FirstName LastName
StudentID acct@ucsc.edu

CMPS 102 — Spring 2020 – Homework 4

Five problems, due Saturday May 16 (on canvas) and May 25 (11:59 PM) on Gradescope.
Due class size, it is likely that only a subset of the problems will be graded.
Draft version: a final version and solution template will be posted in a couple of days.

Before you begin the assignment, please read the following carefully.

- Read the Homework Guidelines.
- Every part of each question begins on a new page. Do not change this.
- This does not mean that you should write a full page for every question. Your answers should be short and precise. Lengthy and wordy answers will lose points.
- Do not change the format of this document. Simply type your answers as directed.
- You are **not** allowed to work in teams.

I have read and agree to the collaboration policy. – Isai Lopez Rodas, ilopezro@ucsc.edu
Collaborators: None

---

1. (1 pt) Take the pre-quiz on Canvas by 11 PM on Saturday May 16th. This will be short quiz designed to test if you have read and understood the other problems. The quiz should be available by Friday afternoon.

2. (10 pts) Let $B(n)$ be the number of different binary search trees containing the n keys $\{1, 2, ..., n\}$. For this problem you will develop use a dynamic-programming like methodology (mostly memoization) to create an algorithm that, given n, calculates $B(n)$ efficiently. Note that $B(1) = 1$ since there is only one one-node binary tree. For two nodes $B(2) = 2$ (either node can be the root, and there is only one binary search tree consistent with each choice).

   (a) (1 pt) Calculate by hand $B(3)$.

   Solution.

   The root node can have any random node at its root. Therefore, that leaves us with 2 nodes to pick from and put them in any given order either on the left node or the right node of the tree. This means that for each side we can have 2C1 (2 choose 1 combination) for each side of the tree, we can do this. 2C1 resolves to 2, which means that on each side of the tree, we have 2 possible combinations. Since we already have one at the root, this will be a total of 2+1+2 = 5. Therefore, B(3) = 5.

(b) (1 pt) How many $n = 6$ key binary trees are there with keys $\{1, 2, 3, 4, 5, 6\}$ and key 3 at the root (so the left subtree has two nodes, and the right one has 3 nodes) are there?

Solution.

(c) (4 pts) Construct a recurrence for $B(n)$, I expect something with a sum. Include boundary conditions in your recurrence; what is a convenient boundary value for $B(0)$? Briefly justify (formal proof not required) that your recurrence computes the correct value.

Solution.

(d) (3 pts) Give an iterative (bottom up) algorithm based on the recurrence that computes $B(n)$ by filling in a table. It may help to first create a recursive function with memoization from your recurrence above.

Solution.

(e) (1 pt) What is the running time of your iterative algorithm as a function of n (use asymptotic notation)?

Solution.

3. (8 pts) Assume that you have a list of n home maintenance/repair tasks (numbered from 1 to n) that must be done in numeric order on your house. You can either do each task i yourself at a positive cost (that includes your time and effort) of $c[i]$. Alternatively, you could hire a handyman who will do the next 4 tasks for the fixed cost h (regardless of how much time and effort those 4 tasks would cost you). The handyman always does 4 tasks, so cannot be used if fewer than four tasks remain. You are to create a dynamic programming algorithm that finds a minimum cost way of completing the tasks. The inputs to the problem are h and the array of costs $c[1], \ldots, c[n]$.

   (a) (3 pts) First, find and justify a recurrence (with boundary conditions) giving the the optimal cost for completing the tasks. Use $M(j)$ for the minimum cost required to do the first j tasks.

   Solution.

(b) (2 pts) Give an $O(n)$-time recursive algorithm with memoization for calculating the $M(j)$ values.

Solution.

(c) (1 pt) Give an $O(n)$-time bottom-up algorithm for filling in the array

Solution.

(d) (2 pts) Describe how to determine which tasks to do yourself, and which tasks to hire the handyman for in an optimal solution.

Solution.

(optional challenge for the interested students, not to be turned in: solve the variant of the problem where the handyman can be used at cost h to finish the last one, two, or three tasks on the list)

4. (10 pts) Assume you are planning a canoe trip down a river. The river has n trading posts numbered 1 to n going downstream. You will start your trip at trading post number 1 and end at trading post number n. Let $R(i, j)$ be the cost of renting a canoe at trading post i and returning it at trading post j, where $j > i$. Assume that you always want to go down river, so the costs if $j \leq i$ are irrelevant. Find the cheapest sequence of rentals that allow you to complete your trip. Aim for an algorithm running in $O(n^2)$ time.

   For example, if $n = 4$ and the costs are:

R(i,j)	j		
i	2	3	4
1	15	25	35
2	$--$	12	16
3	$--$	$--$	5

   then the cheapest sequence of canoe rentals to travel the river would be to rent from 1 to 3, and then from 3 to 4 for a cost of $25 + 5 = 30$.

   On the other hand, if the costs were:

R(i,j)	j		
i	2	3	4
1	20	15	30
2	$--$	5	10
3	$--$	$--$	20

   then taking one canoe all the way from 1 to 4 and renting 1 to 2 and then 2 to 4 are the cheapest solutions (both cost 30). (Note that renting from 1 to 3 is cheaper than going 1 to 2, but the 1 to 3 rental is not in any of the cheapest 1 to 4 solutions.)

   Let $C(k)$ be the cost of the cheapest sequence of canoe rentals starting from trading post 1 and returning the last canoe rented at trading post k.

(a) (3 pts) Assume an optimal sequence of rentals changes canoes at some trading post j. What subproblems are also solved optimally by (parts of) this rental sequence? Prove your answer (probably using a proof by contradiction)

Solution.

(b) (2 pts) Derive a recurrence for $C(k)$ in terms of $C(j)$ values where $j < k$.

Solution.

(c) (4 pts) Give a bottom-up iterative algorithm for computing the $C(j)$ values. This algorithm may also compute "signpost" values for use in the following part.

For part (c), it may be helpful to first construct a recursive algorithm for computing the $C(k)$ values.

Solution.

(d) (1 pt) Finally, show how keeping a little (i.e. $O(n)$) additional information allows the a cheapest sequence of canoe rentals to be printed out in $O(n)$ time.

Solution.

5. (10 pts) Consider the problem of detecting similar papers in a history course. If the assignment was on a particular topic, then one would expect that many of the key terms related to the topic would appear in most of the essays, but perhaps not in the same order. Let the sequence similarity of two essays be the length of the longest sequence of words such that both essays contain the sequence of words in the same order.

   For example, if one paper was "four score and seven years ago our fathers brought forth upon this continent, a new nation conceived in liberty, and dedicated to the proposition that all men are created equal." and another paper was "four years and seven days ago, fathers helped created a new nation with liberty and justice for all on the continent."

   One sequence I found that has many of the words in both essays in right order is: "four and seven ago fathers a new nation and all" (10 words). Replacing "all" with "the" gives another 10 words occurring in both sequences in order. Although "continent" and "years" also occur in both papers, they cannot be added to this sequence while preserving the sequence's order of occurrence.

   For this problem you are to create a dynamic programming algorithm that takes a papers P and Q as a lists/arrays of words n and m words respectively (possibly with repeats) and determines a longest sequence of words that occur in both papers in the same order.

(a) (4 pts) First focus on the length of the longest sequence of words occurring in both papers in order. Derive a recurrence for this length by considering what can happen with the last words in the papers. What are the boundary conditions for your recurrence? Give a brief rational why the recurrence gives the right values.

Solution.

(b) (3 pts) Give a bottom-up dynamic programming algorithm based on your recurrence that calculates the value of the optimal solution (i.e. the length of the longest sequence of words occurring in both lists in the same order). This algorithm should fill in a table.

Solution.

(c) (1 pt) What is the running time of your dynamic programming algorithm? (use asymptotic notation, assume that any two words can be compared in $O(1)$ time)

Solution.

(d) (2 pts) Describe how to output an actual longest sequence of words occurring in both lists in the same order. What is the (asymptotic) worst-case running time of your output method (assuming the table from part (b) has already been calculated).

Solution.