GROUP 22

Lovely Digra IIT2017090
Abhishek Vishwakarma IIT2017091
Anuj Raj IIT2017092

Problem Statement

Our required aim is to design and analyze an optimal algorithm to convolve 3*3 mask and check if it is sorted,if not find largest sorted subsequence.

Introduction

What is 3*3 mask?
The 3*3 mask(i,j) is the small matrix of 3*3 order starting at MATRIX[i][j] and ending at MATRIX[i+2][j+2] where MATRIX[][] is provided matrix.
The Criteria used to define sorted order
The sequence here is defined as row-wise sequence of elements that is firstly check for first row first then 2nd row and at last for 3rd.

Algorithm Approach

Traversal of 3*3 mask

Traversal is done in column fashion. Suppose a mask(0,0) is selected. This means it has elements from 0,0 to 3,3. Then mask(1,0) is selected. this means only 0,0 - 0,3 elements i.e elements of the first row of previous mask is removed and 4th row 4,0 - 4,3 is added.
This **reduces the number of selections** from 9 to 6 for each shift of masks for same value of starting column.

Approach1 (Using dynamic programming)

> Let arr[0..n-1] be the input array and L(i) be the length of the LIS ending at index i such that arr[i] is the last element of the LIS.
> Then, L(i) can be recursively written as: L(i) = 1 + max(L(j)) where 0 < j < i and arr[j] < arr[i]; or L(i) = 1, if no such j exists. To find the LIS for a given array, we need to return max(L(i)) where 0 < i < n.
> Considering the above implementation,following is recursion tree for an array of size 4. lis(n) gives us the length of LIS for arr[].
lis(4) calls lis(3) lis(2) lis(1)
lis(3) calls lis(2) lis(1)
lis(2) calls lis(1)
> Therefore,Overlapping substructure property and recomputation of same subproblems can be avoided by either using Memoization or Tabulation.

Approach2(Greedy algorithm Using Binary Search)

> Consider an input array A = {2, 5, 3, 7, 11} By observation we know that the LIS is either {2, 3, 7, 11} or {2, 5, 7, 11.Now, what if we need to add 8 to LIS?

> Note that the latest element 8 is greater than smallest element of any active sequence.Therefore, it should come after 7 (by replacing 11).

> We are not sure whether adding 8 will extend the series or not.

> Assume there is 9 in the input array,say{2, 5, 3, 7, 11, 8, 7, 9 }. We can replace 11 with 8, as there is potentially best candidate (9) that can extend the new series {2, 3, 7, 8} or {2, 5, 7, 8}.

> Possible cases during insertion of new element A[i] assume that the end element of largest sequence is E.
=> We can add current element A[i] to the existing sequence if there is an element A[j] (j > i) such that E < A[i] < A[j].
=> We can replace E with current element A[i] if E > A[i] < A[j].

> The observation is, new occurring element need to be a potential candidate to start new sequence.
> In general, we have set of active lists of varying length.We are adding an element A[i] to these lists. We scan the lists (for end elements) in decreasing order of their length. We will verify the end elements of all the lists to find a list whose end element is smaller than A[i] (floor value).

Our strategy determined by the following conditions

> If A[i] is smallest among all end candidates of active lists, we will start new active list of length 1.
> If A[i] is largest among all end candidates of active lists, we will clone the largest active list, and extend it by A[i].
> If A[i] is in between, we will find a list with largest end element that is smaller than A[i]. Clone and extend this list by A[i]. We will discard all other lists of same length as that of this modified list.

The length of the Longest Increasing Subsequence:

> All We need is to maintain a list(array) of end elements of the active lists.Initialise the list with A[0].i.e list[0] = A[0] and length = 1(length is list length) for each A[i] i : 1 to n-1(end of A).
> Strategy 1 is followed with replacement of first element of list(array) with A[i]. Strategy 2 is followed with addition of A[i] at the end of list(array). Strategy 3 is followed with binary search of the least upper bound of A[i] in the list(array) followed by its replacement with A[i].
> At the end length of the list is the length of longest increasing subsequence.

Pseudo Code

PseudoCode for LUB of key in given Array

```
procedure GetCeilIndex(vector arr, T, intl, r, key
    find the lowest Upper bound(LUB)
    of the key in T(that is TailIndices)
    return the position of the LUB where
    key position is to be inserted
```

PseudoCode for LIS in given Array in NlogN

```
procedure LIS(vector arr, int n)
    finds the LIS in the given array arr
    vector tailIndices(n,0)//Initialised with 0
    vector prevIndices(n,-1)//Intialised with -1
    int len ← 1
    for all i = 1 to n do
        if arr[i] is smallest then
            insert i to tailIndices[0]
        if arr[i] is largest then
            insert i to tailIndices[len]
            insert tailIndices[len-1] to prevIndices[i]
        else find pos with CeilIndex
        and then update both arrays
        i ← tailIndices[len-1] (i != -1)
backtracking of the LIS
        myvector ← arr[i]
        i ← prevIndices[i]
myvector stores LIS in reverse order
```

Experimental Analysis

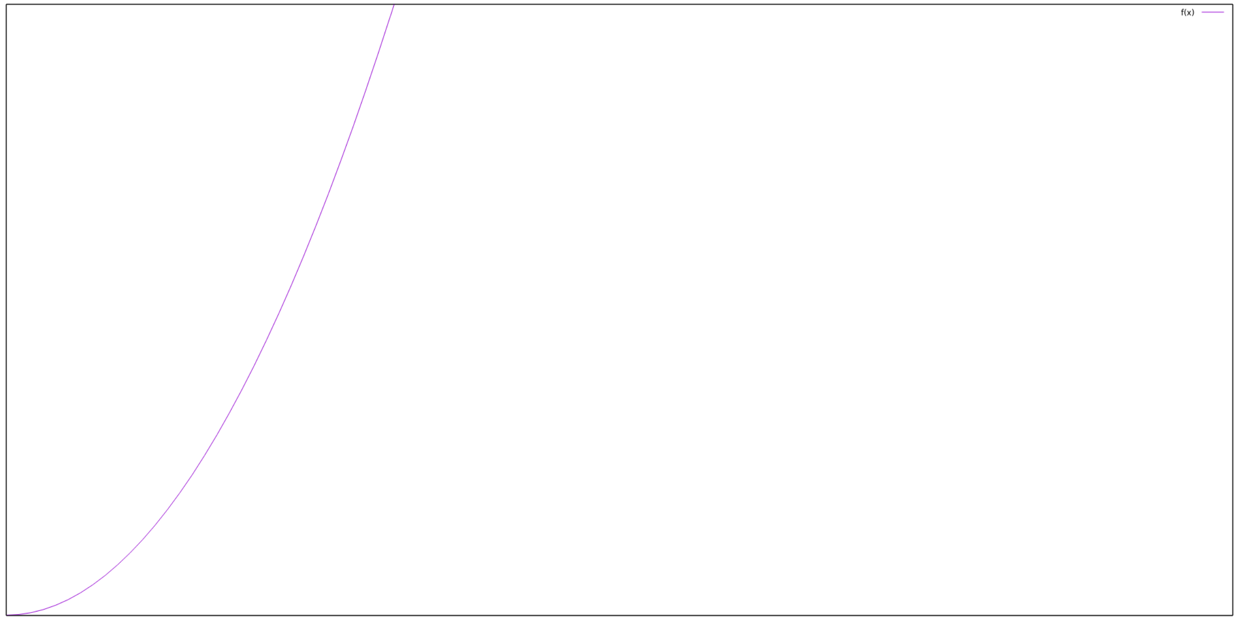
As we have to generate 3*3 mask from 100 * 100 matrix, we have to run two nested loops that would take O(n*n)complexity, where n= size of parent matrix (in this case ,n=100).

Time Complexity

Now for checking the longest increasing subsequence , we iterate the whole set of elements and perform modification of binary search that takes O(logm) time. So, time complexity becomes O(mlogm) , where m = no. of elements in the mask (in this case , m=9).

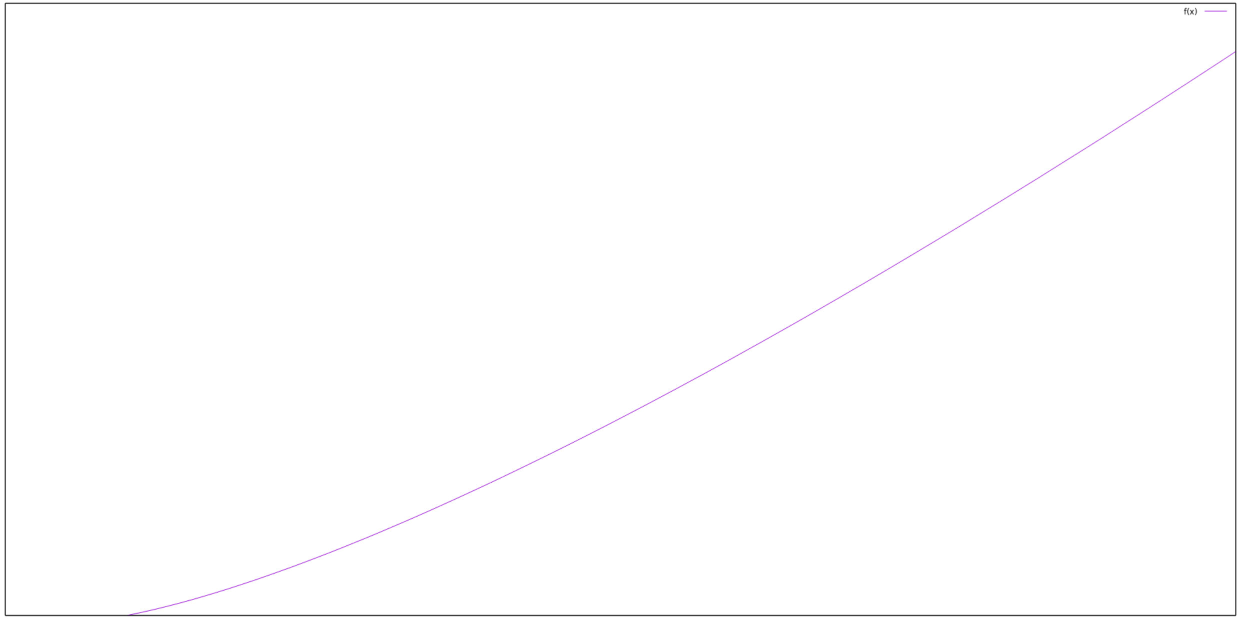
General Time Complexity: θ(n*n*m*logm).
Best Time Complexity : Ω(n*n*m), when the elements are increasingly sorted as we don’t have to perform binary search in any iteration.
Worst Time Complexity: O(n*n*m*logm), when the elements are decreasingly sorted as we have to perform binary search in every iteration.

Dynaminc Programming(n*n)



Time vs Length of Input

Greedy Algorithm Binary search (n*logn)



Time vs Length of Input

Conclusion

The Algorithm is correct and optimal under the constraints defined in the Problem Statement.

Thank You!