Data Structures

The Capacity Trick

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)

Root cause analysis (RCA)

- Root cause analysis (RCA) is the process of discovering the **root causes** of problems in order to identify appropriate solutions
  - Tip: Senior Software Engineers must be clever in that.
- We know our **append** is slow as it does linear number of steps!
  - Hence the whole program is quadratic steps!
- But why we ended with such solution?
- Because with **every** append
  - We create a new array. Move old data ⇒ Linear number of steps per an append!
- Now, we know the **design issue**. How to solve?
  - Intuitively: how can we make a single append takes a few steps (e.g. 2-5)?
    - Hence in total the loop is linear!

The Capacity Trick

- Assume the user asked for an array of length 10
- But, internally: we reserved an array of 3000 values!
  - Let's call this number the **capacity**
- Now, with every **append**, we can just add the value in O(1)
  - But after **2990 additions**, our whole **array** is filled again!
- Now we need to create a **new array** once again, and copy the data
  - New array size? 3001? NO. We will face the original problem again!
  - Let's **multiply** the **current size by 2** ⇒ reserves an array of 6000.
  - Now we have another 3000 values to use.
  - Filled? Create a new array of size 2 x 6000 = 12000 and so on
- We moved from being very slow in every step to being very fast most of the time except on **increasingly rare occasions** (exponentially with size doubling each time)

Capacity Trick: Data

- To implement this idea, we first need now 2 variables
  - Size = The actual elements size from the user
  - Capacity = The actual array size in the memory
    - Capacity >= size, typically larger. Let's start with value like 16
- Try to finish the code :)

```
def __init__(self, size):
    self.size = size                        # user size
    self._capacity = max(16, 2*size)        # actual mrmoty size

    array_data_type = ctypes.py_object * self._capacity
    self.memory = array_data_type()

    for i in range(self._capacity):
        self.memory[i] = None
```

Capacity Trick: Improved append

- Now with every append, if the current array capacity is enough
  - Just add the element (total 3 steps)
- But what if the capacity is not enough?
  - Double the array capacity
  - Move old data
  - This will be a slow step, but it happens no very rare times
- Can you code the expand function?

```
def append(self, value):
    if self.size == self._capacity:
        self.expand_capacity()
    self.memory[self.size] = value
    self.size += 1
```

Capacity Trick: Expanding capacity

- The function logic is exactly what we did before
- The function
  - doubles the capacity
  - moves the old data

```
def expand_capacity(self):
    # Double the actual array size
    self._capacity *= 2
    print(f'Expand capacity to {self._capacity}')

    # create a new array of _capacity
    array_data_type = ctypes.py_object * self._capacity
    new_memory = array_data_type()

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

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

Let’s run again

- How fast?!
- Try to analyze
  - How many times approximately the expansion statement will be printed?

```
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))
```

Output

- We will need around ~log2(1000000) expansions only for 1 million numbers
- This means, we rarely apply the expansion idea
- So for **approximately**
  - 1000000 - 16 steps ⇒ we were constant time
  - For 16 steps ⇒ we were linear time
  - If you thought from the **average** perspective, as if we do on average constant steps
  - This is called **Amortized** Analysis
    - Optionally, read after the complexity section

```
1, 2, 3, 12, hello,
Expand capacity to 32
Expand capacity to 64
Expand capacity to 128
Expand capacity to 256
Expand capacity to 512
Expand capacity to 1024
Expand capacity to 2048
Expand capacity to 4096
Expand capacity to 8192
Expand capacity to 16384
Expand capacity to 32768
Expand capacity to 65536
Expand capacity to 131072
Expand capacity to 262144
Expand capacity to 524288
Expand capacity to 1048576
1000005
```

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."