# Class 10 :( :Dynamic :programming :)

---

Wednesday, September 22, 2021 9:26 PM

## Previous classes:

We studied the following dynamic programming problems in the previous classes:
i) Shortest Path in DAGs
ii) Longest Increasing Subsequence

## Edit distance:

Are we going to edit distance ????
No I think..
Are we going to calculate distance ???
No ..
IS there something to do with distance ???
:( No ..

Why the name "edit distance" of all the things in the world ???
Well I have no idea on why it is so :( :( :( you may be able to answer it better after learning what this algorithm is about.
Do you see the two words EDIT and DISTANCE ..
If you haven't you can see now.. :)
Are they same??? silly me .. why do I ask such a question when they are clearly of not of same length.
So .. They are not same .. Can we say something about how much they are not same ??
I mean, can we measure the deviation of one word from being similar to other.
What does deviation mean ..??

lets see..
we have word EDIT whose first letter matches with only the last letter while second letter D matches with the first letter of the second word.
Similarly the third letter matches with second letter of the second word..
I think it is too wordy already .. so we see the following illustration which will make it clear.

```
E D I T _ _ _ _
D I S T A N C E
```
1 match. And 7 differences

```
E D I T _ _ _ _ _
_ D I S T A N C E
```
1 match. And 7 differences

```
_ _ _ _ _ _ _ E D I T
D I S T A N C E _ _ _
```
1 match. And 10 differences

```
E D I _ T _ _ _ _
_ D I S T A N C E
```
3 matches and 6 differences

So what do think is a good matching..
the 3 matches and 6 differences case.. but how to get this without trial and error as we did above..

lets say we have a word ( nothing but a string ) X[1.. .. m] and another word Y[1.. ..n] having m and n letters respectively.

Lets see, we want to find minimum difference alignment of these two strings..
Can we divide it and conquer?? Hmm .. sounds right ..

lets take a subproblem of alignment of X[1.. .. i] with Y[1.. .. j]

lets call this difference of letters in the alignment as D(i , j)

I see that there are only there possible ways in which one letter of first is aligned with another letter of the second.

It means that last letters X[i] and Y[j] can be only one of the following three

```
X[i]    or    X[i]    or    _
_             Y[j]          Y[j]
```

Now if we want D(i, j) then we can approach it from D(i, j−1) D(i−1, j) or D(i−1, j−1)

Any reason on choosing these??
Yes ..

When you are looking at the last letters ( here it is X[i] and Y[j] ) of both the strings then it means you have
1) Only one letter left from X[i] and no letters from Y[j] i.e., case D( i−1, j )
2) Only one letter left from Y[i] and no letters from X[j] i.e., case D( i, j−1 )
3) Both letters X[i] and Y[j] are left i.e., case D( i−1, j−1 )

when we approach D(i ,j) from any of them(i.e., the three points metioned above ) we have to check if the current X[i] and Y[j] are same. If they are same then we don't have to add any thing to get D(i, j) else we have to add 1 to the D(a,b) from which we are approaching it. We will take the minimum way to aproach it from all three directions.

i.e., D(i, j) is minimum of the following three
1 + D(i−1, j)
1 + D(i, j−1)
AreSame(X[i],Y[j]) + D(i−1, j−1)

where

AreSame(X[i],Y[j]) = 0 if X[i] = Y[j]
= 1 else

lets compute all the values of D(i,j) for all the values of i and j
initially the value i.e., the base case will be having D(i,0) = i and D(0,j) = j

| | | D | I | S | T | A | N | C | E |
|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| E | 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 7 |
| D | 2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| I | 3 | 2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| T | 4 | 3 | 2 | 2 | 2 | 3 | 4 | 5 | 6 |

The last box has 6 which is the minimum number of differences between the two words. The yellow colored blocks show what we are comparing and number of differences till then in the optimal solution.

The vertical coloring says that we are comparing with blanks with a fixed veritcal column letter.
The horizontal coloring says that we are comparing with blanks with a fixed horizontal row letter.
Diagonal coloring implies that non of them are blanks and we check if they are same or not based on which we update the number current diagonal block.

we see that we have to compute m*n values in the table.
So this method has a complexity of O(mn).

So, lets see what we found in this problem..
We found minimum number of differences between two strings..

what to do with them??
Hmm..

Given one string , I can get ( or transoform this ) to other string by changing ( or modifying or EDITING ) first string minimum number of times.
Hence the word "edit".

The number of differences is the distance between the two strings.
Hence the word "distance" .

Thus, we have the name of the algorithm as "EDIT DISTANCE". `\(^_^)/`

NOT-e :
Hope you all can win the game of edit distance using this algorithm with your young sibblings or friends who are unaware of this nice little algorithm.

(:Don't introduce them to this algorithm immediately )
Help them in developing this algorithm or let them work on it ..
Let them get a feel for it ( steP by step :)
Hope all had a nice feel for it.

- Miryala Narayana Reddy
2020101044
ug2k20