Russian Doll Envelopes

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)

Example:
Given envelopes = `[[5,4],[6,4],[6,7],[2,3]]`, the maximum number of envelopes you can Russian doll is `3` ([2,3] => [5,4] => [6,7]).

Solution 1

1. Sort the array. Ascend on width and descend on height if width are same.
2. Find the longest increasing subsequence based on height.

- Since the width is increasing, we only need to consider height.
- [3, 4] cannot contains [3, 3], so we need to put [3, 4] before [3, 3] when sorting otherwise it will be counted as an increasing number if the order is [3, 3], [3, 4]

```
public int maxEnvelopes(int[][] envelopes) {
    if(envelopes == null || envelopes.length == 0
       || envelopes[0] == null || envelopes[0].length != 2)
        return 0;
    Arrays.sort(envelopes, new Comparator<int[]>(){
        public int compare(int[] arr1, int[] arr2){
            if(arr1[0] == arr2[0])
                return arr2[1] - arr1[1];
            else
                return arr1[0] - arr2[0];
       }
    });
    int dp[] = new int[envelopes.length];
    int len = 0;
    for(int[] envelope : envelopes){
        int index = Arrays.binarySearch(dp, 0, len, envelope[1]);
        if(index < 0)
            index = -(index + 1);
        dp[index] = envelope[1];
        if(index == len)
            len++;
    }
    return len;
}
```

written by TianhaoSong original link here

Solution 2

```
public int maxEnvelopes(int[][] envelopes) {
    if (   envelopes           == null
        || envelopes.length    == 0
        || envelopes[0]        == null
        || envelopes[0].length == 0){
        return 0;
    }

    Arrays.sort(envelopes, new Comparator<int[]>(){
        @Override
        public int compare(int[] e1, int[] e2){
            return Integer.compare(e1[0], e2[0]);
        }
    });

    int   n  = envelopes.length;
    int[] dp = new int[n];

    int ret = 0;
    for (int i = 0; i < n; i++){
        dp[i] = 1;

        for (int j = 0; j < i; j++){
            if (   envelopes[i][0] > envelopes[j][0]
                && envelopes[i][1] > envelopes[j][1]){
                dp[i] = Math.max(dp[i], 1 + dp[j]);
            }
        }

        ret = Math.max(ret, dp[i]);
    }
    return ret;
}
```

written by larrywang2014 original link here

Solution 3

See more explanation in Longest Increasing Subsequence Size (N log N)

```
def maxEnvelopes(self, envelopes):
    def bin_search(A, key):
        l, r = 0, len(A)
        while l < r:
            mid = (l+r)/2
            if A[mid][1] < key[1]:
                l = mid + 1
            else:
                r = mid
        return l
    envelopes.sort(
        cmp = lambda x,y: x[0]-y[0] if x[0] != y[0] else y[1]-x[1])
    n = len(envelopes)
    tails = []
    for i in range(n):
        e = envelopes[i]
        p = bin_search(tails, e)
        if p == len(tails):
            tails.append(e)
        else:
            tails[p] = e
    return len(tails)
```

written by JWZH original link here