

MAP-ADT DOCUMENTATION

(Question-1)

**INTRODUCTION:-**

**Maps** are basically containers that store elements/structures in a mapped fashion. Each element has a key value and a mapped value. It allows fast access to the value using its unique key!

**Available Functions:-**

* constructor
* copy constructor
* operator=
* destructor
* at
* operator[]
* begin
* end
* empty
* size
* clear
* insert
* erase
* count
* find

**Functions Details:-**

**Constructor():-**

Creates a dummy head, points all the pointers (left, right, parent) to itself.

**Copy-Constructor():-**

Uses Pre-Order Recursion to copy the nodes one by one, has the case complexity of n log n.

**Operator = ():-**

Works similarly to copy constructor, but it first empties the map if its not empty.

**Destructor:-**

Uses clear function to delete all nodes, then deleted the dummy head node in the end as well.

Complexity is n

**at():-**

Returns the value found at the given key, the key is provided as a parameter, if it does not exist, the program throws an exception. The at() function calls find function to search the value first.

**Parameters:** Takes key as a parameter

**Return Type:** Returns the value found at the key, if not found, throws an exception

Complexity is log n

**Begin():-**

Returns an iterator to the smallest value in the map.

**Parameters:** none

**Return Type:** Returns the iterator to the smallest value in the map

Complexity is constant 1

**End():-**

Returns an iterator pass the last node, i.e. the dummy head node.

**Parameters:** none

**Return Type:** Returns the iterator to the Dummy head.

Complexity is constant 1

**rbegin():-**

Returns an iterator to the greatest node of the map.

**rend():-**

Returns an iterator to the Dummy Head node of the map.

**Empty():-**

Returns a bool value of either the map is empty or not!

**Parameters:** none

**Return Type:** bool either true of false.

Complexity is constant 1

**Size():-**

Returns a size of the map that is an integer value that is maintained in the program as variable n.

**Parameters:** none

**Return Type:** Returns the int size of the map

Complexity is constant 1

**Clear():-**

Visits all the nodes and deletes them one by one starting from the leaf nodes.

**Parameters:** none

**Return Type:** none

Complexity is n

**Insert():-**

Takes a pair as parameter, finds its correct place using the key value, inserts the value maintaining the properties of binary search tree.

**Parameters:** pair(key type, value type)

**Return Type:** iterator to the inserted node or the node if it already exists.

Complexity is log n

**Erase():-**

Takes an iterator as a parameter, Deletes it in such a way that the properties of binary search tree is not harmed,

There are three cases,

* The node is leaf node, in this case, we simply delete the node.
* The node has one child, in this case, we attach the child to the parent of the node to be deleted and then delete the node.
* The node has two Childs, in this case, we find the immediately smaller or larger value than the node to be deleted, then we will replace them and delete the node to be deleted, in this case, we make sure to link them accordingly.

**Parameters:** takes iterator of the node to delete

**Return Type:** iterator to the node followed by deleted node

Complexity is constant 1.

**find():-**

Takes key of the node to be searched, searches the node and then returns an iterator of that node!

**Parameters:** takes key value to find

**Return Type:** returns the iterator of the found key, if not found, returns node to the head node.

Complexity is log n.

**count():-**

Returns the number of elements with the provided key, in this case it will be either 0 or 1 since we don’t allow duplicates. It calls find function to see if the key exists or not!

**Parameters:** takes key value

**Return Type:** returns either 1 (if such key exists) and returns 0 if it doesn’t exist.

Complexity is log n.

**Iterators:-**

The ADT contains all the iterators;

* Iterators
* reverse iterators
* constant iterators
* reverse constant iterators

**Iterator overloads:-**

**++operator:-**

When an iterator is ++, it moves to the immediately next larger node.

**--operator:-**

When an iterator is --, it moves to the immediately next smaller node.

**Operator \*():-**

When an iterator is dereferenced, it returns the reference of the pair.

**Operator ->():-**

When an arrow is used with an iterator, it returns the value part of the pair.