University of Toronto - Computer Science - Image

Searching for a Value in a List: Linear Search

Exploring `list.index(value)`

`list.index(value)` returns the index of the first occurrence of `value` in the list. For instance:

```
>>> L = ['a', 'b', 'c', 'a', 'd']
>>> L.index('a')
0
>>> L.index('c')
2
>>> L.index('d')
4
>>>
```

Notice that the value produced by `L.index('a')` is `0` (the index of the first occurrence of `'a'`), and not `3` (the index of the second occurrence of `'a'`).

When `L.index('d')` is executed, it first examines the item at index `0` of `L`, which is `'a'`. Since that is not the value we are looking for, it moves on to index `1`, which contains `'b'`. Again, this is not what we are looking for, so it moves on. This pattern continues, until it finds `'d'` at index `4`. This way of searching is called *linear*, which refers to items arranged in a line. Essentially, we are searching for items in a linear manner.

Linear Search

Let us understand more deeply how linear search works. Here is a list that we want to search through:

'a'	'b'	'c'	'a'	'd'

Next, let us index the items in the list:

0	1	2	3	4
'a'	'b'	'c'	'a'	'd'

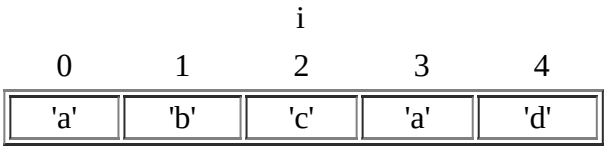
Now, let us search for value `'d'` in the list. We will now draw our indices above the position that we are currently inspecting:

	i			
0	1	2	3	4
'a'	'b'	'c'	'a'	'd'

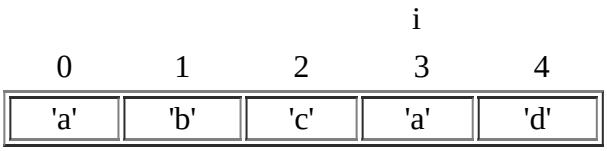
We can now start our linear search for the value `'d'`. Since at index `0` we cannot find `'d'`, we will advance `i` to refer to the next index and search at that new index. `i` now refers to `1`:

		i		
0	1	2	3	4
'a'	'b'	'c'	'a'	'd'

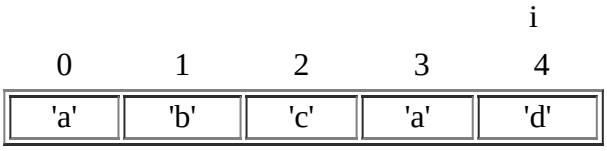
Since we did not find what we were looking for, we move to the item at the next index:



Again, we did not find what we were looking for, so we move to the item at the next index:



Again, we move to the item at the next index:



Finally, we have found the value we were searching for, which is at index 4. One question that we must ask ourselves is, what do we return if we do not find the item that we are searching for? In that case, we will return -1.

Implementation

For the linear search algorithm, here is the code we have developed:

```
def linear_search(L, v):
    """ (list, object) -> int

    Return the index of the first occurrence of v in L, or
    return -1 if v is not in L.

    >>> linear_search([2, 3, 5, 3], 2)
    0
    >>> linear_search([2, 3, 5, 3], 5)
    2
    >>> linear_search([2, 3, 5, 3], 8)
    -1
    """

    i = 0

    while i != len(L) and v != L[i]:
        i = i + 1

    if i == len(L):
        return -1
    else:
        return i
```

Here are some important details for the above function:

1. The `while` loop has two conditions for when to end the loop. First, we will continue looping while the index `i` is not equal to the length of the list. And since the index starts from 0, we will be examining each item in the list in a forward manner. Second, we will continue looping as long as `v`, the value we are searching for, is not in the current position of the list `v!=L[i]`. In other words, if `v` is in the current position of the list, then we know we have found the item we are looking for, and we can exit the `while` loop.

2. There is an `if` statement after the while loop ends. This is because there are two possible reasons why the while loop could have exited. First is the case when we have not found the item we are looking for, and this happens when the index `i` is equal to the length of the list `(i == len(L))`. In this case we return -1. Otherwise, if `i` is not equal to the length of the list, then `i` is less than the length of the list, so we must return `i`. This is because `i` is the index at which we found the value `v`, and that is precisely why the while loop ended before `i` became equal to the length of the list.

Jennifer Campbell • Paul Gries
University of Toronto
