Preface

Consider this document as your cheat-sheet and the upcoming session to be a trial version (what I mean is, PLEASE COOPERATE). We are going to start our Data Structure sessions with the most basic one i.e, an Array.

Although we will have a quick recap of the concepts discussed below, it is recommended to read the document once before the session as I really want to keep the live session *'all about implementing'.*

Programming Language

We are going to use CPP as it is considered to be one of the fastest programming languages when it comes to Competitive Programming and also because **CPP is tough.** There are too many concepts you need to know about the **I am sure** using CPP during these sessions will help us all clear and build our basics more!

Content Pattern

It is possible that we might lose sight (and hope!) of important topics as we start to struggle with the basics of DS. Therefore, here is a list of topics (along with the reasons why you must learn it) that we will discuss foremost for every single Data Structure we pick and then dwell further.

1. Definition
   May sound boring but you should be able to give a one line explanation when a recruiter (or possibly a junior) asks you – what is an array?

2. What type of data does it handle and how?

3. Different ways of declaring and initializing that DS.

4. What operations does it perform?
   Example: With an array, the most basic operations we perform are insertion, deletion, traversal (iterating through the data structure), searching, sorting.

5. Applications

6. Programming questions based on it.

Please note that we are not going to print patterns here. We will discuss some what about the syntax of CPP which was introduced in the latest version because it is new (and I like it!). Therefore, please brush up your programming teeth before you start with DS!

Also, TIME and SPACE Complexity is not an individual topic. It is attached to everything that comes under the paradigm that is computer engineering. So, each operation we discuss, we talk about its complexities **without a doubt!**

Session 1: Arrays.

…you have heard about them, you know them, you have probably used them many times, like MANY times, nevertheless…

1. Definition

- It is a physical data structure.
- An array is a **collection** of **similar type** of items stored in contiguous memory locations.
- Multiple items of similar type can be referenced via. only one reference variable using an array.
- i.e, you need only a single variable, say 'table' to reference an array containing multiplication table of 2.
  table -> {2, 4, 6, 8, 10}

Advantage

- Accessing any element is efficient in an array since data is stored in contiguous memory locations. Therefore, you can access any element via its index.

  i.e, if you need to know the product of 2*4, you just need to access the 4^{th} element of the array 'table'.

  table[3] = 8 …index starts from 0.

Drawback

- We will discuss drawbacks while discussing the next DS. So as to understand why we need an alternate data structure.

2. Data handling

- An array can only store homogenous data depending upon the data type of your array's reference variable.
  i.e, if your reference variable 'table' is of type 'int', your array will only store integer type data.

- Implicit type conversion (conversion of one data type to another data type by the complier) is possible.

  For example:
  int characters [ ] = { 'A', 'B', 'C' } // gets converted to {65, 66, 67}
  Since the type of the array reference variable 'characters' is 'int'.

3. Declaration and Initialization

- Link to Code…

4. Operations

- Inserting an element
  - In the front // O(N)
  - In the end // O(1), O(N)
  - In the middle // O(N)

- Deleting an element
  - In the front // O(N)
  - In the end // O(1)
  - In the middle //O(N)

- Accessing an Element
  - Via. Index number // O(1)

- Traversal
  - Iterating through the data structure // O(N)

- Searching and Sorting
  - using pre-defined libraries.
  - By building functions.

5. Application and Case Study

- An array is generally used when you have similar type of data. For example, if all the data elements are integers or if you want to store a list of names (strings).

- For any problem that you want to implement, you need to first look for the **operations that your problem will require.**

- For example, if I ask you to maintain a list of names of all the people that are present inside a temple near my house do you think you can use an array to maintain this list?

- Let's go over this. Suppose you choose an array of type string to maintain this list of names. Now since an array needs to be declared, you use your logic, calculate the capacity of the hall and declare an array 'PEOPLE' with size 500.

- The next day you managed to insert around 300 names into the array and the 19^{th} person leaves the hall. So now you have to remove her name from the list but you cannot leave the 19^{th} index empty! So eventually you reach the conclusion that you will have to **left shift** the name at 20^{th} index to 19^{th} index and then you realize now you will have to shift all the elements on the right of the 19^{th} index to left (20^{th} to 19^{th}, 21^{st} to 20^{th}…..) and then the big realization dawns that you will have to do this every time someone leaves the room! (Lots of shifting).

- Let's take it further. Suppose the next day I ask you to add a feature to your little program and maintain this list of names in Alphabetical order.

- So now if and when a guy named "Aaron" visits the hall you will have to **right shift** every element in you array PEOPLE: ["Bob", "Charlie", "David", "Elen" …]. Again, a lot of shifting right!

- So, technically, choosing a data structure like an Array can prove to be efficient when you do not have to perform a lot of modification operations (insertion, deletion) and a have a load of traversing or searching to do. Got it?

NOTE: We will look at 'Vector' later which is a popular data structure. It provides us with many useful methods such as size( ) that array does not. It is included in <vector>.

6. Practice Questions

1. Maximum consecutive ones

Given a binary array, find the maximum number of consecutive 1s in this array.

Example 1:

```
Input: [1,1,0,1,1,1]
Output: 3
Explanation: The first two digits or the last three digits are consecutive 1s.
    The maximum number of consecutive 1s is 3.
```

Note:

- The input array will only contain `0` and `1`.
- The length of input array is a positive integer and will not exceed 10,000

Solution: Click here!

2. Find Numbers with Even Number of Digits

Given an array `nums` of integers, return how many of them contain an **even number** of digits.

Example 1:

```
Input: nums = [12,345,2,6,7896]
Output: 2
Explanation:
12 contains 2 digits (even number of digits).
345 contains 3 digits (odd number of digits).
2 contains 1 digit (odd number of digits).
6 contains 1 digit (odd number of digits).
7896 contains 4 digits (even number of digits).
Therefore only 12 and 7896 contain an even number of digits.
```

Solution: Click here!

3. Check If N and Its Double Exist

Given an array `arr` of integers, check if there exists two integers `N` and `M` such that `N` is the double of `M` (i.e. `N = 2 * M`).

More formally check if there exists two indices `i` and `j` such that :

- `i != j`
- `0 <= i, j < arr.length`
- `arr[i] == 2 * arr[j]`

Example 1:

```
Input: arr = [10,2,5,3]
Output: true
Explanation: N = 10 is the double of M = 5,that is, 10 = 2 * 5.
```

Solution: **Click Here!**

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