H-Index II

Follow up for H-Index: What if the `citations` array is sorted in ascending order? Could you optimize your algorithm?

1. Expected runtime complexity is in $O(\log n)$ and the input is sorted.

Solution 1

Just binary search, each time check citations[mid] case 1: citations[mid] == len-mid, then it means there are citations[mid] papers that have at least citations[mid] citations. case 2: citations[mid] > len-mid, then it means there are citations[mid] papers that have moret than citations[mid] citations, so we should continue searching in the left half case 3: citations[mid] < len-mid, we should continue searching in the right side After iteration, it is guaranteed that right+1 is the one we need to find (i.e. len-(right+1) papars have at least len-(righ+1) citations)

```
class Solution {
public:
    int hIndex(vector<int>& citations) {
        int left=0, len = citations.size(), right= len-1,  mid;
        while(left<=right)
        {
            mid=(left+right)>>1;
            if(citations[mid]== (len-mid)) return citations[mid];
            else if(citations[mid] > (len-mid)) right = mid - 1;
            else left = mid + 1;
        }
        return len - (right+1);
    }
};
```

written by dong.wang.1694 original link here

Solution 2

The basic idea of this solution is to use **binary search** to find the minimum `index` such that

```
citations[index] >= length(citations) - index
```

After finding this `index`, the answer is `length(citations) - index`.

This logic is very similar to the C++ function `lower_bound` or `upper_bound`.

Complexities:

- Time: O(log n)
- Space: O(1)

C++:

```
class Solution {
public:
    int hIndex(vector<int>& citations) {
        int size = citations.size();

        int first = 0;
        int mid;
        int count = size;
        int step;

        while (count > 0) {
            step = count / 2;
            mid = first + step;
            if (citations[mid] < size - mid) {
                first = mid + 1;
                count -= (step + 1);
            }
            else {
                count = step;
            }
        }

        return size - first;
    }
};
```

Java:

```java
public class Solution {
    public int hIndex(int[] citations) {
        int len = citations.length;

        int first = 0;
        int mid;
        int count = len;
        int step;

        while (count > 0) {
            step = count / 2;
            mid = first + step;
            if (citations[mid] < len - mid) {
                first = mid + 1;
                count -= (step + 1);
            }
            else {
                count = step;
            }
        }

        return len - first;
    }
}
```

Python:

```python
class Solution(object):
    def hIndex(self, citations):
        """
        :type citations: List[int]
        :rtype: int
        """

        length = len(citations)

        first = 0
        count = length

        while count > 0:
            step = count / 2
            mid = first + step
            if citations[mid] < length - mid:
                first = mid + 1
                count -= (step + 1)
            else:
                count = step

        return length - first
```

@daviantan1890 @ruichang Thank you for your comments and discussions.

I am very sure that two-branch binary search is more efficient than three branch binary search. and (low + high) is not good idea since it may rely on the overflow behavior. In fact, using `count` `step` `first` `mid` is the standard implement way of C++, so I do not think there are better ways to implement the binary search.

written by zhiqing_xiao original link here

Solution 3

The basic idea comes from the description**h of his/her N papers have at least h citations each**. Therefore, we know `if "mid + 1" is a valid h index, it means value of position "citationsSize - mid - 1" must exceed "mid"`. After we find a valid h index, we go on searching on the right part to see if we can find a larger h index. If it's not a valid h index, the h index can be found in the left part and we simply follow the standard binary search to solve this problem.

```
int hIndex(int* citations, int citationsSize) {
    int lo = 0, hi = citationsSize, mid, index = 0;
    while (lo <= hi) {
        mid = lo + ((hi - lo) >> 1);
        if (citations[citationsSize - mid - 1] > mid) {
            lo = mid + 1;
            index = lo;
        } else {
            hi = mid - 1;
        }
    }
    return index;
}
```

written by hdchen original link here