Test cases

1.Van Embde Boas Tree:

a. Insertion:

Test Case 1: Inserting Elements

Attempt 1:

Code:

```
// Test Case 1 - Inserting a element
TEST(Insert, InsertValue)
{
    VEBTree veb(SIZE);

    veb.insert(3);
    veb.insert(10);

    EXPECT_EQ(veb.getmin(), 3);
    EXPECT_EQ(veb.getmax(), 10);
    EXPECT_EQ(veb.search(3), true);
    EXPECT_EQ(veb.search(10), true);
}
```

Expectation:

The elements are added to the Van Emde Boas Tree and the test case is passed

Result:

Result Matches the Expectation

```
[ RUN      ] Insert.InsertValue
[       OK ] Insert.InsertValue (0 ms)
```

Test Case 2: Inserting multiple/ Large amount of Values

Attempt 1:

Code:

```
void test_insertion_and_min_max() {
    try {
        VEBTree veb(4); // Adjust the universe size accordingly

        veb.insert(10);
        veb.insert(3);
        veb.insert(11);

        assert(veb.getmin() == 3);
        assert(veb.getmax() == 10);

        // Inserting duplicate elements
        //veb.insert(10);
        assert(veb.getmin() == 3);
        assert(veb.getmax() == 10);

        std::cout << "Insertion and Min/Max test passed for 4 elements.\n";

    } catch (...) {
        std::cerr << "Insertion and Min/Max test failed for 4 elements.\n";

    }
```

Expectation:

```
compilation terminated.


=== Code Exited With Errors ===
```

The large number of values are added to the VEB Tree and the test case is passed

Result:

The Result does not match with the Expectations since the test gets terminated due to segmentation fault

Attempt 2: After Optimizing the code to handle large amount of clusters

Code:

```
// Test Case 2 - Inserting many values
TEST(Insert, InsertingManyValues)
{
    VEBTree veb(SIZE);

    for (int i = 0; i < 10000; i++)
    {
        veb.insert(i);
    }

    EXPECT_EQ(veb.getmin(), 0);
    EXPECT_EQ(veb.getmax(), 9999);
}
```

Expectation:

The large number of values are added to the VEB Tree and the test case is passed

Result:

The result matches the expectations

```
[ RUN      ] Insert.InsertingManyValues
[       OK ] Insert.InsertingManyValues (31 ms)
```

Test Case 3: Inserting Duplicate elements

Attempt 1:

Code:

```
// Test Case 3 - Inserting duplicates
TEST(Insert, InsertDuplicates)
{
    VEBTree veb(SIZE);

    for (int i = 0; i < 10000; i++)
    {
        veb.insert(1);
    }

    EXPECT_EQ(veb.getmin(), 1);
    EXPECT_EQ(veb.getmax(), 1);
}
```

Expectations:

The Duplicate Values get replaced due to hashing features of the tree and the test case is passed

Result:

The result matches the expectations

```
[ RUN      ] Insert.InsertDuplicates
[       OK ] Insert.InsertDuplicates (18 ms)
```

b. Deletion:

Test case 4: Deleting an element

Attempt 1:

Code:

```
// Test Case 4 - Delete a value
TEST(Delete, DeleteValue)
{
    VEBTree veb(SIZE);

    veb.insert(3);
    veb.insert(5);
    veb.insert(10);
    veb.insert(15);

    veb.remove(3);
    veb.remove(15);

    EXPECT_EQ(veb.getmin(), 5);
    EXPECT_EQ(veb.getmax(), 10);
}
```

Expectations:

The elements are deleted and the test case is passed

Result:

```
[ RUN      ] Delete.DeleteValue
[       OK ] Delete.DeleteValue (0 ms)
```

Test Case 5: Delete an already deleted element

Attempt 1:

Code:

// Test Case 5 - Delete an already deleted value

```
TEST(Delete, DeleteValueAgain)
```

```
{
    VEBTree veb(SIZE);

    veb.insert(3);
    veb.insert(5);
    veb.insert(10);
    veb.insert(15);

    veb.remove(3);
    veb.remove(15);

    EXPECT_EQ(veb.getmin(), 5);
    EXPECT_EQ(veb.getmax(), 10);

    veb.remove(3);
    veb.remove(15);

    EXPECT_EQ(veb.getmin(), 5);
    EXPECT_EQ(veb.getmax(), 10);
}
```

Expectation:

Deletion of an already deleted element would not cause any errors and the test case is passed

Result:

The Result matches the expectations

```
[ RUN      ] Delete.DeleteValueAgain
[       OK ] Delete.DeleteValueAgain (0 ms)
```

Test Case 6: Deleting Multiple Elements

Attempt 1:

// Test Case 6 – Deleting Multiple Elements Twice

```
TEST(Delete, DeleteMultiple)
{
    VEBTree veb(SIZE);

    for (int i = 0; i < 10000; i++)
    {
        veb.remove(i);
    }

    EXPECT_EQ(veb.getmin(), -1);
    EXPECT_EQ(veb.getmax(), -1);

    veb.insert(1);

    for (int i = 0; i < 10000; i++)
    {
        veb.remove(i);
    }

    EXPECT_EQ(veb.getmin(), -1);
    EXPECT_EQ(veb.getmax(), -1);

    veb.remove(3);
    veb.remove(15);

    for (int i = 0; i < 10000; i++)
    {
        veb.remove(1);
    }

    EXPECT_EQ(veb.getmin(), -1);
    EXPECT_EQ(veb.getmax(), -1);
```

}

Expectations:

The multiple elements which are inserted gets deleted and when deleting the already deleted elements no error should occur and the test case is passed

Result:

```
[ RUN      ] Delete.DeleteMultiple
[       OK ] Delete.DeleteMultiple (0 ms)
```

c. Successor

Test Case 7: To find the successor for a value entered

Attempt 1:

Code:

```
// Test Case 7 - Find successors
TEST(Successor, FindSuccessor)
{
    VEBTree veb(SIZE);

    veb.insert(3);
    veb.insert(5);
    veb.insert(10);
    veb.insert(15);

    EXPECT_EQ(veb.successor(1), 3);
    EXPECT_EQ(veb.successor(5), 10);
    EXPECT_EQ(veb.successor(15), -1);
}
```

Expectations:

The successor is returned and the test case is passed

Result:

```
[ RUN      ] Successor.FindSuccessor
[       OK ] Successor.FindSuccessor (0 ms)
```

d. Predecessor

Test Case 8: To find the Predecessor for a value entered

Attempt 1:

Code:

```
// Test Case 8 - Find predecessor
TEST(Predecessor, FindPredecessor)
{
    VEBTree veb(SIZE);

    veb.insert(3);
    veb.insert(5);
    veb.insert(10);
    veb.insert(15);

    EXPECT_EQ(veb.predecessor(1), -1);
    EXPECT_EQ(veb.predecessor(3), -1);
    EXPECT_EQ(veb.predecessor(5), 3);
    EXPECT_EQ(veb.predecessor(20), 15);
}
```

Expectations:

The predecessor is returned and the test case is passed

Result:

```
[ RUN      ] Predecessor.FindPredecessor
[       OK ] Predecessor.FindPredecessor (0 ms)
```

e. Searching

Test Case 8: To search for the presence of an element

Attempt 1:

Code:

```
// Test Case 9 - Search
TEST(Search, SearchValue)
{
    VEBTree veb(SIZE);

    veb.insert(5);
    veb.insert(10);
    veb.insert(3);
    veb.insert(15);

    EXPECT_EQ(veb.search(5), true);
    EXPECT_EQ(veb.search(10), true);
    EXPECT_EQ(veb.search(20), false);
}
```

Expectations:

The presence of the elements in the VEB tree is returned properly and the test case is passed

Result:

The result matches the Expectations

```
[ RUN      ] Search.SearchValue
[       OK ] Search.SearchValue (0 ms)
```