Theory: Linear search in Python

30 minutes 0 / 5 problems solved

Skip this topic Start practicing

482 users solved this topic. Latest completion was 28 minutes ago.

Current topic:

Linear search in Python

Topic depends on:

✓ Linear search

✓ Algorithms in Python

Topic is required for:

Binary search in Python

Linear search is a simple algorithm for searching a specified value in a list or a similar collection. The main idea of the algorithm is to sequentially compare each element of the collection with the target element and return the first element equal to the target. In case none of the elements are equal to the target one, the algorithm returns a special value indicating that the target element hasn't been found. For a list of the size n, the worst-case time complexity of the algorithm is $O(n)$.

§1. Implementation in Python

Now, let's consider how the linear search algorithm can be implemented in Python:

```
def search(elements, target):
    index = -1

    for i, elem in enumerate(elements):
        if elem == target:
            index = i
            break

    return index
```

The function takes a list named `elements`, a value named `target` and returns the index of the first element equal to `target` or -1 if `target` is not in the list. The implementation is simple and straightforward: first, we create a variable named `index` to store the index of the found element and initialize it with -1. Then, we apply the `enumerate` function for the input list to get a new list of its elements along with their indexes. Next, we start iterating through this new list using the `for` loop and compare the current element with `target`. If the elements are equal, we update the index and break the loop. Finally, we return the `index` variable from the function.

§2. Usage examples

Here is how the described algorithm can be used:

```
names = ["Alice", "Bob", "Mike", "Kate", "John", "Bill", "Andrew", "David"]
print(search(names, "Alice"))  # 0
print(search(names, "Bob"))  # 1
print(search(names, "Andrew"))  # 6
print(search(names, "Tomas"))  # -1
print(search(names, "Ann"))  # -1
print(search(names, "Oliver"))  # -1
```

Note that there is the `in` operator in Python that can also be used to search elements in lists:

```
names = ["Alice", "Bob", "Mike", "Kate", "John", "Bill", "Andrew", "David"]
print("Alice" in names)  # True
print("Bob" in names)  # True
print("Andrew" in names)  # True
print("Tomas" in names)  # False
print("Ann" in names)  # False
print("Oliver" in names)  # False
```

The difference with the given algorithm is that this operator returns a boolean value: `True` if the target element is contained in the list and `False` otherwise.

§3. Searching values in sorted lists

Note that the described algorithm works both for unsorted and sorted lists. However, if we know in advance that the input list is sorted, we may modify the algorithm to make it a bit more efficient:

```
def search(elements, target):
    index = -1

    for i, elem in enumerate(elements):
        if elem == target:
            index = i
            break

        if elem > target:
            break

    return index
```

Here, we add an extra condition to check if the current element is greater than `target`. If the condition holds, we may break the iteration since we know that the remaining elements are not equal to `target`. This allows us to avoid unnecessary comparisons. Let's see an example:

```
numbers = [4, 8, 15, 16, 23, 42]
print(search(numbers, 14))  # -1
```

In this case, the algorithm performs only 6 comparisons while in the initial version all the elements would be considered. Although this implementation might work better in some cases, the worst-case time complexity is still $O(n)$. Note that there is a more efficient algorithm for searching elements in sorted lists named **binary search.** This algorithm will be considered in the next topic.

Report a typo

54 users liked this theory. **1** didn't like it. **What about you?**

Start practicing

Comments (6) Hints (0) Useful links (0) Show discussion

Table of contents: