Finding the longest sorted Subsequence started with different positions in a Matrix

Karan Kunwar[IIT2019001], Anirudh Arora[IIT2019003], Naina Kumari[IIT2019004]
IV Semester, Department of Information Technology
Indian Institute of Information Technology Allahabad

Abstract: In this Paper, we have devised two algorithms based on concepts of dynamic programming and Binary Search to find the longest sorted subsequence in a given matrix in row major as well as column major order.
And we have discussed and compared time and space complexities of both the algorithms using Apriori Analysis as well as Aposteriori Analysis.

I. INTRODUCTION

Dynamic Programming is an optimized form of recursion technique. In simpler words, it is recursion with memorization. In recursion, a function for lower values is called in order to get the value of function at a particular value. In dynamic programming, we define the base cases. As we move to upper values, we keep on storing the function values, so that we can retrieve these values in constant time whenever they are needed. The values are stored in a data structure which can later be called to get the desired value. For example:
For Fibonacci sequence,
fib[0]=1 //base case
fib[1]=1 //base case
fib[n]=fib[n-1]+fib[n-2] //function
Now, when we need to find the fibonacci number at index x, it calls the number at index x-1, x-2. Subsequently index at x-1 calls number at x-2,x-3 and so on. So the same values are being called multiple times, increasing the time complexity.
In dynamic programming, we maintain an array(data structure) containing numbers on different indexes of a fibonacci series. So, whenever a number at index x is needed we just need to check the array at indexes x-1 and x-2 without any need to call further indexes, reducing the time complexity quite effectively.

II. Algorithm 1 and Analysis

STEP 1: We convert the matrix into linear arrays rm, cm going in row major order as well as column major order respectively. We will implement the algorithm twice for both row major array and column major array.

STEP 2: we initialize an array (say dparr) of length rows*column. Initially each element is equal to 1. Along with this, we initialize array of vectors vecr of length rows*columns each with a single element (ie. ith vector will have ith element of the arrays rm) We initialize a variable ans=1 and an answer vector vr.

STEP 3: Starting from index 1 through the end of array rm, we check for all the non greater elements present in the array on the left of the current index. For each satisfying element we update the dparr and vecr at current index as:

```
Current index = curr
Satisfying element index = x
if(dparr[curr]<=dparr[x]+1)
{
        dparr[curr] = dparr[x]+1;
        vecr[curr] = vecr[x];
        vecr[curr].push_back(rm[curr]);
}
```

STEP 4: We update the value of variable ans at each satisfying index as:

```
if(anr<=dparr[curr])
{
        ans = dparr[curr];
        vr=vecr[curr];
}
```

We repeat the given steps traversing through the array and the length of the longest sorted subsequence will be stored in the variable ans and the subsequence will be stored in the vector vr.

```
Analysis                                 Time
for i = 0 to L :                         (c1)
    for j = 0 to i :                     (c2)
        if arr[ j ] <= arr[ i ]          (c3)
            if dpr[ j ]+1 >=dpr[ i ]
                dpr[i]=dpr[j]+1;
                vecr[i]=vecr[j];
                vecr[i].push_back(arr[i]);
            if(dpr[i]>=ansr)
                ansr = dpr[i];
                vr = vecr[i];
```

Here c3 is a constant time for all the

T(n)= c1.L + c2. $\sum_{j=0}^{L} t_j + c3.\sum_{j=0}^{L} t_j$

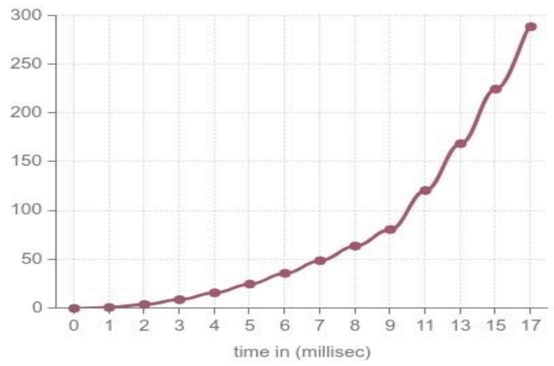
Here $t_j =$ j.

$\sum_{j=0}^{L} t_j = L(L+1)/2$

Therefore,

Time Complexity: O(n^2)

Space Complexity: O(n^2)



III. Algorithm 2 and Analysis

STEP 1: We convert the matrix into linear arrays rm, cm going in row major order as well as column major order respectively. We will implement the algorithm twice for both row major array and column major array.

STEP 2: We initialize a vector(say vans). Initially only the element at 0th index is pushed into this vector.

STEP 3: Traversing 1 through the end of the array rm. If the element on the current index greater than or equal to the last element of the array we will push this element. It is done as:

```
Current index = curr
if(vans.back<=rm[curr])
{
        vans.push_back(rm[curr]);
}
```

STEP 4: Now, We will check if the last element of the vector is the upper bound of the element at the current index. We will replace the last element with the element at current index in such case. We will use the binary search to find the upper bound or we can also use the inbuilt method to find the upper bound. As shown below:

```
Upper bound index = up
if(up == vans.size()-1)
{
        vans[up]=vans[curr]
}
```

We traverse through the array and the length of the vector vans will be the length of longest sorted subsequence and the subsequence will be stored in the vector vans

```
 Analysis                                     Time
for(int i=1;i<l;i++)                          (c1)
      if(vc[vc.size()-1] <= arc[i])            (c2)
         vc.push_back(arc[i]);                (c3)
      else
         int low=0, high=vc.size();           (c4)
         int ac=high;                         (c5)
         while(low<=high)                     (c6)
            int mid = (low+high)/2;           (c7)
            if(vc[mid]>arc[i])                (c8)
              ac=mid;                         (c9)
               high=mid-1;                    (c10)
            else
               low=mid+1;                     (c11)
         if(ac == vc.size()-1)                (c12)
            vc[ac]=arc[i];
(c13)
```

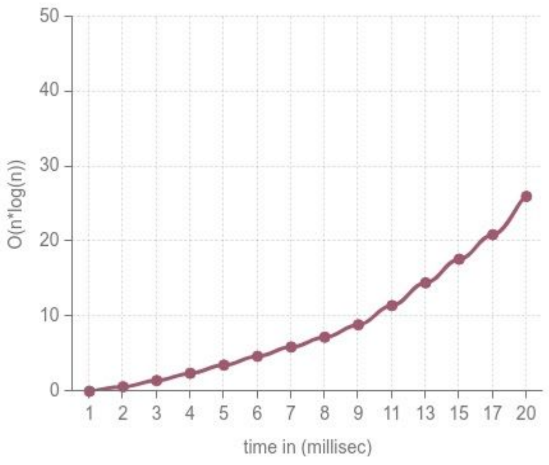
T(n)=c1.L+(c2+..+c5).(L-1)+(c6+..c13)(Llog L)

T(n) $\approx$ L . log(L)

Therefore,

Time Complexity: O($nlog(n)$ **)**

Space Complexity: O(n **)**



IV. Conclusion

We can conclude that we can find the longest sorted subsequence in a matrix with best complexity of O(nlogn).

V. References

- Algorithm Design by J Kleinberg, and E Tardos
- wikipedia.com