Longest Increasing Subsequence

Stanislav Ostapenko

January 29, 2024

DP solution

Let's define $L(i)$ as the length of the longest strictly increasing subsequence ending at index i. The recurrence formula for the longest strictly increasing subsequence is given by:

$$L(i) = 1 + \max_{\substack{j<i \\ \text{arr}[j]<\text{arr}[i]}} L(j)$$

This equation states that the length of the longest increasing subsequence ending at index i is 1 plus the maximum length obtained by considering all indices j less than i, where the corresponding element arr[j] is less than arr[i].

Complexity :

$T(n) = \mathcal{O}(n^2)$
$M(n) = \mathcal{O}(n)$

```java
class Solution {
 private int max(int[] L) {
  int maxLength = Integer.MIN_VALUE;
  for (final int length : L) {
   maxLength = Math.max(maxLength, length);
  }
  return maxLength;
 }

 public int lengthOfLIS(int[] nums) {
  int n = nums.length;
  int[] L = new int[n];
  // Initialize the array with minimum length 1 for each index
  Arrays.fill(L, 1);

  // Iterate to fill in the values of L(i) using the recurrence relation
  for (int i = 1; i < n; i++) {
   for (int j = 0; j < i; j++) {
    if (nums[i] > nums[j] && L[i] < L[j] + 1) {
     L[i] = L[j] + 1;
    }
   }
  }
  // Find the maximum value in the array L
  return max(L);
 }
}
```

Naive solution

```
public class Solution {
 private List<List<Integer>> generateSubsequences(int[] arr) {
  List<List<Integer>> allSubsequences = new ArrayList<>();
  generateSubsequencesHelper(arr, 0, new ArrayList<>(), allSubsequences);
  return allSubsequences;
 }

 private void generateSubsequencesHelper(int[] arr, int index,
    List<Integer> current, List<List<Integer>> allSubsequences) {
  if (index == arr.length) {
   // Base case: add the current subsequence to the result
   allSubsequences.add(new ArrayList<>(current));
   return;
  }
  // Exclude the current element
  generateSubsequencesHelper(arr, index + 1, current, allSubsequences);
  // Include the current element
  current.add(arr[index]);
  generateSubsequencesHelper(arr, index + 1, current, allSubsequences);
  // Backtrack to exclude the current element
  current.removeLast();
 }

 private boolean isStrictlyIncreasing(List<Integer> list) {
  for (int i = 1; i < list.size(); i++) {
   if (list.get(i) <= list.get(i - 1)) {
    return false;
   }
  }
  return true; // Strictly increasing
 }

    public int lengthOfLIS(int[] nums) {
     List<List<Integer>> allSubsequences = generateSubsequences(nums);
     int max = 1;
     for (List<Integer> subsequence : allSubsequences) {
      if (isStrictlyIncreasing(subsequence)) {
       max = Math.max(max, subsequence.size());
      }
     }
     return max;
    }
}
```