COMPETITIVE PROGRAMMING

Assignment-02

2303A51463

B-07

Assignment I: Fibonacci using Memoization

Imagine a financial planning software that predicts future savings growth based on a recursive formula similar to the Fibonacci sequence. If the software calculates Fibonacci numbers using plain recursion, it repeats the same calculations many times, making it slow for large inputs.

Example:
To compute F(10):
- F(10) calls F(9) and F(8)
- F(9) again calls F(8) and F(7)
- F(8) is computed multiple times

Using memoization, once F(8) is computed, it is stored and reused,
making the program fast and efficient.

Sample Input
10
Expected Output:
55
Explanation:
The 10th Fibonacci number is
0 1 1 2 3 5 8 13 21 34 55

Algorithm: Fibonacci using Memoization

1. Read integer **n**.
2. Create an array (or dictionary) **memo** to store already computed Fibonacci values.
3. Define a recursive function **fib(n)**:
    a. If `n == 0`, return `0`.
    b. If `n == 1`, return `1`.
    c. If `memo[n]` is already computed, return `memo[n]`.
    d. Otherwise:
        i. Compute `fib(n-1) + fib(n-2)`.
        ii. Store the result in `memo[n]`.
        iii. Return the stored value.
4. Call `fib(n)`.
5. Print the result.

Code and output:

ass_2..1.py > ...

```python
n=int(input())
memo=[-1]*(n+1)

def fib(n):
    if n==0:
        return 0
    if n==1:
        return 1
    if memo[n]!=-1:
        return memo[n]
    memo[n]=fib(n-1)+fib(n-2)
    return memo[n]

print(fib(n))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PO

```
PS C:\AAC_> & C:/Users/harsh/AppData/Local/Prog
10
55
PS C:\AAC_>
```

Assignment 2: 0/1 Knapsack Problem

Algorithm:

1. Read integers **n** (number of items) and **W** (maximum capacity).
2. Read **n** pairs of integers **(value, weight)** for each item.
3. Create a 2D DP table dp of size `(n+1) × (W+1)`.
   a. `dp[i][w]` = maximum value using first `i` items with capacity `w`.
4. Initialize:
   a. `dp[0][w] = 0` for all `w` (no items → no value).
5. Fill the DP table:
   a. For `i` from `1` to `n`:
      i. For `w` from `0` to `W`:
         1. If `weight[i-1] ≤ w`:
            a. `dp[i][w] = max(dp[i-1][w], value[i-1] + dp[i-1][w - weight[i-1]])`
         2. Else:
            a. `dp[i][w] = dp[i-1][w]`
6. The answer is `dp[n][W]`.
7. Print the result.

Code and output:

ass_2.2.py > ...

```python
n,W=map(int,input().split())
values=[]
weights=[]
for i in range(n):
    v,w=map(int,input().split())
    values.append(v)
    weights.append(w)
dp=[[0]*(W+1) for _ in range(n+1)]
for i in range(1,n+1):
    for w in range(W+1):
        if weights[i-1]<=w:
            dp[i][w]=max(dp[i-1][w],values[i-1]+dp[i-1][w-weights[i-1]])
        else:
            dp[i][w]=dp[i-1][w]

print(dp[n][W])
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\AAC_> & C:/Users/harsh/AppData/Local/Programs/Python/Python314/python.exe c:/AAC
3 50
60 10
100 20
120 30
220
```

Assignment 3: Longest Common Subsequence (LCS)

Algorithm:

1. Read two strings **X** and **Y**.
2. Let **m = length of X**, **n = length of Y**.
3. Create a 2D DP table dp of size `(m+1) × (n+1)`.
    a. `dp[i][j]` stores the length of the LCS of `X[0…i-1]` and `Y[0…j-1]`.
4. Initialize:
    a. `dp[i][0] = 0` for all `i`
    b. `dp[0][j] = 0` for all `j`
5. Fill the table:
    a. For `i` from `1` to `m`:
        i. For `j` from `1` to `n`:
            1. If `X[i-1] == Y[j-1]`:
                a. `dp[i][j] = dp[i-1][j-1] + 1`
            2. Else:
                a. `dp[i][j] = max(dp[i-1][j], dp[i][j-1])`
6. The value `dp[m][n]` is the length of the LCS.
7. Print `dp[m][n]`.

Code and output:

ass_2.3.py > ...

```python
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])

print(dp[m][n])
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\AAC_> & C:/Users/harsh/AppData/Local/Programs/Python/Pyt
AGGTAB
GXTXAYB
4
PS C:\AAC_>
```

Assignment 4: Coin Change – Minimum Coins

Algorithm:

1. Read integer **n** (number of coin denominations).
2. Read **n** integers representing the coin values.
3. Read integer **amount** (total value to be made).
4. Create a DP array dp of size `amount + 1`.
   a. `dp[i]` stores the minimum number of coins needed to make amount `i`.
5. Initialize:
   a. `dp[0] = 0`
   b. For all `i > 0`, set `dp[i] = ∞` (a very large value).
6. For each amount `i` from `1` to `amount`:
   a. For each coin value `c`:
      i. If `c ≤ i`:
         1. `dp[i] = min(dp[i], dp[i - c] + 1)`
7. After filling the array, `dp[amount]` contains the minimum number of coins.
8. Print `dp[amount]`.

Code and output:

ass_2.4.py > ...

```python
n=int(input())
coins=list(map(int,input().split()))
amount=int(input())

dp=[10**9]*(amount+1)
dp[0]=0

for i in range(1,amount+1):
    for c in coins:
        if c<=i:
            dp[i]=min(dp[i],dp[i-c]+1)

print(dp[amount])
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\AAC_> & C:/Users/harsh/AppData/Local/Programs/Python/Pyt
3
1 3 4
6
2
PS C:\AAC_>
```

Assignment 5: Matrix Chain Multiplication

Algorithm: Matrix Chain Multiplication (Dynamic Programming)

1. Read integer **n** (number of matrices).
2. Read **n** integers representing matrix dimensions in an array p,
   where matrix **Ai** has dimensions `p[i−1] × p[i]`.
3. Create a 2D DP table dp of size `n × n`.
   a. `dp[i][j]` stores the minimum number of scalar multiplications
      needed to multiply matrices from **Ai** to **Aj**.
4. Initialize:
   a. `dp[i][i] = 0` for all `i` (single matrix → no multiplication).
5. Consider chain lengths from 2 to `n−1`:
   a. For each starting index `i`:
      i. Let `j = i + length − 1`.
      ii. Set `dp[i][j] = ∞`.
      iii. For each possible split point k from `i` to `j−1`:
         1. Compute cost
            `cost = dp[i][k] + dp[k+1][j] + p[i−1] * p[k] * p[j]`
         2. Update
            `dp[i][j] = min(dp[i][j], cost)`
6. The answer is stored in `dp[1][n−1]`.
7. Print the result.

Code and output:

ass_2.5.py > ...

```python
n=int(input())
p=list(map(int,input().split()))
dp=[[0]*n for _ in range(n)]
for length in range(2,n):
    for i in range(1,n-length+1):
        j=i+length-1
        dp[i][j]=10**18
        for k in range(i,j):
            cost=dp[i][k]+dp[k+1][j]+p[i-1]*p[k]*p[j]
            dp[i][j]=min(dp[i][j],cost)

print(dp[1][n-1])
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\AAC_> & C:/Users/harsh/AppData/Local/Programs/Python/Python3
4
10 20 30 40
18000
PS C:\AAC_>
```