Problem 4.1 In the line code of LOOK-CHAIN (checking slides or p.388 from the text book), if the first two codes namely

```
1 if $m[i,j] < \infty$
2     return $m[i,j]$
```

are removed, what will happen ? (2pts)

Answer

this part code is called by memoization step, in this function. The purpose is to check whether the m[i,j] is already calculated or not, if this already calculated then the below the line should not executed implies more efficient.

if that lines removed so, the same subproblems recomputed multiple times, leading inefficiency.

Problem 4.2 Consider a variant of the matrix-chain multiplication problem in which this goal is parethesize the sequence of matrices so as to maximize, rather than minimize, the number of scalar multiplication. Does this problem exhibit optimal substructure.

Answer

base on lookup chain algorithm, we can change the line

if $q < m[i,j]$ to if $q > m[i,j]$

as we know the q:

$$q = LC(p,i,k) + LC(p,k+1,j) + p(i-1)p(k)p(j)$$

this shows that q is depends on the subproblems solution, hence this problem exhibits optimal substructures because the optimal solution can be constructed from the optimal solutions to it's subproblems.

Problem 4.3 Determine an LCS of {1,0,1,0,0,1,0,1} and {0,1,0,1,1,0,1,1,0}, moreover as discussed in our class, generalize the recursive formula used in the textbook Eq. 15.9 p(393) to the following one:

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ max(c[i-1,j-1] + (x_i == y_j), c[i,j-1], c[i-1,j]) & \text{if } i,j > 0 \end{cases}$$

($x_i == y_i$ is 1 if $x_i = y_i$ or 0 if not), so that all LCS's of two sequences can be found. Use the recursive formula to find all the LCS's. (2pts)

Answer

to more visualize the answer and the visualize is more way better in common way, then the LCS from the table is given by $\{0,1,0,1,0,1\}$

c[i,j]	-1	1	0	1	0	0	1	0	1
-1	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1	1	1
1	0	1	1	2	2	2	2	2	2
0	0	1	2	2	3	3	3	3	3
1	0	1	2	3	3	3	4	4	4
1	0	1	2	3	3	3	4	4	5
0	0	1	2	3	4	4	4	5	5
1	0	1	2	3	4	4	5	5	6
1	0	1	2	3	4	4	5	5	6
0	0	1	2	3	4	5	5	6	6

Problem 4.4 Give an $O(n^2)$ algorithm to find the longest monotonically increasing subsequence of a sequence o n numbers. Discuss two cases separately: (1) simple increasing subsequences (2) strictly increasing subsequences. (2pts)

Answer

```python
def LIS(arr):
    n = len(arr)
    dp = [1] * n

    for i in range(1, n):
        for j in range(i):
            if arr[i] >= arr[j]:
                dp[i] = max(dp[i], dp[j] + 1)

    return max(dp)
```

the code for simple increasing subsequences, and stricly increasing are same, because every stricly increasing sequence is also increasing sequence it's trivial. if we want separatly differ the case we can change on this code part

```
if arr[i] >= arr[j] (for increasing subsequence)
if arr[i] > arr[j] (for stricly increasing subsequence)
```

Problem 4.5 Not just any greedy approach to the activity-selection problem produces a maximum-size set of mutually compatible activities. Give an example to show that the approach of selecting the activity of least duration from among those that are compatible with previously selected activities does not work. Do the same for the approaches of always selecting the compatible activity that overlaps the fewest other remaining activities and always selecting the compatible remaining activity with the earliest start time. (2pts)

(a) an example that always selecting least duration activity, when it can (compatible), this approach will disprove by given a counter example.

Activity i	Starting Time	Finishing Time	Duration Time
1	3	6	3
2	1	5	4
3	5	9	4

The solution obtained by this approach is {1}, but the optimal solution should be {2, 3}.

(b) an example that always selecting the overlaps the fewest wit other remaining activities is not always best solution

Activity i	Starting Time	Finishing Time	overlaps
1	3	5	2
2	0	2	2
3	6	8	2
4	2	4	3
5	4	6	3
6	1	3	3
7	1	3	3
8	5	7	3
9	5	7	3

the solution obtained by this approach is {1,2,3} bu the optimal solution should be {2,3,4,5}

(c) an example for always selecting compatible remaining activity with the earliest start time.
the optimal solution by this approach is {1}, but the optimal solution should be {2,3}

Activity i	Starting Time	Finishing Time
1	1	5
2	2	3
3	4	5

Problem 4.6 Reformulate the minimum coin use problem that we discussed in Chapter 15 to a Knapsack problem by using the similar formulation in Slide no. 18? (2pts)

Answer

Knapsack Capacity(W): T

Items

(a) 5 items with weight for each item $w_i \in \{1, 5, 10, 21, 25, 50\}$ just an example

(b) value of item $i : v_i = 1$

Objective: Minimize the total value number of coins used to fill Knapsack capacity

the objective function of this problem is given by:

$$min \sum_{i=1}^{5} x_i$$

This represents minimizing the total number of coins used.

Contraint:

$$\sum_{i=1}^{5} x_i.d_i = T \qquad x_i \in \mathbb{Z}^+ \cup \{0\},$$

This ensures that the total value of the coins equals the target amount.

initialization:

$$DP(0) = 0 \quad \text{0 Coins are needed to make amount 0}$$

Recursive formula:

$$DP(T) = \min_{d_i < T} DP(T - d_i) + 1$$

Algorithm

```
initialize an array 'DP' of size T+1 with infinity
except 'DP[0]', which is zero
for i from 1 to T:
    for each coin denomination d_i \in {1,5,10,21,25}
        if d_i <= T
            DP(t) = min(DP(t),DP(t-d_i)+1)
the result is DP(T)
```

the code

```
def min_coins(coins, target):
    # Initialize a DP table with maximum value
    dp = [float('inf')] * (target + 1)

    # Base case: 0 coins needed to make amount 0
    dp[0] = 0

    # Iterate over each amount from 1 to target
    for t in range(1, target + 1):
        # Try each coin denomination
        for coin in coins:
            # If the coin denomination is less than or equal to the current amount
            if coin <= t:
                # Update dp[t] with the minimum of current value and the value if
                    this coin is used
                dp[t] = min(dp[t], dp[t - coin] + 1)

    # Return the minimum number of coins needed to make the target amount
    return dp[target]

    # Test the function
    coins = [1, 5, 10, 21, 25]
    target = 63
    print("Minimum number of coins needed:", min_coins(coins, target))
```