4. Edit distance

Given two string A and B, you can convert one string to another by performing these operations

1. Insert a character

2. Replace a character

3. Delete a character

Minimum number of opearations required to convert is called ‘Edit distance’.

For example

A = “GOAT” , B=“GET” Edit distance is 2. We can convert A to B by deleting O and replacing ‘A’ with ‘E’.

Write a program which takes two strings and returns the edit distance.

1. State

1. State

Parameters

i - last index of A

j - last index of B

Cost function

editDistance(i,j,A,B) - Edit distance to convert substring of A ending at i to substring of B ending at j.

2. Transitions

2. Transitions

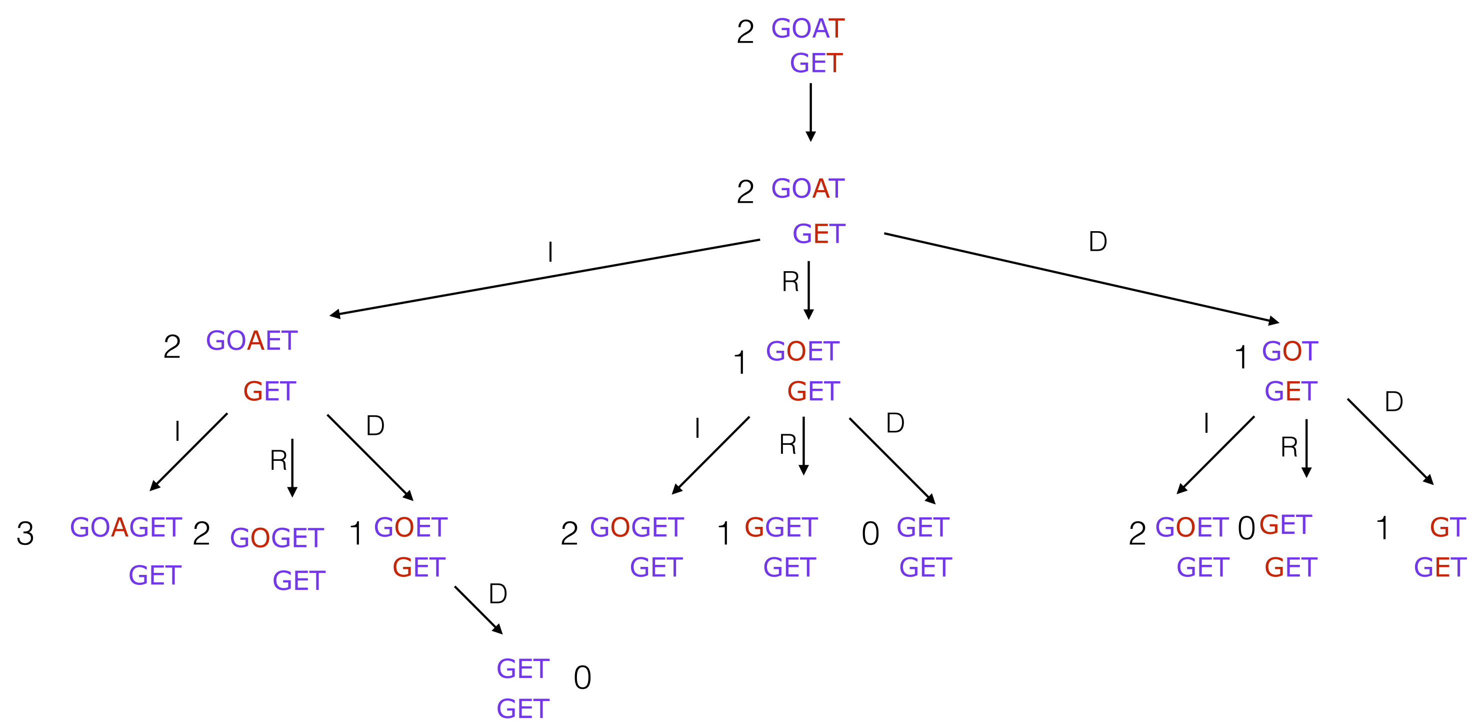
Base case

i = -1, j=-1 , represents empty string

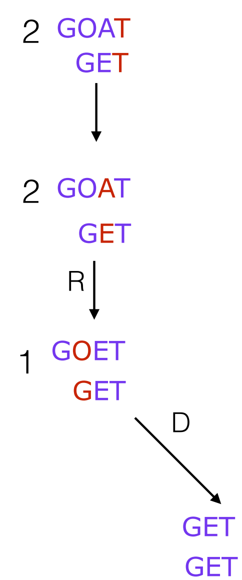
i=-1 , return j+1 , this means A is empty, we can convert it to B by inserting all the j characters of B.

j=-1 , return i+1, this means B is empty , we can convert A ending at i-1 to B which is empty by deleting all the characters.

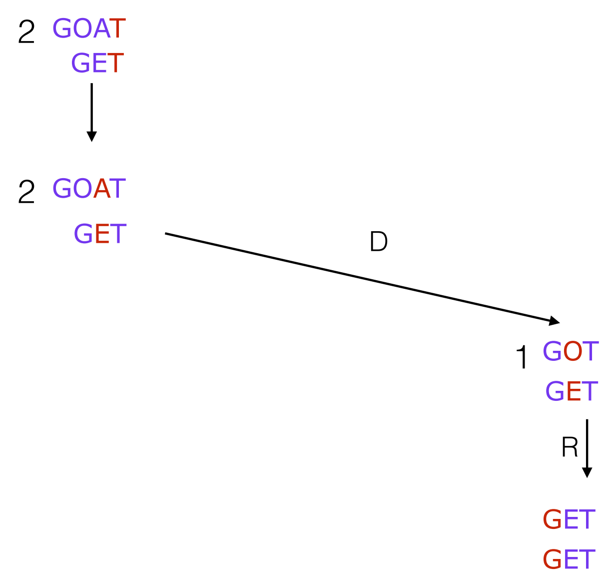
Edit distance



Edit distance



Edit distance



2. Transitions

editDistance(i,j,A,B)

Case 1

If last character of A is equal to last character of B

i.e A[i] == B[j], then we do not need transform, we can move onto smaller subproblem

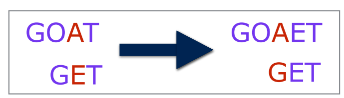
editDistance(i-1,j-1,A,B)

Case 2

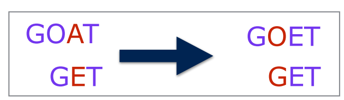
If last character of A is not equal to last character of B

then we have three choices

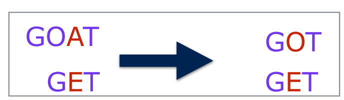
1. Insert - editDistance(i,j-1)+1 ,



2. Replace - editDistance(i-1,j-1)+1



3. Delete - editDistance(i-1,j)+1



2. Transitions

Optimal choice

We are interested in minimum operations required.

We choose the minimum of all the choices.

MIN(editDistance(i,j-1),editDistance(i-1,j-1),editDistance(i-1,j))+1

Recurrence relation

editDistance(i,j,A,B) = j+1 , if i=-1

editDistance(i,j,A,B) = i+1 , if j=-1

editDistance(i,j,A,B) = editDistance(i-1,j-1,A,B) , A[i] = B[j]

editDistance(i,j,A,B) =
MIN(editDistance(i,j-1),editDistance(i-1,j-1),editDistance(i-1,j))+1

3. Recursive solution

3. Recursiuve solution

Pseudo code

```
editDistance(i,j,A,B)
  if i==-1
    return j+1
  if j == -1
    return i+1
  if A[i] == B[j]
     return editDistance(i-1,j-1,A,B)
  else
     return MIN(editDistance(i-1,j),editDistance(i,j-1),editDistance(i-1,j-1))+1
```

Java

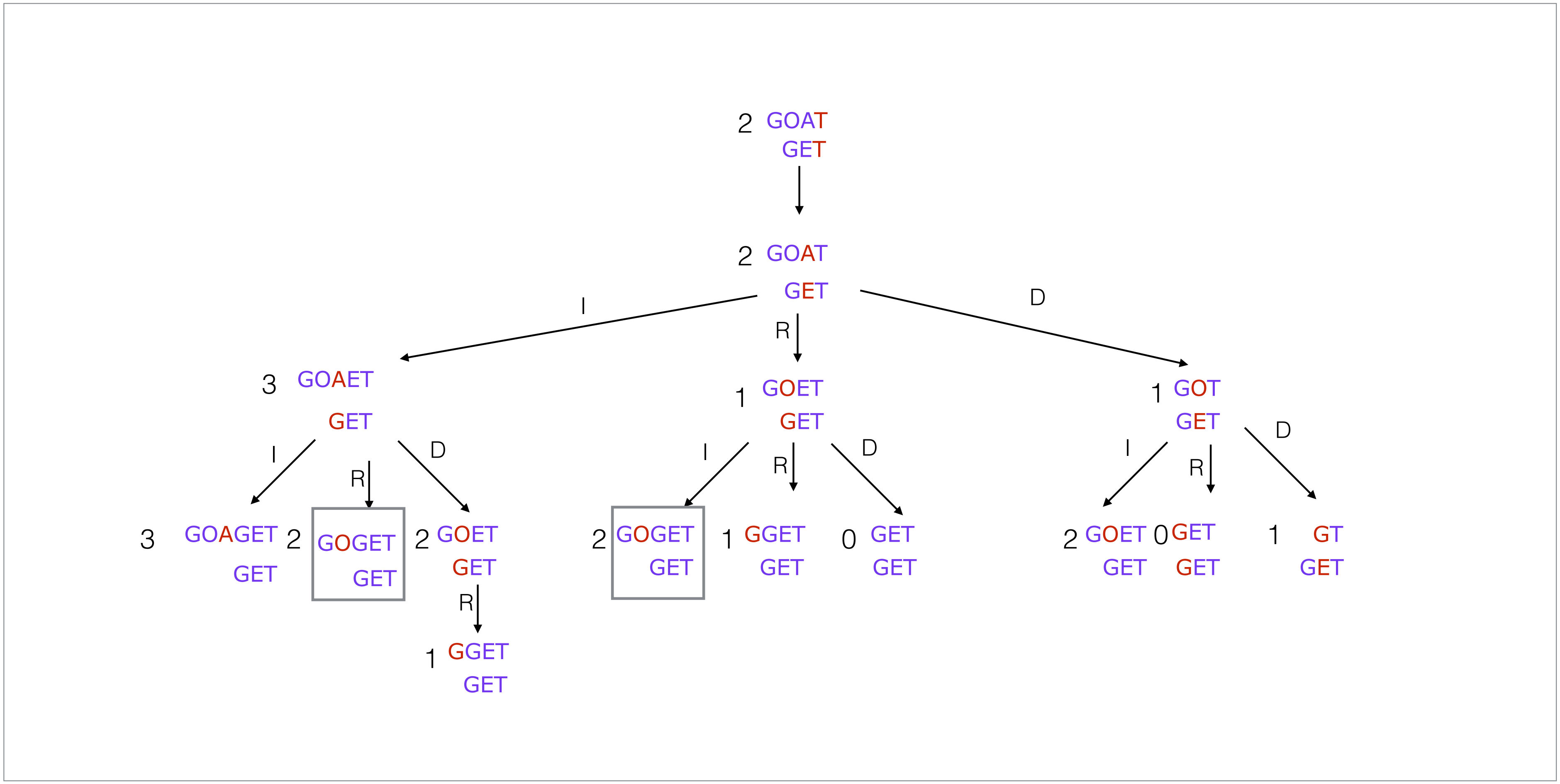
```java
public static int editDistance(int i, int j, String A, String B) {
    if (i == -1) {
        return j+1;
    }
    if (j == -1) {
        return i+1;
    }
    if (A.charAt(i) == B.charAt(j)) {
        return editDistance(i - 1, j - 1, A, B);
    } else {
        return Math.min(editDistance(i, j-1, A, B),
                Math.min(editDistance(i-1, j - 1, A, B),
                        editDistance(i - 1, j, A, B))) + 1;
    }
}
```

Python

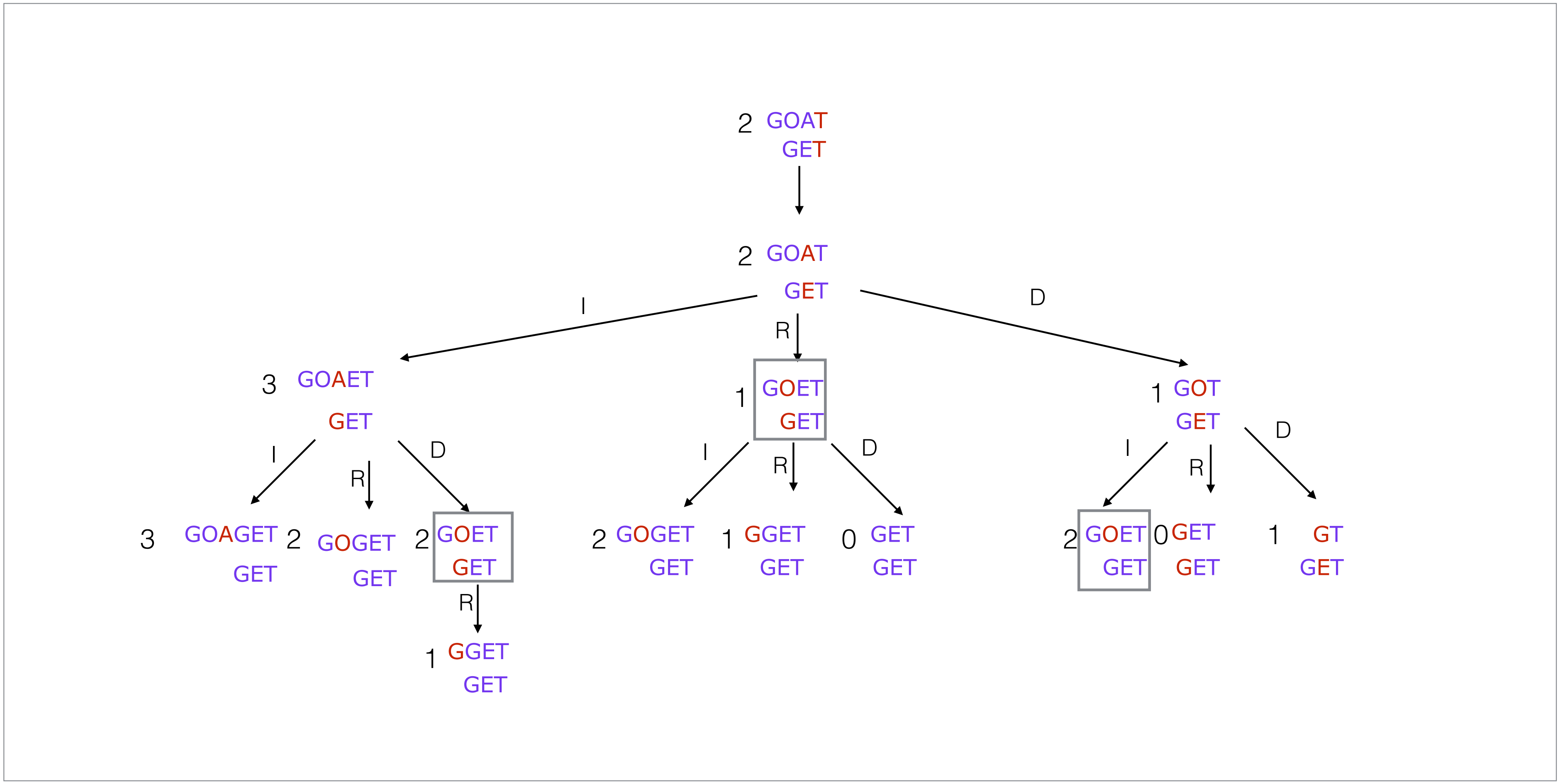
```python
def edit_distance(i, j, A, B):
    if i == -1:
        return j+1
    if j == -1:
        return i+1
    if A[i] == B[j]:
        return edit_distance(i - 1, j - 1, A, B)
    else:
        return min(edit_distance(i, j-1, A, B),
                   min(edit_distance(i-1, j - 1, A, B),
                       edit_distance(i - 1, j, A, B))) + 1
```

4. Memoize

Edit distance



Edit distance



4. Memoize

We can cache the results of the subproblems to avoid solving again and again.

We can use a 2D array of size M X N, M = Length of A, N = Length of B

Key -> (i,j)

Value -> Edit distance to convert A[0:i] to B[0:j]

Default value -> -1

Java

```java
public static int editDistanceMemo(int i, int j, String A, String B, int[][] cache) {
    if (i == -1) {
        return j + 1;
    }
    if (j == -1) {
        return i + 1;
    }
    if (cache[i][j] != -1) {
        return cache[i][j];
    }
    if (A.charAt(i) == B.charAt(j)) {
        int ed = editDistance(i - 1, j - 1, A, B);
        cache[i][j] = ed;
        return cache[i][j];
    } else {
        int ed = Math.min(editDistanceMemo(i, j - 1, A, B, cache),
                Math.min(editDistanceMemo(i, j - 1, A, B, cache),
                        editDistanceMemo(i - 1, j, A, B, cache))) + 1;
        cache[i][j] = ed;
        return ed;
    }
}
```

Python

```
def edit_distance_memo(i, j, A, B, cache):
    if i == -1:
        return j + 1
    if j == -1:
        return i + 1
    if cache[i][j] != -1:
        return cache[i][j]
    if A[i] == B[j]:
        cache[i][j] = edit_distance(i - 1, j - 1, A, B)
        return cache[i][j]
    else:
        cache[i][j] = min(edit_distance_memo(i, j - 1, A, B, cache),
                          min(edit_distance_memo(i - 1, j - 1, A, B,
cache),
                              edit_distance_memo(i - 1, j, A, B,
cache))) + 1
        return cache[i][j]
```

5. Bottom up approach

5. Bottom up approach

Its pretty straight forward to implement a bottom up approach by table filling.

editDistance(i,j,A,B) = j+1 , if i=-1

editDistance(i,j,A,B) = i+1 , if j=-1

editDistance(i,j,A,B) = editDistance(i-1,j-1,A,B) , A[i] = B[j]

editDistance(i,j,A,B) =
MIN(editDistance(i-1,j),editDistance(i,j-1),editDistance(i-1,j-1))+1

dp[i][j] = i , if j==0 , dp[i][j] = j, if i==0

dp[i][j] = dp[i-1][j-1] , if A[i-1] == B[j-1]

dp[i][j] = MIN(dp[i][j-1],MIN(dp[i-1][j-1],dp[i-1][j]))+1

```java
public static int editDistanceDp(String A,String B){
    int M = A.length();
    int N = B.length();
    int[][] dp = new int[M+1][N+1];
    for(int i=0;i<=M;i++){
        for(int j=0;j<=N;j++){
            if(i == 0){
                dp[i][j] = j;
            }else if(j==0){
                dp[i][j]=i;
            }else if(A.charAt(i-1) == B.charAt(j-1)){
                dp[i][j] = dp[i-1][j-1];
            }else{
                dp[i][j] = Math.min(dp[i][j-1],Math.min(dp[i-1][j-1],dp[i-1][j]))+1;
            }
        }
    }
    return dp[M][N];
}
```

Python

```python
def edit_distance_dp(A, B):
    M = len(A)
    N = len(B)
    dp = [[0 for _ in range(0, N + 1)] for _ in range(0, M + 1)]
    for i in range(0, M + 1):
        for j in range(0, N + 1):
            if i == 0:
                dp[i][j] = j
            elif j == 0:
                dp[i][j] = i
            elif A[i - 1] == B[j - 1]:
                dp[i][j] = dp[i - 1][j - 1]
            else:
                dp[i][j] = min(dp[i - 1][j - 1], min(dp[i - 1][j], dp[i][j - 1])) + 1
    return dp[M][N]
```

		0	1	2	3
			G	E	T
0					
1	G				
2	O				
3	A				
4	T				

$$dp[i][j] = j \ , \text{ if } i = 0$$

		0	1	2	3
			G	E	T
0		0			
1	G				
2	O				
3	A				
4	T				

$dp[i][j] = j$, if $i = 0$

		0	1	2	3
			G	E	T
0		0	1		
1	G				
2	O				
3	A				
4	T				

$$dp[i][j] = j \text{ , if } i = 0$$

		0	1	2	3
			G	E	T
0		0	1	2	
1	G				
2	O				
3	A				
4	T				

$$dp[i][j] = j \text{ , if } i = 0$$

		0	1	2	3
			G	E	T
0		0	1	2	3
1	G				
2	O				
3	A				
4	T				

$$dp[i][j] = i \text{ , if } j = 0$$

		0	1	2	3
			G	E	T
0		0	1	2	3
1	G	1			
2	O				
3	A				
4	T				

$$dp[i][j] = dp[i-1][j-1] \text{ , if } A[i-1] == B[j-1]$$

		0	1	2	3
			G	E	T
0		0	1	2	3
1	G	1	0		
2	O				
3	A				
4	T				

dp[i][j] = MIN(dp[i][j-1],MIN(dp[i-1][j-1],dp[i-1][j]))+1

Insert

		0	1	2	3
			G	E	T
0		0	1	2	3
1	G	1	0	1	
2	O				
3	A				
4	T				

$$dp[i][j] = MIN(dp[i][j-1], MIN(dp[i-1][j-1], dp[i-1][j])) + 1$$

Insert

		0	1	2	3
			G	E	T
0		0	1	2	3
1	G	1	0	1	2
2	O				
3	A				
4	T				

$$dp[i][j] = i \text{ , if } j = 0$$

		0	1	2	3
			G	E	T
0		0	1	2	3
1	G	1	0	1	2
2	O	2			
3	A				
4	T				

$$dp[i][j] = MIN(dp[i][j-1], MIN(dp[i-1][j-1], dp[i-1][j])) + 1$$

Delete

		0	1	2	3
			G	E	T
0		0	1	2	3
1	G	1	0	1	2
2	O	2	1		
3	A				
4	T				

$$dp[i][j] = MIN(dp[i][j-1], MIN(dp[i-1][j-1], dp[i-1][j])) + 1$$

Replace

		0	1	2	3
			G	E	T
0		0	1	2	3
1	G	1	0	1	2
2	O	2	1	1	
3	A				
4	T				

$$dp[i][j] = MIN(dp[i][j-1], MIN(dp[i-1][j-1], dp[i-1][j])) + 1$$

Insert

		0	1	2	3
			G	E	T
0		0	1	2	3
1	G	1	0	1	2
2	O	2	1	1	← 2
3	A				
4	T				

$$dp[i][j] = MIN(dp[i][j-1], MIN(dp[i-1][j-1], dp[i-1][j])) + 1$$

Replace

		0	1	2	3
			G	E	T
0		0	1	2	3
1	G	1	0	1	2
2	O	2	1	1	2
3	A				
4	T				

$dp[i][j] = i \text{ , if } j = 0$

		0	1	2	3
			G	E	T
0		0	1	2	3
1	G	1	0	1	2
2	O	2	1	1	2
3	A	3			
4	T				

$$dp[i][j] = MIN(dp[i][j-1], MIN(dp[i-1][j-1], dp[i-1][j])) + 1$$

Delete

		0	1	2	3
			G	E	T
0		0	1	2	3
1	G	1	0	1	2
2	O	2	1	1	2
3	A	3	2		
4	T				

$$dp[i][j] = MIN(dp[i][j-1], MIN(dp[i-1][j-1], dp[i-1][j])) + 1$$

Replace

		0	1	2	3
			G	E	T
0		0	1	2	3
1	G	1	0	1	2
2	O	2	1	1	2
3	A	3	2	2	
4	T				

$$dp[i][j] = MIN(dp[i][j-1], MIN(dp[i-1][j-1], dp[i-1][j])) + 1$$

Delete

		0	1	2	3
			G	E	T
0		0	1	2	3
1	G	1	0	1	2
2	O	2	1	1	2
3	A	3	2	2	
4	T				

$$dp[i][j] = MIN(dp[i][j-1],MIN(dp[i-1][j-1],dp[i-1][j]))+1$$

Delete

		0	1	2	3
			G	E	T
0		0	1	2	3
1	G	1	0	1	2
2	O	2	1	1	2
3	A	3	2	2	2
4	T				

$$dp[i][j] = i \text{ , if } j = 0$$

		0	1	2	3
			G	E	T
0		0	1	2	3
1	G	1	0	1	2
2	O	2	1	1	2
3	A	3	2	2	2
4	T	4			

$$dp[i][j] = MIN(dp[i][j-1], MIN(dp[i-1][j-1], dp[i-1][j])) + 1$$

Delete

		0	1	2	3
			G	E	T
0		0	1	2	3
1	G	1	0	1	2
2	O	2	1	1	2
3	A	3	2	2	2
4	T	4	3		

$$dp[i][j] = MIN(dp[i][j-1], MIN(dp[i-1][j-1], dp[i-1][j])) + 1$$

Replace

		0	1	2	3
			G	E	T
0		0	1	2	3
1	G	1	0	1	2
2	O	2	1	1	2
3	A	3	2	2	2
4	T	4	3	3	

$$dp[i][j] = MIN(dp[i][j-1], MIN(dp[i-1][j-1], dp[i-1][j])) + 1$$

Replace

		0	1	2	3
			G	E	T
0		0	1	2	3
1	G	1	0	1	2
2	O	2	1	1	2
3	A	3	2	2	2
4	T	4	3	3	

$$dp[i][j] = MIN(dp[i][j-1], MIN(dp[i-1][j-1], dp[i-1][j])) + 1$$

Replace

		0	1	2	3
			G	E	T
0		0	1	2	3
1	G	1	0	1	2
2	O	2	1	1	2
3	A	3	2	2	2
4	T	4	3	3	2

		0	1	2	3
			G	E	T
0		0	1	2	3
1	G	1	0	1	2
2	O	2	1	1	2
3	A	3	2	2	2
4	T	4	3	3	2

Replace

		0	1	2	3
			G	E	T
0		0	1	2	3
1	G	1	0	1	2
2	O	2	1	1	2
3	A	3	2	2	2
4	T	4	3	3	2

Delete

		0	1	2	3
			G	E	T
0		0	1	2	3
1	G	1	0	1	2
2	O	2	1	1	2
3	A	3	2	2	2
4	T	4	3	3	2

Delete

		0	1	2	3
			G	E	T
0		0	1	2	3
1	G	1	0	1	2
2	O	2	1	1	2
3	A	3	2	2	2
4	T	4	3	3	2

Delete

		0	1	2	3
			G	E	T
0		0	1	2	3
1	G	1	0	1	2
2	O	2	1	1	2
3	A	3	2	2	2
4	T	4	3	3	2

Replace

		0	1	2	3
			G	E	T
0		0	1	2	3
1	G	1	0	1	2
2	O	2	1	1	2
3	A	3	2	2	2
4	T	4	3	3	2

Delete

		0	1	2	3
			G	E	T
0		0	1	2	3
1	G	1	0	1	2
2	O	2	1	1	2
3	A	3	2	2	2
4	T	4	3	3	2

Replace

		0	1	2	3
			G	E	T
0		0	1	2	3
1	G	1	0	1	2
2	O	2	1	1	2
3	A	3	2	2	2
4	T	4	3	3	2

		0	1	2	3
			G	E	T
0		0	1	2	3
1	G	1	0	1	2
2	O	2	1	1	2
3	A	3	2	2	2
4	T	4	3	3	2

		0	1	2	3
			G	E	T
0		0	1	2	3
1	G	1	0	1	2
2	O	2	1	1	2
3	A	3	2	2	2
4	T	4	3	3	2

Time and space complexity analysis

Time and space complexities

Recursive solution

Time complexity is $O(3^N)$, exponential

Space complexity is $O(1)$

Dynamic programming approach

We use two for loops

outer for loop goes from 0…M

inner for loop goes from 0…N

Time complexity is O(MN)

Space complexity is O(MN)