Data Structures

Appending operation

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

The append operation

- So far, we wrote our basic Array Structure.
- Let’s enhance it with append operations
- **def** append(self, item): this function pushes this new item in the back of the array
  - array = [1, 2, 3, 4]
  - append(29)
  - array = [1, 2, 3, 4, 29]
  - append(‘mostafa’)
  - array = [1, 2, 3, 4, 29, ‘mostafa’]
- Your turn: think in an approach to implement this functionality
  - Think about its speed

append

- To add an element to an array, we need each time
  - Create new array
  - Move old data to it
  - Add new value
  - Use the new data and remove old one
- We can make some approximation for the number of steps
  - Don’t take it seriously

```
def append(self, item):
    # 1) create a new array of bigger size
    array_data_type = ctypes.py_object * (self.size + 1)
    new_memory = array_data_type()

    # 2) copy old data
    for i in range(self.size):
        new_memory[i] = self.memory[i]

    # add the new element and increase size
    new_memory[self.size] = item
    self.size += 1

    # use the new memory and delete old one
    del self.memory
    self.memory = new_memory
```

Usage

- Now our data structure **grows dynamically in size**!
- From **efficiency** perspective, what is wrong with our code?
- Take 10 minutes to think **why?!**
  - Hint: How many steps per append?
  - Hint: How many steps in all program?

Just approximate

```
array = Array(3)

for i in range(len(array)):
    array[i] = i + 1

array.append('12')
array.append('hello')

print(array)

for i in range(10 ** 6):
    array.append(i)

print(len(array))
# takes too much time to finish!
```

Approximately, How many steps per an append?

- Assume array has length **size**, If we tried to estimate the number of steps per a call:
  - **~ 4 size + 5**
  - If size is 10^6, this is around 4 millions in total.
  - In other words, it takes **linear number** of steps to be finished
  - For simplicity, let's assume it takes size steps (drop constants)

```
def append(self, item):
    # 1) create a new array of bigger size: size+1 steps
    array_data_type = ctypes.py_object * (self.size + 1)
    new_memory = array_data_type()

    # 2) copy old data - 2 size steps
    for i in range(self.size):
        new_memory[i] = self.memory[i]

    # add the new element and increase size: 2 steps
    new_memory[self.size] = item
    self.size += 1

    # use the new memory and delete old one: size+2 steps
    del self.memory
    self.memory = new_memory
    # Total approximately: 4size + 5 steps
```

Approximately, What is Overall number of steps?

- Now this loop is iterating n = 10^6 step
  - We call append, where size is increasing in each step, 1, 2, 3, …. n
  - So steps per a call are 1 + 2 + 3 + …..n ⇒ ~ n * (n+1) / 2 = **1/2 (n^2 + n)** steps
  - For simplicity, let's keep only the largest factor here ⇒ n^2
    - Drop constant ½ and the smaller factor n
  - In other words, it takes **quadratic number** of steps for loop/body to be finished
  - So for n=10^6, we take ~ 10^12 (multiplied by some factor, e.g. 5)
- Now, you know, **mathematically**, why this code takes too much time!

```
for i in range(10 ** 6):
    array.append(i)
```

Your turn

- In the next video, we will present a simple idea that speed the code
- Take 15 minutes to guess the trick
- Also, consider today analysis as one good reason why we need to study the data structures **NOT just use them**!

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."