

CSE 3101: Solution for Assignment 3
total 64

Solution 1 [23]

1. Array [4] Let $A[i, j]$ be the cost of getting from $X[1 \dots i]$ to $Y[1 \dots j]$ *without using the Kill operation*. Here $0 \leq i \leq m$, $0 \leq j \leq n$.

2. Initialization and Recurrence [6] Let C, R, D, I, T and K denote the cost of **C**opy, **R**eplace, **D**elete, **I**nsert, **T**widdle, and **K**ill, respectively.

Initialization:

$$A[0, 0] = 0, \quad A[i, 0] = iD, \quad A[0, j] = jI$$

(for $1 \leq i \leq m$, $1 \leq j \leq n$).

Recurrence:

First, we compute $A[i, 1]$ and $A[1, j]$:

- For $i \geq 0$:

- If $X[i + 1] = Y[1]$ then

$$A[i + 1, 1] = \min\{A[i, 0] + C, A[i + 1, 0] + I, A[i, 0] + R, A[i, 1] + D\}$$

- Otherwise,

$$A[i + 1, 1] = \min\{A[i + 1, 0] + I, A[i, 0] + R, A[i, 1] + D\}$$

- For $j \geq 1$:

- If $X[1] = Y[j + 1]$ then

$$A[1, j + 1] = \min\{A[0, j] + C, A[1, j] + I, A[0, j] + R, A[0, j + 1] + D\}$$

- Otherwise,

$$A[1, j + 1] = \min\{A[1, j] + I, A[0, j] + R, A[0, j + 1] + D\}$$

Now for $i, j \geq 1$:

- If $X[i + 1] = Y[j + 1]$ then

$$A[i + 1, j + 1] = \min\{A[i, j] + C, A[i + 1, j] + I, A[i, j] + R, A[i, j + 1] + D, A[i - 1, j - 1] + T\}$$

- Otherwise,

$$A[i + 1, j + 1] = \min\{A[i + 1, j] + I, A[i, j] + R, A[i, j + 1] + D, A[i - 1, j - 1] + T\}$$

3. Program to compute A [5]

1. $A[0, 0] \leftarrow 0$

2. For $i = 0$ to m do $A[i, 0] \leftarrow iD$ End For
3. For $j = 0$ to n do $A[0, j] \leftarrow jI$ End For
4. For $i = 0$ to $m - 1$ do % compute $A[i + 1, 1]$
5. If $X[i + 1] = Y[1]$ do
6. $A[i + 1, 1] \leftarrow \min\{A[i, 0] + C, A[i + 1, 0] + I, A[i, 0] + R, A[i, 1] + D\}$
7. Else
8. $A[i + 1, 1] \leftarrow \min\{A[i + 1, 0] + I, A[i, 0] + R, A[i, 1] + D\}$
9. End If
10. End For
11. For $j = 1$ to $n - 1$ do % compute $A[1, j + 1]$
12. If $X[1] = Y[j + 1]$ do
13. $A[1, j + 1] \leftarrow \min\{A[0, j] + C, A[1, j] + I, A[0, j] + R, A[0, j + 1] + D\}$
14. Else
15. $A[1, j + 1] = \min\{A[1, j] + I, A[0, j] + R, A[0, j + 1] + D\}$
16. End If
17. End For
18. For $i = 1$ to $m - 1$ do
19. For $j = 1$ to $n - 1$ do % compute $A[i + 1, j + 1]$
20. If $X[i + 1] = Y[j + 1]$ do
21. $A[i + 1, j + 1] \leftarrow \min\{A[i, j] + C, A[i + 1, j] + I, A[i, j] + R, A[i, j + 1] + D, A[i - 1, j - 1] + T\}$
22. Else
23. $A[i + 1, j + 1] \leftarrow \min\{A[i + 1, j] + I, A[i, j] + R, A[i, j + 1] + D, A[i - 1, j - 1] + T\}$
24. End If
25. End For
26. End For

4. The edit distance [3]

If we don't use **Kill** then the cost is $A[m, n]$. Otherwise, suppose that we can produce Y from $X[1 \dots i]$, then we need one call to **Kill** (the last operation), and the cost will be $A[i, n] + K$. Thus, the edit distance is

$$\min\{A[m, n], A[0, n] + K, A[1, n] + K, \dots, A[m - 1, n] + K\}$$

5. Running Time and Space [5]

The running time of step 3 is $\Theta(mn)$. To find the min, the running time is $\Theta(m)$. Thus, the total running time is $\Theta(mn)$.

We need an array A of size $m \times n$. The space requirement is $\Theta(mn)$.

Solution 2 [17]

1. Array [2] A of length n , where $A[i]$ the the length of the longest path from v_1 to v_i . If there is no path from v_1 to v_i , then $A[i]$ is some default value (here we use -1).

2. Initialization and Recurrence [5] Initially, $A[1] = 0$. For the recurrence, we have to check for the case where there is no path from v_1 to v_i :

- If

$$\max\{A[j] : 1 \leq j < i \text{ and } (v_j, v_i) \in E\} = -1\}$$

then $A[i] = -1$.

- Otherwise,

$$A[i] = 1 + \max\{A[j] : 1 \leq j < i \text{ and } (v_j, v_i) \in E\}$$

3. Program to compute A [5]:

1. $A[1] \leftarrow 0$
2. For i from 2 to n do
3. $A[i] \leftarrow -1$
4. For j from 1 to $i - 1$ do
5. If $(v_j, v_i) \in E$ and $A[j] > -1$ and $A[i] < 1 + A[j]$ do
6. $A[i] \leftarrow 1 + A[j]$
7. End If
8. End For
9. End For

4. Computing a path [5]

1. $B[n] \leftarrow 1$
2. $i \leftarrow n$
3. While $i > 1$ do
4. $j \leftarrow i - 1$
5. While not $((v_j, v_i) \in E$ and $A[j] > -1$ and $A[i] = 1 + A[j])$ do
6. $B[j] \leftarrow 0$
7. $j \leftarrow j - 1$
8. End While % found v_j that comes before v_i in the longest path from v_1 to v_n
9. $i \leftarrow j$
10. $B[i] \leftarrow 1$
11. End While

Solution 3 [12] [Marking scheme: 3pts for setting up the graph, 5pts for the algorithm, 2pts for argument for the correctness, and 2pts for running time argument]

Represent each wrestler by a node, and connect two wrestlers by an edge if they are rivals. Let G be the resulting graph. Now perform a BFS on this graph to label (i.e., “color”) the nodes with two labels (i.e., colors) “good” and “bad”. We start with a node s .

1. For each node v in G , $label[v] \leftarrow NULL$ End For % start with null label (color).
2. $label[s] \leftarrow GOOD$ % give s any label (good).
3. Queue $Q \leftarrow \{s\}$ % FIFO queue with only s
4. While Q is not empty do
5. $v \leftarrow Dequeue(Q)$ % take an element from the queue
6. For each neighbor u of v do
7. If $label[u] = label[v]$ % two neighbors with the same label
8. Output NO
9. Else If $label[u] = NULL$ do % give label to u different from the label of v
10. If $label[v] = GOOD$ do
11. $label[u] \leftarrow BAD$
12. Else
13. $label[u] \leftarrow GOOD$
14. End If
15. End If
16. End For
17. End While

Theorem The algorithm outputs NO if and only if it is impossible to label the wrestlers using two labels “Good” and “Bad”. Also, if such a labelling exists, the algorithm will output one.

Proof This is because the labelling the wrestlers as “Good” or “Bad” is the same as coloring the nodes of the graph with two colors so that no edge joins two nodes of the same color. So the correctness of the algorithm follows from the correctness proof of the algorithm for the 2-Colorability Problem.

Running time The running time of the algorithm $\mathcal{O}(n+r)$, since BFS in G takes time $\mathcal{O}(|V|+|E|)$, where $|V| = n, |E| = r$.

Solution 4 [12] [Marking scheme: 10pts for the algorithm, 2pts for the running time argument.]

For each node v , let $T(v)$ be the total number of shortest path from s to v . (We need to compute $T(t)$). Let $d(v)$ denote the distance between s and v . The idea comes from the following observation:

Observation: Consider the BFS-tree of G rooted at s . Then for each node v on level $\ell + 1$ (that is, $d(v) = \ell + 1$), $T(v)$ is the sum of all $T(u)$, where u is on level ℓ and there is an edge between u and v :

$$T(v) = \sum_{d(u)=\ell \ \& \ (u,v) \in E} T(u) \tag{1}$$

So we will perform a BFS starting at s , and compute $T(v)$ for every node v in G .

1. For each node v in G , $color[v] \leftarrow White$ End For % start with white color.

2. For each node v in G , $d[v] \leftarrow 0$ End For % $d[v]$ for distance from v to s .
3. For each node v in G , $T[v] \leftarrow 0$ End For % $T[v]$ is the total number of shortest sv -paths.
4. $color[s] \leftarrow Gray$ % start at s
5. $T[s] \leftarrow 1$ % there is only 1 “path” from s to s .
6. Queue $Q \leftarrow \{s\}$ % FIFO queue with only s
7. While Q is not empty do
8. $v \leftarrow Dequeue(Q)$ % take an element from the queue
9. For each neighbor u of v do
10. $Endqueue(Q, u)$
11. If $color[u] = White$ % first time seeing u
12. $color[u] \leftarrow Gray$
13. $d[u] \leftarrow d[v] + 1$ % indicate that u is on the next layer
14. $T[u] \leftarrow T[v]$ % each sv -path gives an su -path
15. Else % already seen u . Check if u is on the next layer
16. If $d[u] = d[v] + 1$
17. $T[u] \leftarrow T[u] + T[v]$ % computing the sum in (1)
18. End If
19. End If
20. End For
21. End While

Running time The algorithm runs in time $\mathcal{O}(m + n)$ as in the generic BFS algorithm. Here in the main loop (lines 7–20) we go through each adjacency list once, and perform a constant work on each edge. The loops on lines 1,2,3 go through each nodes once.