Data Structures

Appending operation

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)

Get data

- 2 nice utilities might be to get the front or the back element
- But what if we want to:
  - Add in the back?
    - We call it append operation
    - Or push_back operation
  - Similarly insert in the front?
    - We call it push_front
- Take 10 minutes:
  - Create function void **push_back**(int value)
  - It adds an element to the current array

```
int get_front() {
    return arr[0];
}

int get_back() {
    return arr[size-1];
}
```

push_back

- 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

```
void push_back(int value) {
    // Add element to the end of the vector
    // 1) create a new array of bigger size
    int *arr2 = new int[size + 1];
    // 2) copy old data
    for (int i = 0; i < size; ++i)
        arr2[i] = arr[i];
    // 3) add the new element and increase size
    arr2[size++] = value;
    // 4) change the pointers
    swap(arr, arr2);
    // 5) remove the useless data
    delete[] arr2;
```

Usage

- One clear advantage now compare to primitive arrays, our data structure can **grows normally in size**!
- From **efficiency** perspective, what is wrong with our code?
- Take 10 minutes to think **why?!**
  - Hint: How many steps per push_back?
  - Hint: How many steps in all program?
  - Just approximate

```
int n = 4;
Vector v(n);
for (int i = 0; i < n; ++i)
    v.set(i, i);

v.push_back(15);
v.push_back(17);
v.push_back(19);
v.print();
// 0 1 2 3 15 17 19

for (int i = 0; i < 1000000; ++i) {
    v.push_back(i);
} // takes tooooo much time! WHY
```

Approximately, How many steps

- Assume array has length size, If we tried to estimate the number of steps
  - ~ 5 x size + 7
  - If size is 10^6, this is around 5 millions.
  - In other words, it takes **linear number** of steps to be finished
- For simplicity, let's assume it takes size steps
  - E.g. we dropped constants

```
void push_back(int value) {
    int *arr2 = new int[size + 1];  // size+1 steps
    for (int i = 0; i < size; ++i)  // 3size+1 steps
        arr2[i] = arr[i];
    arr2[size++] = value;           // 2 steps
    swap(arr, arr2);                // 3 steps
    delete[] arr2;                  // size steps
    // Total: 5size + 7
}
```

Approximately, What is Overall number of steps?

- Now this loop is iterating n = 10^6 step
  - We call push back, 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 (int i = 0; i < 1000000; ++i) {
    v.push back(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 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."