

UE5 Fundamentals of Algorithms

Lecture 3: Lists, search, sort

Ecole Centrale de Lyon, Bachelor of Science in Data Science for Responsible Business

Romain Vuillemot



Outline

- Definition and examples of lists
- Manipulate
- Search
- Sort
- Iterate
- Filter/Map

What is a list?

*A **list** is a linear data structure*

- A sequence of elements
- Each element has a position (index)
- The order of the elements is important
- The elements can be of any type

Examples of lists:

```
In [ ]: numbers_list = [1, 2, 3, 4, 5]
strings_list = ["apple", "banana", "cherry", "date"]
mixed_list = [1, "apple", 3.14, True]
nested_list = [[1, 2, 3], ["a", "b", "c"], [True, False]]
```

List operations

- **Access** an element at a given position
- **Insert** an element at a given position
- **Remove** an element at a given position
- **Search** an element
- **Sort** the list
- **Reverse** the list

Access

Return a value

```
In [1]: L = [1, 2, 3]

for val in L:
    print(val)
```

```
1
2
3
```

Example: is a list ordered?

```
In [2]: def estordonnee(liste):  
  
        for i in range(len(liste) - 1):  
            if liste[i + 1] < liste[i]:  
                return False  
        return True  
  
print(estordonnee([1,2,3,4]))  
print(estordonnee([1,2,3,4,1]))
```

True

False

Insert

Add elements in the list (regardless the index)

```
In [3]: from time import time

def compute_average(n):
    data = []
    start = time()
    for k in range(n):
        data.append(None)
    end = time()
    return (end - start) / n
```

```
In [4]: compute_average(20)
```

```
Out[4]: 1.5497207641601563e-07
```

```
In [5]: my_list = [1, 2, 3, 4, 5]
insert_elements = [99, 100]
my_list = my_list[:2] + insert_elements + my_list[2:]
print(my_list)
```

```
[1, 2, 99, 100, 3, 4, 5]
```

Search

Given a list of elements, find the position of a given element

```
In [6]: def search_element_in_list(element, list):  
        for i in list:  
            if i == element:  
                return True  
        return False
```

```
In [7]: L = [1, 2, 3, 4, 5]  
        element_to_find = 3  
  
        try:  
            index = L.index(element_to_find)  
            print(f"{element_to_find} found at index {index}.")  
        except ValueError:  
            print(f"{element_to_find} is not in the list.")
```

3 found at index 2.

Example: Binary search (pseudo code)

Input: A sorted list (array) and a target value to find.

Initialization:

- Set a pointer `left` to the beginning of the list (index 0).
- Set a pointer `right` to the end of the list (index equal to the length of the list minus one).

Search:

- While `left` is less than or equal to `right`:
 - Calculate the middle index as `mid` by adding `left` and `right` and then dividing by 2.
 - Check if the element at index `mid` in the list is equal to the target value:
 - If it is, you've found the target, so return `mid`.
 - If the element at index `mid` is less than the target:
 - Update `left` to `mid + 1` to search in the right half of the list.
 - If the element at index `mid` is greater than the target:
 - Update `right` to `mid - 1` to search in the left half of the list.

Result:

- If you've gone through the entire loop and haven't found the target, return -1 to indicate that the target is not in the list.

Example: Binary search (pseudo code)

```
In [10]: def binary_search(arr, target):
          left, right = 0, len(arr) - 1

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

              if arr[mid] == target:
                  return mid
              elif arr[mid] < target:
                  left = mid + 1
              else:
                  right = mid - 1

          return -1

ordered_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
target_element = 5
result = binary_search(ordered_list, target_element)

if result != -1:
    print(f"Element {target_element} found at index {result}.")
else:
    print(f"Element {target_element} not found in the list.")
```

Element 5 found at index 4.

Sort

Given a list of elements, sort the elements according to a given order

- **Ascending** order
- **Descending** order
- **Alphabetical** order
- **Reverse** order
- **Custom** order

NB: Sort is more complex and will be studied later on.

Many ways to sort a list

```
In [ ]: my_list = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
sorted_list = sorted(my_list)
print(sorted_list)
```

```
In [36]: numbers = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
sorted(numbers) # in place operation (changes the original variable)
print(sorted_numbers)
```

```
[1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9]
```

```
In [38]: numbers = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
numbers.sort(reverse=True) # did not change the original variable
print(numbers)
```

```
[9, 6, 5, 5, 5, 4, 3, 3, 2, 1, 1]
```

```
In [92]: words = ['apple', 'cherry', 'banana', 'date']
sorted_words = sorted(words, key=len)
print(sorted_words)
```

```
['date', 'apple', 'cherry', 'banana']
```

```
In [93]: import functools
```

```
In [98]: words = ['apple', 'cherry', 'banana', 'date']  
sorted(words, key=functools.cmp_to_key(order_by_alphabetical_order))
```

```
Out[98]: ['apple', 'banana', 'cherry', 'date']
```

```
In [95]: words = ['apple', 'cherry', 'banana', 'date']  
sorted(words, key=functools.cmp_to_key(lambda x, y: ord(x[0]) - ord(y[0])))
```

```
Out[95]: ['apple', 'banana', 'cherry', 'date']
```

```
In [97]: def order_by_alphabetical_order(a, b):  
         return ord(a[0]) - ord(b[0])
```

```
In [99]: order_by_alphabetical_order("cherry", "banana")
```

```
Out[99]: 1
```

Enumerators

Enables to turn a list into a list of index + value

In [6]:

```
a = [1, 2, 3]
b = ["A", "B", "C"]

for index, value in enumerate(L):
    print(index, value)

for i, (x, y) in enumerate(zip(a, b)):
    print(i, x, y)
```

```
0 1
1 2
2 3
0 1 A
1 2 B
2 3 C
```

```
In [7]: iterable = [1, 2, 3]
        iterator = iter(iterable)

        try:
            while True:
                item = next(iterator)
                print(item)
        except StopIteration:
            pass
```

```
1
2
3
```

Iterators

```
In [ ]: iterable = [1, 2, 3, 4, 5]
         iterator = iter(iterable)

         try:
             while True:
                 item = next(iterator)
                 print(item)
         except StopIteration:
             pass
```

Generators

Using generators

```
In [4]: def my_iterator():  
        data = [1, 2, 3]  
        for item in data:  
            yield item  
  
        for item in my_iterator():  
            print(item)
```

```
1  
2  
3
```

Fibonacci with generators

```
In [15]: def fib_generator():  
        a, b = 0, 1  
        while True:  
            yield a  
            a, b = b, a + b
```

```
In [14]: fib = fib_generator()  
         for _ in range(10):  
           print(next(fib))
```

```
0  
1  
1  
2  
3  
5  
8  
13  
21  
34
```

Linked list

*A **linked list** is a sequence of values (or objects) called nodes that are connected to each other in order to facilitate their storage and retrieval.*

- The first node is called the head, the last node is the tail, and it points to `null`.
- This structure allows for a flexible approach to manipulating objects: increasing their number, order, etc.
- Especially allows for **dynamic memory allocation**, whereas an array needs to allocate all the space before being filled.
- On the other hand, it requires linear search time (unlike arrays), and can be problematic for implementing a stack.

```
In [16]: linked_list = None

def append(data):
    global linked_list
    if linked_list is None:
        linked_list = {"data": data, "next": None}
    else:
        current = linked_list
        while current["next"]:
            current = current["next"]
        current["next"] = {"data": data, "next": None}

def traverse():
    current = linked_list
    while current:
        print(current["data"], end=" -> ")
        current = current["next"]
    print("None")

append(1)
append(2)
append(3)

traverse()
```

1 -> 2 -> 3 -> None

Filter

Return a sublist given a criteria.

```
In [11]: x = range(10)
list(x)
list(filter(lambda x : x > 5, x))
```

```
Out[11]: [6, 7, 8, 9]
```

```
In [14]: [1, 2, 3][1:]
```

```
Out[14]: [2, 3]
```

Map

Apply a function a list of values

```
In [12]: list(map(lambda x : x * x, x))
```

```
Out[12]: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
In [15]: L = [1, 2, 3]
```

```
In [16]: L.append(4)
```

```
In [17]: L
```

```
Out[17]: [1, 2, 3, 4]
```

```
In [27]: L = L + [5, 6, 7]
```

```
In [28]: L.extend([2, 4, 5])
```

```
In [29]: L
```

```
Out[29]: [1, 2, 3, 4, [5, 6, 7], 2, 4, 5, 5, 6, 7, 2, 4, 5]
```

