Detailed analysis and design of the algorithm to find Longest Zig-Zag Subsequence

DAA Assignment 5 - Group 21

Divyesh Rana
IIT2019063

Akash Deep
IIT2019064

Gopal Pedada
IIT2019065

***Abstract*—This paper contains the design and the detailed analysis of the algorithm used to solve the following problem:Find length of the longest subsequence of given sequence such that all elements of this are alternating.**

I. Introduction

The longest zig-zag subsequence problem is to find length of the longest subsequence of given sequence such that all elements of this are alternating. A sequence is called a zig-zag sequence if the differences between successive elements strictly alternate between positive and negative. The first difference (if one exists) may be either positive or negative. A sequence with fewer than two elements is trivially a zig-zag sequence.

If a sequence $(x1, x2, \ldots, xn)$ is a alternative sequence then its element satistfy one of the following relation:

$(x1 < x2 > x3 < x4 > \ldots xn)$ or

$(x1 > x2 < x3 > x4 < \ldots xn)$.

This report further contains -

II. Algorithmic Design

III. Algorithm Analysis

IV. Illustration

V. Experimental Study

VI. Conclusion

II. Algorithmic Design

An array Arr of length n consists of integers is given as an input from the user.

A. longest zig-zag subsequence using sign comparision

We will proceed by comparing the mathematical signs(negative or positive) of the difference of two consecutive elements of Arr. To achieve this, we will store the sign of $(Arr[i] - Arr[i-1])$ in a variable, subsequently comparing it with that of $(Arr[i+1] - Arr[i])$.If it is different, we shall increment our result.

In short, we compare the sign of current difference with the sign of the previous difference and if it is different then we increment the result and update the sign for previous difference and return result.

For checking the sign, we shall use a simple Signum Function, which shall determine the sign of a number passed to it. That is,

$$signum(x) = \begin{cases} 1 & x > 0 \\ -1 & x < 0 \\ 0 & x = 0 \end{cases}$$

B. longest zig-zag subsequence using dynamic programming

This algorithm uses Dynamic Programming.

We define a two dimensional array $dp[n][2]$ such that $dp[i][0]$ contains longest zig-zag subsequence ending at index i and last element is greater than its previous element and $dp[i][1]$ contains longest zig-zag subsequence ending at index i and last element is smaller than its previous element, then we have following recurrence relation between them,

For all $j < i$,

$dp[i][0] = max(dp[i][0], dp[j][1] + 1)$ if $Arr[j] < Arr[i]$,
$dp[i][1] = max(dp[i][1], dp[j][0] + 1)$ if $Arr[j] > Arr[i]$

Note that the minimum length of zig-zag subsequence can be 1, so intialize all the elements of $dp[n][2]$ with 1.

Run a loop with index i from 0 to n and inside that loop, run another loop with index j from 0 to i such that $j < i$ satisfies, now assign values in $dp[i][0]$ and $dp[i][1]$ as per relation shown above. Keep updating $result = max(result, max(dp[i][0], dp[i][1]))$. Return result.

Algorithm 1: Longest zig-zag subsequence using sign comparision

Input: Array Arr of size n
Output: length of longest zig-zag subsequence
Require: $n \geq 0$

```
Function LZS(Arr,n):
    length ← 1
    prevSign ← 0
    for i ← 1 to n − 1 do
        sign ← signum(Arr[i] − Arr[i − 1])
        if sign! = prevSign and sign! = 0 then
            length ← length + 1
            prevSign ← sign
    return length
```

Algorithm 2: Longest zig-zag subsequence using dynamic programming

Input: Array Arr of size n
Output: length of longest zig-zag subsequence
Require: $n \geq 0$

```
Function LZS(Arr,n):
    result ← 1
    for i ← 0 to n − 1 do
        dp[i][0] ← 1
        dp[i][1] ← 1
    for i ← 0 to n − 1 do
        for j ← 0 to i − 1 do
            if Arr[j] < Arr[i] then
                dp[i][0] ← max(dp[i][0], dp[j][1] + 1)
            if Arr[j] > Arr[i] then
                dp[i][1] ← max(dp[i][1], dp[j][0] + 1)
        result = max(result, max(dp[i][0], dp[i][1]))
    return result
```

Algorithm 3: signum function

Input: Integer x
Output: sign

```
Function signum(x):
    sign ← 0
    if x > 0 then
        sign ← 1
    else
        sign ← −1
    return sign
```

III. Algorithm Analysis

A. Time Complexity:

Approach 1: Here, Only single traversal is required to complete the algorithm so the time complexity is $O(n)$.

The best case for this algorithm is $n = 1$, so $T_{best} = O(1)$ and $T_{worst} = O(n)$.

Approach 2: Here, we assume that integer operations take O(1) time. Two loops are used, nested one into other. Therefore, time complexity of this algorithm is $O(n^2)$.

The best case for this algorithm is $n = 1$, so $T_{best} = O(1)$ and $T_{worst} = O(n^2)$.

B. Space Complexity

The space complexity for both algorithms is $O(n)$.

IV. Illustration

Suppose we are given an array Arr = {5, 0, 3, 1, 0}
size of Arr = 5.

A. First Algorithm

Initially $prevSign = 0$ and $result = 1$.

now when $i = 1$, $0 - 5 = -5 < 0$ so $sign = -1$ and $sign \neq prevSign$, result++ and $prevSign = -1$.

when $i = 2$, $3 - 0 = 3 > 0$ so $sign = 1$ and $signprevSign$, result++ and $prevSign = 1$.

when $i = 3$, $1 - 3 = -2 < 0$so $sign = -1$ and again $sign \neq prevSign$, result++ and $prevSign = -1$.

when $i = 4$, $0 - 1 = -1 < 0$so $sign = -1$ but this time $sign = prevSign$, so result does not increment and loop terminates.

The function returns $result = 4$.

B. Second Algorithm

inside first for loop,

when i = 1, $dp[i][0] = 1$ and $dp[i][1] = 2$.

when i = 2, $dp[i][0] = 3$ and $dp[i][1] = 2$.

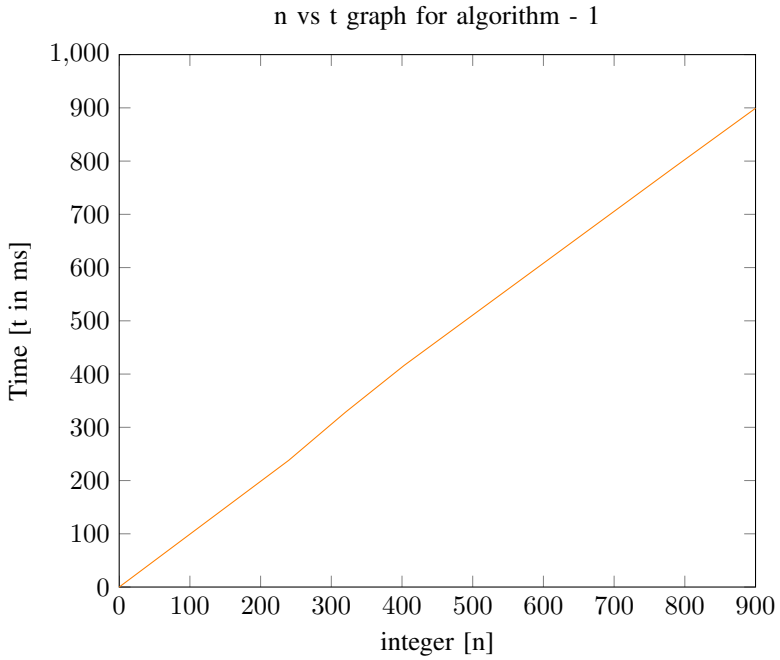
when i = 3, $dp[i][0] = 3$ and $dp[i][1] = 4$.

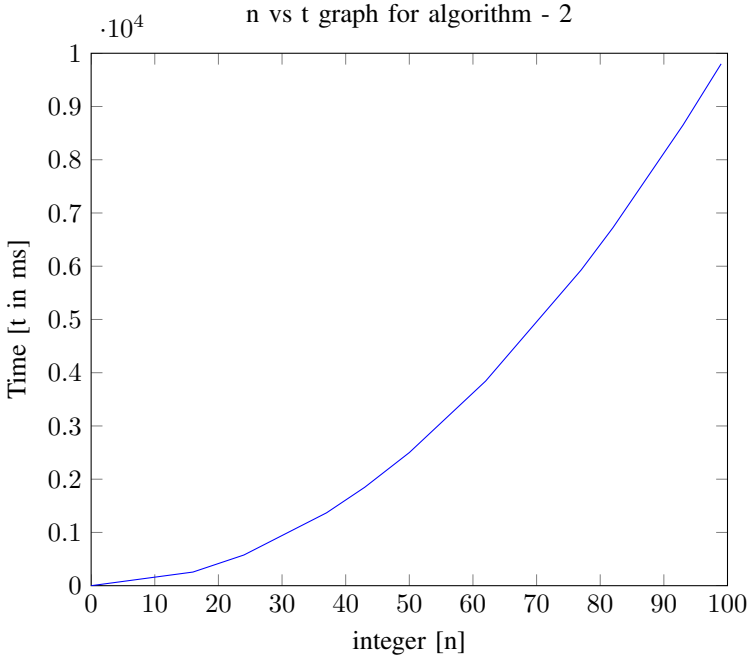
when i = 4, $dp[i][0] = 1$ and $dp[i][1] = 4$.

clearly throught the computation, our $result$ will be 4 and the length of longest zig-zag subsequence is also 4.

V. Experimental Analysis

Graph - 1 : The first graph is showing Linear behaviour.
Graph - 2 : The second graph is showing $O(n^2)$ behaviour.





VI. Conclusion

The second algorithm uses the dynamic programming but has the time complexity of $O(n^2)$, however first algorithm has time complexity of $O(n)$ which is better than $O(n^2)$.

References

1) http://www.algorithmsandme.com/longest-alternating-subsequence/
2) https://www.geeksforgeeks.org/longest-zig-zag-subsequence/