Binary Search

Jan 30, 2021

Agenda

- Concept
- Examples
- Practice problems

Agenda

- Concept
- Examples
- Practice problems

Search

- One of the most common computational problems (along with sorting) is searching.
- In the simplest form, the input to the search problem is a list L and an item k and we are asked if k belongs to L. (The in operator in Python.)
- In a common variant, we might be asked for the index of k in L, if k does belong to L. (The L.index() method in Python.)

Search List

Python provides several built-in operations for searching lists:

- elem in L: evaluates to True if elem is in list L
- L.index(elem): returns the index of the first occurrence of elem in L; is an error if elem is not in L.
- L.count(elem): returns the number of occurrences of elem in L.

Other related operations:

- min(L), max(L): these return the minimum element and maximum element respectively of L.

Linear Search

- If we don’t know anything about L, then the only way to solve the problem is by scanning the list L completely in some systematic manner.
- This takes time proportional to the size of the list, in the worst case.
- And for this reason, this is called linear search.
- Linear search can be quite inefficient for many applications because search is such a common operation in programs.
- The Python search operations mentioned in the previous slide all perform linear search because they are expected to work on any list.

Binary Search

- If the list L is known to be sorted (in ascending or descending order), then we can use a more efficient algorithm called binary search.
- Suppose that L is sorted in ascending order.
- Compare k with the middle element of L.
  - If k == L[middle], we are done
  - If k < L[middle], we need to search the first half of L
  - If k > L[middle], we need to search the second half of L
- After one comparison, the size of the problem shrinks to a half.

Complexity

- Worst case scenario: k is not in L
- After each comparison of k with L[middle], the problem size shrinks to a half of what it was before.

Problem size	# of iterations
N	0
N/2	1
N/2^2	2
…	…
N/2^t	t

Complexity

- Therefore, we need $N = 2^t$ for the problem to shrink to size of 1
- Hence $t = \log_2 N$
- The complexity if $O(\log N)$

Agenda

- Concept
- Examples
- Practice problems

Example 1 Problem

704. Binary Search

Easy 1084 55 Add to List Share

Given a **sorted** (in ascending order) integer array `nums` of `n` elements and a `target` value, write a function to search `target` in `nums`. If `target` exists, then return its index, otherwise return `-1`.

Example 1:

```
Input: nums = [-1,0,3,5,9,12], target = 9
Output: 4
Explanation: 9 exists in nums and its index is 4
```

Example 2:

```
Input: nums = [-1,0,3,5,9,12], target = 2
Output: -1
Explanation: 2 does not exist in nums so return -1
```

Example 1 Solution

```python
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        # explicitly maintain two indices left and right
        left = 0
        right = len(nums) - 1
        # the sublist from left to right is what still remains to be searched
        while left <= right:
            # maintain a third index called middle
            middle = (left + right) // 2
            if nums[middle] < target:
                # look for the target in the right half
                left = middle + 1
            elif nums[middle] > target:
                # look for the target in the left half
                right = middle - 1
            else:
                # find the target and return its index
                return middle
        return -1
```

Example 2 Problem

34. Find First and Last Position of Element in Sorted Array

Medium 4822 185 Add to List Share

Given an array of integers `nums` sorted in ascending order, find the starting and ending position of a given `target` value.

If `target` is not found in the array, return `[-1, -1]`.

Follow up: Could you write an algorithm with `O(log n)` runtime complexity?

Example 1:

```
Input: nums = [5,7,7,8,8,10], target = 8
Output: [3,4]
```

Example 2:

```
Input: nums = [5,7,7,8,8,10], target = 6
Output: [-1,-1]
```

Example 2 Solution

```python
class Solution:
    def searchRange(self, nums, target):
        # write the first binary search algorithm
        def binarySearchLeft(A, x):
            # borrow the template from the previous example
            left = 0
            right = len(A) - 1
            while left <= right:
                middle = (left + right) // 2
                if x > A[middle]:
                    left = middle + 1
                else:
                    right = middle - 1
            # return the index of the element at the very beginning
            return left

        # write the second binary search algorithm
        def binarySearchRight(A, x):
            # borrow the template from the previous example
            left = 0
            right = len(A) - 1
            while left <= right:
                middle = (left + right) // 2
                if x >= A[middle]:
                    left = middle + 1
                else:
                    right = middle - 1
            # return the index of the element at the very end
            return right

        left = binarySearchLeft(nums, target)
        right = binarySearchRight(nums, target)
        if left <= right:
            return [left, right]
        else:
            return [-1, -1]
```

Example 3 Problem

- 温故而知新
- Binary search can be used to reduce complexity

2.3. Longest Increasing Subsequence (Homework)

Given an array x of numbers, return the length of the longest strictly increasing subsequence.

- Our generic DP solution yields $O(n^2)$ time complexity.
- There exists an $O(n \log n)$ solution.

What does this remind you of?

Example 3 Problem

300. Longest Increasing Subsequence

Medium 6315 149 Add to List Share

Given an integer array `nums`, return the length of the longest strictly increasing subsequence.

A **subsequence** is a sequence that can be derived from an array by deleting some or no elements without changing the order of the remaining elements. For example, `[3,6,2,7]` is a subsequence of the array `[0,3,1,6,2,2,7]`.

Example 1:

```
Input: nums = [10,9,2,5,3,7,101,18]
Output: 4
Explanation: The longest increasing subsequence is [2,3,7,101], therefore the length is 4.
```

Example 2:

```
Input: nums = [0,1,0,3,2,3]
Output: 4
```

Example 3 Solution 1

```python
class Solution:
    def lengthOfLIS(self, nums):
        if not nums:
            return 0
        # let dp[i] be the length of the longest increasing subsequence ending with ith element
        dp = [1] * len(nums)

        # loop through all elements in the list
        for i in range(len(nums)):
            # for the ith element, loop through all elements before it one by one
            for j in range(i):
                # compare each of those elements (e.g., jth element) with the ith element
                if nums[i] > nums[j]:
                    # if jth element is smaller than the ith element, we can add ith element after jth element
                    # the updated dp[i] should be the larger between itself and length of subsequence ending with jth plus 1
                    dp[i] = max(dp[i], dp[j] + 1)
        return max(dp)
```

Example 3 Solution 2

```python
class Solution:
    def lengthOfLIS(self, nums):
        # maintain the list of the current Longest Increasing Subsequence
        current_LIS = [0] * len(nums)
        # the initial size of that list is 0
        size = 0
        # loop through all items in the given list nums
        for item in nums:
            # for each item, initialize two indices: left and right
            left = 0
            # the right index is equal to the size of the current Longest Increasing Subsequence
            right = size
            # use a while loop to replace the second "for" loop in the previous slide
            while left != right:
                # maintain a third index called middle
                middle = left + (right - left) // 2
                # compare the current item with the middle number in the current Longest Increasing Subsequence
                if current_LIS[middle] < item:
                    # if the current item is larger, then update the left index and search the right half only
                    left = middle + 1
                else:
                    # otherwise, the current item is smaller, then update the right index and search the left half only
                    right = middle
            # update the current Longest Increasing Subsequence to include the current item
            current_LIS[left] = item
            # update the size of the current Longest Increasing Subsequence
            size = max(size, left + 1)
        return size
```

Example 4 Problem

410. Split Array Largest Sum

Hard 2327 86 Add to List Share

Given an array `nums` which consists of non-negative integers and an integer `m`, you can split the array into `m` non-empty continuous subarrays.

Write an algorithm to minimize the largest sum among these `m` subarrays.

Example 1:

```
Input: nums = [7,2,5,10,8], m = 2
Output: 18
Explanation:
There are four ways to split nums into two subarrays.
The best way is to split it into [7,2,5] and [10,8],
where the largest sum among the two subarrays is only 18.
```

Example 2:

```
Input: nums = [1,2,3,4,5], m = 2
Output: 9
```

Example 4 Approach

- Split an array, called A, into m different sub-arrays
- The sum of each sub-array forms an m-element array
  - This array is named sum_sub_array
- The smallest possible value of sum_sub_array is $\max(A)$
  - This number must be in one of those m different sub-arrays
- The largest possible value of sum_sub_array is $\text{sum}(A)$
  - This number comes from a sub-array of all elements in A
- Problem reformulation: search within the range $\max(A)$ to $\text{sum}(A)$ and find the minimum value such that we can split the array A into m sub-arrays and none of those sub-arrays has a sum larger than that.

Example 4 Solution

```python
class Solution():
    # write a helper function that splits nums into sub-arrays and compares their sum to middle
    def helper(self, nums, m, middle):
        cut_off = 0
        current_sum = 0
        for item in nums:
            current_sum = current_sum + item
            if current_sum > middle:
                cut_off = cut_off + 1
                current_sum = item
        return (cut_off + 1 <= m)

    def splitArray(self, nums, m):
        # search between the smallest possible value and largest possible value
        low = max(nums)
        high = sum(nums)
        ans = -1
        # apply the binary search algorithm
        while low <= high:
            middle = (low + high) // 2
            # decide whether we can find m sub-arrays, each of which has a sum smaller than middle
            if self.helper(nums, m, middle):
                # if True, search the smaller half
                ans = middle
                high = middle - 1
            else:
                # if False, search the larger half
                low = middle + 1
        return ans
```

```
import inspect
```

```
Solution().splitArray([7,2,5,10,8], 2)
```

```
function splitArray()
low 10
high 32
        self=<__main__.Solution object at 0x000001FF243246A0>
        nums=[7, 2, 5, 10, 8]
        m=2
function helper()
item 7
        self=<__main__.Solution object at 0x000001FF243246A0>
        nums=[7, 2, 5, 10, 8]
        m=2
        middle=21
current_sum 7
cut_off 0
function helper()
item 2
        self=<__main__.Solution object at 0x000001FF243246A0>
        nums=[7, 2, 5, 10, 8]
        m=2
        middle=21
current_sum 9
cut_off 0
function helper()
item 5
        self=<__main__.Solution object at 0x000001FF243246A0>
        nums=[7, 2, 5, 10, 8]
        m=2
        middle=21
current_sum 14
cut_off 0
```

```
function helper()
item 10
        self=<__main__.Solution object at 0x000001FF243246A0>
        nums=[7, 2, 5, 10, 8]
        m=2
        middle=21
current_sum 24
Because current_sum is larger than middle, we update current_sum to 10
cut_off 1
function helper()
item 8
        self=<__main__.Solution object at 0x000001FF243246A0>
        nums=[7, 2, 5, 10, 8]
        m=2
        middle=21
current_sum 18
cut_off 1
function splitArray()
low 10
high 20
        self=<__main__.Solution object at 0x000001FF243246A0>
        nums=[7, 2, 5, 10, 8]
        m=2
function helper()
item 7
        self=<__main__.Solution object at 0x000001FF243246A0>
        nums=[7, 2, 5, 10, 8]
        m=2
        middle=15
current_sum 7
cut_off 0
```

Agenda

- Concept
- Examples
- Practice problems

Practice Problems

- Leetcode 35. Search Insert Position: https://leetcode.com/problems/search-insert-position/
- Leetcode 287. Find the Duplicate Number: https://leetcode.com/problems/find-the-duplicate-number/
- Leetcode 378. Kth Smallest Element in a Sorted Matrix: https://leetcode.com/problems/kth-smallest-element-in-a-sorted-matrix/
- Leetcode 719. Find K-th Smallest Pair Distance: https://leetcode.com/problems/find-k-th-smallest-pair-distance/
- Easy, Medium, Medium, and Hard
- Set a timer for 90 minutes, like a leetcode contest