Week 1: Competitive Coding QSTP

STL Data Structures

Summer 2021

1 Stacks

1.1 Some Jargon

A stack is a collection of elements (or technically, *a container of objects*) which stores elements according to the **LIFO (Last in First out)** principle. Hence, an element inserted most recently will be removed first. Insertion and deletion can be done from only one end of the stack popularly called as top of the stack. Inserting an element on the top of the stack is referred to as the push operation and removal of the top most element is referred to as the pop operation.

1.2 Array Implementation of a Stack

Please refer here for the Array Implementation of the Stack. We shall use STL for all practical purposes :)

1.3 STL Implementation of Stack

Please refer here for STL functions for stack. Make yourself comfortable with the syntactic sugar of push, pop, top, size and empty functions.

Note that in C++ STL pop function is of void return type.

1.4 The Famous Balanced Parenthesis Problem

Given a string with only 2 symbols, '(' and ')', you have to check whether the string is balanced. A balanced string means if used in a mathematical expression, it is syntactically correct.
For example, (()) is a balanced string, whereas ((()) is not balanced. Also,)()(and)(() are also not balanced due to obvious reasons.

Note that just counting the opening and closing brackets will not work because)()(is not a balanced string.

1.5 An extension to the Balanced Parenthesis Problem

An extension to the Balanced Parenthesis Problem is when we have more than 1 type of parenthesis. This can be found here.

TASK: Find out the error in this code for the Extension Problem:
The error is somewhere in between syntax and a logical error.

```
stack <char> s t ; s
t . push ( s [ 0 ] ) ;
i nt f l a g= 1 ;
f o r ( i nt i =1; i <s . l ength ( ) ; i ++)          {
     i f ( s [ i ]== '( '          ||        {          ||        {
               s t . push ( s [ i ] ) ;
     }

     e l s e i f ( s [ i ]== ' ) ' )    {
          i f (   s t . top ()== ' ( ' , s    {
               t . pop ( ) ;
            [illegible]
              f l a g = 0 ;
               break ;
          }
     }
     e l s e   i f ( s [ i ]== ' [illegible]
          i f (   s t . top ()== ' [illegible] ') [illegible]
               t . pop ( ) ;
            [illegible]
              f l a g = 0 ;
               break ;
          }
     }
     e l s e i f ( s [ i ]== ' ] ' )   {
          i f ( s t . top ()== ' [ ' ) s    {
               t . pop ( ) ;
            [illegible]
              f l a g = 0 ;
               break ;
          }
     }
}
i f ( f l a g  && s t . empty()== true ) cout
     <<  ”YES”  <<  endl ;
e l s e  cout  <<  ”NO”  <<   endl ;
     }
```

1.6 Creating a Customized Stack

The problem is basically to create a customized stack which can perform the getMin operation in $O(1)$ time. So, perform the following queries -

- 1 x meaning push x in the stack.
- 2 meaning pop the topmost element from the stack.
- 3 basically implementing the getMin operation which returns the minimum element present in the stack uptil now.

You can try and code the problem yourself for practice here.

We shall discuss a $O(n)$ space solution here. Another solution which uses $O(1)$ space can be found here. *Important from interviews point of view.*

1.7 Other Problems on Stacks

- Linked list implementation of a stack can be found here.
- Variation of Balanced Parenthesis Problem Link.
- Nearest Smaller Element. Problem Link.
- Next Greater Element. Problem Link.
- Number of Next greater elements to the right. (Processing Q queries). Problem Link.
- Prefix to Posstfix conversion and similar. (Important from Interview point of view). Problem Link.
- Maximum area under Histogram Problem Link.
- Almost regular bracket sequence. Problem Link.

2 Queues

2.1 Some Jargon

A stack is a collection of elements (or technically, *a container of objects*) which stores elements according to the **FIFO (First in First out)** principle. Hence, an element inserted least recently will be removed first. Elements are always added to the back and removed from the front. Hence, insertion takes place at the back and deletion is done from the front. Inserting an element in the queue is referred to as the push or enqueue operation and removal of an element from the queue is referred to as the pop or dequeue operation.

2.2 Array Implementation of a Queue

Please refer here for the Array Implementation of the Queue. We shall use STL for all practical purposes :)

2.3 STL Implementation of Queue

Please refer here for STL functions for queue. Make yourself comfortable with the syntactic sugar of push, pop, front, size and empty functions.

2.4 Implementing a Stack using Queue

We shall implement a Stack using 2 Queue lets say $q1$ and $q2$. The two stack operations are defined as follows -

Push Operation: Simply push the incoming element to $q1$.

Pop Operation: For Pop operation, we will dequeue everything from $q1$ except the last element (which is basically the element to be popped) and enqueue it in $q2$. Now dequeue the last element of $q1$. (The pop operation has taken place). Simply swap the names of $q1$ and $q2$. The top of the stack is the back of the queue $q1$

Implementation of a Stack using a single queue can be found here. Important from Interview point of view.

Implementation of a Queue using 2 Stacks can be found here.

2.5 Circular Tour

There is a circular track which has n petrol pumps. Each petrol pump has certain units of petrol which is given to us. Also, the distance between the successive petrol pump is given to us. The task is to find the first petrol pump from which if we start, the entire circular tour can be completed. A formal formal statement of the problem can be found here. There are a lot of variants of this problem. Another one can be found here.

2.6 About Deque

A deque is a generalized form of queue which allows insertion and deletion at both ends. The STL implementation of a deque can be found here. A very famous problem which involves an elegant use of a deque is the Sliding Window Maximum Problem. Given an array and an integer k, find the maximum element in all subarrays of size k.

2.7 Other Problems

- Sliding Window Maximum Problem
- Variation of Sliding Window Maximum
- Another variant of Circular Tour problem

Vectors

Some Jargon

Vectors are same as dynamic arrays with the ability to resize itself automatically when an element is inserted or deleted, with their storage being handled automatically by the container. Vector elements are placed in contiguous storage so that they can be accessed and traversed using iterators. 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 array. Removing the last element takes only constant time because no resizing happens. Inserting and erasing at the beginning or in the middle is linear in time

STL Implementation of Vector

Syntax: vector < DataType > vectorName;

Adding elements in vector : vectorName.push_back(element);

Size of vector: vectorName.size()

Last element: vectorName.back()

First element : vectorName.front()

Remove last element : vectorName.pop_back()

Erase whole vector : vectorName.erase()

Pair

Some Jargon

The pair container is a simple container defined in <utility> header consisting of two data elements or objects.

STL Implementation of Pair

Syntax: pair<data type,data type> pairName

First Element : pairName.first;

Second Element: pairName.first;

Maps

Some Jargon

Maps are associative containers that store elements in a mapped fashion. Each element has a key value and a mapped value. No two mapped values can have same key values.

STL Implementation of Map

Syntax: map < keyDataType, valueDataType > mapName;

unordered_map < keyDataType, valueDataType > mapName

Giving value to a key : mapName[key]=value;

Size of map: mapName.size()

Iterator end:: mapName.end()

Iterator start : mapName.begin()

Erase a key: mapName.erase(keyName)

Get iterator to a key if present : mapName.find(keyName);

Problem Section

1. Winner
2. Zero Quantity Maximisation
3. Power Products

Sets

Some Jargon

2 Types of Sets:

Sets :- It stores the values in a sorted manner. The key point about set is that no duplicate values are stored.

Multi Sets :- It stores the values in a sorted manner, but unlike set, duplicate values can be stored.

STL Implementation of Sets

Syntax: set < DataType > setName;

multiset < DataType > setName

Inserting a value : setName.insert(value);

Size of set: setName.size()

Iterator end: setName.end()

Iterator start : setName.begin()

Erase a value: setName.erase(valueName)

Get iterator to a value if present : setName.find(valueName);

Problem Section

1. Second Order Statistics
2. Distinct Characters Queries
3. Glass Carving

With the help of Sarthak Ajmera and Manav Jain