## CS 575: Design and Analysis of Algorithms, Spring 2020

**Solution to Theory Assignment 3.1**

1. [40 points] 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:
   1. [10 points] Write down the principle of optimality for the minimum edit distance problem, and prove that the problem satisfies the principle of optimality.

**Answer**: A subsequence P’ of a sequence P with minimum edit operations between two strings *S*1 and *S*2, is a sequence with minimum edit operations between two substrings *S*1s and *S*2s induced by P’.

If P’ was not the subsequence with minimum edit operations between two substrings, we can substitute the subsequence between two substrings *S*1s and *S*2s in P by the shortest subsequence between the two substrings. The result is a shorter sequence between two strings *S*1 and *S*2 than P. This contradicts our assumption that P is a shortest sequence between two strings *S*1 and *S*2.

* 1. [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 *S*1 and the substring of the first *j* characters of *S*2. Then consider the prefixes of the two strings in a way similar to the analysis for the LCS problem.)

**Answer**: Given two strings *S*1 and *S*2, depending on whether their last characters are the same, the edit distance between them can be reduced to 1 or 0 plus the edit distance between certain prefixes of *S*1 and *S*2 (after removing the last character from one or both of the original strings). Clearly, the edit distance of the prefixes must also be the number of minimum edit operations.

Let *d*[*i*, *j*] denote the edit distance between *S*1[*i*] and *S*2[*j*]. What we need to find is *d*[*m*, *n*]. The recurrence equation for computing *d*[*i*, *j*] is



* 1. [10 points] Provide pesudocode for Edit-Distance(*S*1, *S*2).

**Answer**:

EditDistance(*S*1, *S*2) // *S*1 and *S*2 are input strings

1. *d*[0, 0] = 0;

2. for *i* = 1 to *m*

3. *d*[*i*, 0] = *i* ;

4. for *j* = 1 to *n*

5. *m=d*[0, *j*] = *j* ;

6. for *i* = 1 to *m*

7. for *j* = 1 to *n*

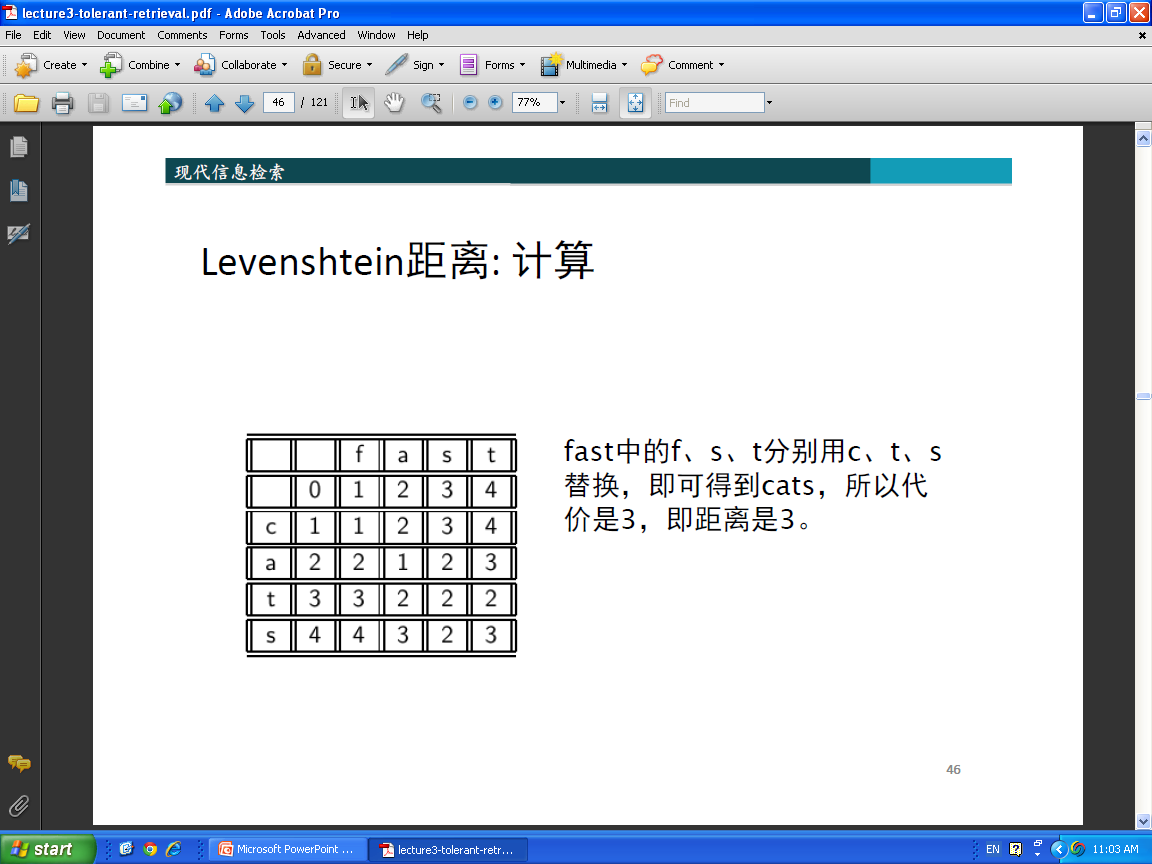
8. *d*[*i*, *j*] = min{ *d*[*i*–1, *j*–1] + (if (*S*1[*i*]==*S*2[*j*]) then 0 else 1),

9. *d*[*i*–1, *j*] + 1, *d*[*i*, *j*–1] + 1 };

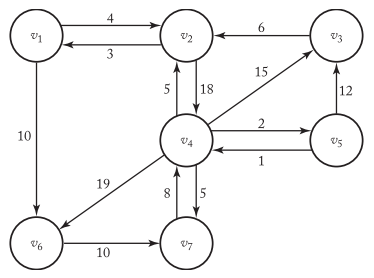
10. return *d*[*m*, *n*];

* 1. [10 points] 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.

**Answer**: Completed table *d*:



1. [35 points] Use Floyd’s algorithm to find all pairs shortest paths in the following graph.



* 1. [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 D0 to D7 and P1 to P7 (i.e. matrix P updated along with D step by step). You can use computer program to output them or do it manually.

Constructing D:

D0:

0 4 INF INF INF 10 INF

3 0 INF 18 INF INF INF

INF 6 0 INF INF INF INF

INF 5 15 0 2 19 5

INF INF 12 1 0 INF INF

INF INF INF INF INF 0 10

INF INF INF 8 INF INF 0

D1:

0 4 INF INF INF 10 INF

3 0 INF 18 INF 13 INF

INF 6 0 INF INF INF INF

INF 5 15 0 2 19 5

INF INF 12 1 0 INF INF

INF INF INF INF INF 0 10

INF INF INF 8 INF INF 0

D2:

0 4 INF 22 INF 10 INF

3 0 INF 18 INF 13 INF

9 6 0 24 INF 19 INF

8 5 15 0 2 18 5

INF INF 12 1 0 INF INF

INF INF INF INF INF 0 10

INF INF INF 8 INF INF 0

D3:

0 4 INF 22 INF 10 INF

3 0 INF 18 INF 13 INF

9 6 0 24 INF 19 INF

8 5 15 0 2 18 5

21 18 12 1 0 31 INF

INF INF INF INF INF 0 10

INF INF INF 8 INF INF 0

D4:

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 18 5

9 6 12 1 0 19 6

INF INF INF INF INF 0 10

16 13 23 8 10 26 0

D5:

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

INF INF INF INF INF 0 10

16 13 22 8 10 26 0

D6:

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

INF INF INF INF INF 0 10

16 13 22 8 10 26 0

D7:

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

Constructing P:

P0:

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

P1:

0 0 0 0 0 0 0

0 0 0 0 0 1 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

P2:

0 0 0 2 0 0 0

0 0 0 0 0 1 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

P3:

0 0 0 2 0 0 0

0 0 0 0 0 1 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

P4:

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

P5:

0 0 5 2 4 0 4

0 0 5 0 4 1 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

P6:

0 0 5 2 4 0 6

0 0 5 0 4 1 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

P7:

0 0 5 2 4 0 6

0 0 5 0 4 1 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 0 4 4 0

* 1. [10 points] Use the Print Shortest Path algorithm 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.

Answer: The shortest path from v7 to v3 is: v7 -> v4 -> v5 -> v3

The call tree similar to slide 51 is OK too.

Steps:

1) Given P7 from Exercise 5

P7:

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 0 4 4 0

and q=7, r=3, call path(7, 3).

2) Call path(7, 5).

3) Call path(7, 4) -> P[q][r]==0, so return.

4) Print v4.

5) Call path(4, 5) -> P[q][r]==0 so return.

6) Return from call to path(7, 5).

7) Print v5.

8) Call path(4 ,5) -> P[q][r] ==0, so return.

9) Return from call to path(7, 3).

10) Done. Intermediate vertices v4 and v5 on shortest path from v7 to v3 have been printed in order.

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

To evaluate complexity, we count the number of array accesses P[i][j].

The path from any vertex q to any vertex r has in the worst case n-1 edges (n is the nr. of vertices in the graph). For each of the n-1 edges, Algorithm 3.4 evaluates P[i][j] four times, so the worst-case complexity is 4(n-1) ∈(n).