assign the people to the rooms with a gap $\geq 0$, then output “IMPOSSIBLE”. If only one person uses a room then this does not contribute any gap, because gaps only happen between successive people in the same room. In particular, if $t \geq n$ then person $i$ can use room $i$ and you should output $\text{infty}$ (your pseudocode can just use “infty” rather than a specific large value).

Hints: You might pre-compute $G(i,j) = \text{the gap value obtained by placing people } i, i+1, \ldots, j \text{ into one room.}$ After that, use dynamic programming and aim for $O(nt)$ subproblems.

Your solution must include the four parts (i)–(iv). Here are further instructions about each part:

(i) Describe your algorithm using these steps:
   (a) Describe the subproblems you will solve.
   (b) Give a recursive formula for solving a subproblem, and give the base case(s).
   (c) Describe the order in which subproblems will be solved.

(ii) Give pseudocode for your algorithm.

(iii) Argue that your algorithm is correct. (Show/state that: the base case is correct; the recursive formula is correct; and the order of solving subproblems guarantees that the solutions to smaller subproblems are available when you solve a subproblem. This can be very brief.)

(iv) Analyze the runtime, stating the number of subproblems, the time to solve each subproblem and the overall runtime (as a function of $n$). Use a word RAM model, i.e., assume that each of the input numbers, $q_i$, $t$, $L$ fit in one word of $O(\log n)$ bits, and, for the run time, count the number of word operations, i.e., assume that each word operation takes $O(1)$ time.

In Programming Assignment 1 you will be asked to implement your algorithm and to return the actual assignment of people to rooms. However, for this question, you do not need to return the assignment of people to rooms.

Challenge Question. This is for fun and enrichment only. Do not hand it in.
1. Suppose you are given a sequence of integers separated by + and − signs; for example:

\[ 1 + 3 - 2 - 5 + 1 - 6 + 7 \]

You can change the value of this expression by adding parentheses in different places. For example:

\[ 1 + 3 - 2 - 5 + 1 - 6 + 7 = -1 \]

\[ (1 + 3 - (2 - 5)) + (1 - 6) + 7 = 9 \]

\[ (1 + (3 - 2)) - (5 + 1) - (6 + 7) = -17. \]

Give an algorithm to find the maximum value that can be attained by adding parentheses to a given sequence of the form \( x_1, s_1, x_2, s_1, \ldots, s_{n-1}, x_n \) where each \( x_i \) is an integer and each \( s_i \) is an arithmetic operator + or −. Note that you may not use parenthesis to create implicit multiplications as in \( 1 + 3(-2)(-5) + 1 - 6 + 7 = 33. \)