An Algorithm to find length of the longest sub sequence of given sequence such that all elements of this are alternating using dynamic programming.

DAA ASSIGNMENT-2 , GROUP 21

Saloni Singla
IIB2019004

Sandeep Kumar
IIB2019005

Amanjeet Kumar
IIB2019006

***Abstract*—This Paper contains the algorithm for finding the length of the longest sub sequence of given sequence such that all elements of this are alternating.**

I. Problem Statement

The longest Zig-Zag sub sequence problem is to find length of the longest sub sequence of given sequence such that all elements of this are alternating.
If a sequence {x1, x2, .. xn} is alternating sequence then its element satisfy one of the following relation :
x1 <x2 >x3 <x4 >x5 <.... xn or
x1 >x2 <x3 >x4 <x5 >.... xn
Solve using Dynamic programming.

II. Keywords

Recursion, Dynamic Programming

III. Introduction

Let's first formally define what subsequence is.
A subsequence is a sequence that can be derived from another sequence by zero or more elements, without changing the order of the remaining elements as an example for the array $[1, 2, 3, 4]$ the subsequences are (1), (2), (3), (4), (1,2), (1,3),(1,4), (2,3), (2,4), (3,4), (1,2,3), (1,2,4), (1,3,4), (2,3,4), (1,2,3,4).

IV. Algorithmic Design

A. *Approach 1(Using Recursion)*

1) During recursion, we will determine the state of the problem if
   a) State is 0 then we want an element smaller than it as subsequence is already been started
   b) State is 1 then t we want an element greater than it as subsequence is already been started
   c) State is 2 then we can take any element as subsequence is yet to be started.
2) Then in the Base case, when we have no further elements left. (n==0).
3) After knowing the state, either discard the number or include the number in the current sequence if the previous element is smaller with state = 1 or previous element is greater with state = 0 and if state =2 we take all of the possibilities.

B. *Approach 2(Using DP)*

1) At start, we get to know about the state of the problem whether it is 0, 1 or 2.
2) To store the values of each element, there is a two dimensional matrix of size n x 3 (dp[n][3]). Here 2D matrix will store each element's state in best possible way.
3) If the value of dp[i][state] is precalculated then directly pass on it for further calculation, else calculate dp[i][state] and store it for further use.
4) Finally we return the lenght of the subsequence.

Algorithm 1: To find the length of the longest subsequence as per the given condition.

```
Input:  One array arr with size n
Output: Return the length of the longest zigzag
        subsequence
Function Helper(arr,n,prev,state):
    if n = 0 then
        return 0
    if state <0 then
        a ← INT_MIN
        b ← INT_MIN
        a←Helper(arr, n − 1, prev, state)
        if arr[n − 1] >prev then
            b←1 + Helper(arr, n − 1, arr[n − 1], 1)
        return max(a, b)
    else if state >0 then
        a ← INT_MIN
        b ← INT_MIN
        a←Helper(arr, n − 1, prev, state)
        if arr[n − 1] <prev then
            b←1 + Helper(arr, n − 1, arr[n − 1], −1)
        return max(a, b)
    else
        a ← INT_MIN
        b ← INT_MIN
        c ← INT_MIN
        a←Helper(arr, n − 1, prev, state)
        b←1 + Helper(arr, n − 1, arr[n − 1], −1)
        c←1 + Helper(arr, n − 1, arr[n − 1], 1)
         return max(a, max(b, c))
```

V. Algorithm Analysis

A. *Approach 1(Using recursion)*

Time Complexity Analysis

The time complexity will be $O(2^n)$ because it checks out all the possibilities by recursing.

Space Complexity Analysis

The space complexity is $O(n)$,for storing the input array.

B. *Approach 2(Using DP)*

Time Complexity Analysis

The time complexity will be $O(n * 3)$ because we are in a way doing memoization our approach 1 code and saving time by using values of the precalculated subproblems.

Space Complexity Analysis

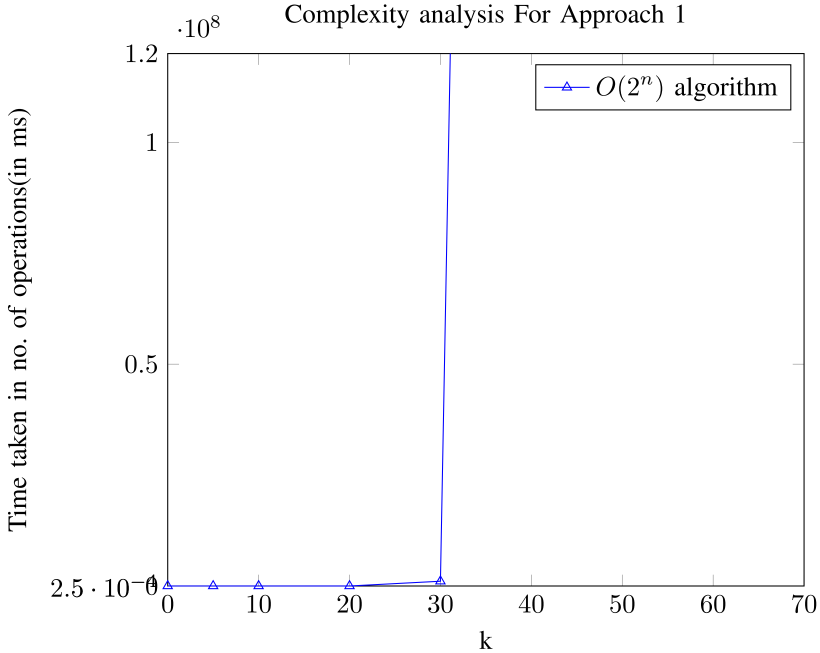
The space complexity will be $O(n)$ for input array and $O(n * 3)$ for storing the values of each condition by dynamic programming.

Algorithm 2: To find the length of the longest subsequence as per the given condition.

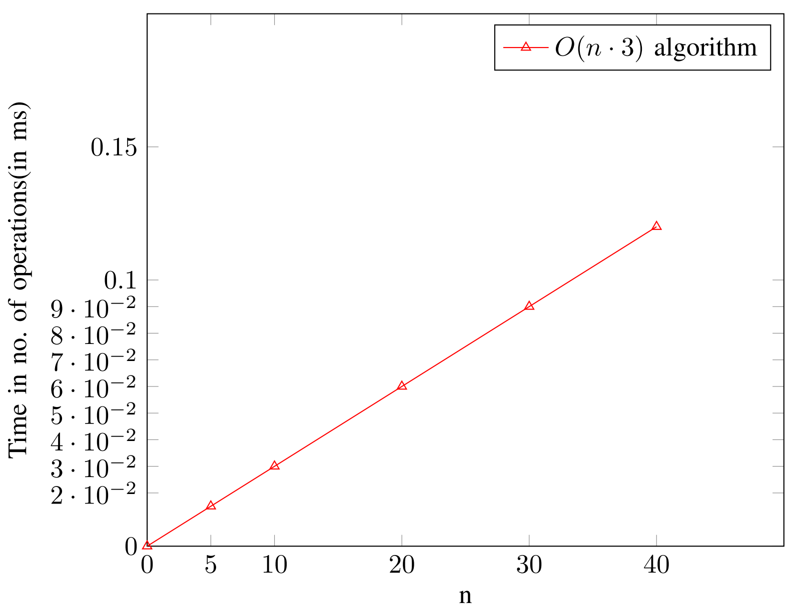
```
Input:  One array arr with size n
Output: Return the length of the longest zigzag
        subsequence
Function Helper(n,prev,state,dp[][],arr):
    if n = 0 then
        return 0
    if dp[n][state] ≠ −1 then
        return dp[n][state]
    if state = 0 then
        a ← INT_MIN
        b ← INT_MIN
        a←Helper(n − 1, prev, state, dp, arr)
        if arr[n − 1] >prev then
            b←1 +
              Helper(n − 1, arr[n − 1], 1, dp, arr)
        dp[n][state]←max (a, b)
        return max(a, b)
    else if state = 1 then
        a ← INT_MIN
        b ← INT_MIN
        a←Helper(n − 1, prev, state, dp, arr)
        if arr[n − 1] <prev then
            b←1 +
              Helper(n − 1, arr[n − 1], 0, dp, arr)
        dp[n][state]←max (a, b)
        return max(a, b)
    else
        a ← INT_MIN
        b ← INT_MIN
        c ← INT_MIN
        a←Helper(n − 1, prev, state, dp, arr)
        b←1 + Helper(n − 1, arr[n − 1], 0, dp, arr)
         c←1 + Helper(n − 1, arr[n − 1], 1, dp, arr)
         dp[n][state]←max (a, max (b, c))
        return max(a, max(b, c))
```

VI. Experimental study

Complexity analysis For Approach 1



Complexity analysis for Approach 2



VII. Conclusion

Above two methods have different time complexities and meet to fulfill the problem statement. The order in which they are good can be listed as:
I. Approach 2
II. Approach 1
Based on the time complexities.

VIII. REFERENCES

1) Utkarsh Trivedi, 'Longest Zig-Zag Subsequence', GeeksforGeeks, 2018. [Online]. [Accessed: 27-Mar-2021]