DAA Assignment No. : 5
Convolve 3*3 mask and check if it is sorted, if not find largest sorted subsequence.

Lovely Digra
IIT2017090

Abhishek Vishwakarma
IIT2017091

Anuj Raj
IIT2017092

***Abstract*—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. For our approach we have used Binary Search. While designing our problem we calculated the time complexity for the algorithm and plotted graphs.**

I. INTRODUCTION

What is 3X3 mask? The 3X3 mask(i,j) is the small matrix of 3X3 starting at MATRIX[i][j] and ending at MATRIX[i+2][j+2]. How to sorted order defined? The sequence here is defined as row-wise sequence of elements. that is first row first then 2nd row then 3rd.

II. ALGORITHM DESIGN AND EXPLANATION

Traversal of 3X3 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. To find longset increasing subsequence two approach is followed

Approach 1: 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. Thus, we see the LIS problem satisfies the optimal substructure property as the main problem can be solved using solutions to subproblems.

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)

We can see that there are many subproblems which are solved again and again. So this problem has Overlapping Substructure property and recomputation of same subproblems can be avoided by either using Memoization or Tabulation. Puedo code for the algorithm gives the further depiction.see Agorithm 1.

Approach 2: Greedy algorithm Using Binary Search

Consider an input array A = {2, 5, 3} By observation we know that the LIS is either {2, 3} or {2, 5}

Let us add two more elements, say 7, 11 to the array. These elements will extend the existing sequences. Now the increasing sequences are {2, 3, 7, 11} and {2, 5, 7, 11} for the input array {2, 5, 3, 7, 11}

Further, we add one more element, say 8 to the array i.e. input array becomes {2, 5, 3, 7, 11, 8}. Note that the latest element 8 is greater than smallest element of any active sequence (will discuss shortly about active sequences). How can we extend the existing sequences with 8? First of all, can 8 be part of LIS? If yes, how? If we want to add 8, 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}.

Our observation is, when checking for the newest element A[i] there are three possible cases: assume that the end element of largest sequence is E.

1. We can add current element A[i] to the existing sequence if there is an element A[j] $(j > i)$ such that $E < A[i] < A[j]$.

2. We can replace E with current element A[i] if $E > A[i] < A[j]$.

In the above example, E = 11, A[i] = 8 and A[j] = 9. thus E will be replaced with A[i].

The question is, when will it be safe to add or replace an element in the existing sequence?

Let us consider another sample A = {2, 5, 3}. Say, the next element is 1. How can it extend the current sequences {2, 3} or

$\{2, 5\}$. Obviously, it cant extend either. Yet, there is a potential that the new smallest element can be start of an LIS. To make it clear, consider the array is $\{2,5,3,1,2,3,4,5,6\}$. Making 1 as new sequence will create new sequence which is largest.

3. The observation is, when we encounter new smallest element in the array, it can be a potential candidate to start new sequence.

From the observations, we need to maintain lists of increasing sequences.

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 thelists to find a list whose end element is smaller than A[i] (floor value).

Our strategy determined by the following conditions

1. If A[i] is smallest among all end candidates of active lists, we will start new active list of length 1.

2. 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].

3. 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:

This is easy as we do not need to maintain the whole list of acitve lists.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) will follow the above strategy.

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.

The construction of Longest Increasing Subsequence:

For construction in NlogN the idea is to maintain a two arrays. One previousIndices array and another tailIndices array. TailIndices contains inputArrayIndex of end element of the active lists. This array is similar to the list in the previous algorithm of length of LIS.(InputArrayIndex of E means position of E in the input Array.)previousIndices stores previous value of the current operated tailIndices element. prevIndices[i] = tailIndices[pos-1]; where pos is the position of the current operated tailIndices. Operation means the three operations discussed in the previous algorithm. i.e. for each A[i] i:1 to n(end of A) pos is the position of tailIndidces found by Algo 1 and operated accordingly. then , prevIndices[i] = tailIndices[pos-1].

The LIS is extracted by backtracking of previous Indices As:

i = tailIndices[len-1]//last element while$(i! = -1)$ LIS.push back(A[i]) i = prevIndices[i] end LIS is in reverse order.

III. EXPERIMENTAL STUDY

For the experimental analysis of this algorithm, we have generated graphs of both algos : Dynammic Programming (Fig 2) and Using Binary Search (Fig 1).

IV. TIME COMPLEXITY AND DISCUSSION

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).

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).

Therefore , General Time Complexity: O(n*n*m*logm).

Best Time Complexity : O(n*n*m), when the elements are increasingly sorted as we dont have to perform binary search in any iteration.

Worst Time Complexty: O(n*n*m*logm), when the elements are decreasingly sorted as we have to perfrom binary search in every iteration.

V. CONCLUSION

We designed an algorithm to convolve 3*3 mask and check if it is sorted, if not find largest sorted subsequence. We also analyzed the algorithm and plotted the graph for both the algos.

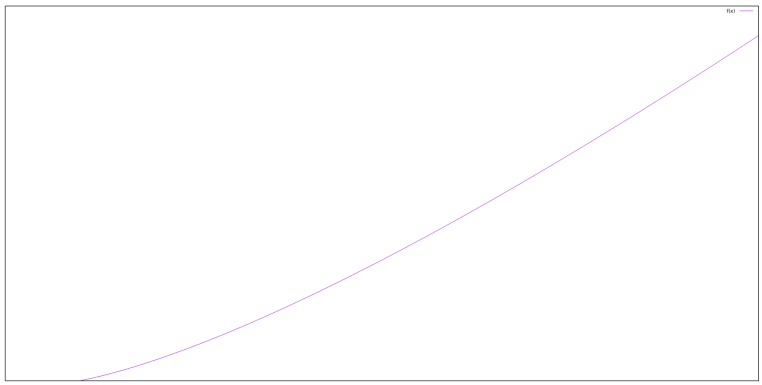

Fig. 1. Using Binary Search(nlogn)

Algorithm 1 PseudoCode for Approach

$order \leftarrow$ matrix size
procedure $generateMatrix(intorder)$
for all $i = 0$ to order **do**
for all $j = 0$ to order **do**
v[i][j] = $rand()\%1000 + 1$

Store largest size of sub-sequence in particular 3*3 mask
$maxcount \leftarrow 1$
for all $i = 0$ to order-2 **do**
for all $j = 0$ to order-2 **do**
$m \leftarrow 0$
for all $k = 0$ to 3 **do**
for all $l = 0$ to 3 **do**
arr[m] = v[k][l];
$m \leftarrow m$ + 1
Create vector contianing vector which
will store sub-sequence for that
particular index look_up[i] will
represent possible sub-sequence
$look\_up(arr.size()) \leftarrow$ store all sub-sequences
$look\_up[0] \leftarrow$ arr[0]
for all $k = 1$ to arr.size() **do**
for all $j = 0$ to k **do**
if $arr[k] > arr[l] look\_up[k].size() < look\_up[l].size() + 1$ **then**
$look\_up[k] \leftarrow$ look_up[l]
$look\_up[k] \leftarrow$ arr[k]

max will store largest sub-sequence
$max \leftarrow$ look_up[0]
for all $i = look\_up[0]$ to look_up.size() **do**
if $max.size() < i.size()$ **then**
$max \leftarrow$ i

sub-sequence will store all the
largest sub-sequence of 3*3 mask
$count \leftarrow$ max.size()
if $count >= max\_count$ **then**
$sub\_sequence[n] \leftarrow$ max

Store size of largest sub-sequence
$max\_size \leftarrow 0$
for all $i = 0$ to subsequence.size() **do**
if $subsequence[i].size() > max\_size$ **then**
$max\_size \leftarrow$ subsequence[i].size()
for all $i = 0$ to subsequence.size() **do**
if $subsequence[i].size() == max\_size$ **then**
for all $j = 0$ to max_size **do**
$print(subsequence[i][j)$

Algorithm 2 PseudoCode for LUB of key in given Array

procedure $GetCeilIndex(vectorarr, T, intl, r, key$
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

Algorithm 3 PseudoCode for LIS in given Array in NlogN

procedure $LIS(vectorarr, intn$
finds the LIS in the given array arr
vector tailIndices(n,0)//Initialised with 0
vector prevIndices(n,-1)//Intialised with -1
int $len \leftarrow 1$
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

$i \leftarrow$ tailIndices[len-1] (i != -1)
backtracking of the LIS
$mysvector \leftarrow$ arr[i]
$i \leftarrow$ prevIndices[i]
myvector stores LIS in reverse order

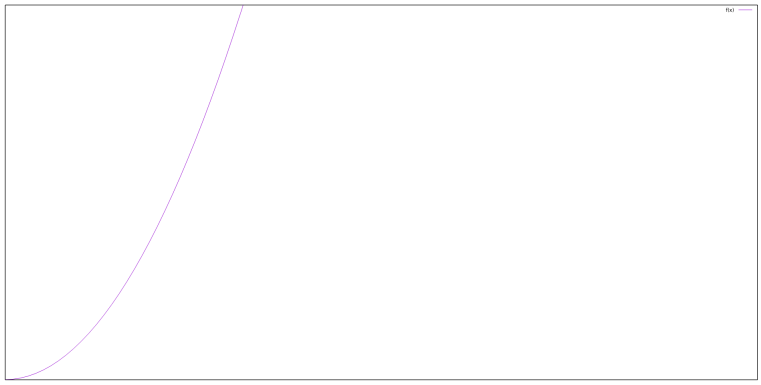

Fig. 2. Dynamminc Programming(n*n)