354. Russian Doll Envelopes

Nov. 12, 2019 | 8.1K views

Average Rating: 4.62 (13 votes)

You have a number of envelopes with widths and heights given as a pair of integers `(w, h)`. One envelope can fit into another if and only if both the width and height of one envelope is greater than the width and height of the other envelope.

What is the maximum number of envelopes can you Russian doll? (put one inside other)

Note:
Rotation is not allowed.

Example:

```
Input: [[5,4],[6,4],[6,7],[2,3]]
Output: 3
Explanation: The maximum number of envelopes you can Russian doll is 3
 ([2,3] => [5,4] => [6,7]).
```

Solution

Intuition

The problem boils down to a two dimensional version of the longest increasing subsequence problem (LIS).

We must find the longest sequence `seq` such that the elements in `seq[i+1]` are greater than the corresponding elements in `seq[i]` (this means that `seq[i]` can fit into `seq[i+1]`).

The problem we run into is that the items we are given come in arbitrary order - we can't just run a standard LIS algorithm because we're allowed to rearrange our data. How can we order our data in a way such that our LIS algorithm will always find the best answer?

Notes on the LIS algorithm

You can find the longest increasing subsequence problem with a solution here. If you're not familiar with the $O(N \log N)$ algorithm please go visit that question as it's a prerequisite for this one.

For the sake of completeness here's a brief explanation on how the LIS algorithm used below works:

`dp` is an array such that `dp[i]` is the smallest element that ends an increasing subsequence of length `i + 1`. Whenever we encounter a new element `e`, we binary search inside `dp` to find the largest index `i` such that `e` can end that subsequence. We then update `dp[i]` with `e`.

The length of the LIS is the same as the length of `dp`, as if `dp` has an index `i`, then it must have a subsequence of length `i+1`.

Approach 1: Sort + Longest Increasing Subsequence

Algorithm

We answer the question from the intuition by sorting. Let's pretend that we found the best arrangement of envelopes. We know that each envelope must be increasing in `w`, thus our best arrangement has to be a subsequence of all our envelopes sorted on `w`.

After we sort our envelopes, we can simply find the length of the longest increasing subsequence on the second dimension (`h`). Note that we use a clever trick to solve some edge cases:

Consider an input `[[1, 3], [1, 4], [1, 5], [2, 3]]`. If we simply sort and extract the second dimension we get `[3, 4, 5, 3]`, which implies that we can fit three envelopes (3, 4, 5). The problem is that we can only fit one envelope, since envelopes that are equal in the first dimension can't be put into each other.

In order fix this, we don't just sort increasing in the first dimension - we also sort *decreasing* on the second dimension, so two envelopes that are equal in the first dimension can never be in the same increasing subsequence.

Now when we sort and extract the second element from the input we get `[5, 4, 3, 3]`, which correctly reflects an LIS of one.

Implementation

Java | Python — Copy

```python
from bisect import bisect_left

class Solution:
    def maxEnvelopes(self, arr: List[List[int]]) -> int:
        # sort increasing in first dimension and decreasing on second
        arr.sort(key=lambda x: (x[0], -x[1]))

        def lis(nums):
            dp = []
            for i in range(len(nums)):
                idx = bisect_left(dp, nums[i])
                if idx == len(dp):
                    dp.append(nums[i])
                else:
                    dp[idx] = nums[i]
            return len(dp)
        # extract the second dimension and run the LIS
        return lis([i[1] for i in arr])
```

Complexity Analysis

- Time complexity : $O(N \log N)$, where N is the length of the input. Both sorting the array and finding the LIS happen in $O(N \log N)$
- Space complexity : $O(N)$. Our `lis` function requires an array `dp` which goes up to size N. Also the sorting algorithm we use may also take additional space.

Rate this article:

Previous Next

Comments: 7

Sort By

Type comment here... (Markdown is supported)

Preview Post

arctalex ★2 November 19, 2019 12:27 AM

Correction to python code

```
        def lis(nums):
            dp = []
            for i in range(len(nums)):
```

Read More

2 Share Reply

SHOW 1 REPLY

LeiH ★56 March 27, 2020 4:04 AM

Why is this problem labelled as Dynamic Programming? It should be just Binary Search...

1 Share Reply

SHOW 1 REPLY

yebenbenben ★0 June 24, 2020 10:52 AM

why is this thing a time-out for DP? i mean you can always create something that can be TLE for a slower algo, but what is the point of marking it as DP?

0 Share Reply

little_late ★53 June 20, 2020 12:26 PM

Even if you sort it based on increasing height and decreasing width (when the height is equal) then apply LIS on width. you get the correct answer.

```
static bool compare(const vector<int> &a, const vector<int> &b){
    return a[1] < b[1] or (a[1] == b[1] and a[0] > b[0]);
```

Read More

0 Share Reply

ping_pong ★827 March 29, 2020 11:04 AM

Can anyone explain rationale behind this sorting order. I couldn't get through this. when first dimension is equal, why are we sorting based on decreasing order of second dimension.

0 Share Reply

SHOW 1 REPLY

ningaloo ★138 January 20, 2020 9:59 AM

Is this problem possible with brute force recursion? My solution can't get more than 80/85 test cases without TLE. I'm sure there must be a better way of optimizing the tabulation criteria

```
class Solution:
    def maxEnvelopes(self, e: List[List[int]]) -> int:
```

Read More

0 Share Reply

SHOW 1 REPLY

ApoorvaGaurava ★0 December 2, 2019 3:11 PM

I doubt if there is any resemblance to longest increasing subsequence problem here, as you can't change the order of elements is LIS problem. This is just a variant of sorting with two comparators I think.

0 Share Reply

SHOW 2 REPLIES