Data Structures

Capacity Trick

Mostafa S. Ibrahim
Teaching, Training and Coaching since 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 push_back is slow as it does quadratic number of steps!
- But why we ended with such solution?
- Because with every push_back
  - We create a new array. Move old data ⇒ Linear number of steps per a push back!
- Now, we know the design issue. How to solve?
  - Intuitively: how can we make a single push back is taking a few steps (e.g. 2-5)

Capacity Trick

- Assume the user asked for a vector of length 10
- But, internally: we reserved array of 3000 values!
- Now, with every push_back, we can just put the value (in 2 steps only)
  - But after 2990 steps, our whole array is filled again!
- Now, we have to again create new array and copy data
  - New array size? 3001? NO. We will be slow again
  - Let’s multiply with 2. Reserve array of 6000.
  - Now we have another 3000 values to use.
  - Filled? New array of size 2 x 6000 = 12000
- In this way, we only are **slow once** in every too many steps
  - Instead of being slow in **every** step!

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. Capacity >= size, typically larger

```
5  class Vector {
6  private:
7      int *arr { nullptr };
8      int size { 0 };          // user size
9      int capacity { };        // actual size
10
```

Capacity Trick: Constructor

- It is up to the design the initial value for the capacity
  - Here I selected capacity = size + 10
    - Some guys use 0
    - Others use size * 2
- Observe:
  - The new array is based on capacity!
- Tip
  - Many bugs happens due to wrongly mixing size with capacity in the code

```
Vector(int size) :
        size(size) {
    if (size < 0)
        size = 1;
    capacity = size + 10;
    // The actual size array will
    // be bigger than needed
    arr = new int[capacity] { };
}
```

Capacity Trick: Improved push_back

- Now with every push_back, 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

```
void push_back(int value) {
    // we can't add any more
    if (size == capacity)
        expand_capacity();
    arr[size++] = value;
}
```

Capacity Trick: Expanding capacity

- The function logic is exactly what we did before
- The new change is
  we now do it **only a few times**
  when we need for more space!
- The function
  - Doubles the capacity
  - Moves the old data
- Future reading
  - **Amortized** Analysis of push_back

```
void expand_capacity() {
    // Double the actual array size
    capacity *= 2;
    cout << "Expand capacity to "
         << capacity << "\n";
    int *arr2 = new int[capacity] { };
    for (int i = 0; i < size; ++i)
        arr2[i] = arr[i];   // copy data

    swap(arr, arr2);
    delete[] arr2;
}
```

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."