CS 344 Problem Set 4: Dynamic Programming and Greedy algorithms

Due : (August 16, 2023 11:55 pm)

The objective of this assignment is to get you to work on developing greedy algorithms and dynamic programming based algorithms.
You must attempt all problems, even if you are unable to completely solve them.
You may discuss these problems with your classmates - and you are strongly encouraged to discuss them with me.

1. (**Subset Sum**) You are given as input a list of n positive integers $a_1, a_2, a_3, ..., a_n$ and a positive integer k. You have to answer whether or not there is some subset of these integers that add up to k.
   (a) Provide a brute force algorithm to solve this problem. What is the time complexity of your algorithm?
   (b) Let us construct a Dynamic programming based algorithm to solve this problem :
   - How do you want to define a subproblem for this problem? This will require you to consider a few choices :
     i. $S(i)$: True iff there is some subset of $a_1, a_2, a_3, .., a_i$ that adds up to k.
     ii. $S(i, j)$: True iff there is some subset of $a_1, a_2, a_3, .., a_i$ that adds up to j.

     For *each* of the choices above identify the DAG of subproblems that will help you answer the main question. Which of these choices would you pick to proceed?
   - Next, for a general subproblem, find out all the subproblems that it depends on. What is the exact dependency?
   - Use the dependency information from the previous step to develop an algorithm that efficiently computes solutions to each of the subproblems.
   - Once all the subproblems have been solved, how do you compute the final answer? What is the time complexity of your algorithm?

   (c) (True or False) The DP based algorithm is **always** more efficient than the brute force algorithm for this problem. **Justify**.

2. (**Vertex Cover**) A *Vertex Cover* for a graph $G = (V, E)$ is a subset of vertices $S \subseteq V$ that includes at least one vertex from every edge $e \in E(G)$. In this problem we will develop an algorithm that computes the minimum vertex cover for **Trees**.
   (a) Consider the tree on vertices {A, B, C, D, E, F, G} having edges {AB, BC, CD, DE, DF,FG}. What is the size of the smallest vertex cover for this tree? Give the smallest vertex cover for this tree next to a diagram of the tree.
   (b) We will now design a DP based algorithm to solve this problem.
   - Define an appropriate choice of subproblems for this problem.
   - For a general subproblem write down the dependency on other subproblems.
   - Use the dependency information from the previous step to develop an algorithm that efficiently computes solutions to each of the subproblems.
   - Once all the subproblems have been solved, how do you compute the final answer? What is the time complexity of your algorithm?

3. (**Document reconstruction**) You are given a string of n characters $s[1...n]$, which is potentially a corrupted text document in which all punctuation has vanished. You want to reconstruct the document using a dictionary which is available in the following form :

$$D(w) = \begin{cases} true & \text{if w is a valid word} \\ false & \text{otherwise} \end{cases}$$

Assume that any call to the dictionary takes $O(1)$ time.

   (a) Which of the following text documents have a valid reconstruction - "biggreyelephant", "nopainnogain", "commondaylotf" ?

   (b) Give a DP algorithm that determines whether a given string $s[1...n]$ has a valid reconstruction.

   - Consider the following choices for defining a subproblem:
     - $S(i)$: True if $s[1...i]$ has a valid reconstruction.
     - $S(i, j)$: True if $s[i...j]$ has a valid reconstruction.

     Which of these choices would you select? Why?
   - Next, for a general subproblem, find out all the subproblems that it depends on. What is the exact dependency?
   - Use the dependency information from the previous step to develop an algorithm that efficiently computes solutions to each of the subproblems.
   - Once all the subproblems have been solved, how do you compute the final answer? What is the time complexity of your algorithm?

4. (**Contiguous subsequence**) A *contiguous subsequence* of an array A is a subsequence made up of consecutive elements of A. For example, if A = [7, 17, -31, 11, -13, 43, 2] then 17, -31, 11 is a contiguous subsquence whereas -31, -13, 43 is not. Give a **linear time** algorithm for the following task :

   Input : An array of n integers $a_1, a_2, ..., a_n$.

   Output : The contiguous subsequence of maximum sum (assume a subsequence of length zero has sum 0).

   (Hint : For each $i \in \{1, 2, ..., n\}$ consider contiguous subsequences ending exactly at position i)

5. (**Longest common subsequence**) Given two strings $x = x_1x_2...x_n$ and $y = y_1y_2...y_m$ we want to find the length of their longest common subsequence, that is, the largest k such that there are indices $i_1 < i_2 < ...i_k$ and $j_1 < j_2... < j_k$ satisfying $x_{i_1}x_{i_2}...x_{i_k} = y_{i_1}y_{i_2}...y_{i_k}$. Give an $O(mn)$ time algorithm to solve this problem.

6. (**Spanning intervals**) You are given a set of n intervals on a line:

$$(a_1, b_1], (a_2, b_2], ..., (a_n, b_n]$$

   where you may assume that all intervals have integer end points. Design an algorithm to select the minimum number of intervals whose union is the same as the union of all intervals.
   For example, if the following 4 intervals are given to you:

$$(2, 5], (3, 9], (5, 10], (4, 8], (6, 10]$$

   then your answer must be

$$(2, 5], (5, 10]$$