Generic Programming
Splay Tree as a Container

Language : C++

-Shubham Kumar , Shubham Gupta
PES1201800268 , PES1201801295

Splay Tree

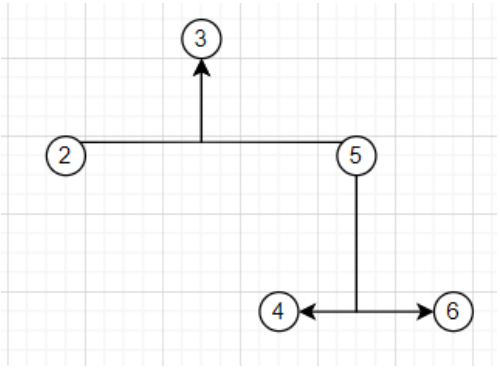
Splay Trees are self balancing trees much like red black trees , the main idea behind the splay trees is to bring the recently accessed node to the root of the tree , such a tree incorporates the idea of locality of reference where a large amount of searches occur to a small number of nodes, this helps in bringing nodes more recently accessed closer to the root node and enables the tree to have closer to constant time complexity in searching for these nodes. To implement this, Splay trees can be changed simply by search operations which occur on the tree , if the node is found it is splayed to the root of the tree , else the previous node is splayed to the top. The rotations which take place while determining the node which needs to be splayed to the top are of different kinds like:

Zag Rotation

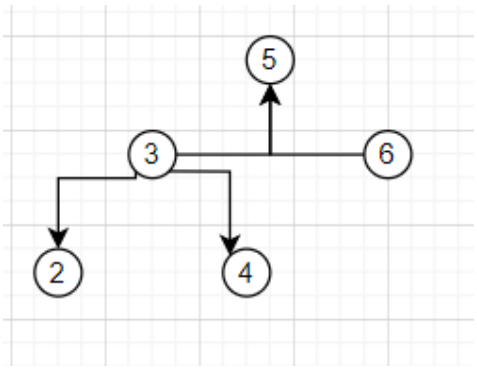
The **Zag Rotation** in the splay tree is similar to the single left rotation in AVL Tree rotations. In zag rotation, every node moves one position to the left from its current position.

Condition: Node does not have a grandparent and it is the right child of the parent

Before Rotation



After Rotation

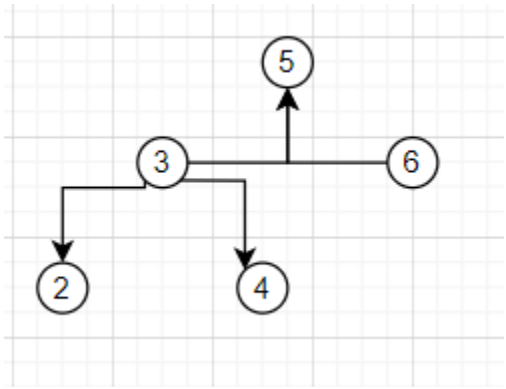


Zig Rotation

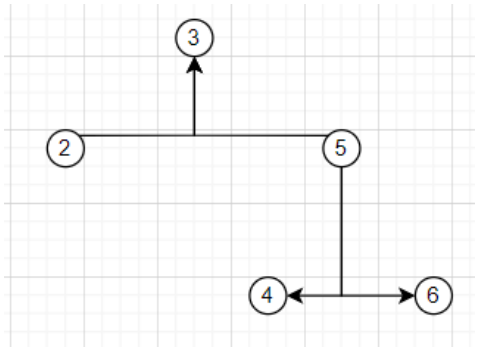
In zig rotation, every node moves one position to the right from its current position.

Condition :If Node does not have a grandparent and is the left child of the parent.

Before Rotation



After Rotation



Zig-Zig Rotation

The **Zig-Zig Rotation** in a splay tree is a double zig rotation. In zig-zig rotation, every node moves two positions to the right from its current position.

Condition:If the node is the right of the parent and the parent is also right of its parent

Zag-Zag rotation

In zag-zag rotation, every node moves two positions to the left from its current position.

Condition: If the node is right of the parent and the parent is right of its parent

Zig-Zag Rotation

In zig-zag rotation, every node moves one position to the right followed by one position to the left from its current position.

Condition: If the node is left of a parent, but the parent is right of its parent

Zag-Zig Rotation

In zag-zig rotation, every node moves one position to the left followed by one position to the right from its current position.

Condition: If the node is right of a parent, but the parent is left of its parent

Map

Maps are containers that store elements formed by a combination of a *key value* and a *mapped value*, following a specific order(generally ascending).

In a map, the *key values* are generally used to sort and uniquely identify the elements, while the *mapped values* store the content associated to this *key*. The types of *key* and *mapped value* may differ, and are grouped together in member type value_type, which is a pair type combining both.

Internally, the elements in a map are always sorted by its *key* following which the internal order of the elements are decided by the internal comparison object

Our implementation seeks to create a container similar to a map for Splay Trees. Keeping in mind the main idea of a key value pair used to map the nodes of the splay tree, we implemented the container using the following functions ,each followed by a brief description of their function in the program.

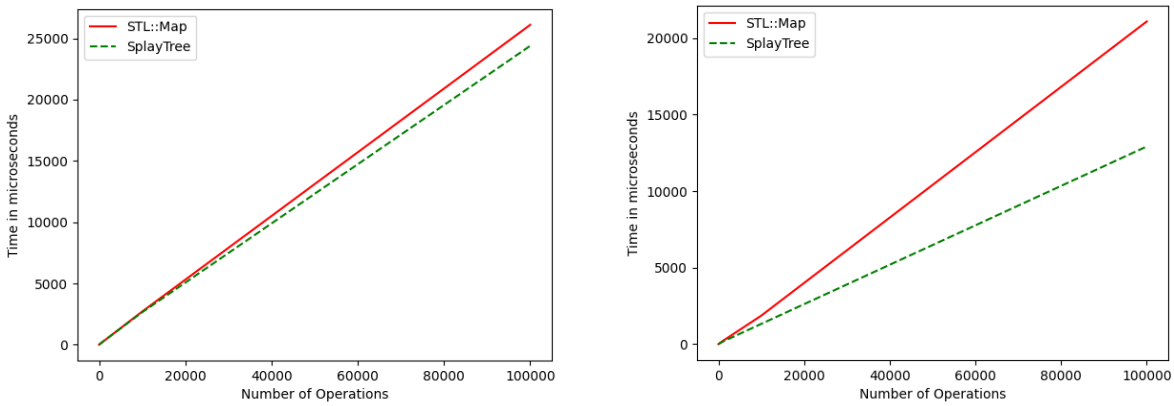
Member types and functions

Member Type	Definition
key_type	template type of the key in the pair
mapped_type	template type of the value in the pair
root_	stores the root node of the splay tree.
iterator	a bidirectional iterator to mapped_type.
const_iterator	a bidirectional iterator to constant mapped_type.
reverse_iterator	a bidirectional iterator to mapped_type.
const_reverse_iterator	a bidirectional iterator to constant mapped_type.

Member Function	Definition
(constructor)	Splay tree constructor.

(destructor)	Splay tree destructor.
operator=	Copy container contents.
Iterators	
begin	Return the iterator to the first in-order element.
end	Return iterator to beyond-the-last element.
cbegin	Return a constant iterator to the first in-order element.
cend	Return a constant iterator to beyond-the-last element.
rbegin	Return iterator to last in-order element.
rend	Return iterator to beyond-the-first element.
crbegin	Return a constant iterator to the last in-order element.
Element Access	
operator[]	Access element using key. If not present, (default) value is inserted.
at	Access element using key. If not preseed, out_of_range exception is thrown.
Modifiers	
insert	Insert a new key-value pair into the tree. Incase key already exists, update value and return an iterator to the inserted element.
erase	Delete a key-value pair based on the key. If the key does not exist, do nothing.
clear	Free the entire splay tree.
Operations	
find	Search for a key-value pair based on a given key. Return an terator to pair if found else return end()

Comparison of performance with STL::Map



a **b**

Figure 1. *Comparison between STL::Map and SplayTree for time vs operations(100000) using maximum of* ***(a).*** *100 unique keys,* ***(b).*** *10 unique keys.*

We compare the performance of the splay-tree container to the STL::Map container provided by C++. We plot time(in microseconds) vs number of operations which could be insertions/deletions/updates in any random order. We observe that the performance of splay-trees is inversely proportional to the number of unique keys for a given number of operations. This is demonstrated in Figure 1a. where both STL::Map and splay-tree linearly increase with the number of operations. While in Figure 1b. where only 10 unique keys are utilized achieves a significantly lower run time.

We conclude our container would be useful in situations where limited data is constantly updated/inserted/deleted as it would in a small caching system.

Future Work:

- Overload output operator to directly print the tree rather than call member function
- Code optimization
- Test more STL algorithms
- Compare performance with other types of key-value pair containers

References:

[1] https://www.cplusplus.com/reference/map/map/

[2] https://www.cs.odu.edu/~zeil/cs361/sum18/Public/treetraversal/index.html

[3] https://stackoverflow.com/questions/7758580/writing-your-own-stl-container

[4] http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2011/n3242.pdf

[5] https://stackoverflow.com/questions/9725675/is-there-a-standard-format-for-command-line-shell-help-text

[6] https://stackabuse.com/command-line-arguments-in-python/