Session 02: Vectors

People don't usually jump to vectors right after arrays but our sessions are centered to competitive programming and vectors are very important if you want to have a head start in it!

Vectors allow to implement the logic right away without worrying much about the functions employed in implementing the logic.

1. Definition

- Vector is a **template class** i.e, a CONTAINER.
- Vectors are **same as dynamic arrays** with the ability to resize itself automatically when an element is inserted or deleted.
- Operations on a vector offer the **same time complexity** as their counterparts on an array.
- Vector elements are placed in **contiguous memory locations** so that they can be **accessed and traversed using indices** as well as **ITERATORS** (new term alert)

2. Data Handling and Memory allocation

- Vectors, like arrays, store **homogeneous data.**
- Unlike static arrays, which are always of a fixed size, vectors can be resized.
- This can be done either **explicitly** (using pre-defined functions like resize()) or by adding/deleting data into them.
- In vectors, data is **inserted at the end**.
- Inserting at the end takes differential time, as **sometimes there may be a need of extending the Vector.**
- Removing the last element takes only **constant tim**e because no resizing happens.

NOTE: We as Competitive Programmers do not need to worry about how Vector allocates the memory, but we as Software developers might need to.

- In order to do this **efficiently**, the typical vector implementation grows by **doubling its allocated space** (rather than incrementing it by data elements added).
- It often allocates more space than required at any point because reallocating memory is usually an expensive operation.

3. Declaration and Initialization

- Link to code

4. Operations

- In vectors, inserting at the end takes **differential time (Amortized O(1))**, as sometimes there may be a need of extending the array.
- Removing the last element takes only **constant time O(1)** because no resizing happens.
- Inserting and erasing at the beginning or in the middle is **linear in time O(N)** if vector has the capacity or else _____ (fill in the blank!).

// Assuming that resizing of the vector will not be required

- **Insertion**
  - In the Front - O(N)
  - In the End - O(1)
  - In the middle - O(N)
- **Deletion**
  - In the Front - O(N)
  - In the End - O(1)
  - In the middle - O(N)
- **Accessing the Element**
  - Using indices - O(1)
  - Using iterators[1] - O(1)
- **Traversal**
  - Iterating through the data structure - O(N)
- **Sorting and Searching**
  - Using Predefined Libraries (STL – Standard Template Library)
  - Building functions (User defined functions)
  - **Conditional sorting - possible using Compare function.**

[1] An iterator is a variable that points to an element in a data structure.
You can consider it as a pointer variable with a fancy name specific to its application i.e, ITERATION.

Operations on Vectors

- LINK TO CODE (VectorsOperation.cpp)

More on Iterators and Sizing of Vectors

- LINK TO CODE (IteratorsAndSizing.cpp)

Passing Vectors as Arguments

- LINK TO CODE (ValueReference.cpp)

5. Practice Questions

1. Reverse String

Write a function that reverses a string. The input string is given as an array of characters `char[]`.

Do not allocate extra space for another array, you must do this by **modifying the input array in-place** with O(1) extra memory.

You may assume all the characters consist of printable ascii characters.

Example 1:

```
Input: ["h","e","l","l","o"]
Output: ["o","l","l","e","h"]
```

SOLUTION: LINK TO CODE

2. Move Zeros

Given an array `nums`, write a function to move all `0`'s to the end of it while maintaining the relative order of the non-zero elements.

Example:

```
Input: [0,1,0,3,12]
Output: [1,3,12,0,0]
```

Note:

1. You must do this **in-place** without making a copy of the array.
2. Minimize the total number of operations.

SOLUTION: LINK TO CODE

3. Sort Colors

Given an array with *n* objects colored red, white or blue, sort them **in-place** so that objects of the same color are adjacent, with the colors in the order red, white and blue.

Here, we will use the integers 0, 1, and 2 to represent the color red, white, and blue respectively.

Note: You are not suppose to use the library's sort function for this problem.

Example:

```
Input: [2,0,2,1,1,0]
Output: [0,0,1,1,2,2]
```

Follow up:

- A rather straight forward solution is a two-pass algorithm using counting sort.
  First, iterate the array counting number of 0's, 1's, and 2's, then overwrite array with total number of 0's, then 1's and followed by 2's.
- Could you come up with a one-pass algorithm using only constant space?

SOLUTION: LINK TO CODE

END NOTE

Hope this proves to be useful to you. If you have any doubts regarding the content in this doc or any other related (or unrelated) topic please contact me I am as excited for this as you are. Any and every feedback is appreciated.

I will try to continue preparing documents suited to your need so that you can have a look on your own whenever you feel like it. Remember that programming is actually a self-taught thing. There exists no one who can teach you programming, just the ones who do it with you and you all learn in the process.

Good day!

Rinku Monani
monanira@gmail.com