IPO

Suppose LeetCode will start its IPO soon. In order to sell a good price of its shares to Venture Capital, LeetCode would like to work on some projects to increase its capital before the IPO. Since it has limited resources, it can only finish at most **k** distinct projects before the IPO. Help LeetCode design the best way to maximize its total capital after finishing at most **k** distinct projects.

You are given several projects. For each project **i**, it has a pure profit $\mathbf{P_i}$ and a minimum capital of $\mathbf{C_i}$ is needed to start the corresponding project. Initially, you have **W** capital. When you finish a project, you will obtain its pure profit and the profit will be added to your total capital.

To sum up, pick a list of at most **k** distinct projects from given projects to maximize your final capital, and output your final maximized capital.

Example 1:

```
Input: k=2, W=0, Profits=[1,2,3], Capital=[0,1,1].

Output: 4

Explanation: Since your initial capital is 0, you can only start the project indexe
d 0.
             After finishing it you will obtain profit 1 and your capital becomes 1
.
             With capital 1, you can either start the project indexed 1 or the proj
ect indexed 2.
             Since you can choose at most 2 projects, you need to finish the projec
t indexed 2 to get the maximum capital.
             Therefore, output the final maximized capital, which is 0 + 1 + 3 = 4.
```

Note:

1. You may assume all numbers in the input are non-negative integers.
2. The length of Profits array and Capital array will not exceed 50,000.
3. The answer is guaranteed to fit in a 32-bit signed integer.

Solution 1

The idea is each time we find a project with `max` profit and within current capital capability.
Algorithm:

1. Create (capital, profit) pairs and put them into PriorityQueue `pqCap`. This PriorityQueue sort by capital increasingly.
2. Keep polling pairs from `pqCap` until the project out of current capital capability. Put them into
PriorityQueue `pqPro` which sort by profit decreasingly.
3. Poll one from `pqPro`, it's guaranteed to be the project with `max` profit and within current capital capability. Add the profit to capital `W`.
4. Repeat step 2 and 3 till finish `k` steps or no suitable project (pqPro.isEmpty()).

Time Complexity: For worst case, each project will be inserted and polled from both PriorityQueues once, so the overall runtime complexity should be `O(NlgN)`, N is number of projects.

```
public class Solution {
    public int findMaximizedCapital(int k, int W, int[] Profits, int[] Capital) {
        PriorityQueue<int[]> pqCap = new PriorityQueue<>((a, b) -> (a[0] - b[0]))
;
        PriorityQueue<int[]> pqPro  = new PriorityQueue<>((a, b) -> (b[1] - a[1])
);

        for (int i = 0; i < Profits.length; i++) {
            pqCap.add(new int[] {Capital[i], Profits[i]});
        }

        for (int i = 0; i < k; i++) {
            while (!pqCap.isEmpty() && pqCap.peek()[0] <= W) {
                pqPro.add(pqCap.poll());
            }

            if (pqPro.isEmpty()) break;

            W += pqPro.poll()[1];
        }

        return W;
    }
}
```

written by shawngao original link here

Solution 2

Loop k times:
Add all possible projects (Capital <= W) into the priority queue with the priority = -Profit.
Get the project with the smallest priority (biggest Profit).
Add the Profit to W

```
def findMaximizedCapital(self, k, W, Profits, Capital):
    import Queue
    q = Queue.PriorityQueue()
    projects = sorted(zip(Profits, Capital), key=lambda l: l[1])
    j = 0
    for i in range(k):
      while j < len(projects):
        if projects[j][1] > W:
          break
        else:
          q.put((-projects[j][0], projects[j][0]))
        j = j + 1
      if q.empty():
        break
      else:
        W += q.get()[1]
    return W
```

written by lee215 original link here

Solution 3

Keep a max-heap of current possible profits. Insert possible profits as soon as their needed capital is reached.

```python
def findMaximizedCapital(self, k, W, Profits, Capital):
    current = []
    future = sorted(zip(Capital, Profits))[::-1]
    for _ in range(k):
        while future and future[-1][0] <= W:
            heapq.heappush(current, -future.pop()[1])
        if current:
            W -= heapq.heappop(current)
    return W
```

written by StefanPochmann original link here