Dept. of Electronics and Electrical Communication Engineering

Indian Institute of Technology Kharagpur

ALGORITHMS
(EC31205)

CODING TASK-4

AUTHOR: JAYA KISHNANI
ROLL NUMBER: 20EC30020
DATE: 13/11/2022
THEME: DYNAMIC PROGRAMMING AND LINEAR PROGRAMMING

Problem 1: Longest Common Subsequence

1) Naïve recursive approach (without memoization)
   Let m and n be the length of the strings.
   - So the base case will be when both the input strings are empty.
   - If the last two alphabets of the strings are same then we return 1 and recursively add the result of m-1 and n-1 remaining alphabets.
   - Else we then return the max value of the recursively computed result between last alphabet of string1 and the remaining alphabets (other than last) of string2 and vice-versa.

   Code snippet

```
def LCS(str1, str2) -> int:
  m = len(str1)
  n = len(str2)
  if(m==0 or n==0):
    return 0
  elif (str1[m-1]==str2[n-1]):
    return 1+LCS(str1[:m-1], str2[:n-1])
  else:
    return max(LCS(str1[:m-1], str2), LCS(str1, str2[:n-1]))
```

   Output:

```
X = "September"
Y = "December"
print(LCS(X,Y))
```

```
6
```

   Time complexity:
   Worst case: exponential : $\boldsymbol{O}(\mathbf{2}^{\boldsymbol{n}})$

2) Recursive approach (with memoization)

   Let m and n be the length of the strings.

   Then to store the previously calculated results (to avoid re-calculation due to recursion and reduce computation complexity) we maintain a 2-D array dp of size (m+1)(n+1).

Code snippet:

```
def LCS2(str1, str2) -> int:
  m = len(str1)
  n = len(str2)
  dp = [[-1 for i in range(n+1)] for j in range(m+1)]
  if(m==0 or n==0):
    return 0;
  if(dp[m][n]!=-1):
    return dp[m][n]
  if(str1[m-1]==str2[n-1]):
    dp[m][n] = 1+LCS2(str1[:m-1], str2[:n-1])
    return dp[m][n]

  dp[m][n] = max(LCS2(str1[:m-1], str2), LCS2(str1, str2[:n-1]))
  return dp[m][n]
```

Result:

```
X = "September"
Y = "December"
print(LCS2(X,Y))
```

```
6
```

Time complexity:

$O(mn)$

3) Dynamic programming (bottom up approach)

Let m and n be the lengths of the string 1 and string 2 respectively.

- In this case we create a 2-D matrix dp of size (m+1)(n+1).
- Then we initialize first row and first column of matrix with zero.
- Then we fill up the table as per the algorithm implemented in the code snippet. i.e. if the alphabets match then we add +1 diagonally else we take the maximum value of the value from top and left.

Code snippet:

```
def LCS3(str1, str2) -> int:
  m = len(str1)
  n = len(str2)
  dp = [[0 for i in range(n+1)]for j in range(m+1)]

  for i in range(m+1):
    for j in range(n+1):
      if (i==0 or j==0):
        dp[i][j]=0
      elif(str1[i-1]==str2[j-1]):
        dp[i][j] = 1 + dp[i-1][j-1]
      else:
        dp[i][j] = max(dp[i-1][j], dp[i][j-1])

  return dp[m][n]
```

Result:

```
X = "September"
Y = "December"
print(LCS3(X,Y))
```

```
6
```

Time complexity:

$O(mn)$

The time taken by the three algorithms are as follows:

```
print('Time taken by Naive recursive approach is: ', time_naive*1000, 'ms')
print('Time taken by recursive approach is: ', time_recur*1000, 'ms')
print('Time taken by dynamic programming is: ', time_dp*1000, 'ms')
```

```
Time taken by Naive recursive approach is:  5.633354187011719 ms
Time taken by recursive approach is:  1.901388168334961 ms
Time taken by dynamic programming is:  0.33402442932128906 ms
```

We can see that dynamic programming approach gives us the solution in optimal time.

Problem 2: Revised Simplex Algorithm

The revised simplex algorithm can be performed as:

Objective:

$$Max\ z = 4x1 + 3x2$$

Constraints:

$$2x1 + 3x2 \leq 6$$
$$-3x1 + 2x2 \leq 3$$
$$2x2 \leq 5$$
$$2x1 + x2 \leq 4$$
$$x1, x2 \geq 0$$

After introducing slack variables, we get:

$$Z - 4x1 - 3x2 = 0$$
$$2x1 + 3x2 + s1 = 6$$
$$-3x1 + 2x2 + s2 = 3$$
$$2x2 + s3 = 5$$
$$2x1 + x2 + s4 = 4$$

For selecting the vector to be entered into basis, we calculate:

$$z_k - c_k = Min\{(z_j - c_j) < 0\} = \min\{(first\ row\ of\ B_1^{-1})(Columns\ a_j\ not\ in\ basis)\}$$

For selecting basic variable to leave basis, we calculate:

$$y_k = B_1^{-1} a_1$$

To calculate the minimum ratio to select the basic variable to leave basis

$$\frac{x_{Br}}{y_{rk}} = \min\left\{\frac{x_{Bi}}{y_{ik}}, y_{ik} > 0\right\}$$

Then we perform row operations.

The output of the following revised simplex algorithm is as follows:

```
time_revised_simplex = end-start
```

```
A =
 [[ 2  3]
 [-3  2]
 [ 0  2]
 [ 2  1]]

b =
 [6 3 5 4]

c =
 [-4 -3]

------------------------------------------
Iteration:  1
[[1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 6. 0. 0.]
 [0. 0. 1. 0. 0. 3. 0. 0.]
 [0. 0. 0. 1. 0. 5. 0. 0.]
 [0. 0. 0. 0. 1. 4. 0. 0.]]
------------------------------------------
Iteration:  2
[[ 1.   0.   0.   0.   2.   8.  -4.   0. ]
 [ 0.   1.   0.   0.  -1.   2.   2.   3. ]
 [ 0.   0.   1.   0.   1.5  9.  -3.   0. ]
 [ 0.   0.   0.   1.   0.   5.   0.   0. ]
 [ 0.   0.   0.   0.   0.5  2.   2.   2. ]]
------------------------------------------
Iteration:  3
[[ 1.          0.5         0.          0.          1.5         9.
  -1.          0.        ]
 [ 0.          0.5         0.          0.         -0.5         1.
   2.          1.        ]
 [ 0.         -1.75        1.          0.          3.25        5.5
   3.5         2.57142857]
 [ 0.         -1.          0.          1.          1.          3.
   2.          2.5       ]
 [ 0.         -0.25        0.          0.          0.75        1.5
   0.5         4.        ]]

Optimized value:
9.0
```

The output of simplex algorithm is as follows:

\+ Code + Text

```
Starting Tableau:
ind         x_0    x_1    s_0    s_1    s_2    s_3
     0.0    4.0    3.0    0.0    0.0    0.0    0.0
2    6.0    2.0    3.0    1.0    0.0    0.0    0.0
3    3.0    -3.0   2.0    0.0    1.0    0.0    0.0
4    5.0    0.0    2.0    0.0    0.0    1.0    0.0
5    4.0    2.0    1.0    0.0    0.0    0.0    1.0
----------------------------------
Iteration : 1
ind         x_0    x_1    s_0    s_1    s_2    s_3
     0.0    4.0    3.0    0.0    0.0    0.0    0.0
2    6.0    2.0    3.0    1.0    0.0    0.0    0.0
3    3.0    -3.0   2.0    0.0    1.0    0.0    0.0
4    5.0    0.0    2.0    0.0    0.0    1.0    0.0
5    4.0    2.0    1.0    0.0    0.0    0.0    1.0
Pivot Column: 2
Pivot Row: 4
Pivot Element:  2.0
----------------------------------
Iteration : 2
ind         x_0    x_1    s_0    s_1    s_2    s_3
     -8.0   0.0    1.0    0.0    0.0    0.0    -2.0
2    2.0    0.0    2.0    1.0    0.0    0.0    -1.0
3    9.0    0.0    3.5    0.0    1.0    0.0    1.5
4    5.0    0.0    2.0    0.0    0.0    1.0    0.0
0    2.0    1.0    0.5    0.0    0.0    0.0    0.5
Pivot Column: 3
Pivot Row: 1
Pivot Element:  2.0
----------------------------------
Iteration : 3
ind         x_0    x_1    s_0    s_1    s_2    s_3
     -9.0   0.0    0.0    -0.5   0.0    0.0    -1.5
1    1.0    0.0    1.0    0.5    0.0    0.0    -0.5
3    5.5    0.0    0.0    -1.75  1.0    0.0    3.25
4    3.0    0.0    0.0    -1.0   0.0    1.0    1.0
0    1.5    1.0    0.0    -0.25  0.0    0.0    0.75
----------------------------------
Final Tableau reached in 3 iterations
ind         x_0    x_1    s_0    s_1    s_2    s_3
     -9.0   0.0    0.0    -0.5   0.0    0.0    -1.5
1    1.0    0.0    1.0    0.5    0.0    0.0    -0.5
3    5.5    0.0    0.0    -1.75  1.0    0.0    3.25
4    3.0    0.0    0.0    -1.0   0.0    1.0    1.0
0    1.5    1.0    0.0    -0.25  0.0    0.0    0.75
```

```
Coefficients:
[1.5 1. ]
Optimal value:
9.0
```

After comparing the time taken by the above two algorithms we observed that revised simplex algorithm performed better as it improves the computational efficiency.

```
Time taken by simplex algorithm:  14.704227447509766 ms
Time taken by revised simplex algorithm 2.381563186645508 ms
```