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

Northeastern University

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

---

**Student NAME:** Xijing Zhang

**Student ID:** 002205911

---

**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.*

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

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

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

*(a) Let the input string be $s$ of length $n$. Define the subproblem as:*

$$L(i, j) = \textit{Length of the longest palindromic subsequence in } s[i \dots j]$$

*We want to compute $L(0, n-1)$.*

*(b) If $s[i] = s[j]$, then both characters can be included:*

$$L(i, j) = 2 + L(i+1, j-1)$$

*If $s[i] \neq s[j]$, try excluding one character:*

$$L(i, j) = \max(L(i+1, j), L(i, j-1))$$

*(c)*

```
If s[i] == s[j]:
  L(i, j) = 2 + L(i+1, j-1)
Else:
  L(i, j) = max(L(i+1, j), L(i, j-1))
```

*Base Cases:*

- `L(i, i) = 1` *for all* `i` *(a single character is a palindrome of length 1)*
- `L(i, j) = 0` *if* `i > j` *(empty substring)*

*(d) There are $O(n^2)$ subproblems, and each takes $O(1)$ time. So the total time complexity is:*

$$O(n^2)$$

*Space complexity is also:*

$$O(n^2)$$

**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.*

***(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.*

*(a)*

*Let $L = 5$, and four books as follows:*

$$(t_1, h_1) = (3, 1), \quad (t_2, h_2) = (2, 10), \quad (t_3, h_3) = (2, 10), \quad (t_4, h_4) = (3, 1).$$

- *Greedy packing:*
  *Shelf 1: books 1 & 2 (thickness 5, height 10)*
  *Shelf 2: books 3 & 4 (thickness 5, height 10)*
  *Total height $= 10 + 10 = 20$.*

- *Optimal packing:*
  *Shelf 1: book 1 alone (height 1)*
  *Shelf 2: books 2 & 3 (thickness 4, height 10)*
  *Shelf 3: book 4 alone (height 1)*
  *Total height $= 1 + 10 + 1 = 12$.*

*Thus stuffing each shelf full is not always optimal.*

*(b)*

*subproblem:*

*Let dp[i] be the minimum total height for placing books from b1 to bi.*

*Decisions:*

*To compute dp[i], we try placing the last k books (ending at i) on the same shelf.*
*We go from $j = i$ down to 1, and for each j:*

*- Compute the total thickness from book j to i as sum_thickness.*
*- Track the maximum height from book j to i as max_height.*
*- If sum_thickness  L, then this is a valid shelf placement, and we consider:*

$$dp[i] = \min(dp[i], dp[j-1] + max\_height)$$

*Recursion:*

$$dp[i] = \min_{j \le i, \sum_{k=j}^{i} t_k \le L} (dp[j-1] + \max(h_j, ..., h_i))$$

*Base case:*
*dp[0] = 0 (no books means no height)*
*Final answer:*
*dp[n] gives the minimum total height to place all n books.*
*Time complexity:*
*We consider up to i options for each i, so the total time complexity 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?*

*(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?*

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

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

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

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

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

*(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?*

(a) With $S = \{25, 10, 1\}$ and $N = 40$, the greedy algorithm picks

$$25 + 10 + 1 + 1 + 1 + 1 + 1 = 40$$

for a total of $1 + 1 + 5 = 7$ coins, whereas the optimal is

$$10 + 10 + 10 + 10 = 40$$

using only 4 coins. Thus greedy (7 coins) > optimal (4 coins).

(b) Define a recursive function

$$\text{Brute}(i) = \begin{cases} 0, & i = 0, \\ \min\limits_{c \in S,\, c \le i} \big\{1 + \text{Brute}(i - c)\big\}, & i > 0. \end{cases}$$

Running time: Exponential in $N$, roughly $O\big(|S|^{N/\min S}\big)$.

(c) This is the *unbounded knapsack* (or equivalently, the rod-cutting) DP formulation.

(d) Let

$$f(i) = \text{minimum number of coins needed to make amount } i, \quad i = 0, 1, \ldots, N.$$

(e) For each subproblem $f(i)$, choose which coin $c \in S$ to use last, then solve $f(i - c)$ and add 1.

(f)

$$f(0) = 0,$$
$$f(i) = \min_{c \in S,\, c \le i} \{f(i-c) + 1\}, \quad i = 1, 2, \ldots, N.$$

(g)

- Number of subproblems: $N + 1$.
- Work per subproblem: $O(|S|)$.
- Total: $O(N \cdot |S|)$.

(h)

$(S = \{25, 10, 1\}, N = 40)$.

Compute $f(0)$ through $f(40)$ in increasing order:

| $i$ | $f(i)$ | Choice giving $f(i)$ |
|---|---|---|
| 0 | 0 | – |
| 1 | 1 | 1¢ |
| 2 | 2 | 1¢ + 1¢ |
| ⋮ | ⋮ | ⋮ |
| 9 | 9 | 1¢ × 9 |
| 10 | 1 | 10¢ |
| 11 | 2 | 10¢ + 1¢ |
| ⋮ | ⋮ | ⋮ |
| 20 | 2 | 10¢ + 10¢ |
| 25 | 1 | 25¢ |
| 30 | 3 | 10¢ × 3 |
| 35 | 2 | 25¢ + 10¢ |
| 40 | 4 | 10¢ × 4 |

The algorithm fills 41 entries ($i = 0$ to 40), so it terminates after 41 iterations.