Master algorithms together on Binary Search! Create a room, invite your friends, and race to finish the problems. ✕

Daily Coding Problem #75

Problem

This problem was asked by Microsoft.

Given an array of numbers, find the length of the longest increasing subsequence in the array. The subsequence does not necessarily have to be contiguous.

For example, given the array [0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15], the longest increasing subsequence has length 6: it is 0, 2, 6, 9, 11, 15.

Solution

This naive, brute force way to solve this is to generate each possible subsequence, testing each one for monotonicity and keeping track of the longest one. That would be prohibitively expensive: generating each subsequence already takes

O(2^N)!

Instead, let's try to tackle this problem using recursion and then optimize it with dynamic programming.

Assume that we already have a function that gives us the length of the longest increasing subsequence. Then we'll try to feed some part of our input array back to it and try to extend the result. Our base cases are: the empty list, returning 0, and an array with one element, returning 1.

Then,

- For every index `i` up until the second to last element, calculate `longest_increasing_subsequence` up to there.
- We can only extend the result with the last element if our last element is greater than `arr[i]` (since otherwise, it's not increasing).
- Keep track of the largest result.

```python
def longest_increasing_subsequence(arr):
    if not arr:
        return 0
    if len(arr) == 1:
        return 1

    max_ending_here = 0
    for i in range(len(arr)):
        ending_at_i = longest_increasing_subsequence(arr[:i])
        if arr[-1] > arr[i - 1] and ending_at_i + 1 > max_ending_here:
            max_ending_here = ending_at_i + 1
    return max_ending_here
```

This is really slow due to repeated subcomputations (exponential in time). So, let's use dynamic programming to store values to recompute them for later.

We'll keep an array `A` of length N, and `A[i]` will contain the length of the longest increasing subsequence ending at `i`. We can then use the same recurrence but look it up in the array instead:

```python
def longest_increasing_subsequence(arr):
    if not arr:
        return 0
    cache = [1] * len(arr)
    for i in range(1, len(arr)):
        for j in range(i):
            if arr[i] > arr[j]:
                cache[i] = max(cache[i], cache[j] + 1)
    return max(cache)
```

This now runs in O(N^2) time and O(N) space.

© Daily Coding Problem 2019 Privacy Policy Terms of Service Press