Dynamic Programming (3 Marks)

Given the two strings:

X ="ABCBDAB"

Y ="BDCABA"

Answer the following:

Define the Dynamic Programming (DP) state used

to solve the Longest Common Subsequence (LCS) problem.

Construct table with size (m+1) X (n+1)

LCS[i][j] = length of the longest common subsequence of the first **i** characters of string **A** and the first **j** characters of string **B** fill all cells in table as follows

i ranges from 0 to len(A)

j ranges from 0 to len(B)

LCS[0][j] = 0 for all j (empty prefix of A)

LCS[i][0] = 0 for all i (empty prefix of B)

case 1. if there is a match add 1 to previses diagonal element

case 2. if there is no a match take max value from either previses row (up) or column (left)

Write the recurrence relation for the LCS problem.

Recurrence (Simple Form)

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

Construct the DP table and compute the length of the LCS.

	0	B	D	C	A	B	A
0	0	0	0	0	0	0	0
A	0	0	0	0	1	1	1
B	0	1	1	1	1	2	2
C	0	1	1	2	2	2	2
B	0	1	1	2	2	3	3
D	0	1	2	2	2	3	3
A	0	1	2	2	3	3	4
B	0	1	2	2	3	4	4

LCS length = dp[7][6]=4

Write one possible LCS sequence.

BCBA

State the time complexity and space complexity of the DP solution.

Time Complexity= O(m×n)

m=length of string X

n=length of string Y

Space Complexity= O(m×n) size of table

Graph Algorithms (2 Marks)

Apply Dijkstra's algorithm on a given weighted graph starting from node A. Show the steps and write the time complexity of the algorithm.

Consider the following weighted graph:

$A \rightarrow B = 4$

$A \rightarrow C = 2$

$B \rightarrow C = 1$

$B \rightarrow D = 5$

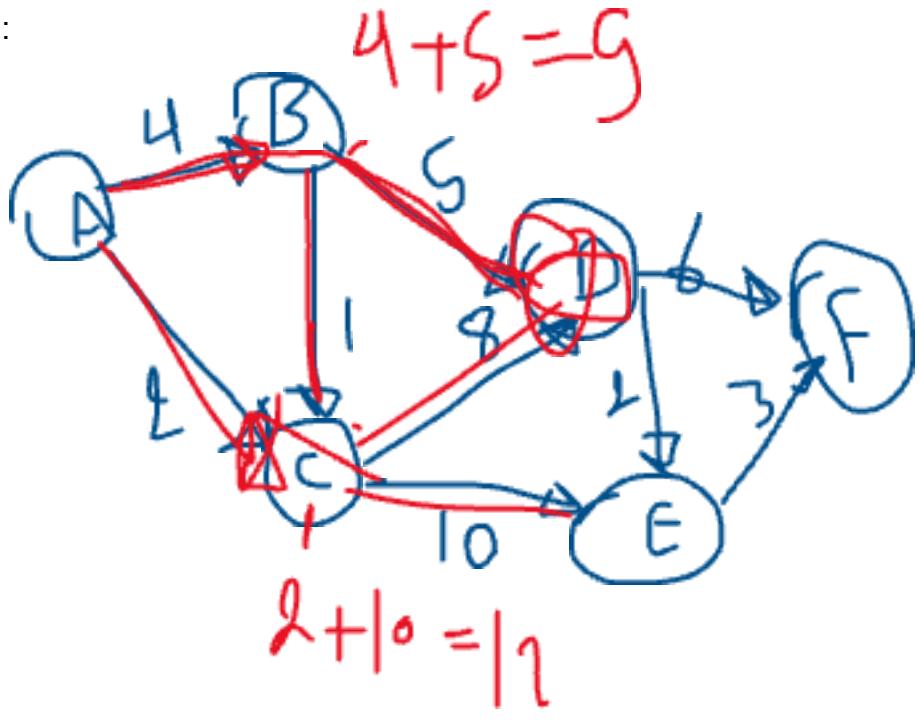
$C \rightarrow D = 8$

$C \rightarrow E = 10$

$D \rightarrow E = 2$

$D \rightarrow F = 6$

$E \rightarrow F = 3$



Answer the following:

Show the distance table after each iteration of the algorithm.

Initial Step (Iteration 0)

Distances:

Node	A	B	C	D	E	F
Dist	0	∞	∞	∞	∞	∞
parent	–	–	–	–	–	–
Selected	–	–	–	–	–	–

Iteration 1

Select **A** (dist 0, min unvisited)

Update neighbors:

A→B: 0+4 = 4 (update B)

A→C: 0+2 = 2 (update C)

Node	A	B	C	D	E	F
Dist	0	4	2	∞	∞	∞
parent	–	A	A	–	–	–
Selected	✓	–	–	–	–	–

Selected this step: A

Iteration 2

Unvisited nodes: B(4), C(2), D(∞), E(∞), F(∞)
Min dist = C (2) → **Select C**

Update from C:

C→D: 2+8 = 10 (update D)

C→E: 2+10 = 12 (update E)

Node	A	B	C	D	E	F
Dist	0	4	2	10	12	∞
parent	–	A	A	C	C	–
Selected	✓	–	✓	–	–	–

Selected this step: C

Iteration 3

Unvisited: B(4), D(10), E(12), F(∞)
Min dist = B(4) → **Select B**

Update from B:

B→C: 4+1=5 (C visited, ignore)

B→D: 4+5=9 → 9 < 10 → update D to 9

Node	A	B	C	D	E	F
Dist	0	4	2	9	12	∞
parent	-	A	A	B	C	-
Selected	✓	✓	✓	-	-	-

Selected this step: B

Iteration 4

Unvisited: D(9), E(12), F(∞)
Min dist = D(9) → **Select D**

Update from D:

D→E: 9+2=11 → 11 < 12 update E to 11

D→F: 9+6=15 → update F to 15

Node	A	B	C	D	E	F
Dist	0	4	2	9	11	15
parent	-	A	A	B	D	D
Selected	✓	✓	✓	✓	-	-

Selected this step: D

Iteration 5

Unvisited: E(11), F(15)
Min dist = E(11) → **Select E**

Update from E:

E→F: 11+3=14 → 14 < 15 → update F to 14

Node	A	B	C	D	E	F
Dist	0	4	2	9	11	14
parent	-	A	A	B	D	E
Selected	✓	✓	✓	✓	✓	-

Selected this step: E

Iteration 6

Unvisited: F(14) → **Select F** (no outgoing edges given)

Node	A	B	C	D	E	F
Dist	0	4	2	9	11	14
Selected	✓	✓	✓	✓	✓	✓

Selected this step: F

Write the final shortest distance from node A to every other

Final shortest distances from A

Node	A	B	C	D	E	F
Dist	0	4	2	9	11	14

A:0, B:4, C:2, D:9, E:11, F:

State the time complexity of Dijkstra's algorithm:

1. Using an adjacency matrix

$O(V^2)$

2. Using a priority queue (binary heap) with an adjacency list

$O((V+E)\log(V)$