Design and Analysis of Algorithms
CS 575, Spring 2023
Theory Assignment 3.1

1. The edit distance between two strings S1 and S2 is the minimum number of operations to convert one string to the other string. We assume that three types of operations can be used: Insert (a character), Delete (a character), and Replace (a character by another character). For example, the edit distance between dof and dog is 1 (one Replace), between cat and act is 2 (one Delete and one Insert or two Replace), between cat and dog is 3 (3 Replace). Design a dynamic programming algorithm to compute the edit distance between two strings by following the steps below:
   a. [10 points] Write down the principle of optimality for the minimum edit distance problem, and prove that the problem satisfies the principle of optimality.

Solution:

- The principle of optimality states that an optimal solution to a problem contains within it optimal solutions to subproblems.
- In the case of minimum edit distance, this means that the minimum edit distance between two strings S1 and S2 can be computed by finding the minimum edit distance between prefixes of S1 and S2, and using those minimum edit distances to compute the minimum edit distance between S1 and S2.
- For example, suppose we want to find the minimum edit distance between the strings "cat" and "bat". We can break the problem down into smaller subproblems by considering the prefixes of the two strings:
  - The minimum edit distance between the prefixes "" and "" is 0.
  - The minimum edit distance between the prefixes "c" and "" is 1 (one deletion).
  - The minimum edit distance between the prefixes "c" and "b" is 1 (one substitution).
  - The minimum edit distance between the prefixes "ca" and "b" is 2 (one substitution and one deletion).
  - The minimum edit distance between the prefixes "cat" and "b" is 3 (one substitution and two deletions).
- Using the minimum edit distances between prefixes, we can compute the minimum edit distance between "cat" and "bat" as 1 (one substitution). This shows that the problem satisfies the principle of optimality, as the optimal solution to the problem contains within it optimal solutions to subproblems.

b. [10 points] Show the recurrence equation for computing the edit distance. (Hint: Let d[i, j] be the edit distance between the substring of the first i characters of S1 and the substring of the first j characters of S2. Then consider the prefixes of the two strings in a way similar to the analysis for the LCS problem.)

Solution:

1 b) Recurrence Equation :

$$eD(i,j) = \begin{cases} 0 \\ i \\ j \\ eD(i-1, j-1) & \text{if } S_1[i] = S_2[j] \\ \min \begin{Bmatrix} eD(i, j-1) + 1 \\ eD(i-1, j) + 1 \\ eD(i-1, j-1) + cost \end{Bmatrix} & \begin{matrix} \text{if } S1[i] = S2[j] \\ \text{then } cost = 0 \\ \text{else } cost = 1 \end{matrix} \end{cases}$$

c. [10 points] Provide pseudocode for Edit-Distance(S1, S2).

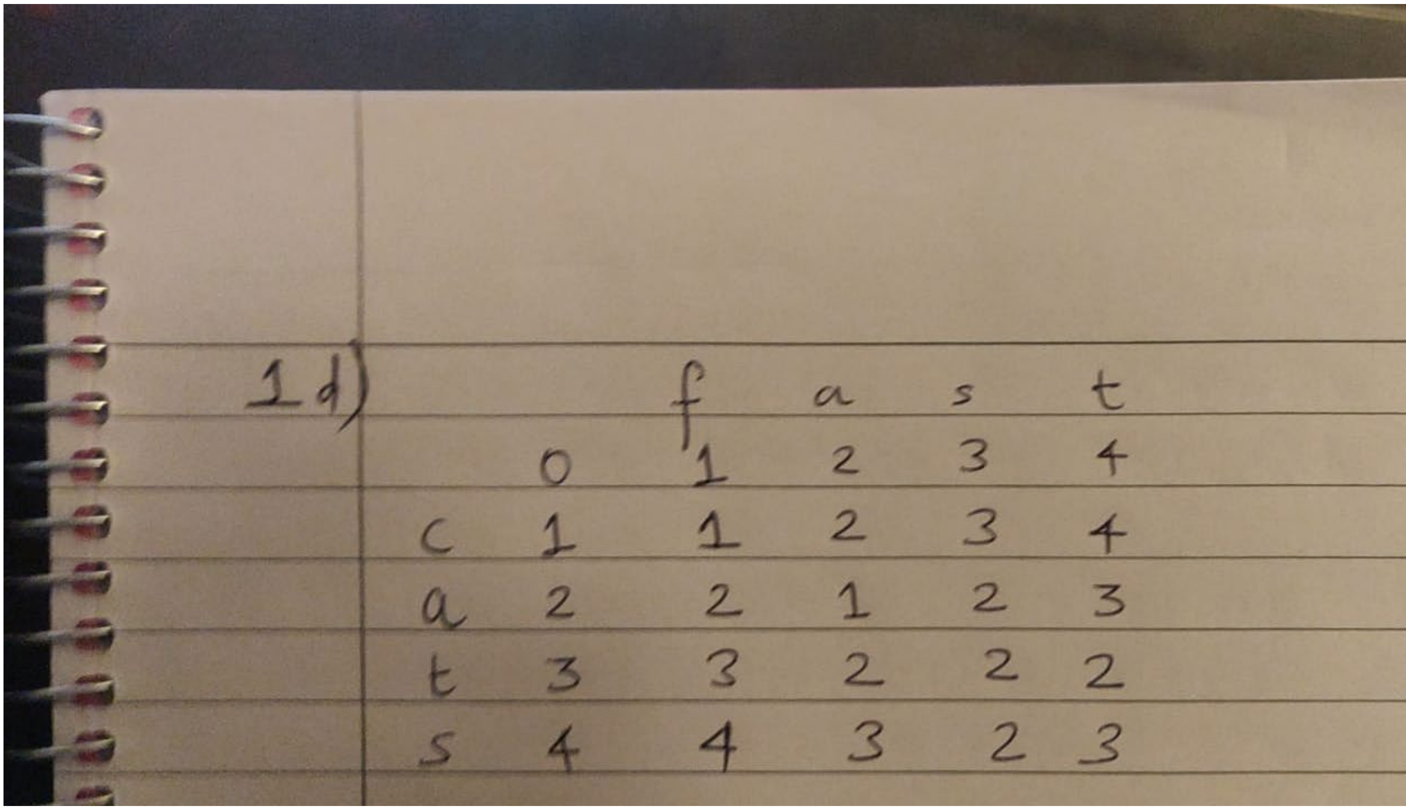
Solution:

1$c)

```
int editDistance (S1, S2)
    x = length(S1)
    y = length(S2)
    eD [x+1] [y+1]

    for i = 0 to x do:
        eD [i] [0] ← i

    for j = 0 to y do:
        eD [0] [j] ← j

    for i = 1 to x do:
        for j = 1 to y do:

            if S1[i-1] == S2[j-1] do:
                eD[i][j] ← eD[i-1] [j-1]

            else do:
                eD [i][j] ← 1 + min(eD[i][j-1], eD [i-1][j]
                                   , eD [i-1] [j-1])

    return eD [x][y]
```

d. [10 points] Use Edit-Distance() to create the table d (d[i, j] is defined above) for S1 = cats and S2 = fast. The entry at d[4, 4] should show the correct edit distance between the two words.

Solution:



1d)

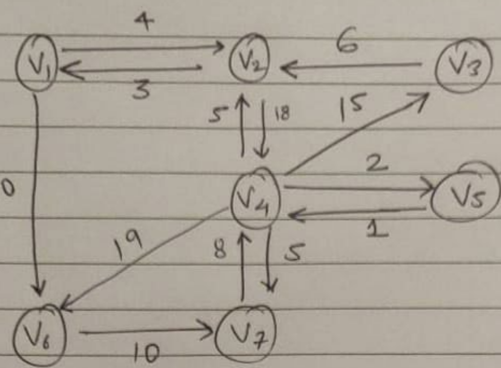
		f	a	s	t
	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	2	2
s	4	4	3	2	3

2. [35 points] Use Floyd's algorithm to find all pairs shortest paths in the following graph.

   a. [15 points] construct the matrix D, which contains the lengths of the shortest paths, and the matrix P, which contains the highest indices of the intermediate vertices on the shortest paths. Show the actions step by step. You need to show D^0 to D^7 and P^0 to P^7 (i.e. matrix P updated along with D step by step). You can use your computer program to output them or do it manually.

Solution:

2 a)



$$D^{k}[i,j] = \min\{D^{k-1}(i,j),\ D^{k-1}[i,k] + D^{k-1}[k,j]\}$$

$$D_0 = \begin{bmatrix} 0 & 4 & \infty & \infty & \infty & 10 & \infty \\ 3 & 0 & \infty & 18 & \infty & \infty & \infty \\ \infty & 6 & 0 & \infty & \infty & \infty & \infty \\ \infty & 5 & 15 & 0 & 2 & 19 & 5 \\ \infty & \infty & 12 & 1 & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & 10 \\ \infty & \infty & \infty & 8 & \infty & \infty & 0 \end{bmatrix} \quad P^0 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$D_1 = \begin{bmatrix} 0 & 4 & \infty & \infty & \infty & 10 & \infty \\ 3 & 0 & \infty & 18 & \infty & 13 & \infty \\ \infty & 6 & 0 & \infty & \infty & \infty & \infty \\ \infty & 5 & 15 & 0 & 2 & 19 & 5 \\ \infty & \infty & 12 & 1 & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & 10 \\ \infty & \infty & \infty & 8 & \infty & \infty & 0 \end{bmatrix} \quad P^1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$D_2 = \begin{bmatrix} 0 & 4 & \infty & 22 & \infty & 10 & \infty \\ 3 & 0 & \infty & 18 & \infty & 13 & \infty \\ 9 & 6 & 0 & 24 & \infty & 19 & \infty \\ 8 & 5 & 15 & 0 & 2 & 18 & 5 \\ \infty & \infty & 12 & 1 & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & 10 \\ \infty & \infty & \infty & 8 & \infty & \infty & 0 \end{bmatrix} \quad P^2 = \begin{bmatrix} 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 2 & 0 & 2 & 0 \\ 2 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$D_3 = \begin{bmatrix} 0 & 4 & \infty & 22 & \infty & 10 & \infty \\ 3 & 0 & \infty & 18 & \infty & 13 & \infty \\ 9 & 6 & 0 & 24 & \infty & 19 & \infty \\ 8 & 5 & 15 & 0 & 2 & 18 & 5 \\ 21 & 18 & 12 & 1 & 0 & 31 & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & 10 \\ \infty & \infty & \infty & 8 & \infty & \infty & 0 \end{bmatrix} \quad P^3 = \begin{bmatrix} 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 2 & 0 & 2 & 0 \\ 2 & 0 & 0 & 0 & 0 & 2 & 0 \\ 3 & 3 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$D_4 = \begin{bmatrix} 0 & 4 & 37 & 22 & 24 & 10 & 27 \\ 3 & 0 & 33 & 18 & 20 & 13 & 23 \\ 9 & 6 & 0 & 24 & 26 & 19 & 29 \\ 8 & 5 & 15 & 0 & 2 & 13 & 5 \\ 9 & 6 & 12 & 1 & 0 & 19 & 6 \\ \infty & \infty & \infty & \infty & \infty & 0 & 10 \\ 16 & 13 & 23 & 8 & 10 & 26 & 0 \end{bmatrix} \quad P^4 = \begin{bmatrix} 0 & 0 & 4 & 2 & 4 & 0 & 4 \\ 0 & 0 & 4 & 0 & 4 & 0 & 4 \\ 2 & 0 & 0 & 2 & 4 & 2 & 4 \\ 2 & 0 & 0 & 0 & 0 & 2 & 0 \\ 4 & 4 & 0 & 0 & 0 & 4 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 4 & 4 & 0 & 4 & 4 & 0 \end{bmatrix}$$

$$D_5 = \begin{bmatrix} 0 & 4 & 36 & 22 & 24 & 10 & 27 \\ 3 & 0 & 32 & 18 & 20 & 13 & 23 \\ 9 & 6 & 0 & 24 & 26 & 19 & 29 \\ 8 & 5 & 14 & 0 & 2 & 18 & 5 \\ 9 & 6 & 12 & 1 & 0 & 19 & 6 \\ \infty & \infty & \infty & \infty & \infty & 0 & 10 \\ 16 & 13 & 22 & 8 & 10 & 26 & 0 \end{bmatrix} \quad P^5 = \begin{bmatrix} 0 & 0 & 5 & 2 & 4 & 0 & 4 \\ 0 & 0 & 5 & 0 & 4 & 0 & 4 \\ 2 & 0 & 0 & 2 & 4 & 2 & 4 \\ 2 & 0 & 5 & 0 & 0 & 2 & 0 \\ 4 & 4 & 0 & 0 & 0 & 4 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 4 & 5 & 0 & 4 & 4 & 0 \end{bmatrix}$$

$$D_6 = \begin{bmatrix} 0 & 4 & 36 & 22 & 24 & 10 & 20 \\ 3 & 0 & 32 & 18 & 20 & 13 & 23 \\ 9 & 6 & 0 & 24 & 26 & 19 & 29 \\ 8 & 5 & 14 & 0 & 2 & 18 & 15 \\ 9 & 6 & 12 & 1 & 0 & 19 & 6 \\ \infty & \infty & \infty & \infty & \infty & 0 & 10 \\ 16 & 13 & 22 & 8 & 10 & 26 & 0 \end{bmatrix} \quad P^6 = \begin{bmatrix} 0 & 0 & 5 & 2 & 4 & 0 & 6 \\ 0 & 0 & 5 & 0 & 4 & 0 & 4 \\ 2 & 0 & 0 & 2 & 4 & 2 & 4 \\ 2 & 0 & 5 & 0 & 0 & 2 & 0 \\ 4 & 4 & 0 & 0 & 0 & 4 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 4 & 5 & 0 & 4 & 4 & 0 \end{bmatrix}$$

$$D_7 = \begin{bmatrix} 0 & 4 & 36 & 22 & 24 & 10 & 20 \\ 3 & 0 & 32 & 18 & 20 & 13 & 23 \\ 9 & 6 & 0 & 24 & 26 & 19 & 29 \\ 8 & 5 & 14 & 0 & 2 & 18 & 5 \\ 9 & 6 & 12 & 1 & 0 & 19 & 6 \\ 26 & 23 & 32 & 18 & 20 & 0 & 10 \\ 16 & 13 & 22 & 8 & 10 & 26 & 0 \end{bmatrix} \quad P^7 = \begin{bmatrix} 0 & 0 & 5 & 2 & 4 & 0 & 6 \\ 0 & 0 & 5 & 0 & 4 & 0 & 4 \\ 2 & 0 & 0 & 2 & 4 & 2 & 4 \\ 2 & 0 & 5 & 0 & 0 & 2 & 0 \\ 4 & 4 & 0 & 0 & 0 & 4 & 4 \\ 7 & 7 & 7 & 7 & 7 & 0 & 0 \\ 4 & 4 & 5 & 4 & 4 & 4 & 0 \end{bmatrix}$$

b. [10 points] Use the Print Shortest Path algorithm (slide 48 of the dynamic programming lecture notes) to find the shortest path from vertex v7 to vertex v3 using the matrix P you constructed from the previous step. Show the actions step by step (either trace the algorithm or show the call tree). You can take the slide 51 as an example of the call tree.

Solution:

2 b) Shortest path from V_7 to V_3

The path is $V_7 \longrightarrow V_4 \longrightarrow V_5 \longrightarrow V_3$

- P_7 : $q = 7$, $r = 3$, callpath $(7, 3)$
- Call path $(7, 5)$
- Call path $(7, 4) \longrightarrow p(q)(r) == 0$, hence we return.
- Printing $\underline{V_4}$
- Calling path $(4, 5) \rightarrow p(q)(r) == 0$, hence we return.
- Returning from call to path $(7, 5)$
- Printing $\underline{V_5}$
- Calling path $(4, 5) \longrightarrow p(q)(r) == 0$, hence we return.
- Returning from call to path $(7, 3)$.
- Complete.

  V_7, V_4, V_5, V_3.

c. [10 points] Analyze the Print Shortest Path algorithm and show that it has a linear-time complexity (input size is the number of vertices in the graph).
(Hint: You can consider each array access to P[i][j] as a basic operation.)

Solution:

2 c) Array access P[i][j] is the basic operation for the print ~~all~~ shortest path.

For vertex V_i to any vertex V_j, the path as, in worst case $(n-1)$ edges., where n is the no. of ~~edges~~ vertices in the graph.

For each $(n-1)$ edge, the algorithm evaluates the basic operation P[i] [j] 4 times, which makes the worst case complexity as $4(n-1)$, that is nothing but ~~$\Theta(1)$~~ $\Theta(n)$, which is linear in nature.