Previous Next

502. IPO

June 15, 2019 | 3.1K views

Average Rating: 3.67 (6 votes)

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 indexed (
             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 project
             Since you can choose at most 2 projects, you need to finish the project i
             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

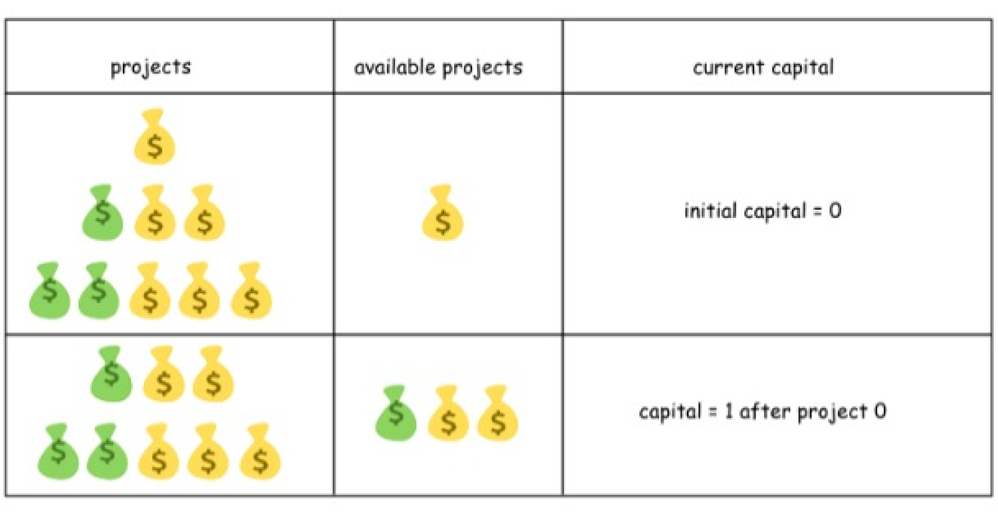
Approach 1: Greedy with Heap

Intuition

This is a greedy problem, and the only hard moment here is that capital is changing and so the list of available projects.

That could be solved by using two data structures:

- `projects` to track all the projects which are not implemented yet.
- `available` to track projects available with the current capital.



Algorithm

- To speed up, first check if here is a situation when all the projects are available with the initial capital `W >= max(Capital)`. If so, return the sum of kth largest elements in `Profits`.
- Build structure `projects` which
  - contains an information about capital and profit from each project,
  - is sorted by capitals, and
  - provides pop operation to remove already taken projects.
  - That could be min heap in Java and array of sets in Python.
- Iterate over k to choose k projects. At each step
  - Update a list of projects available with the current capital. One could choose max heap as a structure for available projects to simplify the peek of the most profitable one on the next step.
  - If there are any, choose the most profitable one, update `W` and proceed further.
  - Break, if the capital isn't large enough to start any project.
- Return `W`.

Implementation

Java Python3 Copy

```python
from heapq import nlargest, heappop, heappush
class Solution:
    def findMaximizedCapital(self, k: int, W: int, Profits: List[int], Capital: List[int]) -> int:
        # to speed up: if all projects are available
        if W >= max(Capital):
            return W + sum(nlargest(k, Profits))

        n = len(Profits)
        projects = [(Capital[i], Profits[i]) for i in range(n)]
        # sort the projects :
        # the most available (= the smallest capital) is the last one
        projects.sort(key = lambda x : -x[0])

        available = []
        while k > 0:
            # update available projects
            while projects and projects[-1][0] <= W:
                heappush(available, -projects.pop()[1])
            # if there are available projects,
            # pick the most profitable one
            if available:
                W -= heappop(available)
            # not enough capital to start any project
            else:
                break
            k -= 1
        return W
```

Complexity Analysis

- Time complexity:
  - $\mathcal{O}(N \log k)$ in the best case when all projects are available with the initial capital.
  - Otherwise, one needs $\mathcal{O}(N \log N)$ time to create and sort projects, and another $\mathcal{O}(N \log N)$ to update the available projects, and finally $\mathcal{O}(k \log N)$ to compute the capital.
  - Hence, the overall time complexity is $\mathcal{O}(N \log N + N \log N + k \log N)$. Assuming that $k < N$, we would have $\mathcal{O}(N \log N)$ time complexity at the end.
- Space complexity : $\mathcal{O}(N)$.

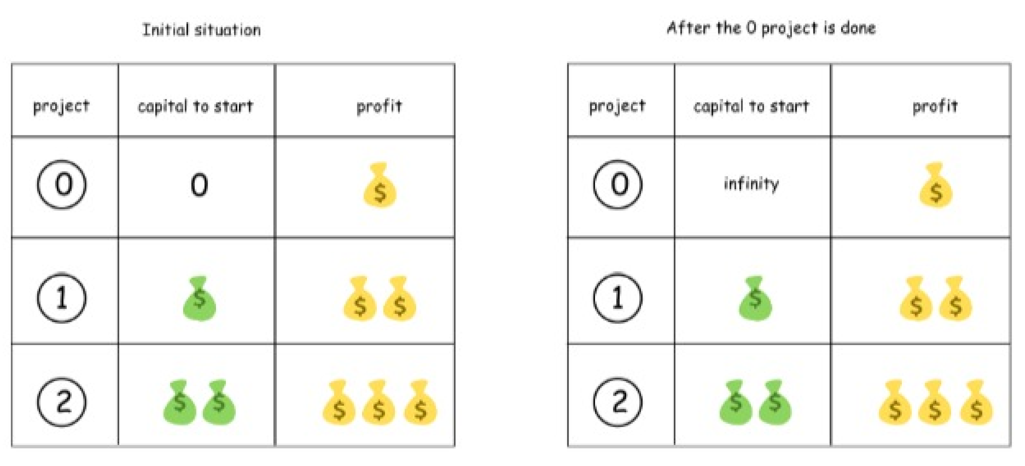
Approach 2: Greedy with Array

Intuition

In the previous approach, we applied the **Heap** data structure to track the available projects and the ones that are implemented. We could actually implement the Greedy algorithm without the need of Heap.

> The idea is to keep all projects in the list, and use the technique of **in-place** modification to mark the ones that have been selected.

Here one could set the capital to start as infinity for the projects which are already done.



Algorithm

- To speed up, first check if here is a situation when all the projects are available with the initial capital `W >= max(Capital)`. If so, return the sum of kth largest elements in `Profits`.
- Iterate over k to choose k projects. At each step
  - Choose the most profitable project. For that, iterate over all N projects and between the ones with `W >= Capital[j]`, choose the project with max `Profits[j]`.
  - Break, if the capital isn't large enough to start any project.
  - Update `W` to add the profit from the chosen project `W += Profits[idx]` and then discard this project from the further consideration `Capital[j] = Integer.MAX_VALUE`.
- Return `W`.

Implementation

Java Python3 Copy

```python
from heapq import nlargest
class Solution:
    def findMaximizedCapital(self, k: int, W: int, Profits: List[int], Capital: List[int]) -> int:
        # to speed up: if all projects are available
        if W >= max(Capital):
            return W + sum(nlargest(k, Profits))

        n = len(Profits)
        for i in range(min(n, k)):
            idx = -1
            # if there are available projects,
            # pick the most profitable one
            for j in range(n):
                if W >= Capital[j]:
                    if idx == -1:
                        idx = j
                    elif Profits[idx] < Profits[j]:
                        idx = j

            # not enough capital to start any project
            if idx == -1:
                break

            # add the profit from chosen project
            # and remove the project from further consideration
            W += Profits[idx]
            Capital[idx] = float('inf')
```

Complexity Analysis

- Time complexity:
  - $\mathcal{O}(N \log k)$ in the best case when all projects are available with the initial capital.
  - Otherwise, $\mathcal{O}(k \cdot N)$, assuming $k < N$.
- Space complexity:
  - If all projects are available with the initial capital, then $\mathcal{O}(k)$ in Java and $\mathcal{O}(1)$ in Python.
  - Otherwise, it is a constant space solution $\mathcal{O}(1)$.

Rate this article:

Previous Next

Comments: 6 Sort By

Type comment here... (Markdown is supported)

Preview Post

basi4869 94 June 29, 2019 1:32 AM

I think the time complexity for **Approach 1** is `O((N+k)lgN)` rather than `O(kNlgN)`. You never push into the heap more than `N` times or pop from it more than `k` times, and both heappush and heappop are `O(lgN)` operations.

7 Share Reply

SHOW 3 REPLIES

ZoroDuncan 22 February 25, 2020 9:54 AM

I think time complexity of the first solution is O(N*lgN)
Because all the projects will be pushed to both the heaps once at most and pop out from minHeap at most once. The time complexity is O(NlogN + KlogN), where 'N' is the total number of projects and 'K' is the number of projects we are selecting.

0 Share Reply

SHOW 1 REPLY

abhinvsinh 5 June 28, 2019 9:38 AM

Test input is incorrect. k=2, W=0, Profits=[1,2,3], Capital=[0,1,1].
How can capital[1] & capital[2] which are equal inn this example produce different profits.
This input needs to change as it fails a valid solution.

0 Share Reply

jol-jol 0 June 14, 2020 10:51 PM

After reading all the comments, should the optimal solution be Approach 1? The official solution still says Approach 1 is not the optimal one.

0 Share Reply

AlgorithmImplementer 583 January 4, 2020 10:35 PM

The naming of variables needs to be improved. Otherwise, its so hard to comprehend the code.

0 Share Reply

beidan 2 June 17, 2019 11:08 PM

Is there an assumption here: project[i] is guaranteed to be larger than capital[j] ?

0 Share Reply

SHOW 4 REPLIES