**Name:** Swapnil S. Gite

**MIS:** 112003046

**Subject:** DSA

**Project Topic:** Faster website accessing using Splay Tree and comparing search efficiency with that of AVL and vanilla tree

**Splay Trees:**

A splay tree is a binary search tree with the additional property that recently accessed elements are quick to access again.

Like **self-balancing binary search trees**, a splay tree performs basic operations such as insertion, and look up in **O (log n) amortized time**.

**Amortized Time:**

We go with amortized time complexity when we feel that **not all operations are worst** and **some can be efficiently done**. In splay trees **not all** splay operations will **lead to O (log n)** worst case complexity.

For example, let us consider the search operation in Splay Trees:

In Splay Trees, the first search operation on any of the element present in the tree has

**O (log n)** has its worst time complexity.

But after an element is searched, it is splayed to the root, and then whenever it is accessed again in the program in continuity, it takes **O (1) time**.

Therefore, **amortized time** is the way to express the time complexity when an algorithm has the very **bad time complexity only once in a while** besides the time complexity that happens most of time.

Good performance for a splay tree depends on the fact that it is **self-optimizing**, in that **frequently accessed nodes will move nearer to the root** where they can be **accessed more quickly**. The worst-case height—though unlikely—is O(*n*), with the **average being O (log *n*).**

**AVL Trees:**

For insertion operation, the running time complexity of the AVL tree is **O (log n)** which is similar for searching the position of element and performing rotations to keep the tree balanced.

AVL is a **height balanced** tree.

The height of an AVL tree storing n keys is O (log n).

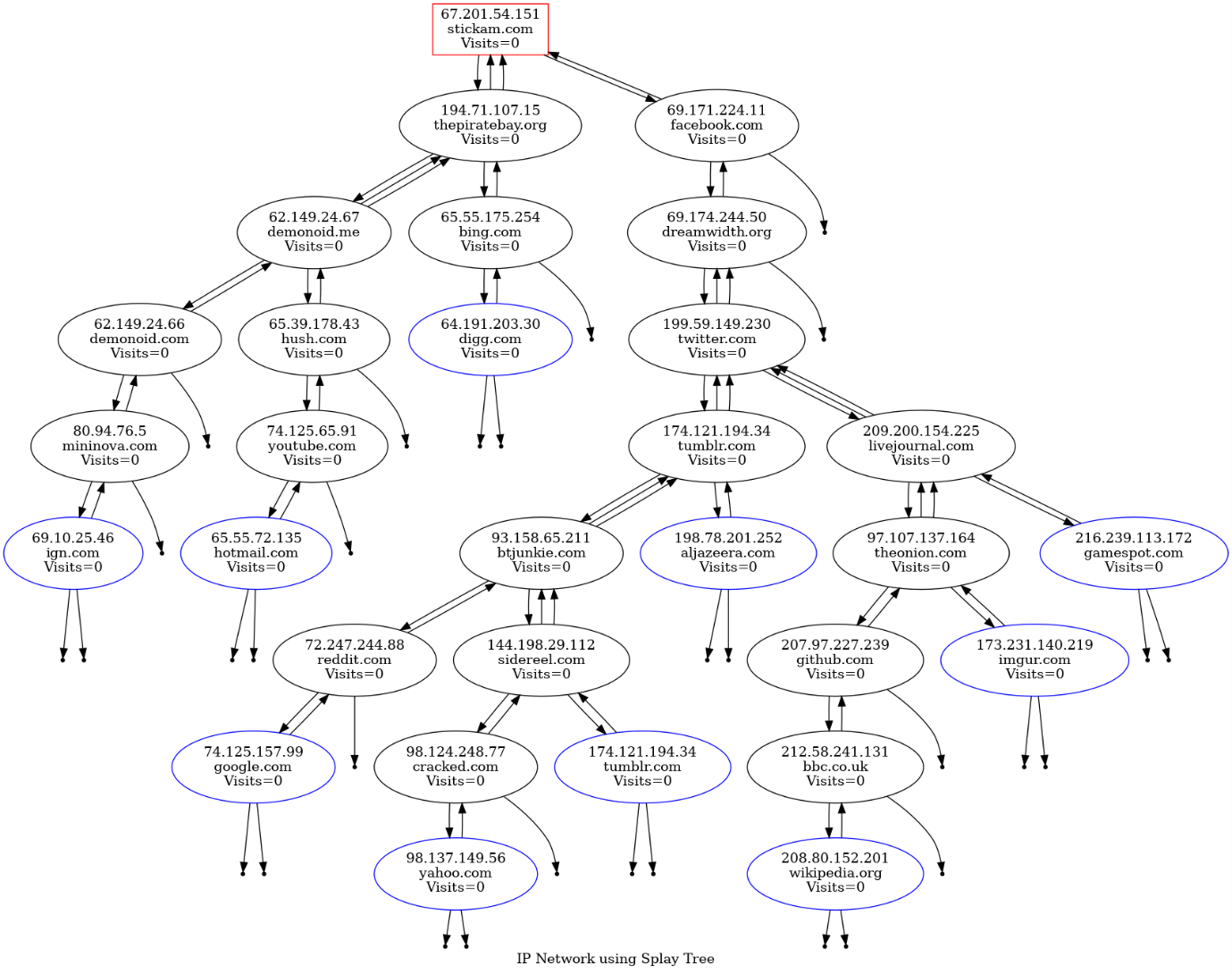
It is efficient than BST as AVL tree never makes a skewed tree.

Hence it reduces the search operation time complexity as compared to Vanilla Trees.

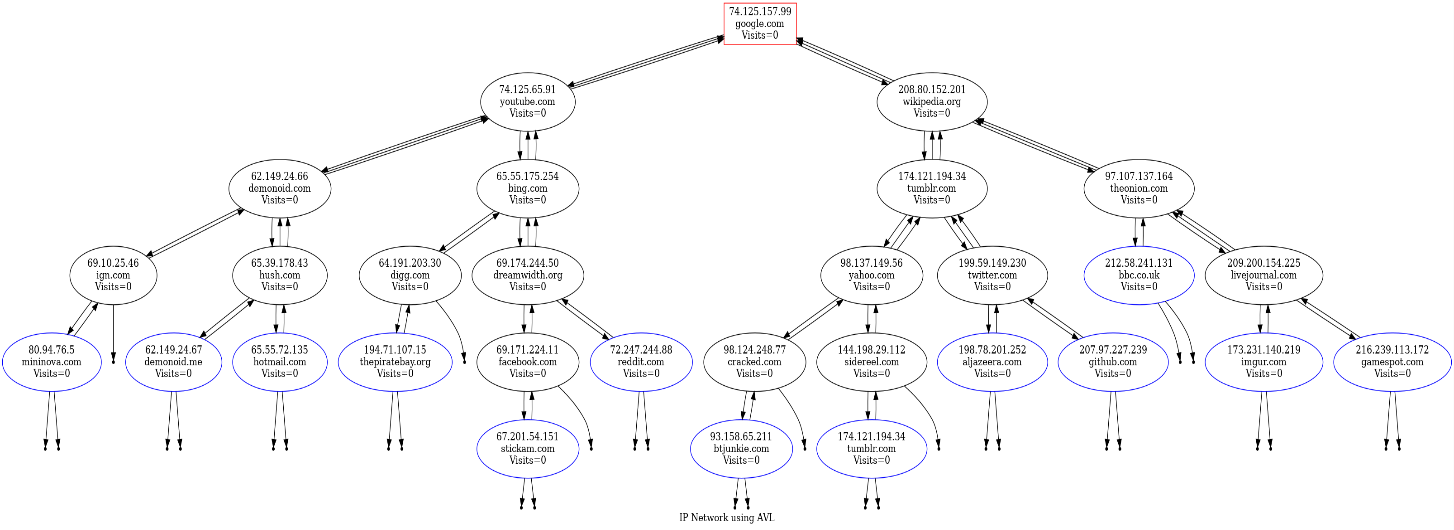
|  |  |  |
| --- | --- | --- |
| **Function** | **Amortized** | **Worst Case** |
| Search | O (log n) | O (n) |
| Insert | O (log n) | O (n) |
| Delete | O (log n) | O (n) |

**Images Generated after inserting elements for the first time:**

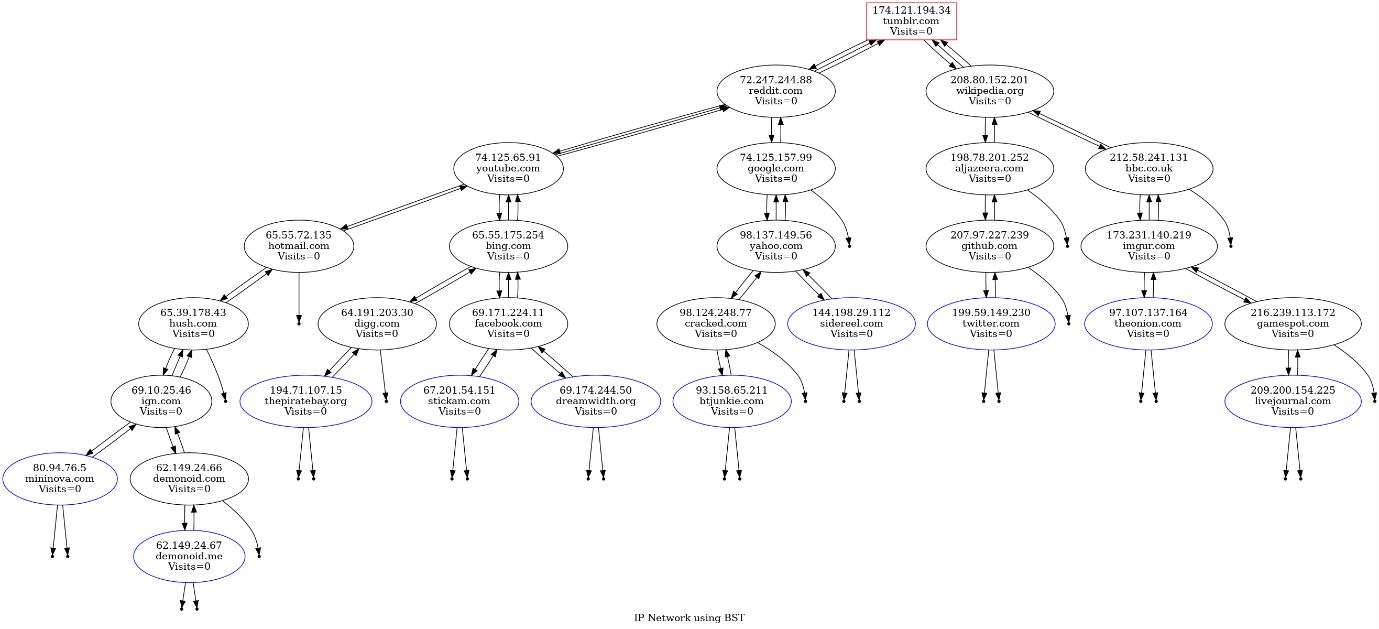
**1)Splay Tree:**



**2.AVL Tree:**



**3.Vanilla Tree:**



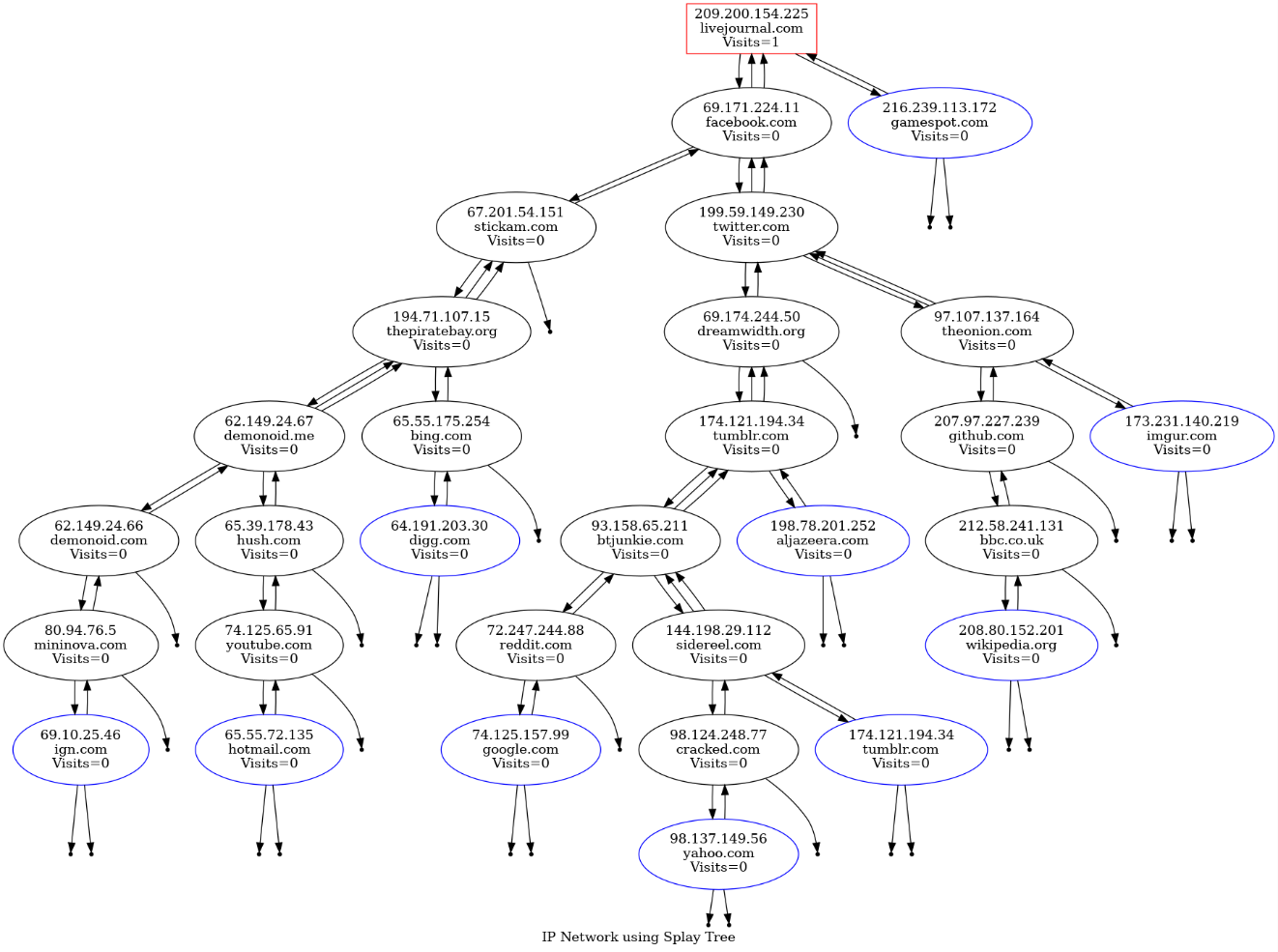
Now let us search for an element in all the trees:

Element Searched:

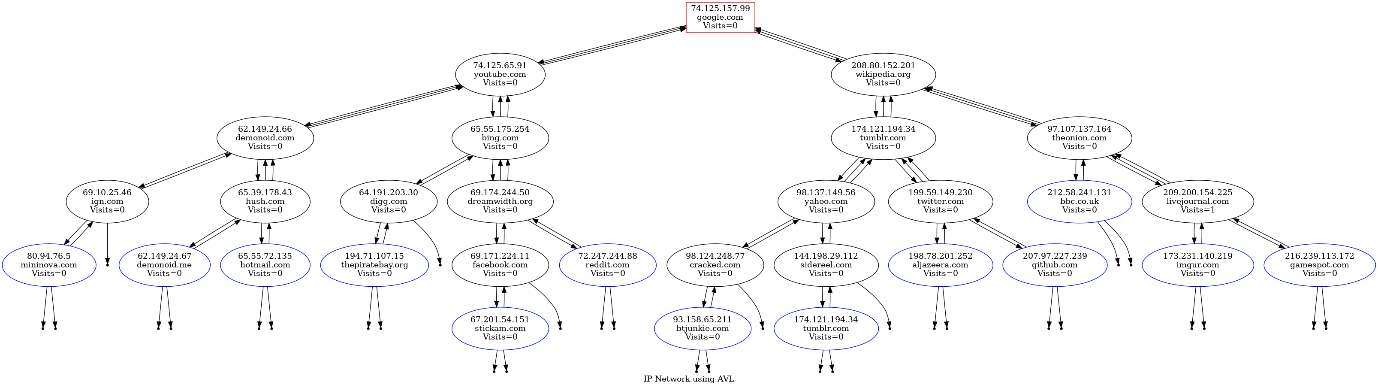
Website: livejournal.com

Ip Address: 209.200.154.225

**1.Splay Tree:**

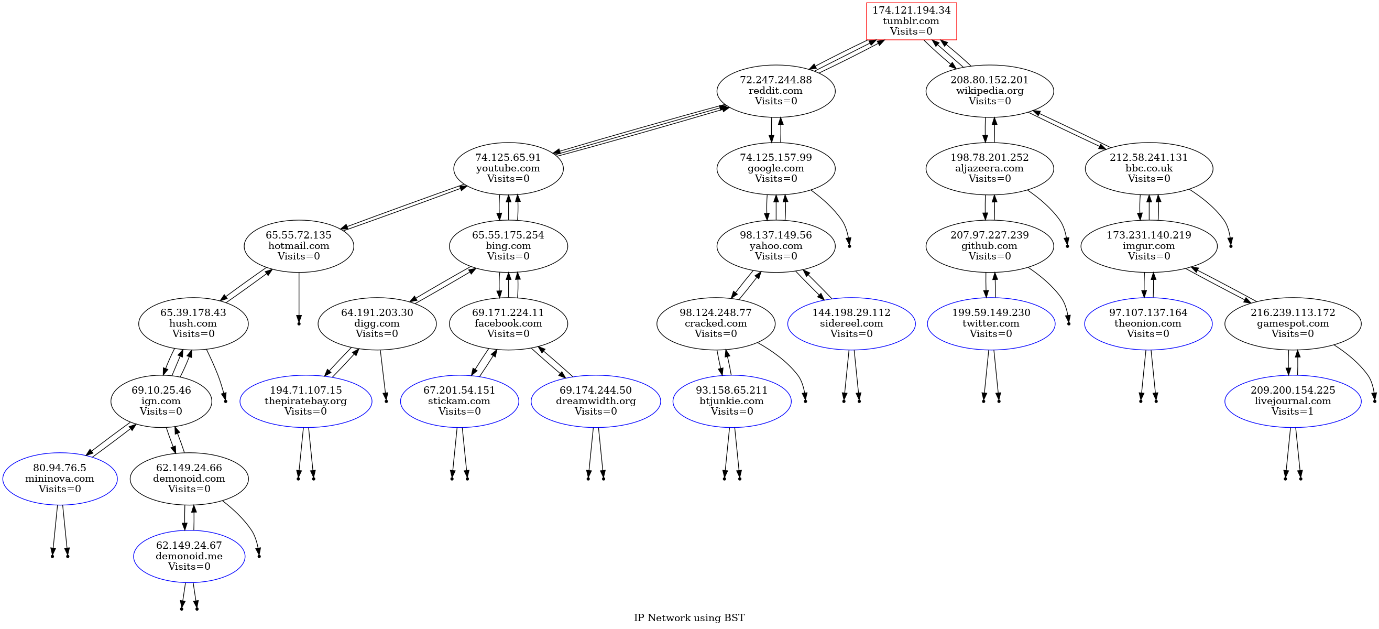


**2.AVL Tree:**



The tree remains in same condition and the number of accesses to the element increases by one.

**3.Vanilla Tree:**



The tree remains in same condition and the number of accesses to the element increases by one.