

278. First Bad Version

March 5, 2016 | 408.5K views

★★★★★
Average Rating: 4.65 (237 votes)

You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.

Suppose you have n versions `[1, 2, ..., n]` and you want to find out the first bad one, which causes all the following ones to be bad.

You are given an API `bool isBadVersion(version)` which will return whether `version` is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API.

Example:

Given $n = 5$, and `version = 4` is the first bad version.

```
call isBadVersion(3) -> false
call isBadVersion(5) -> true
call isBadVersion(4) -> true
```

Then 4 is the first bad version.

Summary

This is a very simple problem. There is a subtle trap that you may fall into if you are not careful. Other than that, it is a direct application of a very famous algorithm.

Solution

Approach #1 (Linear Scan) [Time Limit Exceeded]

The straight forward way is to brute force it by doing a linear scan.

```
Java Copy
1 public int firstBadVersion(int n) {
2     for (int i = 1; i < n; i++) {
3         if (isBadVersion(i)) {
4             return i;
5         }
6     }
7     return n;
8 }
```

Complexity analysis

- Time complexity : $O(n)$. Assume that `isBadVersion(version)` takes constant time to check if a `version` is bad. It takes at most $n - 1$ checks, therefore the overall time complexity is $O(n)$.
- Space complexity : $O(1)$.

Approach #2 (Binary Search) [Accepted]

It is not difficult to see that this could be solved using a classic algorithm - Binary search. Let us see how the search space could be halved each time below.

Scenario #1: `isBadVersion(mid) == false`

```
1 2 3 4 5 6 7 8 9
G G G G G B B B      G = Good, B = Bad
|       |       |
left    mid     right
```

Let us look at the first scenario above where `isBadVersion(mid) == false`. We know that all versions preceding and including `mid` are all good. So we set `left = mid + 1` to indicate that the new search space is the interval `[mid + 1, right]` (inclusive).

Scenario #2: `isBadVersion(mid) == true`

```
1 2 3 4 5 6 7 8 9
G G G B B B B B      G = Good, B = Bad
|       |       |
left    mid     right
```

The only scenario left is where `isBadVersion(mid) == true`. This tells us that `mid` may or may not be the first bad version, but we can tell for sure that all versions after `mid` can be discarded. Therefore we set `right = mid` as the new search space of interval `[left, mid]` (inclusive).

In our case, we indicate `left` and `right` as the boundary of our search space (both inclusive). This is why we initialize `left = 1` and `right = n`. How about the terminating condition? We could guess that `left` and `right` eventually both meet and it must be the first bad version, but how could you tell for sure?

The formal way is to [prove by induction](#), which you can read up yourself if you are interested. Here is a helpful tip to quickly prove the correctness of your binary search algorithm during an interview. We just need to test an input of size 2. Check if it reduces the search space to a single element (which must be the answer) for both of the scenarios above. If not, your algorithm will never terminate.

If you are setting `mid = (left+right)/2`, you have to be very careful. Unless you are using a language that does not overflow such as [Python](#), `left + right` could overflow. One way to fix this is to use `left + (right-left)/2` instead.

If you fall into this subtle overflow bug, you are not alone. Even Jon Bentley's own implementation of binary search had this [overflow bug](#) and remained undetected for over twenty years.

```
Java Copy
1 public int firstBadVersion(int n) {
2     int left = 1;
3     int right = n;
4     while (left < right) {
5         int mid = left + (right - left) / 2;
6         if (isBadVersion(mid)) {
7             right = mid;
8         } else {
9             left = mid + 1;
10        }
11    }
12    return left;
13 }
```


Complexity analysis

- Time complexity : $O(\log n)$. The search space is halved each time, so the time complexity is $O(\log n)$.
- Space complexity : $O(1)$.

Rate this article: ★★★★★

Comments: 52

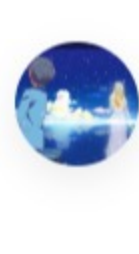
Sort By



Type comment here... (Markdown is supported)

Preview


Post

 kevincongcc ★491 January 30, 2018 6:39 AM

well,got TLE five times because I use (left + right) / 2,now I know why should use left + (right - left) / 2.


174 1 2 Share Reply

SHOW 16 REPLIES

 terrible_whiteboard ★633 May 19, 2020 6:15 PM

I made a video if anyone is having trouble understanding the solution (clickable link)


<https://youtu.be/Pj1eKrBx4E>



Read More

24 1 2 Share Reply

SHOW 1 REPLY

 ankitshah009 ★32 March 12, 2019 4:29 AM

Efficient python library

```
import bisect
class Solution():
    def firstBadVersion(self, n):
```

Read More

31 1 2 Share Reply


SHOW 3 REPLIES

 sotondolphin ★9 November 30, 2017 3:11 PM

why not just start from the lastest version - 1? since the bad version is likely to be near the latest version

10 1 2 Share Reply


SHOW 3 REPLIES

 idomiralin ★8 October 5, 2017 11:55 PM

Why when I use mid = (left + right) / 2 time limit exceeded is returned and when we use mid = left + (right - left) / 2 it works?

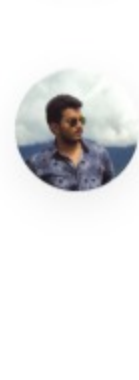
8 1 2 Share Reply

SHOW 2 REPLIES

 suhaboncukcu ★2 January 29, 2018 1:56 AM

@Hidestor, I got the same result till I try low + ((high-low) >>> 1) (language: js)

2 1 2 Share Reply


 codeXcode ★30 January 10, 2018 11:11 PM

To my utter surprise low+(high-low)>>1 is giving TLE and low+(high-low)/2 is getting AC. How is this possible?

We know bit manipulation is faster than arithmetic calculations or I am wrong in my assumption?

2 1 2 Share Reply

SHOW 2 REPLIES

 just_morris ★1 May 22, 2019 2:57 AM

Python3 error:

TypeError: firstBadVersion() takes 2 positional arguments but 3 were given in main:

... ret = Solution().firstBadVersion(n, bad) ...


after i fix it (def firstBadVersion(self, n, bad): OR ret = Solution().firstBadVersion(n))

NameError: name 'isBadVersion' is not defined

Read More

1 1 2 Share Reply

SHOW 1 REPLY

 Try_Your_Best ★7 March 21, 2019 12:13 PM

For Java, here is the JDK 11's implementation of **Binary Search**:

```
private static int binarySearch0(long[] a, int fromIndex, int toIndex, long key)
{
    int low = fromIndex;
```

Read More

1 1 2 Share Reply

SHOW 2 REPLIES

 asrajavel267 ★47 May 9, 2019 9:18 AM

What are we supposed to return if there are no bad versions?

0 1 2 Share Reply

SHOW 1 REPLY