Instructor Name: Siddhartha Nath
Email: s.nath@northeastern.edu

Northeastern University

Program Structure and Algorithms (INFO 6205)
Homework #5 –SAMPLE SOLUTIONS – 100 points

Student NAME:

Student ID:

Question 1 (30 points)**.** *A **palindrome** is a non-empty string over some alphabet that reads the same forward and backward. Examples of palindromes are all strings of length 1, civic, racecar, and aibohphobia (fear of palindromes).*
Give an efficient algorithm to find the longest palindrome that is a subsequence of a given input string of length n. For example, given the input $\{character\}$, your algorithm should return $\{carac\}$.

(a) (8 points) Please describe your subproblems.

We define $LPS[i,j]$ be the longest palindrome in the input sequence x_i to x_j. We want $LPS[1,n]$.

(b) (8 points) Please describe the decisions.

There are mainly two decisions to make. (i) if $x_i = x_j$, then $LPS[i,j] = 2 + LPS[i+1, j-1]$. (ii) Otherwise, we need to look at $LPS[i+1,j]$ and $LPS[i,j-1]$ and pick the longest.

Additional cases, are $LPS[i,j] = 2$ if $x_i = x_j$ and $j = i+1$, and $LPS[i,j] = 1$ if $x_i \neq x_j$ and $j = i+1$. Since we do not know i and j, we must search for these in $1 \leq i < j \leq n$.

(c) (10 points) Please write down the recursion and base cases.

Recursion.

$$LPS[i,j] = \begin{cases} 1, & \text{if } i = j, \\ 1, & \text{if } j = i+1 \text{ and } x_i \neq x_j, \\ 2, & \text{if } j = i+1 \text{ and } x_i = x_j, \\ max\{LPS[i+1,j], LPS[i,j-1]\}, & \text{if } j > i+1 \text{ and } x_i \neq x_j. \end{cases}$$

Base cases:

- $LPS[i,i] = 1$;
- $LPS[i,j] = 1$ *if* $j = i+1$ *and* $x_i \neq x_j$; *and*
- $LPS[i,j] = 2$ *if* $j = i+1$ *and* $x_i = x_j$.

(d) (4 points) Please describe the running time of your algorithm in terms of the number of subproblems and the running time per subproblem.

We have $O(n^2)$ subproblems and each takes $O(1)$ time to solve, so the running time is $O(n^2)$.

Question 2 (30 points). *You have to store n books $(b_1, b_2, \ldots, b_n)$ on shelves in a library. The order of books is fixed by the cataloging system and cannot be rearranged. A book b_i has a thickness t_i and height h_i, where $1 \leq i \leq n$. The length of each bookshelf at the library is L. The values of h_i are not necessarily assumed to be all the same. In this case, we have the freedom to adjust the height of each bookshelf to that of the tallest book on the shelf. Thus, the cost of a particular layout is the sum of the heights of the largest book on each shelf.*

(a) *(10 points) Give an example to show that the greedy algorithm of stuffing each shelf as full as possible does not always give the minimum overall height.*

Consider a set of three books, each of thickness 1, with the first book b_1 has height 1, and the second (b_2) and third (b_3) books having height 2 each. Let's suppose that $L = 2$. A greedy algorithm will sort books by height (let's assume increasing order) and then pick books to place on shelf based on the sorted order. Clearly, b_1 must be placed on the first shelf. The greedy algorithm then places b_2 on the first shelf. This now fills the first shelf, and so we then must place b_3 on the second shelf.

Since the largest book on each shelf is of height 2, the cost of this layout is 4. Instead of this layout, if we place the first book on the first shelf, and the last two books on the second shelf, the first shelf has only one book of height 1, and the second shelf has books only of height 2. Thus, the cost of this layout is 3, which is better than what the greedy algorithm produced.

(b) *(20 points) Describe in English an algorithm based on dynamic programming to achieve the minimum overall height. No need to provide pseudocode, but you must state your subproblems, decisions, recursion, base case(s) and analyze the running time complexity.*

This is similar to the 0-1 Knapsack problem without repetition. The thickness t_i is analogous to weight, value is analogous to height h_i and the capacity is analogous to the shelf width L. The sum of thickness of books on any shelf must be $\leq L$. The height of a shelf is the height of the tallest book placed in the shelf.
Subproblems: *Let $msh(i)$ be the minimum shelf height after placing the first i books.*

Decisions: *Take min of the following.*

- *Place the i^{th} book on a new shelf. $msh(i) = msh(i-1) + h_i$.*
- *Place the i^{th} book on current shelf. Iterate from $(i-1)$ to 1, check the sum of t_i with the sum of thickness of previous books to be either $\leq L$ or the count reaches 0. Thus, $msh(i) = max_j h_j$ (in the current shelf) $+msh(j-1)$ (i.e., we can fit $(i-1)$ to j books in the current shelf.*

Recursion: *Iterate over books from $i = 1$ to n, execute the above decisions for each i. Base cases are $msh(0) = 0, msh(1) = h_1$.*

Running time: *We have $O(n)$ subproblems. For each subproblem, we may iterate in the worst-case from $(i-1)$ to 0, so $O(n)$ time per subproblem. Thus, the overall running time is $O(n^2)$.*

Question 3 (40 points)**.** *You are given an unlimited supply of coins of a given denomination S and a target amount N. Your goal is to find the minimum number of coins required to make change for N using the given denominations.*

Example #1. *Suppose the denominations as $S = \{1, 3, 5, 7\}$ and $N = 18$, the minimum number coins is 4 (i.e., $\{7, 7, 3, 1\}$ or $\{5, 5, 5, 3\}$ or $\{7, 5, 5, 1\}$).*

Example #2. *Suppose the denominations are $S = \{25, 10, 1\}$ and $N = 40$, the minimum number of coins is 4 (i.e., $\{10, 10, 10, 10\}$).*

(a) *(6 points) Can you show that the greedy choice (used in HW4) for coin changing is suboptimal for Example #2?*

The greedy choice of picking the largest denomination first will result in using 7 coins, $\{25, 10, 1, 1, 1, 1, 1\}$, whereas the optimal solution is 4 coins, $\{10, 10, 10, 10\}$.

(b) *(4 = 2 + 2 points) Can you please describe a brute-force (non-DP) approach to solve Example #2? What will be the running time of the brute-force approach?*

We check if the total N can be reached by including each denomination or not. If choosing the current coin is possible, update the balance and the count of coins, else pick the next coin. Finally, return the minimum number of coins to get N. This will result in a recursion with the worst-case tree height of N. Thus, the runtime will be exponential $O(2^N)$.

(c) *(2 points) Suppose we use dynamic programming, which problem that we have solved in the class is the closest to this problem?*

Knapsack (0-1) with repetion.

(d) *(6 points) Please describe your subproblems based on your answer in (c)?*

$MC(k)$ is the minimum number of coins need to make change of $k \leq N$. We need $MC(N)$. Since we do not know i, we must search for these in $0 < i < k \leq N$.

(e) *(6 points) Please explain the decisions you need to make to solve each of your subproblems.*

If the optimal solution includes coin denomination i, then the remaining subproblems to solve will be $MC(k - i) + 1$.

(f) *(6 points) Please write down the recursion for this problem and the base case(s).*

Recursion. $MC[k] = min\{MC[k - i] + 1 : \forall i \leq k\}$

Base case: $MC[0] = 0$.

(g) *(4 points) Please describe the running time of your algorithm in terms of the number of subproblems and the running time per subproblem.*

We have $O(N)$ subproblems, each can taken $O(|S|)$ times to solve, so running time is $O(|S| \cdot N)$.

(h) *(6 points) For Example #2, please describe the value of each of your subproblems that your dynamic programming algorithm (based on previous steps) will compute in each iteration. In how many iterations will your algorithm terminate?*

(1) ∞
(2) 1
(3) 1