COL100 Assignment 5

2020CS10341

31 January 2021

1 Play with Grid

1.1 Time Complexity

Claim: $\Theta(MN)$ is the time-complexity for every case, where the input grid is of size $M \times N$.

Proof

We construct a matrix `A` of size $M \times N$. This matrix is a list of size n in which each element `A[i]`, $0 \leq i \leq N-1$ is a list of size M. Adding one element to each sub-list `A[i]` takes $O(1)$ time, so, in the end we end up doing a total of $M \times N$ such steps and thus, our time complexity is $\Theta(MN)$. Note that this is the time complexity for *every* such possible case.

1.2 Proofs of Correctness

Assertion 1(Main Assertion): `A[i][j] represents the minimum penalty from the cell (m-i,n-j) to the last cell. So, we wish to calculate A[-1][-1] or A[m-1][n-1] which is equivalent to the least penalty from the cell (1,1) to the last cell.`
We prove this assertion based on the other assertions and invariants.
Base Case: `A[0][0]=grid[-1][-1]`
We know that the total penalty to reach the last cell from the last cell is the value of the penalty of the cell itself. So, our base case holds true.
For the first row of the Matrix A: `When we reach the last row of the grid, we can only move rightwards. So, we add the value of the current cell to A[0][i-1] to get the value of A[0][i].`

For loop 1

Pre-Condition: `Assertion 1 holds for all elements of A[0] till A[0][i-1].`
Post-Condition: `Now, we have the first row of the matrix A.`
Base case of our assumption holds true as initially `i=1` and we do know the value of `A[0][0]`

```
A[0].append(A[0][i-1]+grid[-1][-i-1])
```

This way, we set `A[0][i]=A[0][i-1]+grid[-1][-i-1]`
For the first column of the Matrix A: `When we reach the last column of the grid, we can only move downwards. So, we add the value of the current cell to A[i-1][0] to get the value of A[i][0].`

For loop 2

Pre-Condition: `Assertion 1 holds for all elements A[k][0] till A[i-1][0].`
Post-Condition: `Now, we have the first column of the matrix A.`
Base case of our assumption holds true as the initially `i=1` and we do know the value of `A[0][0]`

```
A.append([A[i-1][0]+grid[-i-1][-1]])
```

This way, we set `A[i][0]=A[i-1][0]+grid[-i-1][-1]`

Assertion 2: `Now we know how to go the last cell from the last row and the last column. So, we fill A row-wise from the top, as the minimum of the elements above, to the left and to the top-left of the current element in A will tell us in which direction we should move (down, right, down-right respectively).`
This is our assertion preceding the *Main* `Outer For Loop`. Note that we have already proved that the first row and column of the matrix `A` is correct. So, our Assertion 2 holds true.

Outer For Loop

Pre-Condition: `First we fill the i-th row and then we fill the next row and so on. Assertion 1 holds for all rows till i-th row.`
Post-Condition: `We have all the rows of the matrix A and A[-1][-1] represents the answer so we return it.`
Base case of our assertion holds true as initially `i=1` and we do know the value of `A[0]`.
Now, we need to prove the correctness of the inner while loop to prove the correctness of the outer loop.

Inner For Loop

Pre-Condition: `Assertion 1 holds for all elements till the i-th row and all elements of i-th row till A[i][j-1].`
Post-Condition: `We have the i-th row of the matrix A.`
Base case of our assumption holds true as initially `j=1` and we do know the value of `A[i][0]`.

```
A[i].append(min(A[i-1][j-1],A[i-1][j],A[i][j-1])+grid[-i-1][-j-1])
```

This results from `Assertion 2` as we take the minimum of the 3 available paths, which are to the right, bottom and to the bottom-right and we add the penalty of the current cell to it.
Now, since we have `A[i]`, we can use the invariants of the outer loop to conclude that we have the full matrix `A` and thus, we can return `A[-1][-1]` as the answer.

2 String Problem

2.1 Time Complexity

Claim: $\Theta((M+1)(N+1))$ is the time-complexity for every case, where the input strings `(A,B)` are of size M, N respectively.

Proof

We construct $M+1$ lists `c` of size $N+1$ each. Adding one element to `c` takes $O(1)$ time, so, in the end we end up doing a total of $(M+1) \times (N+1)$ such steps and thus, our time complexity is $\Theta((M+1)(N+1))$. Note that this is the time complexity for *every* such possible case.

2.2 Proofs of Correctness

2.2.1 vowelcheck

This function compares the input string to each element of the list

```
vowels = ["a","e","i","o","u"]
```

and returns `True` if the input string is a vowel and returns `False` otherwise.

2.2.2 stringProblem

Assertion 1: `c[k] represents the required value if we consider the first k characters of the string B and the all the characters before the current character in consideration for the string A.`
We prove this assertion based on the other assertions and invariants.

For Loop 1

Pre-Condition: `Initially we have not considered any characters of the string A. So c[k] = k, as we just have to insert all the k characters in the correct order and that will take k steps.`
Post-Condition: `Now we know the required output if we do not consider any character of the first string.`
We simply append `i` for $0 \leq i \leq N$ `(len(B)=N)`.

```
c.append(i)
```

Thus, `c` represents the minimum number of steps required when we have not considered any characters of the first string.
Assertion 2: `d represents the list of minimum number of changes required if we take all the characters of the string A including the current character ii.`
We build `d` in such a way so as to ensure that this assertion holds at every point.

Outer For Loop

Pre-Condition 1: `Assertion 1 holds`
Pre-Condition 2: `d represents the list of minimum number of changes required if we take the first characters of the string A till the character ii and d[k] represents minimum number of changes required to convert this part of the string A to the first k characters of string B.`
Post-Condition: `As we consider all elements of the string A, c[-1] represents the minimum number`

of changes required if we want to convert string A to the string made up by considering all the characters of the string B. So, we return c[-1] as the output.
Base case of our assumptions hold true as when we consider the first character of the string A, *Assertion 1* holds for c and for d, we set

```
d = [i+1]
```

as initially, we do not consider any element of the second string. So, to convert A to B (empty string), we will have to delete all the characters and that would take i+1 steps. So, initially d[0]= i+1.
Now, we need to prove the correctness of the inner while loop to prove the correctness of the outer loop.

Inner For Loop

Pre-Condition: All elements of d, that is d[k] represents the minimum number of changes required if we take the first k characters of the second string B and all characters of A till the current character.
Post-Condition: Assertion 2 holds.
Base case of our assumption holds true as initially d[0]= i+1. To find the next element of the list d,

```
if ii == jj:
    d.append(c[j])
else:
    if vowelcheck(ii):
        if vowelcheck(jj):
            d.append(1 + min(d[j],c[j+1],c[j]))
        else:
            d.append(1 + min(d[j],c[j+1]))
    else:
        d.append(1 + min(d[j],c[j+1],c[j]))
```

This results from our assertions and pre-conditions as we if the two characters of the strings are equal, the minimum number of ways changes is required is the same as the minimum number of changes required if we remove these characters from the respective strings and then convert A to B. So, in this case, we set d[j+1]=c[j] by appending c[j] to d (as we already have j number of elements in d, the appended element will be d[j+1].

In case these characters are not equal, we can do 3 operations:
Replace: If we replace the current character by the character required, we make 1 change and thus the minimum number of additional changes required will be c[j] as now we convert the remaining characters of A to remaining characters of B (first j characters).
Insert: If we insert the required character just after the current character, we make 1 change and thus the minimum number of additional changes required will be d[j] as now we convert the characters of A including ii to remaining characters of B (first j characters).
Remove: If we remove the current character ii, we make 1 change and thus the minimum number of additional changes required will be c[j+1] as now we convert the remaining characters of A to characters of B (first j+1 characters).
At each step, we find the *minimum* of the minimum number of additional steps corresponding to each *possible* operation. Let us call this quantity as x. Note that we can only perform the *replace* operation when both ii and jj are both vowels or ii is a consonant. So, we append $1 + x$ to d depending on whether ii and jj are vowels or consonants and this way we get d[j+1].
Now, since we have c and d, we set c=d and continue in this manner for all characters of the first string. In the

end, we see that the post-condition holds and we return `c[-1]` as the answer as at that point, we have considered all elements of the first string and we want the last element of the list `c` (which represents the minimum number of changes required to convert the full string `A` to the full string `B`).