Module-5: Data Structures

CRUx Bootstrap

5.A: Stacks

A **stack** is a linear data structure in which the insertion of a new element and removal of an existing element takes place at the same end, which is represented as the top of the stack.

Stacks follow the **LIFO** principle (Last In First Out), i.e., the element that was inserted last will be removed first.

There are mainly 3 types of operations in Stacks

1. push(): Adding an element to the top of stack in O(1)
2. pop(): Removes and returns the top element from a stack in O(1)
3. peek(): Returns the top element of the stack without removing it in O(1) time.

Reading Material:

1. CodeForces Blog
2. GFG

Questions:

5.1: Balanced Parenthesis

5.2: SPOJ: Histogram

5.3: LC: Trapping Rain Water

5.4: MAX special Product

5.5: Bus of Characters

5.B: Queues

A **queue** is a linear data structure that follows a **First-In, First-Out (FIFO)** order.

This means that the first element added to the queue is the first element to be removed.

There are 3 main functions of Queue:

1. Enqueue() : Adding element to end of queue in O(1)
2. Dequeue(): Removes and returns the front element of queue in O(1)
3. Peek(): Returns the front element of the queue without removing it in O(1) time.

Reading Material:

1. GFG

Questions:

5.6: Character Blocking (Read set also)

5.7: Social Network (Read set also)

5.8: Table Tennis

5.C: Priority Queues

A **priority queue** is a type of queue that arranges elements based on their priority values. Elements with higher priority values are retrieved before elements with lower priority values.

The main difference between a priority queue and a normal queue is that the elements in a priority queue do not follow FIFO.

The main functions of priority queue are:

1. empty(): checks whether the queue is empty, returns a boolean value in O(1)
2. size(): returns the size of the priority queue in O(1)
3. top(): returns the first element of the priority queue in O(1)
4. push(): inserts an element into the priority queue in O(log n)
5. pop(): deletes the first element of the priority queue in O(log n)

Reading Material:

1. GFG
2. Programiz

Questions:

5.9: Heap Operations

5.10: Minimise the error

5.11: Kolya and Movie Theatre

5.D: Maps

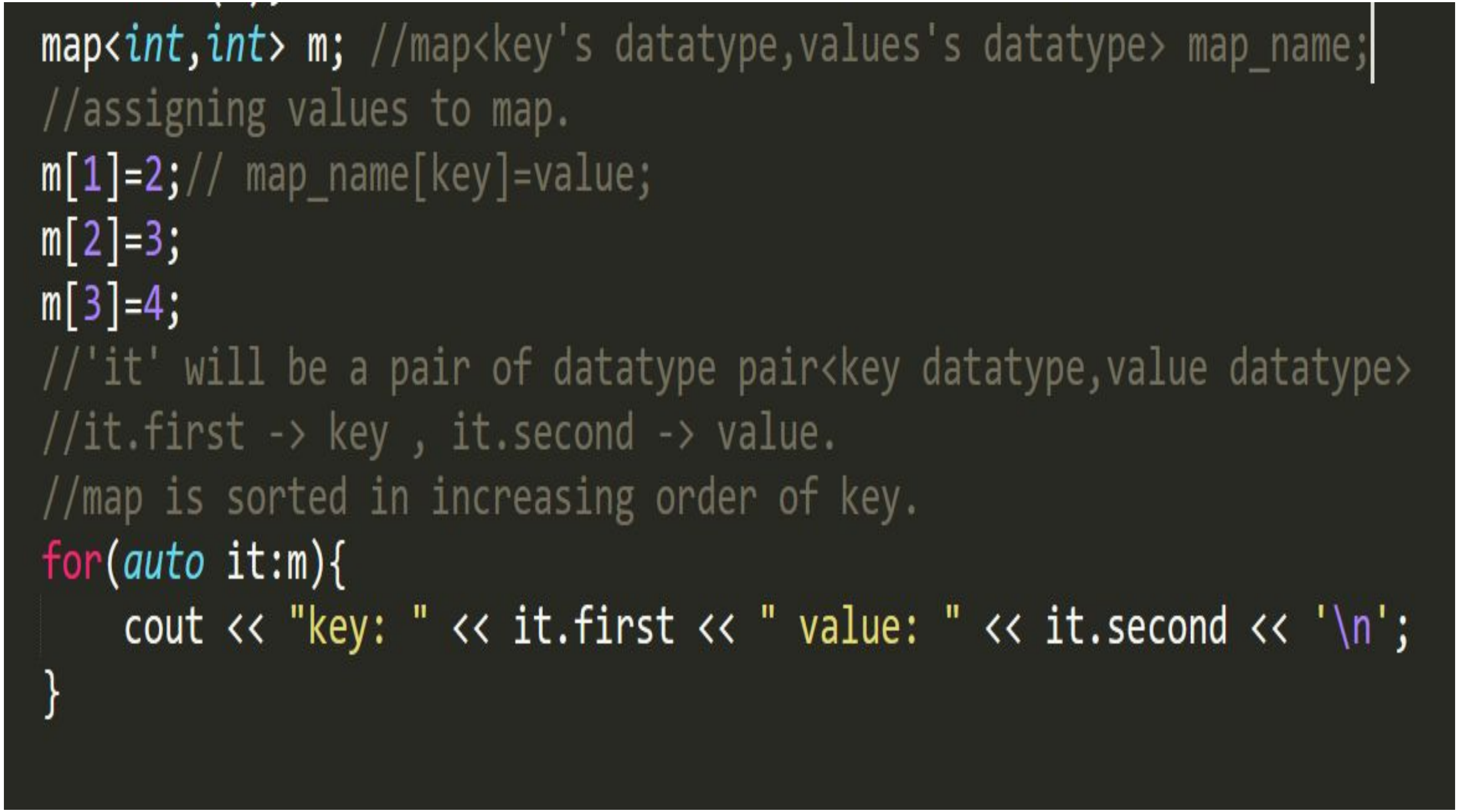
Maps are associative containers that store elements in a mapped fashion. Each element has a key value and a mapped value. No two mapped values can have the same key values.

Maps can be used to store data in ascending order of keys.

The main functions of Map are:

1. m[key]=value; you can use this statement to assign a key some value.O(log(n)).
2. size() – Returns the number of elements in the map.(O(1)).
3. clear() – Removes all the elements from the map.(O(map_size)).

You can iterate through maps like this:-



```
map<int,int> m; //map<key's datatype,values's datatype> map_name;
//assigning values to map.
m[1]=2;// map_name[key]=value;
m[2]=3;
m[3]=4;
//'it' will be a pair of datatype pair<key datatype,value datatype>
//it.first -> key , it.second -> value.
//map is sorted in increasing order of key.
for(auto it:m){
    cout << "key: " << it.first << " value: " << it.second << '\n';
}
```

Reading Material:

1. GFG
2. Programiz

Questions:

5.13: Yarik and Musical Notes

5.14: Train and Queries

5.15: Weird Sum

5.E: Sets and Multisets

Set is a data structure that stores unique elements in sorted order.

The only difference between multisets and sets is that you can store duplicate elements in a multiset.

The main functions of sets and multisets are:

1. s.erase(iterator position)-Removes the element at the position pointed by the iterator. O(log(N)).
2. s.find(element)-Returns an iterator pointing to element, if element is not found in the set, it returns s.end().O(log(N)).
3. s.insert(element)- Inserts element to the set.O(log(N)).

lower_bound and upper_bound in sets and multisets:

- s.lower_bound(element)- Returns an iterator pointing to the first element in the set which is greater than or equal to element,if there is no element in the set which is greater than or equal to element it returns s.end().O(log(N)).

- s.upper_bound(element)-Returns an iterator pointing to the first element in the set which is greater than element,if there is no element in the set which is greater than element it returns s.end(). O(log(N)).

Reading Material:

1. GFG (Set)
2. Programiz (Set)
3. GFG (Multiset)
4. Programiz (Multiset)
5. GFG (Set lower_bound)
6. GFG (Set upper_bound)

Note:-
All the functions mentioned in 5.E are applicable for both sets and multisets. For erase, multiset removes only one occurrence of element from the multiset, for find,lower_bound and upper_bound, multiset gives an iterator pointing to the first occurrence of element in the multiset.

Questions:

5.16: Merge Equals
5.17: Distinct Characters Queries
5.18: Rooks Defenders

5.F: Lower_bound and upper_bound for vectors and arrays

Similar to sets and multisets, you can use upper_bound and lower_bound on vectors and arrays, given that the array or vector is sorted.

- lower_bound(iterator first,iterator last, val) - returns an iterator pointing to the first element in the range[first,last) in the vector/array greater than or equal to val. O(log(N)).

- upper_bound(iterator first,iterator last, val) - returns an iterator pointing to the first element in the range[first,last) in the vector/array greater than val. O(log(N)).

Reading Material:

1. GFG
2. Codeforces Blog

Questions:

5.19: Number of Pairs

5.20: Yet Another Problem About Pairs Satisfying an Inequality