CS 344: Design and Analysis of Computer Algorithms **Rutgers: Spring 2021**

Dynamic Programming Practice

February 17, 2021

Instructor: Sepehr Assadi

Problem 1. We have a rod of length n that we want to sell in the market. In order to do so, we need to cut the rod into multiple smaller rods since the only lengths of rods we can sell are $1, 2, 3, 4$ and 5, each with prices $1, 5, 8, 9$ and 10, respectively. Note that we can cut the rod into any number of smaller rods as long as size of each cut is $1, 2, 3, 4$ or 5.

Our goal is to find the best way to cut the rod of length n into smaller rods so as to maximize our profit. Design a dynamic programming algorithm to find the amount of best profit we can get for this problem with worst case runtime of $O(n^2)$.

Example. A couple of examples for this problem:

- When the rod length is $n = 4$, the output should be 10.

  Explanation: The best way is to cut the rod into two pieces of length 2 each to gain profit of $5+5 = 10$.

  The following lists all possible ways of cutting the rod:

Cut	Profit
4	9
$1, 3$	$(1 + 8) = 9$
$2, 2$	$(5 + 5) = 10$
$3, 1$	$(8 + 1) = 9$
$1, 1, 2$	$(1 + 1 + 5) = 7$
$1, 2, 1$	$(1 + 5 + 1) = 7$
$2, 1, 1$	$(5 + 1 + 1) = 7$
$1, 1, 1, 1$	$(1 + 1 + 1 + 1) = 4$

- When rod length is $n = 6$, the output should be 16.

  Explanation: The best way is to cut the rod into two pieces of length 3 each to gain profit of $8+8 = 16$. (You can try out all possibilities to see this is indeed the best case).

Problem 2. We have an array $A[1 : n]$ consisting of n positive integers in $\{1, \ldots, M\}$. Our goal is to check whether it is possible to partition the indices $[1 : n] = \{1, \ldots, n\}$ of the array into two sets S and T ($S \cup T = [1 : n]$ and $S \cap T = \emptyset$) such that the sum of elements in both sets is the same, i.e.,

$$\sum_{s \in S} A[s] = \sum_{t \in T} A[t].$$

Design a dynamic programming algorithm with worst-case runtime of $O(n \cdot M)$ to determine whether or not such a partition exists for this problem.

Example. A couple of examples for this problem:

- Suppose $n = 6$ and $M = 3$ and $A = [3, 1, 1, 2, 2, 1]$.

  In this case, the answer is *Yes*: we can let $S = \{2, 3, 4, 6\}$ and $T = 1, 5$ and have that:

$$\sum_{s \in S} A[s] = A[2] + A[3] + A[4] + A[6] = 1 + 1 + 2 + 1 = 5,$$
$$\sum_{t \in T} A[t] = A[1] + A[5] = 3 + 2 = 5.$$

- Suppose $n = 4$ and $M = 8$ and $A = [1, 2, 3, 8]$.

  In this case, the answer is *No* as there is no way we can partition $\{1, 2, 3, 4\}$ for this problem (to see this, note that $8 > 1+2+3$ and so the set containing 8 (index 4) is always going to have a larger sum).

Problem 3. You are given a set of n boxes with dimensions specified in the (length, width, height) format. A box i fits into another box j if *every* dimension of the box i is *strictly smaller* than the corresponding dimension of the box j. Your goal is to determine the *maximum length* of a sequence of boxes so that each box in the sequence can fit into the next one.

Design an $O(n^2)$ time dynamic programming algorithm that given the arrays $L[1:n]$, $W[1:n]$, and $H[1:n]$, where for every $1 \leq i \leq n$, $(L[i], W[i], H[i])$ denotes the (length, width, height) of the i-th box, outputs the length of the longest sequence of boxes $i_1, \ldots, i_k$ so that box i_1 can fit into box i_2, i_2 can fit into i_3, and so on and so forth. Note that a box i can fit into another box j if $L[i] < L[j]$, $W[i] < W[j]$, and $H[i] < H[j]$ (you are *not* allowed to rotate any box). **(25 points)**

Example: Given the boxes with dimensions $[(1, 2, 2), (5, 3, 2), (9, 4, 5), (2, 3, 4)]$, the correct answer is 3 as we can fit the box $(1, 2, 3)$ inside $(2, 3, 4)$, and $(2, 3, 4)$ inside $(9, 4, 5)$.

Problem 4. Given a $4 \times n$ rectangular grid and bricks of size 1×4, find the number of ways bricks can be arranged to exactly cover the entire grid. A brick can be placed horizontally or vertically.

We call the cell at row i and column j of the grid by (i, j) with $1 \leq i \leq 4$ and $1 \leq j \leq n$. Placing a brick horizontally starting at column k in row ℓ covers the cells $(\ell, k), (\ell, k+1), (\ell, k+2), (\ell, k+3)$. Placing a brick vertically in column k covers cells $(1, k), (2, k), (3, k), (4, k)$.

Example: A couple of examples for this problem:

- For $n = 1$, there is only 1 way: only 1 brick can be placed vertically.
- For $n = 4$, there are 2 ways: we could either place 4 bricks horizontally or 4 bricks vertically.
- For $n = 5$, there are 3 ways:
  - 5 vertical bricks placed side by side.
  - 1 vertical bricks covering $(1, 1), (2, 1), (3, 1), (4, 1)$ followed by 4 horizontal bricks covering columns $2, 3, 4$ and 5.
  - 4 horizontal bricks covering columns $1, 2, 3$ and 4 followed by one vertical brick covering column 5.

Problem 5. In the knapsack problem, we are given n items with positive integer weights $w_1, \ldots, w_n$ and values $v_1, \ldots, v_n$, and a knapsack of size W; we want to pick a subset of items with maximum total value that fit the knapsack, i.e., their total weight is not larger than the size of the knapsack. In Lecture 5, we designed a dynamic programming algorithm for this problem with $O(nW)$ runtime. Our goal in this problem is to design a different dynamic programming solution.

Suppose you are told that the *value* of each item is a *positive integer* between 1 and some integer V. Design a dynamic programming algorithm for this problem with worst case $O(n^2 \cdot V)$ runtime. Note that there is no restriction on the value of W in this problem.

Hint: Think of the following subproblems for your specification: what is the *minimum* size of a knapsack needed to collect j *values* from a subset of items $\{1, \ldots, i\}$? Having this, you can pick the largest value of j as the solution such that the answer to this problem is smaller than or equal to W.

Problem 6. Given two *strings* $A[1:n]$ and $B[1:m]$, the **edit distance** between A and B is defined as the *minimum number* of character insertions, character deletions, and character substitutions required to transform one string into the other. Design a dynamic programming algorithm for this problem with worst case runtime of $O(n \cdot m)$.

Example: A couple of examples for this problem:

- The edit distance between $A =$ `kitten` and $B =$ `sitting` is 3:

$$\texttt{kitten} \implies \underline{\texttt{s}}\texttt{itten} \implies \texttt{sitt}\underline{\texttt{i}}\texttt{n} \implies \texttt{sitting}\underline{\texttt{}}.$$

- The edit distance between $A =$ `cat` and $B =$ `that` is 2:

$$\texttt{cat} \implies \underline{\texttt{t}}\texttt{cat} \implies \texttt{th}\underline{\texttt{a}}\texttt{t}.$$

- The edit distance between $A =$ `happy` and $B =$ `sad` is 4:

$$\texttt{happy} \implies \underline{\texttt{s}}\texttt{appy} \implies \texttt{sa}\underline{\texttt{d}}\texttt{py} \implies \texttt{sadp\_} \implies \texttt{sad\_}.$$

Note: This problem, including its origin and detailed solution is discussed in Chapter 3.7 of your textbook.

Problem 7. We have an array $A[1:n]$ of n integers between $\{-M, \ldots, 0, \ldots, M\}$. Recall that a subsequence of A is any array obtained from A by removing zero or more elements without changing the order of the elements. The goal in this questions is to determine whether or not there exists a subsequence B of the array A with total sum equal to zero, i.e., $\sum_{i \in B} B[i] = 0$. Design a dynamic programming algorithm for this problem with worst case running time $O(n^2 \cdot M)$.

Example: A couple of examples for this problem:

- Suppose $n = 5$ and $M = 4$ and $A = [3, -1, 3, -2, 4]$. Then, the answer is *Yes* as B can be

$$B = [3, -1, -2].$$

- Suppose $n = 5$ and $M = 5$ and $A = [-3, -1, 5, -2, 4]$. Then, the answer is *Yes* as B can be

$$B = [-3, 5, -2].$$

- Suppose $n = 5$ and $M = 8$ and $A = [-3, -2, 8, -2, 6]$. Then, the answer is *No* as no subsequence of A can sum up to zero.

Note: This problem is entirely different than Problem 3 in your Homework 2 as in this problem, we are looking for a subsequence while in your homework, we are looking for a sub-array instead.