**Analysis and Design of Algorithms**

**Semester III**, Year **2021-22**

**Lab - 6**  Date : 18-11-2021

Name: E. Sai Manoj MIS. No: 112015044 Branch: CSE

**AIM:**

1. Implement to find the nth element of Fibonacci number using

a. brute - force approach

b. using dynamic programming using memorisation, additionally also compute the number of calls computed for both the cases.

c. Using bottom-up approach

2. Write a program to find the longest common sub-sequence using Dynamic Programming (memorization and tabular method).

3. Implement a classic 0/1 knapsack using Dynamic Programming using memorization and tabular method.

**Question 1a:**

**Pseudo Code:**

START

counter <- 0

FUNCTION fibonacci(n):

global counter

counter += 1

IF n <= 1:

RETURN n

ENDIF

RETURN fibonacci(n-1)+fibonacci(n-2)

ENDFUNCTION

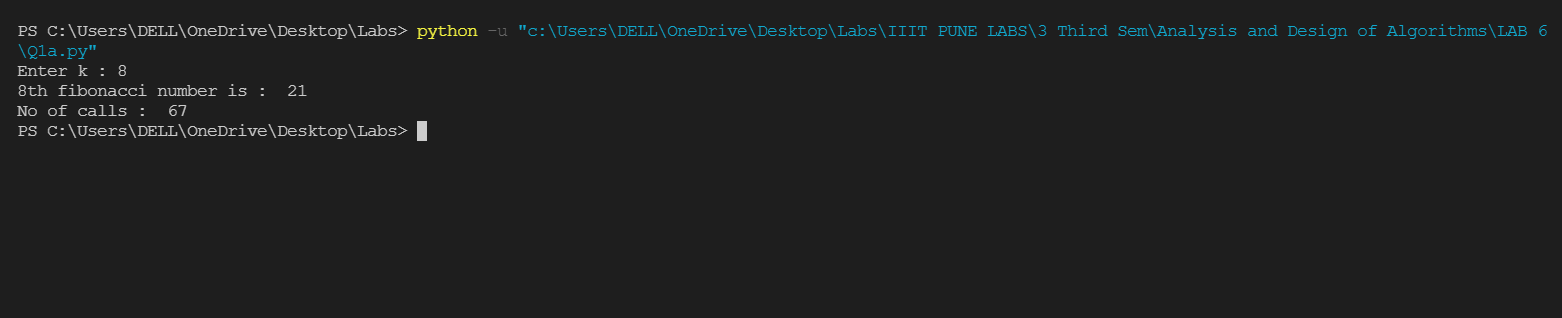
n <- int(input('Enter n : '))

OUTPUT 'nth fibonacci no. is : ',fibonacci(n)

OUTPUT 'No of calls using brute force : ', counter

END

**Output:**

****

**Question 1b:**

**Pseudo Code:**

START

counter <- 0

Fib\_Dict <- [0]\*100

FUNCTION fibonacci\_memo(n):

global counter

counter += 1

IF Fib\_Dict[n] > 0:

RETURN Fib\_Dict[n]

ENDIF

IF n <= 1:

RETURN n

ENDIF

Fib\_Dict[n] <- fibonacci\_memo(n-2) + fibonacci\_memo(n-1)

RETURN Fib\_Dict[n]

ENDFUNCTION

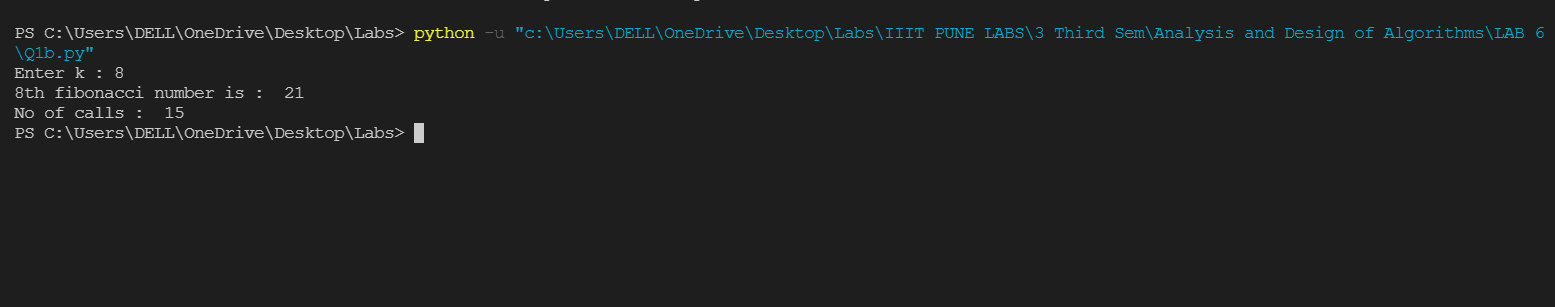
n <- int(input('Enter n : '))

OUTPUT 'nth fibonacci no. is : ', fibonacci\_memo(n)

OUTPUT 'No of calls using memo. : ', counter

END

**Output:**

****

**Question 1c:**

**Pseudo Code:**

START

counter <- 0

FUNCTION fibonacci\_bottom(n):

table <- [0]\*100

table[0] <- 0

table[1] <- 1

for i in range(2, n+1):

table[i] <- table[i-2] + table[i-1]

global counter

counter +=1

ENDFOR

RETURN table[n]

ENDFUNCTION

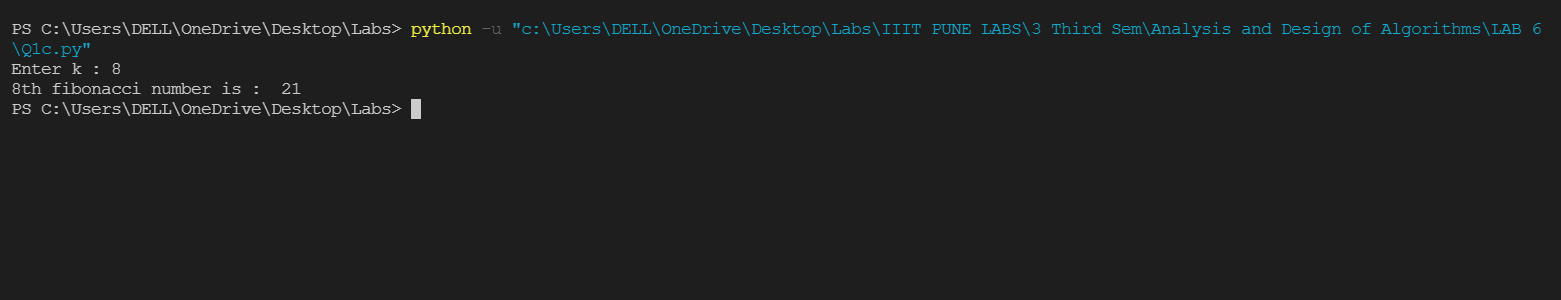
n <- int(input('Enter n : '))

OUTPUT 'nth fibonacci no. is : ',fibonacci\_bottom(n)

OUTPUT 'No of calls using bottom up : ', counter

END

**Output:**

****

**Question 2:**

**Pseudo Code:**

**Memorization approach**

START

CLASS Memorisation:

FUNCTION Memorisation(self, x: str, y: str) -> int:

FUNCTION lcs(i: int, j: int, t=dict()) -> int:

IF i==0 OR j==0:

RETURN 0

ELSE:

key <- (i,j)

IF key not in t:

IF x[i-1] = y[j-1]:

t[key] <- lcs(i-1,j-1,t) + 1

ELSE:

t[key] <- max(lcs(i,j-1,t), lcs(i-1,j,t))

ENDIF

ENDIF

ENDIF

RETURN t[key]

ENDFUNCTION

RETURN lcs(len(x), len(y))

ENDFUNCTION

ENDCLASS

string1 <- input('Enter string 1 : ')

string2 <- input('Enter string 2 : ')

OUTPUT Memorisation().Memorisation(string1, string2)

END

**Tabular approach**

START

CLASS Table:

FUNCTION Table(self, string1: str, string2: str) -> int:

dp <- [[0 for x in range(len(string2)+1)] for y in range(len(string1)+1)]

ENDFOR

IF len(string1) = 0 OR len(string2) = 0:

RETURN 0

ELSE:

for i in range(1,len(string1)+1):

for j in range(1,len(string2)+1):

IF(string1[i-1] = string2[j-1]):

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

ELSE:

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

ENDIF

ENDFOR

ENDFOR

RETURN dp[len(string1)][len(string2)]

ENDIF

ENDFUNCTION

ENDCLASS

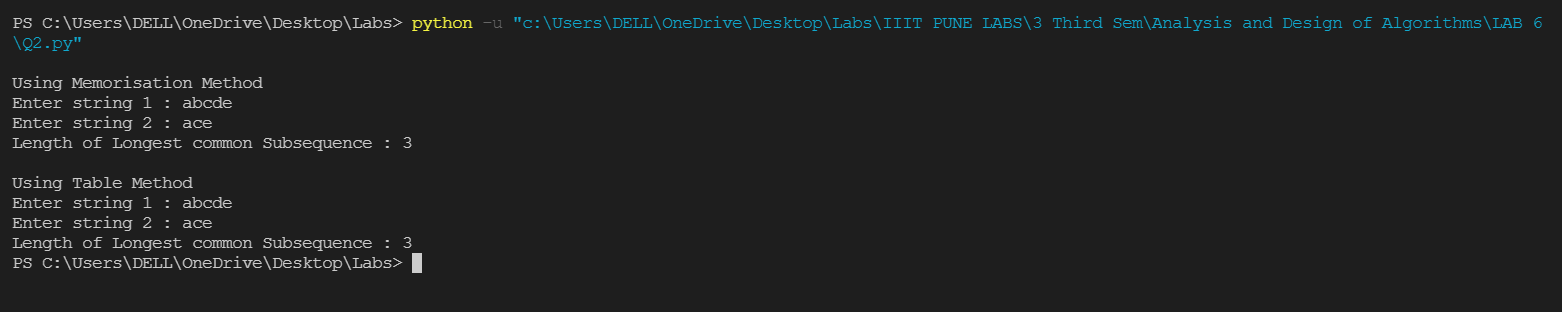
string1 <- input('Enter string 1 : ')

string2 <- input('Enter string 2 : ')

OUTPUT Table().Table(string1, string2)

END

**Output:**

****

**Question 3:**

**Pseudo Code:**

**Memorization approach**

START

FUNCTION knapsack(weights, values, cap, n):

IF n = 0 OR cap = 0:

RETURN 0

ENDIF

IF t[n][cap] != -1:

RETURN t[n][cap]

ENDIF

IF weights[n-1] <= cap:

t[n][cap] <- max(

values[n-1] + knapsack(

weights, values, cap-weights[n-1], n-1),

knapsack(weights, values, cap, n-1))

RETURN t[n][cap]

ELSEIF weights[n-1] > cap:

t[n][cap] <- knapsack(weights, values, cap, n-1)

RETURN t[n][cap]

ENDIF

ENDFUNCTION

values <- list(map(int, input('Enter all the Values : ').split()))

weights <- list(map(int, input('Enter Weights : ').split()))

cap <- int(input('Enter Capacity : '))

n <- len(values)

t <- [[-1 for i in range(cap + 1)] for j in range(n + 1)]

ENDFOR

OUTPUT knapsack(weights, values, cap, n

END

**Tabular approach**

START

FUNCTION Table(cap, weights, value, n):

T <- [[0 for x in range(cap+1)] for x in range(n+1) ]

ENDFOR

for i in range(n+1):

for j in range(cap+1):

IF i = 0 OR j = 0:

T[i][j] <- 0

ELSEIF weights[i-1] <= j:

T[i][j] <- max(value[i-1] + T[i-1][j-weights[i-1]], T[i-1][j])

ELSE:

T[i][j] <- T[i-1][j]

ENDIF

ENDFOR

ENDFOR

RETURN T[n][cap]

ENDFUNCTION

value <- list(map(int, input('Enter all the Values : ').split()))

weights <- list(map(int, input('Enter respective Weights : ').split()))

cap <- int(input('Enter Total Capacity : '))

n <- len(value)

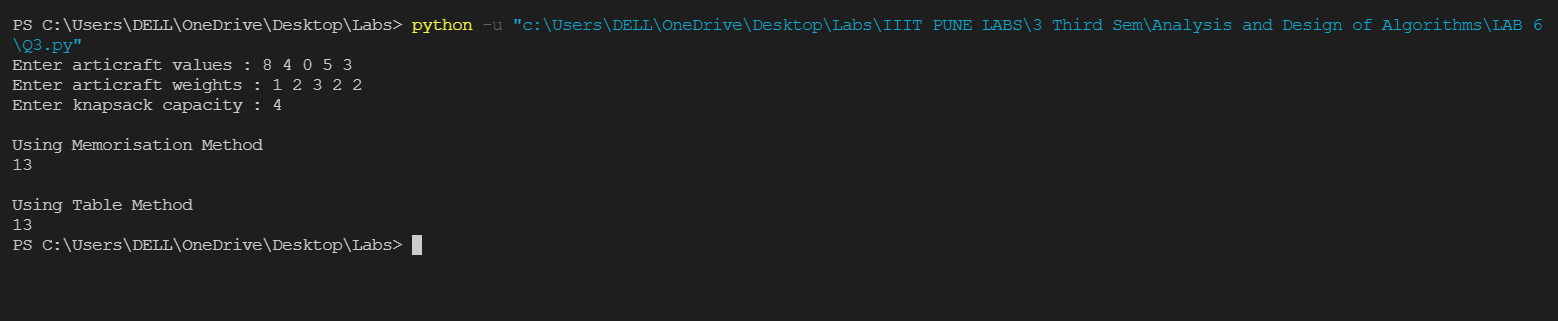
t <- [[-1 for i in range(cap + 1)] for j in range(n + 1)]

ENDFOR

OUTPUT Table(cap, weights, value, n)

END

**Output:**

****