What to hand in. Write a short report addressing each of the questions below. In your report you can assume that the marker knows the context of the questions, so do not spend time repeating material in this handout or in the lecture notes. Submit a PDF file named A2soln.pdf to this course’s CDF site. A legibly hand-written report that is scanned to a PDF file is acceptable. We will not accept a paper copy for this assignment.

Your submitted answers will be marked on their clarity, structure, and style, as well as their correctness. So, when you are formulating your answers, think of being (especially) kind to your marker!

Finally, we may mark only one of the following three questions (the same question for everyone). I will not decide which question(s) will be marked until after the assignment due date. So please do all the questions as if they will be marked.

1. Road Trip. Suppose you are going on a long road trip, which starts at kilometer 0, and along the way there are \( n \) hotels. The \( k \)th hotel is at \( d_k \) kilometers from the start, and the hotels have been sorted so that \( 0 < d_1 < d_2 < \ldots < d_n \). Each night of the trip you must stop at one of the hotels that is further down the road than the previous night. You must stop at the \( n \)th hotel at the end of your trip (on night \( J \), see below). Let \( p_k > 0 \) denote the price (in dollars) of a room at the \( k \)th hotel.

A travel plan specifies the hotels to stay at during each night of the trip. For example, the sequence of hotel indices \( m(1), m(2), \ldots, m(J) \) forms a travel plan, where \( m(j) \) denotes that, on the \( j \)th night you decide to stay at the hotel \( d_{m(j)} \) kilometers from the start of the road. Define \( m(0) = 0 \), and constrain any travel plan \( m(j) \) as follows:

\[
m(0) = 0, \quad m(j + 1) > m(j) \text{ for } 0 \leq j < J, \quad \text{and } m(J) = n.
\] (1)

You can assume \( 0 < J \leq n \).

Unlike the truck driver’s algorithm, you would prefer to drive roughly 600 km per day. However, due to the placement (or prices) of the hotels, this isn’t always possible. To model your preference on the daily driving distance, suppose you use a penalty function of \( C(x) = (600 - x)^2Q \) for driving a distance \( x = d_{m(j)} - d_{m(j-1)} \) on day \( j \). Here \( Q > 0 \) is a constant (with the units of dollars per km²).

You wish to plan your trip in such a way as to minimize the total cost

\[
O(J) = \sum_{j=1}^{J} \left[ C(d_{m(j)} - d_{m(j-1)}) + p_{m(j)} \right],
\] (2)

where \( m \) must satisfy the constraints stated in equation (1).

(a) Consider the following greedy algorithm. Suppose the algorithm chose the \((j-1)\)st hotel to be \( m(j-1) < n \), for some \( j > 0 \). Then the algorithm chooses the \( j \)th hotel to be the \( m(j) \) which minimizes \( C(d_{m(j)} - d_{m(j-1)}) + p_{m(j)} \), subject to \( m(j-1) < m(j) \leq n \).

Prove that this greedy algorithm does not always give the minimum cost travel plan.

(b) Give an efficient algorithm that determines the minimum cost plan. That is, subject to the constraints in (1) with \( 1 \leq J \leq n \), compute a sequence \( m(1), m(2), \ldots, m(J) \) with the minimum possible total cost \( O(J) \), as defined in (2).
(c) Provide a simple implementation of your algorithm in part (b), in either Java or Python. Include a print out of this program in the solution you submit. Also, for \( Q = 5 \times 10^{-3} \) dollars/km², show sample results from your program. Describe what happens to the optimal solution when you increase or decrease the value of \( Q \).

2. **Problem 28, p.334 of text.** This is a scheduling problem with deadlines, with a hard constraint that the selected jobs must be finished before their deadlines. Do both parts (a) and (b).

3. **Optimal Parsing Trees.** Let \( y \) be a string of \( n \) characters, say \( y = (y_1, y_2, \ldots, y_n) \). We use the notation \( y(i \ldots j) \) to denote the substring \( (y_i, y_{i+1}, \ldots, y_j) \) which has length \( j - i + 1 \). Suppose we are also given a sorted list \( d = [d_0, d_1, d_2, \ldots, d_K] \) of string breakpoints, with \( 1 = d_0 < d_1 < d_2 < \ldots < d_K = n + 1 \). For example, these breakpoints could specify the endpoints of individual words, \( s_k = y(d_{k-1} \ldots (d_k - 1)) \), for \( k = 1, 2, \ldots, K \).

Here we are interested in the problem of computing an optimal parse tree for these words \( s_k \). In general, this tree may assign words to phrases, phrases to sentence parts, and so on. For the purposes of this assignment we will simplify the form of the parse tree, but keep the essential elements required to illustrate a dynamic programming approach for computing an optimal parse tree.

We consider binary parse trees with each node \( v \) storing a data item, \( v.data = (i, j) \), along with references to the left and right children, namely \( v.left \) and \( v.right \). These references to children may be null, i.e., \( v.left = null \), representing that there is no such child. The data in each node \( v \) of the parse tree, say \( v.data = (i, j) \), is a pair of integers \( i \) and \( j \), with \( 0 \leq i < j \leq K \). These integers refer to the two string breakpoints \( d_i \) and \( d_j \) which together define the substring \( y(d_i \ldots (d_j - 1)) \).

Given the input string \( y \) and the sorted list of set of breakpoints \( d = [d_0, d_1, d_2, \ldots, d_K] \), as described above, the binary tree \( T \) is said to be a feasible parse tree of \( y \) if and only if \( T \) is a binary tree (with nodes of the form \( v \) described above) which satisfies the following conditions:

- **Root node extends over \( y \).** The root node \( r \) of \( T \) must have \( r.data = (0, K) \). That is, the substring associated with \( r \) is \( y(d_0 \ldots (d_K - 1)) = y(1 \ldots n) = y \).

- **Leaf nodes are words.** Every leaf node \( v \) of \( T \) (i.e., with \( v.left = v.right = null \)) must have data of the form \( v.data = (k-1, k) \) with \( k = 1, 2, \ldots, K \). That is, leaf nodes are associated with single words \( s_k = y(d_{k-1} \ldots (d_k - 1)) \), as described above.

- **The children of non-leaf nodes correspond to left and right subtrings.** For every non-leaf node \( v \) in the tree \( T \), then both \( v.left \) and \( v.right \) are non-null. Moreover, for some integers \( i, k, j \), with \( 0 \leq i < k < j \leq K \), \( v.data = (i, j) \), \( v.left.data = (i, k) \), and \( v.right.data = (k, j) \). That is, the node \( v \) is associated with the substring \( y(d_i \ldots (d_j - 1)) \), and the left and right children are associated with the left and right parts of this string split at the intermediate breakpoint \( d_k \). In particular, the left and right children are associated with the left and right subtrings \( y(d_i \ldots (d_k - 1)) \) and \( y(d_k \ldots (d_j - 1)) \), respectively.

For a concrete example, suppose we are given a string \( y \) of length 50, and the sorted list of breakpoints \( d = [d_0, d_1, d_2, d_3] = [1, 10, 30, 51] \). Then two feasible parse trees are shown for this problem in the figure below. The two integers in each tree node represent the data \((i, j)\) for that node. These trees satisfy the conditions listed above, and are therefore feasible.

We associate a cost with every feasible parse tree \( T \), say \( cost(T) \). The problem we wish to solve is then, given the input string \( y \) and the list of breakpoints \( d \), find a minimal cost, feasible, parse tree.
Here we choose a simple form for this cost function. For any feasible parse tree $T$, define $\text{cost}(T)$ to be the sum of costs for all the nodes $v$ in $T$, and define the cost of any node $v$ to simply be $d_j - d_i$, where $v.data = (i, j)$. With this definition, the cost of any feasible parse tree $T$ is simply the sum of the lengths of all the substrings represented by nodes in $T$. For example, for the parse trees above, the cost of the root node is $d_3 - d_0 = 51 - 1 = 50$, which equals the length of the input string $y$. The cost of the whole tree $T_l$ on the left of this figure is (working from the root downwards) $\text{cost}(T_l) = 50 + 9 + 41 + 20 + 21 = 141$. Similarly, the cost of the tree on the right above is $\text{cost}(T_r) = 50 + 29 + 21 + 9 + 20 = 129$. Therefore, in this example, the right tree is the minimal cost tree.

(a) Describe a dynamic programming approach for computing the minimum cost of any feasible parse tree, given a general input string $y$ and a sorted list of break points $d = [d_0, \ldots, d_K]$. (Don’t compute an optimal parse tree itself, you will do this in part (c) below.) Clearly explain why your approach is correct.

(b) What is the time complexity (in terms of $K$) of your algorithm for computing this cost table $C$? Explain.

(c) Explain how a minimal cost, feasible, parse tree $T$ (not just its cost, as in part (a)) can be computed for this problem given the results of your algorithm in part (a).