APS Project

Aim: Study your implementation of the van Emde Boas tree with application to Prims and compare wrt Union Find and RB Tree

Team Name:

Brute Force

Team Members:

Aman Krishna (2018201070)

Gyanshu Azad Singh (2018201073)

Git Repository:

https://github.com/gyanshu/vEB_Prims

Introductions:

Finding a spanning tree of minimum weight out of the given graph is well known problem. Prim's algorithm is one of the most popular algorithm available to derive the minimum spanning tree (MST). However, the running time of the Prim's algorithm depends largely on the type of data structure used and their respective cost of insert, delete and find operations.

This project does a quantitative analysis of the algorithm across the runtime and space complexities for different data structures. The following have been used to find the minimum spanning tree using Prim's algorithm:

- **Van Emde Boas Tree - vEB Tree**
- **Red-Black Tree - RB Tree**
- **Fibonacci Heap**

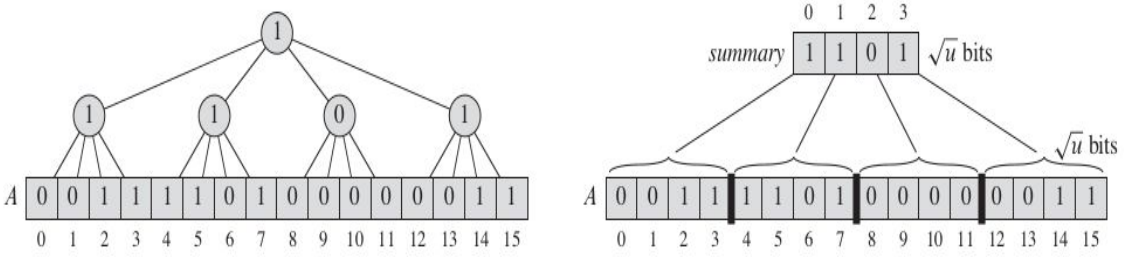
The behaviour of the three structures is noted across different type of graphs to provide a better perspective of their impact on the running of the algorithm. We begin with insights into the data-structures being employed.

Van Emde Boas Tree:

A Van Emde Boas tree, also known as a vEB tree, is a tree data structure which implements an associative array with u-bit integer keys like the one given below.

Cluster and Summary: There is an array, Cluster, which is divided into sub-clusters of size square root of U, where U is the range of element to be inserted. Each sub-cluster is associated with a summary array element which keep track whether the sub-cluster is empty or not.

Min and Max: We store the minimum (Min) and maximum (Max) value of each cluster to improve the time complexity for different operations.



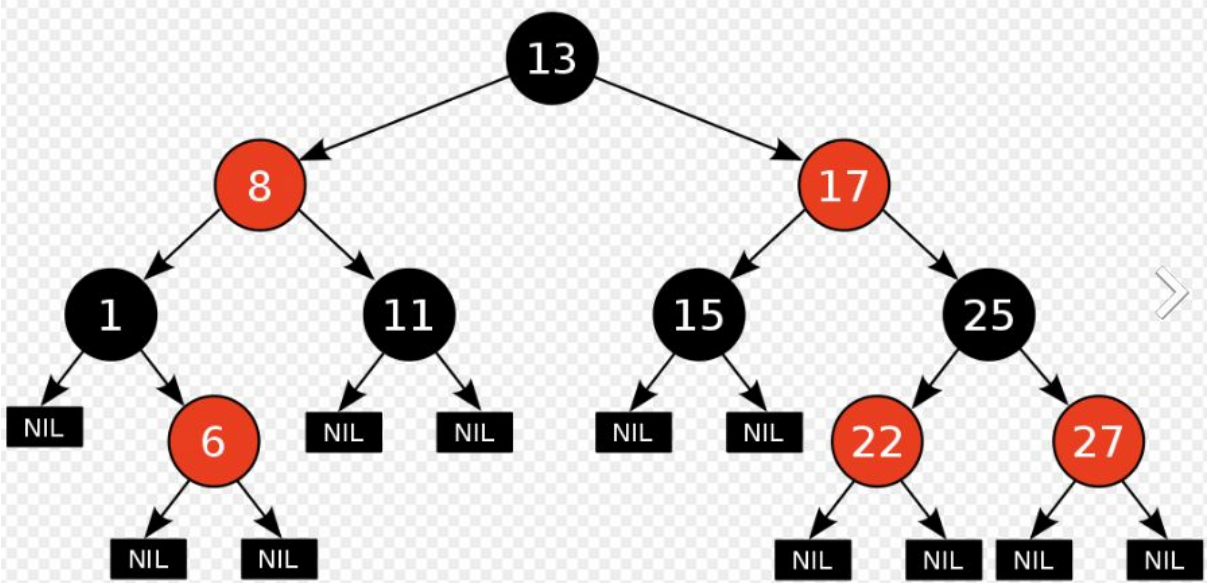
Time Complexity: The time complexity for various operations is as follows (here U is the maximum element to be inserted into the van Emde Boas Tree)

- **Insert:** O(Log(Log U))
- **Delete:** O(Log(Log U))
- **Get Minimum:** O(1)

Space Complexity: The space complexity of vEB tree for the maximum input element U is O(U)

Red-Black Tree:

A Red-Black tree is a kind of self-balancing binary search tree. Each node of a RB tree has a color associated with it which are used to ensure the tree remains approximately balanced during insertion and deletion operations.



Root node of the tree is always Black and no two adjacent nodes can have the color Red. Another property of the tree is that it has the same number of Black nodes to any of its descendant NIL (as seen above).

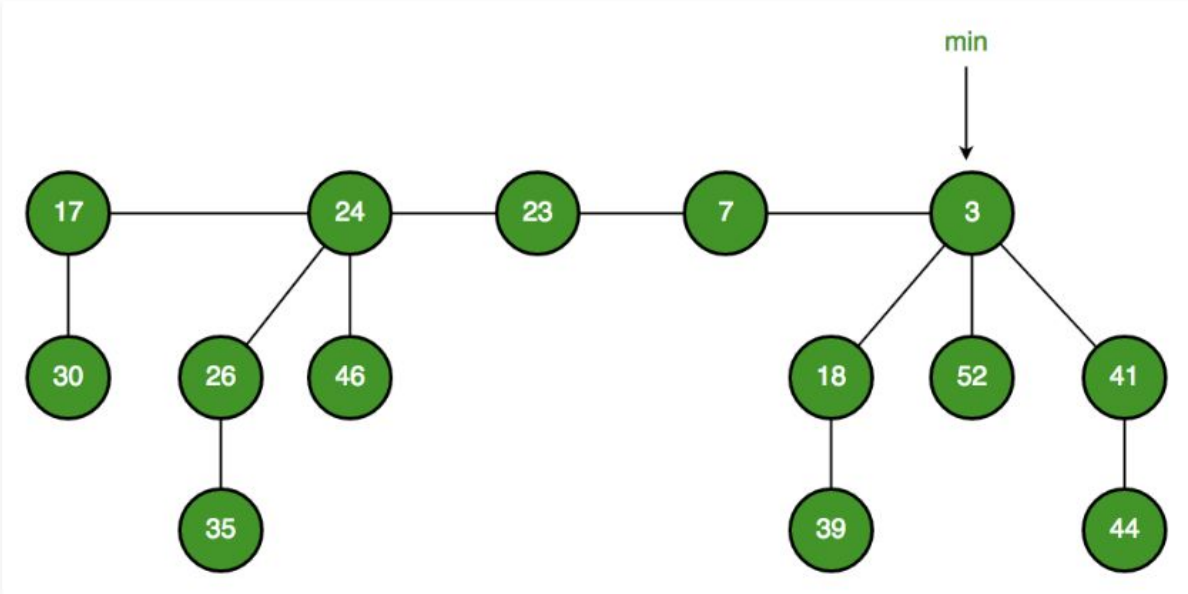
Time Complexity: Following are the time complexities for different operations in average as well as worst cases (Here N is the number of nodes in the tree):

- **Insert**: O(Log N)
- **Delete**: O(Log N)
- **Search**: O(Log N)

Space complexity: The space complexity of a RB Tree is O(N)

Fibonacci Heap:

A Fibonacci Heap is a data structure for priority queue operations consisting of a collection of heap-ordered trees. Fibonacci heaps are named after the Fibonacci numbers which are used in their running time analysis.



The roots of different heaps is stored in a single list called the root list/

Time Complexity: Following are the time complexities for different operations in average as well as worst cases (Here N is the number of nodes in the tree):

- **Insert**: O(1)
- **Delete-min**: O(Log N)
- **Find-Min**: O(1)
- **Decrease-key**: O(1)

Space complexity: The space complexity of a RB Tree is O(N)

Testing:

All three data structures, Van Emde Boas, Red Black Tree and the Fibonacci heap, have been tested against three different type of graphs of different size:

- Completely Connected Graph
- Dense Graph: 70-75% edge density
- Sparse Graph: 20-25% edge density

Number of Nodes in tested Graph: All three structures are tested against different number of input nodes which vary from 10 to 5010. Their respective time complexities are plotted on a graph for every interval of 50 nodes

Edge Weights of tested Graph: The edges are randomly assigned weights which may vary from 1 to 1000

Density of tested Graphs: Density of the graph is defined as:
N = Number of nodes, e = Number of edges currently in tree, E = Number of edges in a Complete tree with Nodes N

$$\text{Density} = (e/E) * 100 \%$$

The test is run on graphs of three densities 100%, 70-75% and 20-25%

How to test: There is a bash file (ba.sh) where user can specify the number of nodes, density and the possible edge weights to be used as input in the three programs. It will output the time being used by each of the three structure onto the terminal

Correctness: We tested the correctness of the programs by comparing the sum of all edge weights of the MST (minimum spanning tree) obtained from the three programs

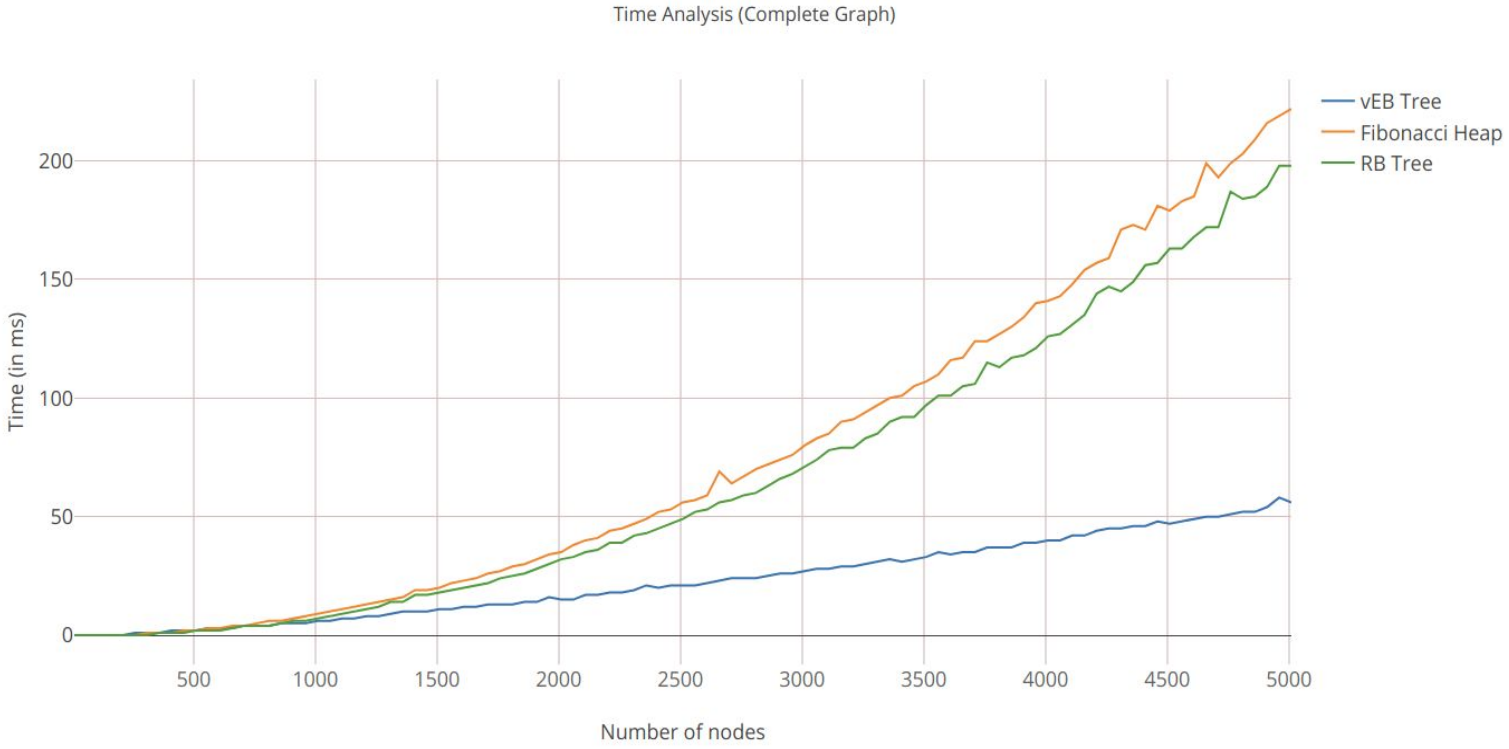
Output: The following slides will contain quantitative analysis of the three structures across the above mentioned constrained and their time complexity and space complexity behaviour are plotted on a line graph

Time Analysis: Complete Graph:

Following graph shows the behaviour of time complexities for the three data structures, Van Emde Boas, Red Black Tree and Fibonacci Heap, for different number of nodes for a complete Graph.

X-Axis: Number of nodes

Y-Axis: Time taken to obtain the MST using Prim's algorithm in millisecond



Observations:

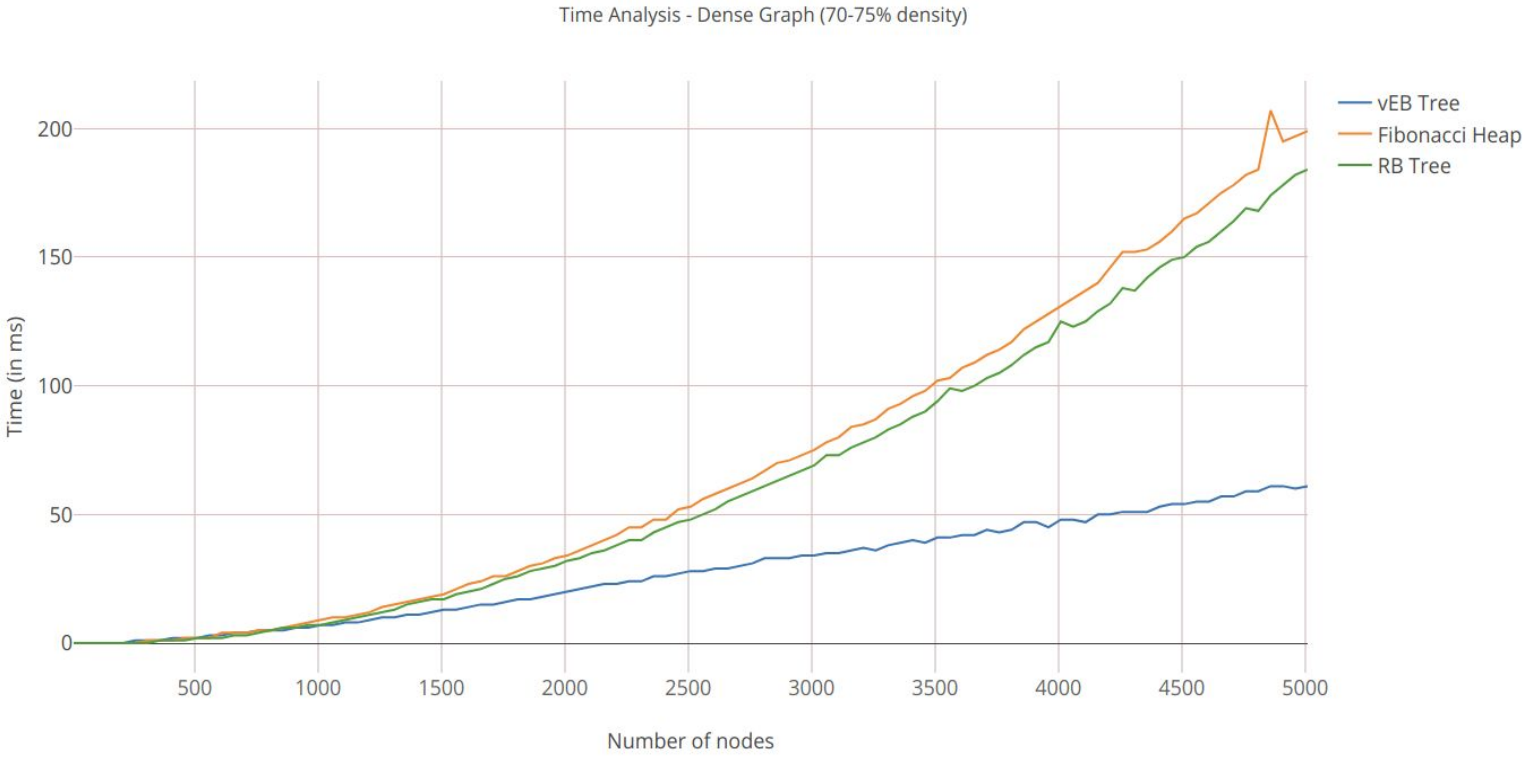
- The three data structure show similar behaviour till the node count is around 1000
- As the number of nodes begin to increase Van Emde Boas shows significant improvement from the other two
- RB Tree and Fibonacci Heap show almost similar behaviour in time complexity for greater number of nodes

Time Analysis: Dense Graph

Following graph shows the behaviour of time complexities for the three data structures, Van Emde Boas, Red Black Tree and Fibonacci Heap, for different number of nodes for a dense Graph.

X-Axis: Number of nodes

Y-Axis: Time taken to obtain the MST using Prim's algorithm in millisecond



Observations:

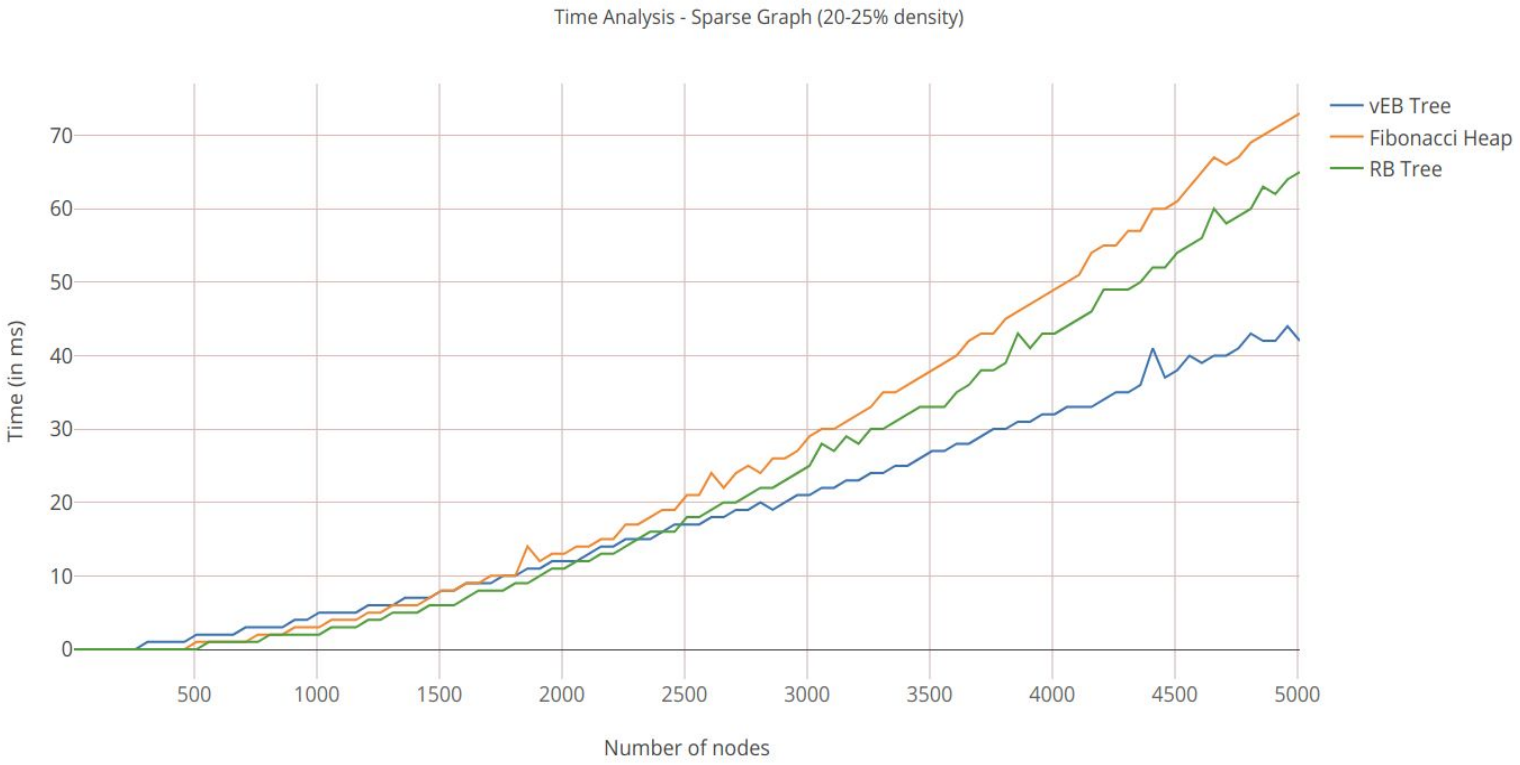
- The overall time taken by RB tree and Fibonacci heap is lesser compared to the complete graph but almost same for vEB tree
- The three data structure show similar behaviour till the node count is around 1200
- As the number of nodes begin to increase Van Emde Boas shows significant improvement from the other two
- RB Tree and Fibonacci Heap show almost similar behaviour in time complexity for greater number of nodes

Time Analysis: Sparse Graph

Following graph shows the behaviour of time complexities for the three data structures, Van Emde Boas, Red Black Tree and Fibonacci Heap, for different number of nodes for a Sparse Graph.

X-Axis: Number of nodes

Y-Axis: Time taken to obtain the MST using Prim's algorithm in ms



Observations:

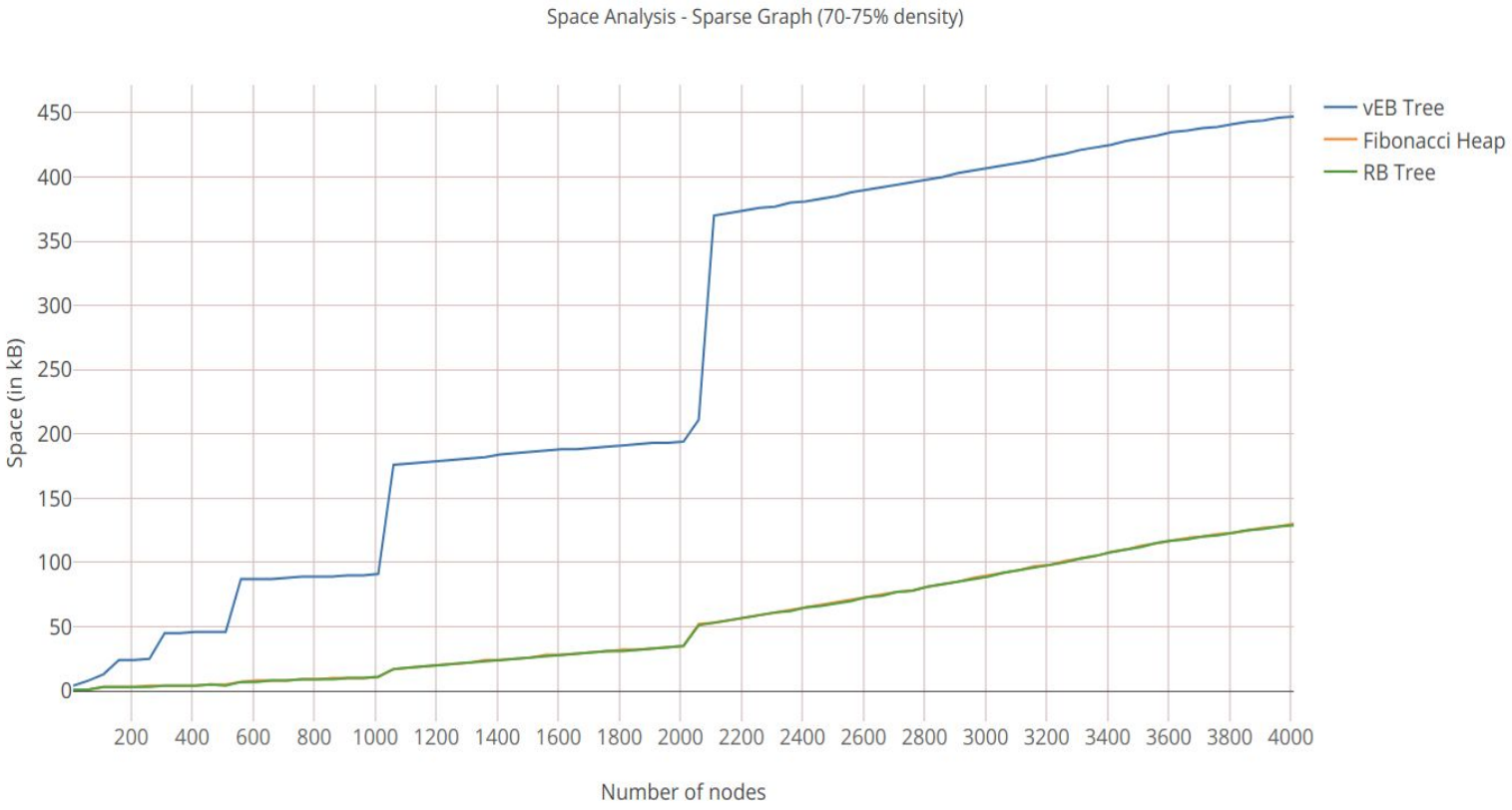
- The overall time taken by RB tree and Fibonacci heap is significantly lesser compared to the complete and dense graph but almost same for vEB tree
- The three data structure show similar behaviour till the node count is around 2500
- As the number of nodes begin to increase Van Emde Boas shows improvement over the other two but they are still close
- RB Tree and Fibonacci Heap show almost similar behaviour in time complexity for greater number of nodes

Space Analysis: Complete Graph:

Following graph shows the behaviour of time complexities for the three data structures, Van Emde Boas, Red Black Tree and Fibonacci Heap, for different number of nodes for a complete Graph.

X-Axis: Number of nodes

Y-Axis: Time consumed to obtain the MST using Prim's algorithm in kB



Observations:

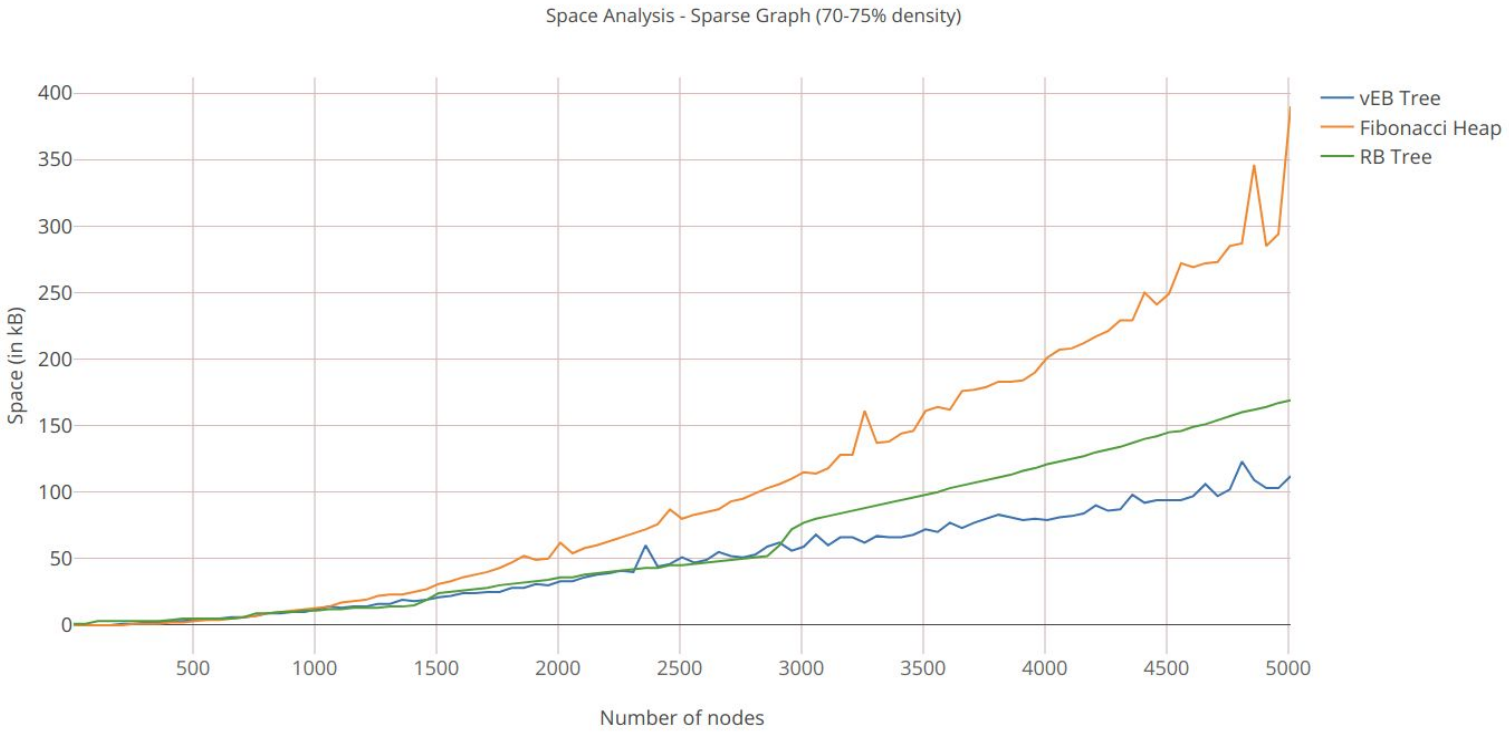
- Space taken by Fibonacci Heap and RB Tree are almost same for different node numbers for a complete graph
- As the number of nodes begin to increase Van Emde Boas consumes significantly large amount of space

Space Analysis: Dense Graph

Following graph shows the behaviour of time complexities for the three data structures, Van Emde Boas, Red Black Tree and Fibonacci Heap, for different number of nodes for a dense Graph.

X-Axis: Number of nodes

Y-Axis: Space consumed to obtain the MST using Prim's algorithm in kB



Observations:

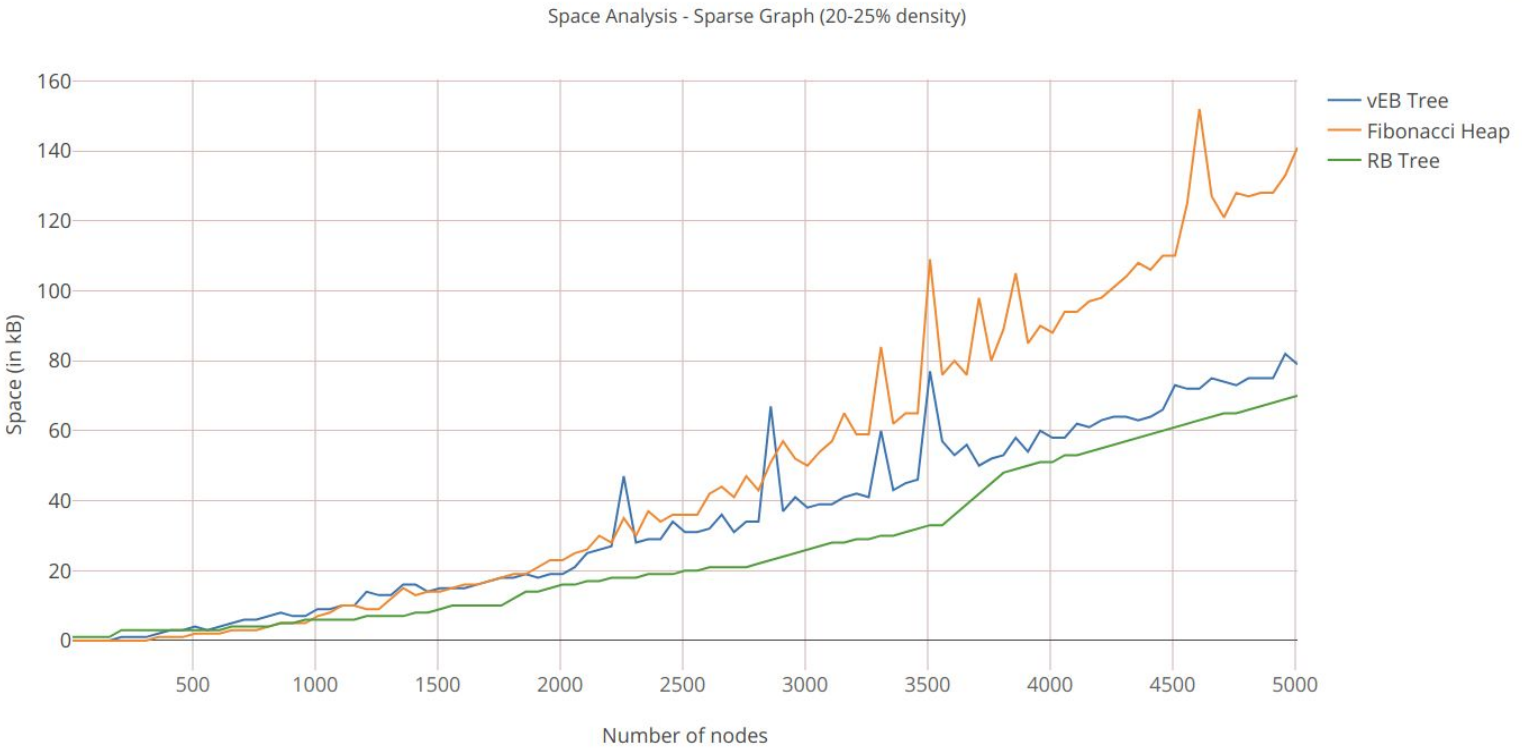
- The three data structure show similar behaviour till the node count is around 1500
- Van Emde Boas takes the least amount of space compared to the other two
- As the number of nodes begin to increase Fibonacci heap takes significantly more space compared to other two

Space Analysis: Sparse Graph

Following graph shows the behaviour of time complexities for the three data structures, Van Emde Boas, Red Black Tree and Fibonacci Heap, for different number of nodes for a Sparse Graph.

X-Axis: Number of nodes

Y-Axis: Space taken to obtain the MST using Prim's algorithm in kB



Observations:

- The three data structure show similar behaviour till the node count is around 2000
- As the number of nodes begin to increase Fibonacci heap takes significantly more space compared to other two
- RB Tree and vEB tree show almost similar behaviour, with some exceptions where vEB space complexity spikes, in space complexity for greater number of nodes with vEB with a higher memory consumption

Handling Duplicates:

One of the issues while constructing Van Emde Boas Tree and Red Black Tree was duplicacy of edge weights in graph. In vEB tree presence of duplicate edge weights would result in collision. This could be tackled in two ways:

- **Open Chaining:** We could construct a linked list at the value where collision occurs and store both the values
- **Transformation:** Transforming the edge weights such that there are no duplicate values

Problem with Open Chaining: If there were a large number of edge with the same weight it would result in a long linked list at the point of collision which may increase the insertion and deletion time over the required O(Log(Log U)). There we used transformation instead

Transformation: We instead of inserting the edge weight into the vEB tree inserted : ($e * (N + 1) + n_i$) where,

- e represents the weight of the current edge being inserted
- N being the total number of nodes in the graph
- n_i represents the source of the node

Pros:

- No extra space needed to store the source as it can be extracted from the value being inserted
- No duplicacy

Conclusions:

After running the above test following conclusions have been derived:

- Van Emde Boas Tree shows significant improvement over RB tree and Fibonacci heap for large number of nodes if the graphs are dense and complete
- vEB shows similar number time complexities across various density graphs. As observed above, the time complexity was between 45-55 for complete, dense as well as sparsely connected graph
- Red Black Tree and Fibonacci heap show similar behaviour with RB tree showing minor improvements over Fibonacci heap over larger number of nodes
- vEB consumes a large amount of space in case of a complete graph compared to the instanced when the graph is dense or sparse
- Space complexity for vEB for sparse and dense graph is similar
- Theoretically, Fibonacci heap takes constant time for insertion and to retrieve min-element and should take less time when compared to RB Tree but it performs a little worse than the RB Tree due to high value of it’s hidden constants