Longest Increasing Subsequence Size (N log N)

After few months of gap posting an algo. The current post is pending from long time, and many readers (e.g. here, here, here may be few more, I am not keeping track of all) are posting requests for explanation of the below problem.

Given an array of random numbers. Find *longest increasing subsequence* (LIS) in the array. I know many of you might have read recursive and dynamic programming (DP) solutions. There are few requests for O(N log N) algo in the forum posts.

For the time being, forget about recursive and DP solutions. Let us take small samples and extend the solution to large instances. Even though it may look complex at first time, once if we understood the logic, coding is simple.

Consider an input array A = {2, 5, 3}. I will extend the array during explanation.

By observation we know that the LIS is either {2, 3} or {2, 5}. ***Note that I am considering only strictly increasing sequences***.

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).

Since the approach is *offline (what we mean by offline?)*, 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, assume that the end element of largest sequence is E. We can add (replace) current element A[i] to the existing sequence if there is an element A[j] (j > i) such that E < A[i] < A[j] or (E > A[i] < A[j] – for replace). In the above example, E = 11, A[i] = 8 and A[j] = 9.

In case of our original array {2, 5, 3}, note that we face same situation when we are adding 3 to increasing sequence {2, 5}. I just created two increasing sequences to make explanation simple. Instead of two

sequences, 3 can replace 5 in the sequence {2, 5}.

I know it will be confusing, I will clear it shortly!

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 can't 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.

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 the lists 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.

Note that at any instance during our construction of active lists, the following condition is maintained.

"end element of smaller list is smaller than end elements of larger lists".

It will be clear with an example, let us take example from wiki {0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15}.

```
A[0] = 0. Case 1. There are no active lists, create one.
0.
-------------------------------------------------------------------------------
A[1] = 8. Case 2. Clone and extend.
0.
0, 8.
-------------------------------------------------------------------------------
A[2] = 4. Case 3. Clone, extend and discard.
0.
0, 4.
~~0, 8~~. Discarded
-------------------------------------------------------------------------------
A[3] = 12. Case 2. Clone and extend.
0.
```

```
0, 4.
0, 4, 12.
--------------------------------------------------------------------------------
A[4] = 2. Case 3. Clone, extend and discard.
0.
0, 2.
~~0, 4~~. Discarded.
0, 4, 12.
--------------------------------------------------------------------------------
A[5] = 10. Case 3. Clone, extend and discard.
0.
0, 2.
0, 2, 10.
~~0, 4, 12~~. Discarded.
--------------------------------------------------------------------------------
A[6] = 6. Case 3. Clone, extend and discard.
0.
0, 2.
0, 2, 6.
~~0, 2, 10~~. Discarded.
--------------------------------------------------------------------------------
A[7] = 14. Case 2. Clone and extend.
0.
0, 2.
0, 2, 6.
0, 2, 6, 14.
--------------------------------------------------------------------------------
A[8] = 1. Case 3. Clone, extend and discard.
0.
0, 1.
~~0, 2~~. Discarded.
0, 2, 6.
0, 2, 6, 14.
--------------------------------------------------------------------------------
A[9] = 9. Case 3. Clone, extend and discard.
0.
0, 1.
0, 2, 6.
0, 2, 6, 9.
~~0, 2, 6, 14~~. Discarded.
--------------------------------------------------------------------------------
A[10] = 5. Case 3. Clone, extend and discard.
0.
0, 1.
0, 1, 5.
~~0, 2, 6~~. Discarded.
0, 2, 6, 9.
--------------------------------------------------------------------------------
A[11] = 13. Case 2. Clone and extend.
0.
```

```
0, 1.
0, 1, 5.
0, 2, 6, 9.
0, 2, 6, 9, 13.
------------------------------------------------------------------------------
A[12] = 3. Case 3. Clone, extend and discard.
0.
0, 1.
0, 1, 3.
0, 1, 5. Discarded.
0, 2, 6, 9.
0, 2, 6, 9, 13.
------------------------------------------------------------------------------
A[13] = 11. Case 3. Clone, extend and discard.
0.
0, 1.
0, 1, 3.
0, 2, 6, 9.
0, 2, 6, 9, 11.
0, 2, 6, 9, 13. Discarded.
------------------------------------------------------------------------------
A[14] = 7. Case 3. Clone, extend and discard.
0.
0, 1.
0, 1, 3.
0, 1, 3, 7.
0, 2, 6, 9. Discarded.
0, 2, 6, 9, 11.
------------------------------------------------------------------------------
A[15] = 15. Case 2. Clone and extend.
0.
0, 1.
0, 1, 3.
0, 1, 3, 7.
0, 2, 6, 9, 11.
0, 2, 6, 9, 11, 15. <-- LIS List
------------------------------------------------------------------------------
```

It is required to understand above strategy to devise an algorithm. Also, ensure we have maintained the condition, "*end element of smaller list is smaller than end elements of larger lists*". Try with few other examples, before reading further. It is important to understand what happening to end elements.

Algorithm:

Querying length of longest is fairly easy. Note that we are dealing with end elements only. We need not to maintain all the lists. We can store the end elements in an array. Discarding operation can be simulated with replacement, and extending a list is analogous to adding more elements to array.

We will use an auxiliary array to keep end elements. The maximum length of this array is that of input. In the worst case the array divided into N lists of size one (*note that it does't lead to worst case complexity*). To

discard an element, we will trace ceil value of A[i] in auxiliary array (again observe the end elements in your rough work), and replace ceil value with A[i]. We extend a list by adding element to auxiliary array. We also maintain a counter to keep track of auxiliary array length.

Bonus: You have learnt Patience Sorting technique partially :).

Here is a proverb, "*Tell me and I will forget. Show me and I will remember. Involve me and I will understand*." So, pick a suit from deck of cards. Find the longest increasing sub-sequence of cards from the shuffled suit. You will never forget the approach. 🙂

Given below is code to find length of LIS,

C++

```
// C++ program to find length of longest increasing subsequence
// in O(n Log n) time
#include <iostream>
#include <string.h>
#include <stdio.h>
using namespace std;

#define ARRAY_SIZE(A) sizeof(A)/sizeof(A[0])

// Binary search (note boundaries in the caller)
// A[] is ceilIndex in the caller
int CeilIndex(int A[], int l, int r, int key)
{
    while (r - l > 1)
    {
        int m = l + (r - l)/2;
        if (A[m]>=key)
            r = m;
        else
            l = m;
    }
    return r;
}

int LongestIncreasingSubsequenceLength(int A[], int size)
{
    // Add boundary case, when array size is one

    int *tailTable   = new int[size];
    int len; // always points empty slot

    memset(tailTable, 0, sizeof(tailTable[0])*size);

    tailTable[0] = A[0];
    len = 1;
    for (int i = 1; i < size; i++)
    {
        if (A[i] < tailTable[0])
            // new smallest value
            tailTable[0] = A[i];

        else if (A[i] > tailTable[len-1])
            // A[i] wants to extend largest subsequence
            tailTable[len++] = A[i];
```

```
        else
            // A[i] wants to be current end candidate of an existing
            // subsequence. It will replace ceil value in tailTable
            tailTable[CeilIndex(tailTable, -1, len-1, A[i])] = A[i];
    }

    delete[] tailTable;
    return len;
}

int main()
{
    int A[] = { 2, 5, 3, 7, 11, 8, 10, 13, 6 };
    int n = ARRAY_SIZE(A);
    printf("Length of Longest Increasing Subsequence is %d\n",
           LongestIncreasingSubsequenceLength(A, n));

    return 0;
}
```

Run on IDE

Java

```
// Java program to find length of longest increasing subsequence
// in O(n Log n) time
import java.io.*;
import java.util.*;
import java.lang.Math;

class LIS
{
    // Binary search (note boundaries in the caller)
    // A[] is ceilIndex in the caller
    static int CeilIndex(int A[], int l, int r, int key)
    {
        while (r - l > 1)
        {
            int m = l + (r - l)/2;
            if (A[m]>=key)
                r = m;
            else
                l = m;
        }

        return r;
    }

    static int LongestIncreasingSubsequenceLength(int A[], int size)
    {
        // Add boundary case, when array size is one

        int[] tailTable   = new int[size];
        int len; // always points empty slot

        tailTable[0] = A[0];
        len = 1;
        for (int i = 1; i < size; i++)
        {
            if (A[i] < tailTable[0])
                // new smallest value
                tailTable[0] = A[i];
```

```
            else if (A[i] > tailTable[len-1])
                // A[i] wants to extend largest subsequence
                tailTable[len++] = A[i];

            else
                // A[i] wants to be current end candidate of an existing
                // subsequence. It will replace ceil value in tailTable
                tailTable[CeilIndex(tailTable, -1, len-1, A[i])] = A[i];
        }

        return len;
    }

    // Driver program to test above function
    public static void main(String[] args)
    {
        int A[] = { 2, 5, 3, 7, 11, 8, 10, 13, 6 };
        int n = A.length;
        System.out.println("Length of Longest Increasing Subsequence is "+
                           LongestIncreasingSubsequenceLength(A, n));
    }
}
/* This code is contributed by Devesh Agrawal*/
```

Run on IDE

Output:

```
Length of Longest Increasing Subsequence is 6
```

Complexity:

The loop runs for N elements. In the worst case (what is worst case input?), we may end up querying ceil value using binary search (log *i*) for many A[i].

Therefore, T(n) < O(log N!) = O(N log N). Analyse to ensure that the upper and lower bounds are also O(N log N). The complexity is THETA (N log N).

Exercises:

1. Design an algorithm to construct the longest increasing list. Also, model your solution using DAGs.

2. Design an algorithm to construct **all** increasing list**s of equal longest size**.

3. Is the above algorithm an *online* algorithm?

4. Design an algorithm to construct the longest *decreasing* list..

— Venki. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

142 Comments Category: Arrays

About Venki

Software Engineer

View all posts by Venki →

Related Posts:

- Longest Span with same Sum in two Binary arrays
- Count Inversions of size three in a give array
- Find the subarray with least average
- Count triplets with sum smaller than a given value
- Find zeroes to be flipped so that number of consecutive 1's is maximized
- Reorder an array according to given indexes
- Find maximum value of Sum(i*arr[i]) with only rotations on given array allowed
- Find maximum average subarray of k length

(Login to Rate and Mark)

3.5 Average Difficulty : **3.5/5.0**
Based on **9** vote(s)

☐ Add to TODO List

☐ Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.