# Lecture Notes

Doina Bein

# The longest decreasing subsequence problem

The *longest decreasing subsequence* problem is to find a subsequence of a given sequence in which the subsequence's elements are in sorted order, highest to lowest, and in which the subsequence is as long as possible. This subsequence is not necessarily contiguous, or unique.

The *longest decreasing subsequence* problem can be formulated as follows:

**input**: *n*, a positive integer, and the array of *n* comparable elements

**output**: array that contains the longest decreasing subsequence

The longest increasing subsequence problem is solvable in time, where denotes the length of the input sequence.[[1]](#footnote-1)

Let us consider an example (an instance to this problem) by mixing the first 14 Fibonacci numbers[[2]](#footnote-2) with 253 and writing them in reverse order:

143, 233, 55, 89, 21, 34, 8, 13, 3, 5, 1, 2, 253, 0, 1

(I took the first 14 Fibonacci numbers and I swapped the first with the second, the third with the fourth, and so on, then added 253 on the third position, and then reverse everything.)

This input sequence has no eight-member decreasing subsequences.

A longest decreasing subsequence is

233, 89, 34, 13, 5, 2, 0

This subsequence has length seven.

The longest decreasing subsequence in this example is not unique. For instance,

143, 55, 21, 8, 3, 1, 0

or

143, 55, 21, 8, 4, 2, 0

are other decreasing subsequences of equal length seven.

The way one can identify a longest decreasing subsequence for this particular instance of the problem is by selecting the largest value in the array as the first element of the subsequence and then continue to select elements in decreasing order:

**Algorithm Max\_then\_Decreasing**

Step 1. Select the maximum value in and add it to . Let *pos* be the position of that value in , i.e. *A[pos]* is the minimum element in . Let *last* be the last element in .

Step 2. For each element in located after *A[pos],* if the element is smaller than *last*, then add it to and adjust *pos* to be the position of that value in and *last* be the last element in .

Step 3. Repeat Step 2 until you reach the end of *.*

Let us see how the algorithm Max\_then\_Decreasing works on another instance of this problem.

For example, let us consider the first 16 terms of the binary Van der Corput[[3]](#footnote-3) sequence, written in reverse order:

15, 7, 11, 3, 13, 5, 9, 1, 14, 6, 10, 2, 12, 4, 8, 0

For this instance of the problem, we start with the maximum element 15, which happens to be the first element in , and we add it to *:*

R: 0

Then the next element 7 is smaller than 15, so we include it:

R: 15, 7

The third element 11 is larger than 7, the last element in , so it cannot be included. The fourth element 3 is smaller than 7, the last element in so we include it:

R: 15, 7, 3

The next three elements, 13, 5, 9, are all larger than 3, the last element in , so they cannot be included. The eighth element 1 is smaller than 3, the last element in so we include it:

R: 15, 7, 3, 1

The rest of the elements, except the last one 0, are all larger than 1, the last element in , so they cannot be included. The last element 0 is smaller than 1, the last element in we include it:

R: 15, 7, 3, 1, 0

We have obtained a subsequence of five elements that is decreasing, but it is not the optimal solution for the problem. This input sequence has no seven-member increasing subsequences.

A longest increasing subsequence is

15, 11, 9, 6, 2, 0.

This subsequence has length six.

The longest increasing subsequence in this example is not unique. For instance,

15, 11, 9, 6, 4, 0

or

15, 13, 9, 6, 4, 0

are other increasing subsequences of equal length six.

There are many instances, including the example given above, for which this strategy will not work.

So this algorithm does not always produce the correct output.

# A simple algorithm for the longest decreasing subsequence problem

There is a straightforward algorithm that solves the problem and has running time, where denotes the length of the input sequence. The algorithm uses an additional array of length *n*, of non-negative integers. The value will indicate how many elements smaller than are further in could be in a longest decreasing subsequence.

Initially, the array is set to 0 (all the elements are 0). The algorithm proceeds by trying to modify the values of starting with the previous to last element and going down to the first element. Then a longest subsequence can be identified by selecting elements of in decreasing order of the values, starting with the largest value in .

**Algorithm End\_to\_Beginning**

Step 1. Set all the values in the array to 0.

Step 2. Starting with and going down to   
try to increase the value of as follows:

Step 3. Starting with index and going up to (the last index in ) repeat Steps 4 and 5:

Step 4. See if any element is smaller than and has its value also bigger or equal to .

Step 5. If yes, then can be followed by that element in an decreasing subsequence, thus set to be 1 plus that value.

Step 6. Calculate the largest (maximum) value in . By adding 1 to that value we have the length of a longest decreasing subsequence.

Step 7. Identify a longest subsequence by identifying elements in that have decreasing values, starting with the largest (maximum) value in .

Let us see how the algorithm End\_to\_Beginning works for the instance:

143, 233, 55, 89, 21, 34, 8, 13, 3, 5, 1, 2, 0, 1

The array has initially the value 0 for all its elements:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| index | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] | [11] | [12] | [13] |
| H | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Leaving blank the cells that are not involved in a computation at some particular step, the array is modified, starting with index and going down to 0.

Since there are no elements smaller than that follow in , then remains 0.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| index | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] | [11] | [12] | [13] |
| A | 143 | 233 | 55 | 89 | 21 | 34 | 8 | 13 | 3 | 5 | 1 | 2 | 0 | 1 |
| H |  |  |  |  |  |  |  |  |  |  |  |  | 0 | 0 |

We compare now with the elements that follow it in , namely and

and thus =1

but so we do not change

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| index | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] | [11] | [12] | [13] |
| A | 143 | 233 | 55 | 89 | 21 | 34 | 8 | 13 | 3 | 5 | 1 | 2 | 0 | 1 |
| H |  |  |  |  |  |  |  |  |  |  |  | 1 | 0 | 0 |

We compare now with the elements that follow it in , namely and

so we do not change

and thus =1

but so we do not change

We continue until we finish calculating . The array has now the values:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| index | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] | [11] | [12] | [13] |
| A | 143 | 233 | 55 | 89 | 21 | 34 | 8 | 13 | 3 | 5 | 1 | 2 | 0 | 1 |
| H | 6 | 6 | 5 | 5 | 4 | 4 | 3 | 3 | 2 | 2 | 1 | 1 | 0 | 0 |

The maximum value in is 6, so a longest increasing subsequence has length 7.

We will look in for the values 6, 5, 4, 3, 2, 1, and 0.

We look in for the value 6: the element with index 0 has H[0]=6, thus A[0] is added to :

R: 143

We look in for the value 5 and check that is smaller than the last element in R: the element with index 2 has H[2]=5, thus A[2] is added to :

R: 143, 55

We look in for the value 4 and check that is smaller than the last element in R: the element with index 4 has H[4]=4, thus A[4] is added to :

R: 143, 55, 21

We look in for the value 3 and check that is smaller than the last element in R: the element with index 6 has H[6]=3, thus A[6] is added to :

R: 143, 55, 21, 8

We look in for the value 2: the element with index 8 has H[8]=2, thus A[8] is added to :

R: 143, 55, 21, 8, 3

We look in for the value 1: the element with index 10 has H[10]=1, thus A[10] is added to :

R: 143, 55, 21, 8, 3, 1

We look in for the value 0: the element with index 12 has H[12]=0, thus A[12] is added to :

R: 143, 55, 21, 8, 3, 1, 0

We are done. We have found a longest decreasing subsequence for this instance.

Let us see how the algorithm End\_to\_Beginning works for the second instance:

15, 7, 11, 3, 13, 5, 9, 1, 14, 6, 10, 2, 12, 4, 8, 0

The array has initially the value 0 for all its elements:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] | [11] | [12] | [13] | [14] | [15] |
| H | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Leaving blank the cells that are not involved in a computation at some particular step, the array is modified, starting with index and going down to 0.

We compare now with the elements that follow in , namely :

and thus =1

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] | [11] | [12] | [13] | [14] | [15] |
| A | 15 | 7 | 11 | 3 | 13 | 5 | 9 | 1 | 14 | 6 | 10 | 2 | 12 | 4 | 8 | 0 |
| H |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 | 0 |

We compare now with the elements that follow it in , namely and

thus so we do not change

and thus =1

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] | [11] | [12] | [13] | [14] | [15] |
| A | 15 | 7 | 11 | 3 | 13 | 5 | 9 | 1 | 14 | 6 | 10 | 2 | 12 | 4 | 8 | 0 |
| H |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 | 1 | 0 |

We compare now with the elements that follow it in , namely and

and thus

and so we do not change

but so we do not change

We continue until we finish calculating . The array has now the values:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] | [11] | [12] | [13] | [14] | [15] |
| A | 15 | 7 | 11 | 3 | 13 | 5 | 9 | 1 | 14 | 6 | 10 | 2 | 12 | 4 | 8 | 0 |
| H | 5 | 3 | 4 | 2 | 4 | 2 | 3 | 1 | 3 | 2 | 2 | 1 | 2 | 1 | 1 | 0 |

The maximum value in is 5, so a longest increasing subsequence has length 6.

We will look in for the values 5, 4, 3, 2, 1, and 0.

We look in for the value 5: the element with index 0 has H[0]=5, thus A[0] is added to :

R: 15

We look in for the value 4 and check that is smaller than the last element in R: the element with index 2 has H[2]=4, thus A[2] is added to :

R: 15, 11

We look in for the value 3 and check that is smaller than the last element in R: the element with index 6 has H[6]=3, thus A[6] is added to :

R: 15, 11, 9

We look in for the value 2 and check that is smaller than the last element in R: the element with index 9 has H[9]=2, thus A[9] is added to :

R: 15, 11, 9, 6

We look in for the value 1 and check that is smaller than the last element in R: the element with index 11 has H[11]=1, thus A[11] is added to :

R: 15, 11, 9, 6, 2

We look in for the value 0 and check that is smaller than the last element in R: the element with index 15 has H[15]=0, thus A[15] is added to :

R: 15, 11, 9, 6, 2, 0

We are done. We have found a longest increasing subsequence for this instance.

1. Schensted, C. (1961), "Longest increasing and decreasing subsequences", Canadian Journal of Mathematics 13: 179–191, doi:10.4153/CJM-1961-015-3 [↑](#footnote-ref-1)
2. https://www.mathsisfun.com/numbers/fibonacci-sequence.html [↑](#footnote-ref-2)
3. https://en.wikipedia.org/wiki/Van\_der\_Corput\_sequence [↑](#footnote-ref-3)