A white paper with black text

Description automatically generated

**(a) Subproblem**

Consider the string x of length n and reverse x is called as y. A subproblem can be defined as finding the longest palindromic subsequence in a substring x[1...i] and y[1...j],

Each subproblem involves finding the LCS between the original string and its reverse. This allows us to use dynamic programming to compute the solution.

1. If characters x[i] and y[j] match, then the current character contributes to the palindrome.
2. If they don't match, then you need to decide whether to exclude x[i] or y[j] to maximize the subsequence.

**(b) Decisions**

Case 1: x[i] == y[j]

dp[i][j] = dp[i-1][j-1] + 1

Case2: x[i] != y[j]

dp[i][j] = max(dp[i-1][j], dp[i][j-1])

**(c) Recursion and Base Cases**

**Recursion:**

If x[i] == y[j], then dp[i][j] = dp[i-1][j-1] + 1

If x[i] != y[j], then dp[i][j] = max(dp[i-1][j], dp[i][j-1])

**Base Case**:

For any i or j where one of the strings is empty (i=0 or j=0), the LCS is 0 because there are no characters to match. Thus:

dp[i][0] = 0 and dp[0][j] = 0

**(d) Running Time**

The number of subproblems are separately **i and j**. The running time per subproblem is **i\*j**. Total Running Time is **n^2**.

Python Code：

def lcs(x, y) -> (int, str):  
 m = len(x)  
 n = len(y)  
 dp = [[0] \* (n + 1) for \_ in range(m + 1)]  
  
 for i in range(1, m + 1):  
 for j in range(1, n + 1):  
 if x[i - 1] == y[j - 1]:  
 dp[i][j] = dp[i - 1][j - 1] + 1  
 else:  
 dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])  
 # Backtrack to get the longest subsequence  
 res = []  
 i, j = m, n  
 while i > 0 and j > 0:  
 if x[i - 1] == y[j - 1]:  
 res.append(x[i - 1])  
 i -= 1  
 j -= 1  
 elif dp[i - 1][j] >= dp[i][j - 1]:  
 i -= 1  
 else:  
 j -= 1  
 return dp[m][n], ''.join(res)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 x = 'character'  
 y = x[::-1]  
 print(lcs(x, y))

A text on a white background

Description automatically generated

a.

|  |  |  |
| --- | --- | --- |
| Item | Thickness | Height |
| book1 | 2 | 4 |
| book2 | 2 | 2 |
| book3 | 2 | 7 |
| book4 | 2 | 9 |

Initial conditions: L = 6

Case1: stuffing each shelf is as full as possible.

The thickness of book1 + book2 + book 3 is 6. The height of first bookshelf is 7;

The height of second bookshelf is 9.

Total height is 16.

Case2: stuffing each shelf is not as full as possible.

Book1 and book2 put in the first bookshelf—The height is 4;

Book3 and book4 put in the second bookshelf – The height is 9;

Total height is 13.

**Therefore, stuffing each shelf as full as possible does not always give the minimum overall height.**

b.

**Subproblem Definition:**

dp[i] = minimum total height needed to arrange books from book 1 to book i.   
Note: i from 1 to n.

**Decision:**

Case1: if i = 0:

return 0

Case2: Book i is placed alone on a new shelf:

dp[i] = dp[i-1] + hᵢ

Case3: Books j+1 to i are placed together on the same shelf:

If tⱼ₊₁ + ... + tᵢ ≤ L, then

dp[i] = min(dp[i], dp[j] + max(hⱼ₊₁, ..., hᵢ))

We try all valid j < i such that the total thickness from j+1 to i does not exceed shelf length L.

**Recursion:**

To compute dp[i], consider placing book i on the same shelf as some earlier books j+1 to i (i.e., ending the previous shelf at position j). We try all valid j < i such that the total thickness from j+1 to i is ≤ L.

dp[i] = min(dp[j] + max\_height(j+1 to i)) for all j where total\_thickness(j+1 to i) <= L

**Base cases:**

dp[0] = 0 — no books → height is 0

**Time Complexity:**

* For each i, we may check all j < i, and for each such j, compute max height and total thickness of books from j+1 to i.
* This gives O(n²) time complexity.

A white and black text with black text

Description automatically generated with medium confidence

1. Firstly, sort coins by their denominations in decreasing order, and the greedy choice will be to select the coin with the largest denomination as long as N − d > d, where d is the denomination. Otherwise, pick the next denomination until N − d = 0.

S = {25,10,1} , N = 40, count = 0(The number of coins)

Select the largest denomination 25: 40 – 25 = 15, count = 1

Select the denomination 10: 15 – 10 = 5, count = 2

Select the denomination 1: total time is 5, count = 7

The final number of count is 7 and not the best result(actual result is 4).

1. 1. Try all combinations of coins (with repetitions allowed) that sum to 40, and track the one with the minimum number of coins used.  
   2. **Running Time**: N = target amount, k = solves for subproblem like tree. Time complexity is O(k^N). About Example #2, time complexity is 2^40
2. P6: Knapsack w/ Repetition
3. dp[i] as the minimum number of coins needed to form amount N (i is from 1 to N). We compute this for all coins get the min dp[i].
4. **Decision:**

If we include coin, we add 1 to the solution of subproblem dp[i - coin]

dp[i] = dp[i - coin] + 1.

1. **Recursion:**

For each coin in the set of denominations S, and for each amount w ≥ coin:

dp[i] = min(dp[i], dp[i - coin] + 1)

**Base case**:

dp[0] = 0 (zero coins needed to make amount 0)

dp[1:N] = ∞(Because we calculate the minimum number of coins)

1. **Running time:**

N = target amount, k = number of denominations

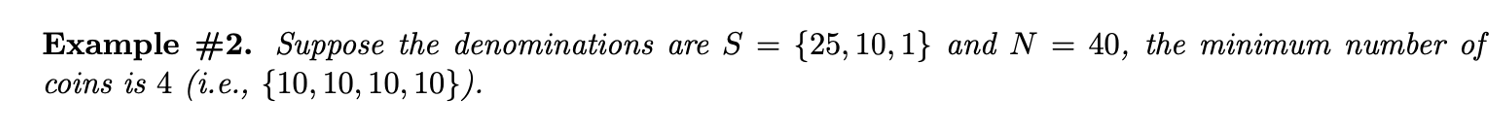
The number of subproblems is N.

The running time per subproblem is k.

**The total running time is N\*k**

1. **Calculation:**

**Firstly, we should sort the S so that we can select the largest denominations.**

****

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Res** | **25** | **10** | **1** | **Iteration** |
| **0** | **0** |  |  |  |  |
| **1** | 1 | **-** | **-** | **dp[1-1] + 1 = 1** | **1** |
| **2** | 2 | **-** | **-** | **dp[2-1] + 1= 2** | **1** |
| **3** | **3** | **-** | **-** | **dp[3-1] + 1= 3** | **1** |
| **4** | **4** | **-** | **-** | **dp[4-1] + 1= 4** | **1** |
| **5** | **5** | **-** | **-** | **dp[5-1] + 1= 5** | **1** |
| **6** | **6** | **-** | **-** | **dp[6-1] + 1= 6** | **1** |
| **7** | **7** | **-** | **-** | **dp[7-1] + 1= 7** | **1** |
| **8** | **8** | **-** | **-** | **dp[8-1] + 1= 8** | **1** |
| **9** | **9** | **-** | **-** | **dp[9-1] + 1= 9** | **1** |
| **10** | **1** | **-** | **dp[10-10] + 1= 1** | **dp[10-1] + 1= 10** | **2** |
| **11** | **2** | **-** | **dp[11-10] + 1= 1** | **dp[11-1] + 1 = 11** | **2** |
| **12-19** | **3-10** | **-** | **Same calculation** | **Same calculation** | **16** |
| **20** | **2** | **-** | **dp[20-10] + 1= 2** | **dp[20-1] + 1= 11** | **2** |
| **21-24** | **3-6** | **-** | **Same calculation** | **Same calculation** | **8** |
| **25** | **1** | **dp[25-25] + 1= 1** | **dp[25-10] + 1= 7** | **dp[25-1] + 1= 7** | **3** |
| **26-29** | **2-5** | **Same calculation** | **Same calculation** | **Same calculation** | **12** |
| **30** | **3** | **dp[30-25] + 1= 6** | **dp[30-10] + 1= 3** | **dp[30-1] + 1= 6** | **3** |
| **31-34** | **4-7** | **Same calculation** | **Same calculation** | **Same calculation** | **12** |
| **35** | **2** | **dp[35-25] + 1= 2** | **dp[35-10] + 1= 2** | **dp[35-1] + 1= 11** | **3** |
| **36-39** | **3-6** | **Same calculation** | **Same calculation** | **Same calculation** | **12** |
| **40** | **4** | **dp[40-25] + 1= 7** | **dp[40-10] + 1= 4** | **dp[40-1] + 1= 7** | **3** |

**Iterations is 87 if coin <= w(w is from 1 to 40).**

**Iterations is 120 if we don’t compare the condition of coin and w.**