Theory: Binary search in Python

1 hour 0 / 5 problems solved

Binary search is an efficient algorithm for searching a target element in a **sorted** list. The algorithm starts with comparing the middle element of a list with the target one. If they are equal, the algorithm returns the index of the found element. If the target element is less or greater than the middle one, the search is recursively applied for the left or for the right half of the list accordingly. The process continues until either the middle and the target elements are equal or the current list is empty. In the last case, the algorithm returns -1 indicating that the target value is not found.

For a list of size n, the algorithm works in $O(\log n)$. In this topic, we will consider two Python implementations of binary search: iterative and recursive.

§1. Iterative implementation

Below is an iterative version of the binary search algorithm implemented in Python:

```
def binary_search(elements, target):
    left, right = 0, len(elements) - 1

    while left <= right:
        middle = (left + right) // 2

        if elements[middle] == target:
            return middle
        elif target < elements[middle]:

            right = middle - 1

        else:

            left = middle + 1



    return -1
```

The `binary_search` function takes a sorted list named `elements` and a value named `target` as arguments. As output, the function returns either the index of the element equal to `target` or -1 if `target` is not found.

In this function, we first create two variables, `left` and `right`, to store the current range of the list where the target element is searched. At each step of the `while` loop, we compare the middle element of the current range with the target value. If the elements are equal, we return the index of the middle element. Otherwise, we update the current range and continue the search in the part of the list where the target element might be presented.

If the current range becomes empty, that is, the condition `left <= right` is `False`, the function returns -1 indicating that the target element is not found.

Below are several examples of how the function can be used:

```
elements = [10, 13, 15, 20, 21, 25]

indexof_10 = binary_search(elements, 10)  # 0
indexof_13 = binary_search(elements, 13)  # 1
indexof_15 = binary_search(elements, 15)  # 2
indexof_20 = binary_search(elements, 20)  # 3
indexof_21 = binary_search(elements, 21)  # 4
indexof_25 = binary_search(elements, 25)  # 5


indexof_19 = binary_search(elements, 19)  # -1

indexof_23 = binary_search(elements, 23)  # -1

indexof_42 = binary_search(elements, 42)  # -1
```

For the elements presented in the list, the function returns their indexes. If an element is not presented in the list, it returns -1, quite as expected.

§2. Recursive implementation

Now, let's see how the binary search algorithm can be implemented using recursion:

```
def binary_search(elements, target, left, right):
    if left > right:
        return -1

    middle = (left + right) // 2

    if elements[middle] == target:
        return middle
    elif target < elements[middle]:

        return binary_search(elements, target, left, middle - 1)

    else:

        return binary_search(elements, target, middle + 1, right)
```

In this version, we pass the boundaries of the current range as function arguments. Initially, we check whether the current range is empty and if it is, we return -1 indicating that the target element is not found. Otherwise, we compare the middle element with the target one and then either return the index of the middle element if the equality holds or update the range boundaries and recursively call the function on these new boundaries.

Below are several usage examples:

```
elements = [7, 10, 15, 20, 25]

indexof_7 = binary_search(elements, 7, left=0, right=len(elements) - 1)  # 0

indexof_10 = binary_search(elements, 10, left=0, right=len(elements) - 1)  # 1

indexof_15 = binary_search(elements, 15, left=0, right=len(elements) - 1)  # 2

indexof_20 = binary_search(elements, 20, left=0, right=len(elements) - 1)  # 3

indexof_25 = binary_search(elements, 25, left=0, right=len(elements) - 1)  # 4


indexof_9 = binary_search(elements, 9, left=0, right=len(elements) - 1)  # -1

indexof_23 = binary_search(elements, 23, left=0, right=len(elements) - 1)  # -1
```

§3. Summary

In this topic, we considered two different implementations of binary search in Python: iterative and recursive. These versions are equivalent but the last one might require a bit more additional memory to store a stack of recursive calls.

Note that in Python, there is a module named `bisect` that has some methods to work with a sorted list. In particular, it has a function called `bisect_left` that can be used similarly to the described binary search method. See the documentation for details.

Report a typo

45 users liked this theory. **0** didn't like it. **What about you?**

Start practicing

Comments (0) Hints (0) Useful links (0) Show discussion