Design and Analysis of Algorithms Assignment - 5

Department of Information Technology,

Indian Institute of Information Technology, Allahabad 211015, India

Sainath Reddy (IIT2019201), Jyoti Verma(IIT2019202), Krishna Kaipa(IIT201920)

Abstract: ***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.***

Index Terms: ***Arrays, Dynamic Programming, Longest Subsequence***

INTRODUCTION

In this problem, we have to find the length of the longest subsequence of a given sequence such that the elements of this sequence are in alternating order.

Dynamic Programming The idea here is to apply dynamic programming. Dynamic programming: it was an optimization over a plain recursive algorithm. The idea here is to simply store the results of subproblems, so that we do not have to re-compute them when needed later.this simple optimization reduces time complexities from exponential to polynomial.

Algorithmic Steps: Our approach to implementing the solution and solving the problem is as follows:

1. We are going to take a dp as 2D array with size as 2*size of the given elements.The 2D array was going to have only two rows one is to store the length of longest alternating order and the other is to store the sign whether the present element in the sequence is greater than the previous or lesser than the previous element in the sequence.

2. The sign has three values if the sign was -1 it means it was not taken part of any sequence (itself it was a sequence of length 1) and if the sign was 0 it means that the element was taken in some sequence and this element is lesser than the preceding element in the sequence. And 1 means that the element is taken in some sequence and this element was greater than the preceding element in the sequence.

3. In this way, we store the longest alternating sequence which ends at present element in the 1st row of 2D array and the sign in 2nd row of 2D array and we take the max of 1st row of 2D array which is same as max length of alternating sequence.

The above returned count is the longest alternating subsequence.

Code:

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
 int arr[100];
 cout<<"Enter number of elements to generate: ";
 int n;
 cin>>n;
 out<<endl;
 cout<<"Randomly generated array: ";
 for(int i=0;i<n;i++)
 {
  arr[i] = rand() % 50 + 1;
  cout<<arr[i]<<" ";
}


int dp[n][2];

for(int i=0;i<n;i++)
{
 //minimum length 1
 dp[i][0] = 1;
 //not part of a subsequence yet
 dp[i][1] =-1;
}
int ma = 1;
for(int i=1;i<n;i++)
{
 int len =1;
 int dec=-1;
 for(int j=0;j<i;j++)
 {
   if(arr[j] < arr[i])
   {
     dec = 1;
     if(dp[j][0] >= dp[i][0])
     {
      if(dp[j][1] == -1 || dp[j][1] == 0)
      //was lesser than its previous element
      {
        len = max(len,dp[j][0]+1);
      }
     }
    }
    else if(arr[j] > arr[i])
    {
      dec = 0;
      if(dp[j][0] >= dp[i][0])
      {
        if(dp[j][1] == -1 || dp[j][1] == 1)
        //was greater than its previous element
```

```
        {
          len = max(len,dp[j][0]+1);
        }
      }
    }
  }
  dp[i][1] = dec;
  dp[i][0]  = len;
  ma = max(ma,dp[i][0]);
}

cout<<"\nLength of longest subsquence is :"<<ma;
return 0;
}
```

ANALYSIS

Apriori Analysis: This is the analysis performed prior to running in a stage where the function is defined using a theoretical model. Therefore, complexity is determined by just examining the algorithm rather than running it on a particular system with a different memory,processor, and compiler.

1. Time Complexity.

   The time complexity of the DP approach is $O(n^2)$. For each element at index i we have to check it with each and every element which are preceding it. As there is a possibility that it can be added with any of preceding elements in order to get the longest altering sequence.

   So for each i we have to do (i-1) checks and there are n elements so the time complexity was going to be $O(n^2)$.

2. Space Complexity.

   The space complexity of the DP approach is o(n) as we are talking a 2D array with size 2*no elements of sequence as the size. Space complexity : O(n)

Aposteriori Analysis: Aposteriori analysis of an algorithm means we per- form analysis of an algorithm only after running it on a system. It directly depends on the system and changes from system to system.So for the a aposteriori analysis of the algorithm,we have run our code on the compiler and get values of the time.

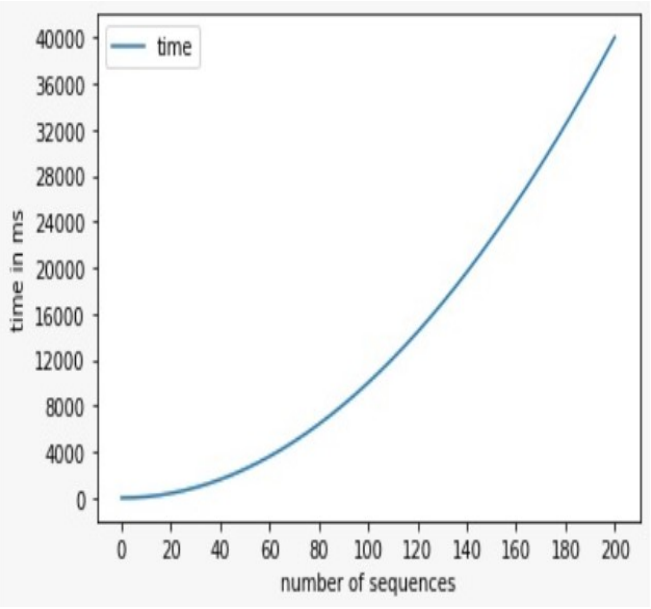

Figure 1: Time Complexity

APPLICATIONS

Some of the applications of the Dynamic Progamming Approach are:

1. 0/1 knapsack problem
2. Mathematical optimization problem
3. All pair Shortest path problem
4. Reliability design problem
5. Longest common subsequence (LCS)
6. Flight control and robotics control

CONCLUSION

In this paper, we had discussed an algorithm which can be used to find the largest alternating subsequence in the given sequence. In this method we had used a dynamic programming approach by storing the answers of the sub-problems and using them to solve the answer. The time complexity of this algorithm is $O(n^2)$ and space complexity is O(n).

REFERENCES

1. Introduction to Dynamic Programming Technique: https://www.geeksforgeeks.org/dynamic-programming/
2. Introduction to Algorithms by Cormen,Charles, Rivest and Stein. https://web.ist.utl.pt/ fabio.ferreira/material/asa

APPENDIX

To run the code, follow the following procedure:

1. Save the code with your own desirable name and extension is .cpp
2. Open the code with any IDE like Sublime Text, VS Code, Atom or some online compilers like GDB.
3. Run the code following the proper running commands(vary from IDE to IDE)
   (a) **For VS Code:** Press Function+F6 key and provide the input on the terminal.
   (b) **For Sublime Text:** Click on the Run button and provide the input.