Harsimran Singh Dhillon B00983136

Design and Analysis of Algorithms
Theory Assignment - 3.1

Q1]

a] Write down the principle of optimality for the minimum edit distance problem, and prove that the problem satisfies the principle of optimality.

Sol The principle of optimality states that for any two strings X and Y, If we have found the minimum edit distance between them and we know the minimum edit distance between their substrings i.e substring of X and substring of Y. We would be able to construct the optimal solution.

Let MED[i][j] be the min edit distance bet" our 2 strings X[1....i] and Y[1....j].

then minimum edit distance MED[i][j] can be calculated using the below values.

There are 3 operations having 3 different values.

MED[i-1][j] + 1 → for deletion in String

MED[i-1][j-1] + cost → for replacing

MED[i][j-1] + 1 → for insertion in string.

here cost will be 1 as well.

For base case we consider i=0, here the minimum edit distance bet" an empty string X and other string Y[1:j] will be simply the length of other string this is known as MED for insertion.

Same base case will act for the other string, where j=0 so the minimum edit distance will be the length of first string and this will be known as MED for deletion

- To minimize the total edit distance, we choose the minimum value out of the 3 operations i.e (deletion, ~~insertion~~ replacement or insertion).
- By selecting the optimal solution for the subproblem will ensure that MED[i][j] is the minimum edit distance between our two strings X & Y

b] Show the recurrence equation for computing the edit distance. &

Do!

We know that there are 3 operations, having different values. Lets divide them into 3 different cases.

Case I :- Where $i = 0$ or $j = 0$.

then our MED (minimum edit distance) will be

$MED[0][j] = j$

$MED[i][0] = i$

Cond^n for insertion & deletion.

Case II:- When both the strings are equal.

~~MED[i][j] = MED[i][j]~~

If. $X[i] = Y[j]$

$MED[i][j] = MED[i-1][j-1]$

for replacement.

Case III :- When both the strings are not equal.

$X[i] \neq Y[j]$

$MED[i][j] = \min(MED[i-1][j], MED[i][j-1], MED[i-1][j-1])$

Creating recurrence eq^n using these values.

$$MED[i][j] = \begin{cases} j & \text{if } i = 0 \\ i & \text{if } j = 0 \\ MED[i-1][j-1] & \text{if } X[i] = Y[j] \\ \min(MED[i-1][j], MED[i][j-1], MED[i-1][j-1]) + 1 & \text{else.} \end{cases}$$

c) Provide pseudocode for Edit-Distance (S_1, S_2).

Soln

```
EditDistance (S1, S2) {
int lenS1 = S1.length
int lenS2 = S2.length.
int [][] MED = new int [lenS1 + 1] [lenS2 + 1]
  // when j = 0
  for (int i = 0 ; i <= lenS1; i++) {
          MED[i][0] = i;
  }

  // when i = 0
  for (int j = 0; j <= lenS2 ; j++){
          MED[0][j] = j
  }
int cost = 1          // initialize to 1
// when (S1 = S2)
for (int i = 1; i <= lenS1; i++) {
   for (int j = 1; j <= lenS2; j++) {
          if (S1[i-1] == S2[j-1]) {
                  MED[i][j] = MED[i-1][j-1]
                  cost = 0;
          }
```

~~MED[i][j] = min(MED[i-1][j], MED[i][j-1], MED[i-1][j-1] + cost)~~

Else {

MED[i][j] = (min (MED[i-1][j], MED[i][j-1], MED[i-1][j-1] + cost))

return MED[lenS1][lenS2]

d) Use Edit-Distance() to create the table d (d[i,j] is defined above) for S_1 = cats and S_2 = fast. The entry at d[4,4] should show the correct edit distance between the two words.

Do! Given :-

$S_1 \Rightarrow$ "cats"

$S_2 \Rightarrow$ "fast"

EditDistance (S_1, S_2)

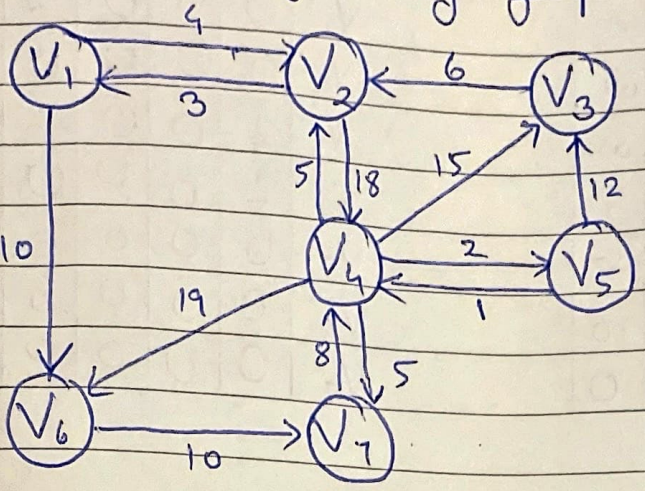
		c	a	t	s
	0	1	2	3	4
f	1	1	2	3	4
a	2	2	1	2	3
s	3	3	2	2	2
t	4	4	3	2	3

as the EditDistance (S_1, S_2) will return MED[lenS1][lenS2] i.e MED[4][4] in this case.

the value will be 3.

Ans So our minimum edit distance will be 3.

q2] Use Floyd's algorithm to find all pairs shortest paths in the following graph.



a] D_0 =

	V_1	V_2	V_3	V_4	V_5	V_6	V_7
V_1	0	4	∞	∞	∞	10	∞
V_2	3	0	∞	18	∞	∞	∞
V_3	∞	6	0	∞	∞	∞	∞
V_4	∞	5	15	0	2	19	5
V_5	∞	∞	12	1	0	∞	∞
V_6	∞	∞	∞	∞	∞	0	10
V_7	∞	∞	∞	8	∞	∞	0

P_0 =

	V_1	V_2	V_3	V_4	V_5	V_6	V_7
V_1	0	0	0	0	0	0	0
V_2	0	0	0	0	0	0	0
V_3	0	0	0	0	0	0	0
V_4	0	0	0	0	0	0	0
V_5	0	0	0	0	0	0	0
V_6	0	0	0	0	0	0	0
V_7	0	0	0	0	0	0	0

D_1 =

	V_1	V_2	V_3	V_4	V_5	V_6	V_7
V_1	0	4	∞	∞	∞	10	∞
V_2	3	0	∞	18	∞	13	∞
V_3	∞	6	0	∞	∞	∞	∞
V_4	∞	5	15	0	2	19	5
V_5	∞	∞	12	1	0	∞	∞
V_6	∞	∞	∞	∞	∞	0	10
V_7	∞	∞	∞	8	∞	∞	0

P_1 =

	V_1	V_2	V_3	V_4	V_5	V_6	V_7
V_1	0	0	0	0	0	0	0
V_2	0	0	0	0	0	1	0
V_3	0	0	0	0	0	0	0
V_4	0	0	0	0	0	0	0
V_5	0	0	0	0	0	0	0
V_6	0	0	0	0	0	0	0
V_7	0	0	0	0	0	0	0

$D_2 =$

	V_1	V_2	V_3	V_4	V_5	V_6	V_7
V_1	0	4	∞	22	∞	10	∞
V_2	3	0	∞	18	∞	13	∞
V_3	9	6	0	24	∞	19	∞
V_4	8	5	15	0	2	18	5
V_5	∞	∞	12	1	0	∞	∞
V_6	∞	∞	∞	∞	∞	0	10
V_7	∞	∞	∞	8	∞	∞	0

$P_2 =$

	V_1	V_2	V_3	V_4	V_5	V_6	V_7
V_1	0	0	0	2	0	0	0
V_2	0	0	0	0	0	1	0
V_3	2	0	0	2	0	2	0
V_4	2	0	0	0	0	2	0
V_5	0	0	0	0	0	0	0
V_6	0	0	0	0	0	0	0
V_7	0	0	0	0	0	0	0

$D_3 =$

	V_1	V_2	V_3	V_4	V_5	V_6	V_7
V_1	0	4	∞	22	∞	10	∞
V_2	3	0	∞	18	∞	13	∞
V_3	9	6	0	24	∞	19	∞
V_4	8	5	15	0	2	18	5
V_5	21	18	12	1	0	31	∞
V_6	∞	∞	∞	∞	∞	0	10
V_7	∞	∞	∞	8	∞	∞	0

$P_3 =$

	V_1	V_2	V_3	V_4	V_5	V_6	V_7
V_1	0	0	0	2	0	10	0
V_2	0	0	0	0	0	1	0
V_3	2	0	0	2	0	2	0
V_4	2	0	0	0	0	2	0
V_5	3	3	0	0	0	3	0
V_6	0	0	0	0	0	0	0
V_7	0	0	0	0	0	0	0

$D_4 =$

	V_1	V_2	V_3	V_4	V_5	V_6	V_7
V_1	0	4	37	22	24	10	27
V_2	3	0	33	18	20	13	23
V_3	9	6	0	24	26	19	29
V_4	8	5	15	0	2	18	5
V_5	9	6	12	1	0	19	6
V_6	∞	∞	∞	∞	∞	0	10
V_7	16	13	23	8	10	26	0

$P_4 =$

	V_1	V_2	V_3	V_4	V_5	V_6	V_7
V_1	0	0	4	2	4	10	4
V_2	0	0	4	0	4	1	4
V_3	2	0	0	2	4	2	4
V_4	2	0	0	0	0	2	0
V_5	4	4	0	0	0	4	4
V_6	0	0	0	0	0	0	0
V_7	4	4	4	0	4	4	0

$D_5 =$

	V_1	V_2	V_3	V_4	V_5	V_6	V_7
V_1	0	4	36	22	24	10	27
V_2	3	0	32	18	20	13	23
V_3	9	6	0	24	26	19	29
V_4	8	5	14	0	2	18	5
V_5	9	6	12	1	0	19	6
V_6	∞	∞	∞	∞	∞	0	10
V_7	16	13	22	8	10	26	0

$P_5 =$

	V_1	V_2	V_3	V_4	V_5	V_6	V_7
V_1	0	0	5	2	4	0	4
V_2	0	0	5	0	4	1	4
V_3	2	0	0	2	4	2	4
V_4	2	0	5	0	0	2	0
V_5	4	4	0	0	0	4	4
V_6	0	0	0	0	0	0	0
V_7	4	4	5	0	4	4	0

$D_6 =$

	V_1	V_2	V_3	V_4	V_5	V_6	V_7
V_1	0	4	36	22	24	10	20
V_2	3	0	32	18	20	13	23
V_3	9	6	0	24	26	19	29
V_4	8	5	14	0	2	18	5
V_5	9	6	12	1	0	19	6
V_6	∞	∞	∞	∞	∞	0	10
V_7	16	13	22	8	10	26	0

$P_6 =$

	V_1	V_2	V_3	V_4	V_5	V_6	V_7
V_1	0	0	5	2	4	0	6
V_2	0	0	5	0	4	1	4
V_3	2	0	0	2	4	2	4
V_4	2	0	5	0	0	2	0
V_5	4	4	0	0	0	4	4
V_6	0	0	0	0	0	0	0
V_7	4	4	5	0	4	4	0

$D_7 =$

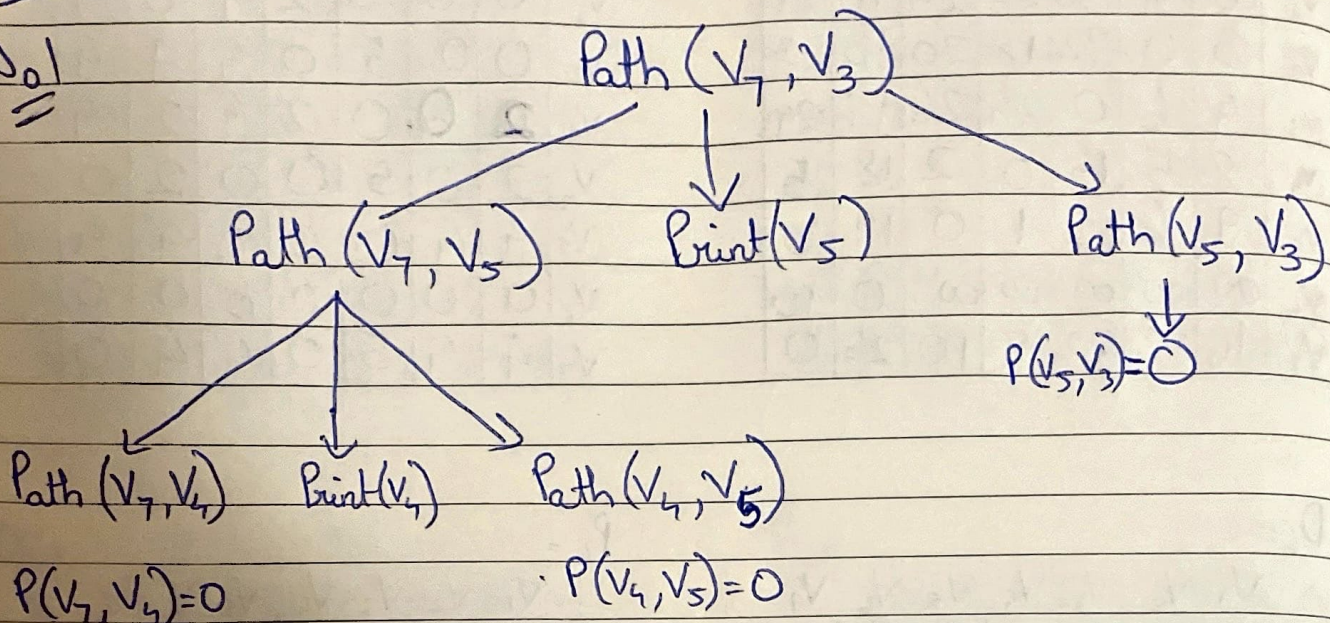
	V_1	V_2	V_3	V_4	V_5	V_6	V_7
V_1	0	4	36	22	24	10	20
V_2	3	0	32	18	20	13	23
V_3	9	6	0	24	26	19	29
V_4	8	5	14	0	2	18	5
V_5	9	6	12	1	0	19	6
V_6	26	23	32	18	20	0	10
V_7	16	13	22	8	10	26	0

$P_7 =$

	V_1	V_2	V_3	V_4	V_5	V_6	V_7
V_1	0	0	5	2	4	0	6
V_2	0	0	5	0	4	1	4
V_3	2	0	0	2	4	2	4
V_4	2	0	5	0	0	2	0
V_5	4	4	0	0	0	4	4
V_6	7	7	7	7	7	0	0
V_7	4	4	5	0	4	4	0

b)

Ans)



The ~~not~~ intermediate nodes on the shortest path from V_7 to V_3 are V_4 & V_5

∴ The shortest path is $V_7 - V_4 - V_5 - V_3$

c)

Ans)

- We ~~have~~ have 7 vertices for which we have implemented the "print shortest path" algorithm.
- The basic operation is told to us as P[i][j] which is used ~~to access~~ by each array to access the elements.
- Since we have 7 vertices ~~of 7 and 7~~ so our matrix size will be $7 \times 7 = 49$.
- Since the print shortest path uses 3 for loops and considering P[i][j] as basic operation, we can consider other operations be performed in the for loop

as 'constant'. So our running time complexity becomes.

$$\underbrace{C}_{\uparrow} \times O(n^3)$$

other operations can be considered as constant.

- Since $n = 7$ over here, Our time complexity becomes $O(7^3)$ [keeping constant aside].
- Time complexity = $O(343)$, this is linear time complexity as the array size or the matrix size is not changing, for this particular example it is 7.
- So, hence we have showed that the Print shortest Path algorithm has linear time complexity ~~only~~ because the vertices are constant.
- As the number of vertices will increase or vary, the linear nature will get affected.