Chapter 48: Longest Increasing Subsequence

Section 48.1: Longest Increasing Subsequence Basic Information

The Longest Increasing Subsequence problem is to find subsequence from the give input sequence in which subsequence's elements are sorted in lowest to highest order. All subsequence are not contiguous or unique.

Application of Longest Increasing Subsequence:

Algorithms like Longest Increasing Subsequence, Longest Common Subsequence are used in version control systems like Git and etc.

Simple form of Algorithm:

1. Find unique lines which are common to both documents.
2. Take all such lines from the first document and order them according to their appearance in the second document.
3. Compute the LIS of the resulting sequence (by doing a Patience Sort), getting the longest matching sequence of lines, a correspondence between the lines of two documents.
4. Recurse the algorithm on each range of lines between already matched ones.

Now let us consider a simpler example of the LCS problem. Here, input is only one sequence of distinct integers `a1, a2, ..., an.`, and we want to find the longest increasing subsequence in it. For example, if input is **7,3,8,4,2,6** then the longest increasing subsequence is **3,4,6**.

The easiest approach is to sort input elements in increasing order, and apply the LCS algorithm to the original and sorted sequences. However, if you look at the resulting array you would notice that many values are the same, and the array looks very repetitive. This suggest that the LIS (longest increasing subsequence) problem can be done with dynamic programming algorithm using only one-dimensional array.

Pseudo Code:

1. Describe an array of values we want to compute.
   For `1 <= i <= n`, let **A(i)** be the length of a longest increasing sequence of input. Note that the length we are ultimately interested in is `max{A(i)|1 ≤ i ≤ n}`.
2. Give a recurrence.
   For `1 <= i <= n`, `A(i) = 1 + max{A(j)|1 ≤ j < i` and `input(j) < input(i)}.`
3. Compute the values of A.
4. Find the optimal solution.

The following program uses A to compute an optimal solution. The first part computes a value m such that **A(m)** is the length of an optimal increasing subsequence of input. The second part computes an optimal increasing subsequence, but for convenience we print it out in reverse order. This program runs in time O(n), so the entire algorithm runs in time O(n^2).

Part 1:

```
m ← 1
for i : 2..n
    if A(i) > A(m) then
```

```
        m ← i
    end if
end for
```

Part 2:

```
put a
while A(m) > 1 do
    i ← m-1
    while not(ai < am and A(i) = A(m)-1) do
        i ← i-1
    end while
    m ← i
    put a
 end while
```

Recursive Solution:

Approach 1:

```
LIS(A[1..n]):
    if (n = 0) then return 0
    m = LIS(A[1..(n - 1)])
    B is subsequence of A[1..(n - 1)] with only elements less than a[n]
    (* let h be size of B, h ≤ n-1 *)
    m = max(m, 1 + LIS(B[1..h]))
    Output m
```

Time complexity in Approach 1 : `O(n*2^n)`

Approach 2:

```
LIS(A[1..n], x):
    if (n = 0) then return 0
    m = LIS(A[1..(n - 1)], x)
    if (A[n] < x) then
        m = max(m, 1 + LIS(A[1..(n - 1)], A[n]))
    Output m

MAIN(A[1..n]):
    return LIS(A[1..n], ∞)
```

Time Complexity in Approach 2: `O(n^2)`

Approach 3:

```
LIS(A[1..n]):
    if (n = 0) return 0
    m = 1
    for i = 1 to n - 1 do
        if (A[i] < A[n]) then
            m = max(m, 1 + LIS(A[1..i]))
    return m

MAIN(A[1..n]):
    return LIS(A[1..i])
```

Time Complexity in Approach 3: `O(n^2)`

Iterative Algorithm:

Computes the values iteratively in bottom up fashion.

```
LIS(A[1..n]):
    Array L[1..n]
    (* L[i] = value of LIS ending(A[1..i]) *)
    for i = 1 to n do
        L[i] = 1
        for j = 1 to i - 1 do
            if (A[j] < A[i]) do
                L[i] = max(L[i], 1 + L[j])
    return L

MAIN(A[1..n]):
    L = LIS(A[1..n])
        return the maximum value in L
```

Time complexity in Iterative approach: `O(n^2)`

Auxiliary Space: `O(n)`

Lets take **{0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15}** as input. So, Longest Increasing Subsequence for the given input is **{0, 2, 6, 9, 11, 15}**.