CS 3000: Algorithms & Data — Spring 2024

Homework 3
Due Monday February 19 at 11:59pm via Gradescope

Name:
Collaborators:

- Make sure to put your name on the first page. If you are using the LaTeX template we provided, then you can make sure it appears by filling in the `yourname` command.
- This homework is due Monday February 19 at 11:59pm via Gradescope. No late assignments will be accepted. Make sure to submit something before the deadline.
- Solutions must be typeset. If you need to draw any diagrams, you may draw them by hand as long as they are embedded in the PDF. We recommend that you use LaTeX, in which case it would be best to use the source file for this assignment to get started.
- We encourage you to work with your classmates on the homework problems, but also urge you to attempt all of the problems by yourself first. If you do collaborate, you must write all solutions by yourself, in your own words. Do not submit anything you cannot explain. Please list all your collaborators in your solution for each problem by filling in the `yourcollaborators` command.
- Finding solutions to homework problems on the web, or by asking students not enrolled in the class is strictly forbidden.

Problem 1. *(3 + 6 + 7 + 4 = 20 points) Stacking packages in a warehouse*

You are working in an Amazon warehouse and are designing a program to optimally store shipments, each of which is a rectangular prism (each face is a rectangle). One piece of this program is to stack the maximum number of packages in a single stack. All packages have the same height but may vary in length, breadth, and weight. So, your program takes as input three lists L, B, and W, each of length n, where $L[i]$, $B[i]$, and $W[i]$ denote the length, breadth, and weight, respectively, of the ith package.

Package i can be placed on top of package j if (i) $W[i] < W[j]$, and (ii) package i can be oriented so that it is strictly smaller than package j in each of the two dimensions. For example, even though the package i with $L[i] = 5$ and $B[i] = 1$ is longer than package j with $L[j] = 2$ and $B[j] = 6$, i can be stacked on j by switching its length and breadth.

(a) Describe an $O(1)$ time algorithm to determine whether one package can be stacked on top of another.

Solution:

(b) Suppose the packages are labeled 1 through n in order of increasing weight (assume all weights are distinct). Let $\textsc{Opt}(i)$ denote the maximum number of packages you can stack using packages 1 through i, with package i at the bottom of the stack. Give a recurrence to compute $\textsc{Opt}(i)$, and write the base case for this recurrence. Write a few sentences explaining why your recurrence is correct.

Solution:

(c) Using your recurrence, design a dynamic programming algorithm with pseudocode to output the optimal set of packages to stack. You may use either a top-down or bottom-up approach. Remember that your algorithm needs to output the optimal set of packages, not only their count. Your algorithm may output the set of packages as a list of indices. The running time of your algorithm must be polynomial in n.

Solution:

(d) Analyze the running time and space usage of your algorithm.

Solution:

Problem 2. *(5 + 6 + 9 = 20 points) Nicest photos*

Alice would like to take nice photos of the Boston skyline. She records the height of the buildings in an array $h[1],\ldots,h[n]$, where h_1 is the height of the leftmost building and h_n is the height of the rightmost building. Each photo cannot stray beyond the recorded skyline and it captures k consecutive buildings. The score of a photo is the total height of all the building in the photo plus the tallest height. For example, if the heights are $5,2,9,1,7$ and $k = 3$ then the best photo would capture $9,1,7$ with score $(9+1+7)+9 = 26$. Alice would like to find the set of non-overlapping photos with the maximum total score. For example, if the heights are $1,1,1,6,6,1,1,1,1$ and $k = 3$ then the best set of photos consist of $1,1,6$ and $6,1,1$ for the total score of 28.

(a) Let $score[i]$ denote the score of the photo capturing buildings from $i-k+1$ to i. Give an algorithm to compute $score[i]$ for all i that runs in $O(nk)$ time.

Solution:

(b) Let $\textsc{Opt}[i]$ denote the maximum score for non-overlapping photos contained entirely between buildings from 1 to i, inclusively. Give a recurrence for computing $\textsc{Opt}[i]$ and the base case(s).

Solution:

(c) Use your recurrence to design a dynamic programming algorithm with pseudocode, which takes as input the h array and k, and computes the best total score as well as the photo locations. Give the running time of your algorithm.

Solution:

(d) **(0 bonus points)** Give an algorithm for part (a) with $O(n)$ runtime. Hint: a viable approach is divide and conquer.

Solution:

Problem 3. *(5 + 6 + 5 + 4 = 20 points) Elegant subsequence*

We are given an array $A[1,\ldots,n]$ of n distinct integers sorted in increasing order. A subsequence of A is a sequence that can be derived from A by deleting some (or none) of the elements without changing the order of the remaining elements. A sequence is elegant if all the differences of the consecutive elements are all the same. For example, $1,4,7,10$ is elegant because the differences between consecutive elements are all 3. We would like to find the longest elegant subsequence of A. For example, if $A = (1,3,4,7,8,10)$ then the longest elegant subsequence is $(1,4,7,10)$. For full credit, your algorithm should run in $O(n^2 \log n)$ time ($O(n^2)$ is possible).

(a) Let $L[i,j]$ be the length of the longest elegant subsequence whose last two indices are i and j where $i < j$. Describe a recurrence to compute $L[i,j]$ using $L[k,i]$ $\forall k < i$. Hint: Define $p[i,j]$ to be the index of the element in A such that $A[p[i,j]] = 2A[i] - A[j]$ i.e. $A[p[i,j]] - A[i] = A[i] - A[j]$ (if such an element exists).

Solution:

(b) Describe a dynamic programming algorithm with pseudocode to compute all $L[i,j]$.

Solution:

(c) Describe how to find the longest elegant subsequence from the $L[i,j]$ you computed.

Solution:

(d) Analyze the running time and space usage of your algorithm.

Solution:

Problem 4. *(6 + 7 + 3 + 4 = 20 points) Resource reservation in video transmission*

Consider the following resource reservation problem arising in video transmission. Suppose we have a video that is a sequence of n frames. We are given an array $s[0 \ldots n-1]$: frame i requires the reservation of at least $s[i]$ units of bandwidth along the transmission link. Since reserving resources separately for each frame may incur a significant overhead, we would like to partition the video into at most k segments (where k is usually much smaller than n), and then reserve bandwidth for each segment. Note that each segment is simply an interval of contiguous frames and that the segments may be of different lengths.

The amount of bandwidth that we need to reserve for a segment is the maximum, over all frames in the segment, of the bandwidth required for the frame. Formally put, for a given interval of frames I, the bandwidth $B(I)$ required for the segment equals $\max_{i \in I} s[i]$.

A segmentation of the video into k segments is a sequence of $k-1$ integers $0 < p[0] < p[1] < \ldots < p[k-2] < k$ indicating that the segments are given by the k intervals $I_0 = [0 \ldots p[0]-1], I_1 = [p[0] \ldots p[1]-1], \ldots, I_{k-1} = [p[k-1] \ldots k-1]$. The cost of the segmentation is the total bandwidth required, which is the following:

$$\sum_{0 \le i < k} |I_i| \cdot B(I_i)$$

where $|I_i|$ is the number of frames in segment S_i.

Example: Suppose $n = 10$, the array $s = [3, 5, 2, 9, 5, 6, 7, 1, 4, 11]$, and $k = 3$. If we consider the segmentation $[0-2], [3-6], [7-9]$, the total bandwidth requirement would be

$$5 \times 3 + 9 \times 4 + 11 \times 3 = 84.$$

On the other hand, if we choose the segmentation $[0-3], [4-8], [9-9]$, the total bandwidth requirement would be

$$9 \times 4 + 7 \times 5 + 11 = 82.$$

Your goal is to design a dynamic programming algorithm that takes input the n values $s[0], s[1], \ldots, s[n-1]$, and the positive integer $k \le n$ and computes a segmentation which has minimum cost.

(a) Let $\textsc{Opt}(i, m)$ denote the minimum cost for segmenting frames 0 through i using m segments. Give a recurrence to compute $\textsc{Opt}(i, m)$, and write the base case(s) for this recurrence. Write a few sentences explaining why your recurrence is correct.

Solution:

(b) Using your recurrence, design a dynamic programming algorithm with pseudocode, which takes as input the s array and k, and outputs the optimal cost for segmenting the video into k segments. You may use either a top-down or bottom-up approach. The output of your algorithm is a single number. The running time of your algorithm must be polynomial in n.

Solution:

(c) Analyze the running time and space usage of your algorithm.

Solution:

(d) Enhance your algorithm of part (b) to return the optimal segmentation, not just the optimal cost. The output of your algorithm can be the array p giving the segmentation.

Solution: