Trie, Digital search tree, Heap and Treap

Trie

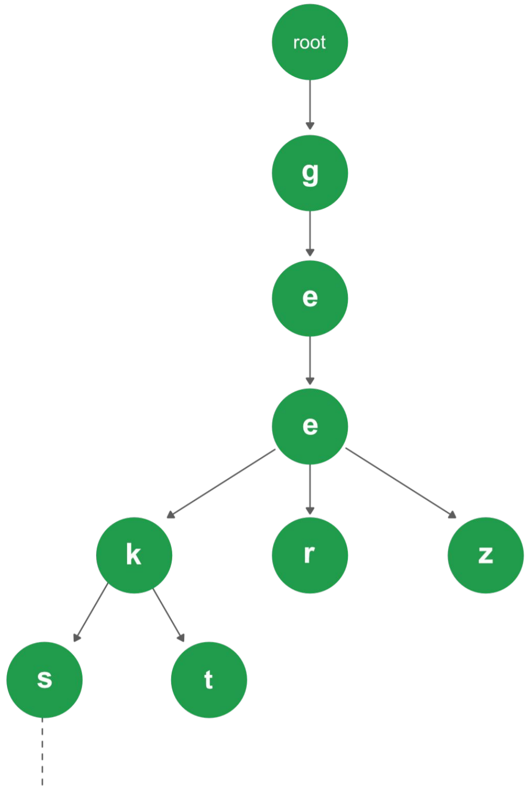
A trie, also called **digital tree** or **prefix tree**, is a type of search tree, a tree data structure used for locating specific keys from within a set. These keys are most often strings, with links between nodes defined not by the entire key, but by individual characters. In order to access a key (to recover its value, change it, or remove it), the trie is traversed depth-first, following the links between nodes, which represent each character in the key.

Unlike a binary search tree, nodes in the trie do not store their associated key. Instead, a node's position in the trie defines the key with which it is associated. This distributes the value of each key across the data structure, and means that not every node necessarily has an associated key.

All the children of a node have a common prefix of the string associated with that parent node, and the root is associated with the empty string. This task of storing data accessible by its prefix can be accomplished in a memory-optimized way by employing a radix tree.

Trie is an efficient information re*Trie*val data structure. Using Trie, search complexities can be brought to optimal limit (key length). If we store keys in binary search tree, a well balanced BST will need time proportional to M * log N, where M is maximum string length and N is number of keys in tree. Using Trie, we can search the key in O(M) time. However the penalty is on Trie storage requirements.

```
      root
   /   \    \
   t   a     b
   |   |     |
   h   n     y
   |   |  \  |
   e   s  y  e
/  |   |
i  r   w
|  |   |
r  e   e
       |
       r
```

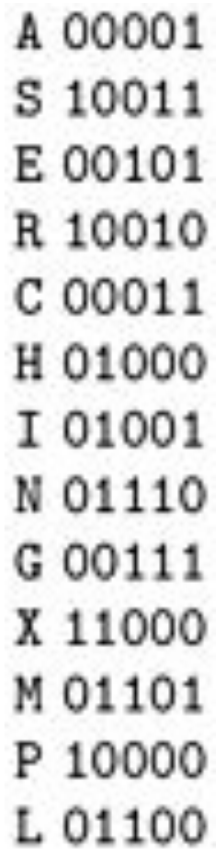


Digital Search Trees

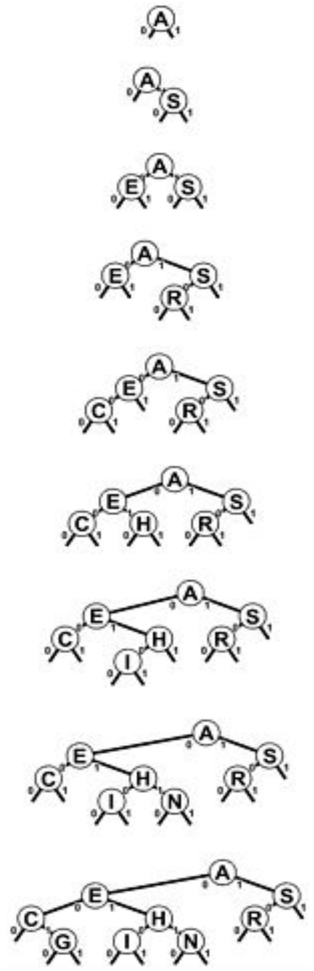
The simplest radix-search method is based on the use of digital search trees (DSTs). The search and insert algorithms are identical to binary tree search and insertion except for one difference: We branch in the tree not according to the result of the comparison between the full keys but rather according to selected bits of the key. At the first level, the leading bit is used; at the second level, the second leading bit is used; and so on, until an external node is encountered.

- DST is one kind of binary tree which contains only binary data.
- If the bit of DST starts with 0, then it is in left subtree and if the bits starts with 1 then it is in right subtree and this process works recursively.
- Searching is based on binary representation of data.
- Insertion, deletion and searching in DST are easier than BST and AVL tree.
- DST requires less memory than BST and AVL tree.

This sequence depicts the result of inserting the keys A S E R C H I N G into an initially empty digital search tree.



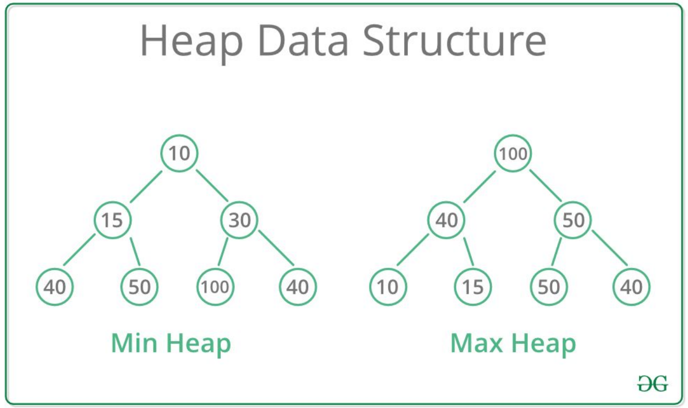
A	00001
S	10011
E	00101
R	10010
C	00011
H	01000
I	01001
N	01110
G	00111
X	11000
M	01101
P	10000
L	01100



Heap

A Binary Heap is a special Tree-based data structure in which the tree is a complete binary tree. Generally, Heaps can be of two types:

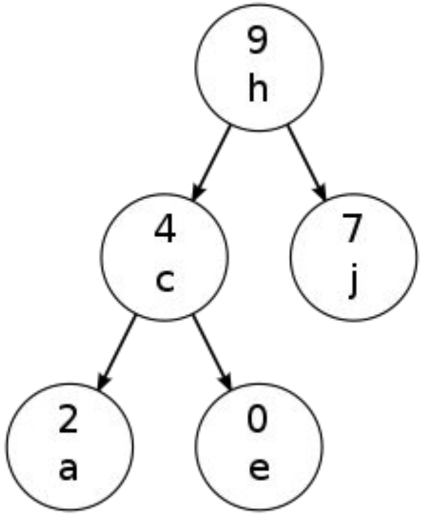
1. **Max-Heap**: In a Max-Heap the key present at the root node must be greatest among the keys present at all of it's children. The same property must be recursively true for all sub-trees in that Binary Tree.
2. **Min-Heap**: In a Min-Heap the key present at the root node must be minimum among the keys present at all of it's children. The same property must be recursively true for all sub-trees in that Binary Tree.



Treap

The **treap** and the **randomized binary search tree** are two closely related forms of binary search tree data structures that maintain a dynamic set of ordered keys and allow binary searches among the keys. After any sequence of insertions and deletions of keys, the shape of the tree is a random variable with the same probability distribution as a random binary tree; in particular, with high probability its height is proportional to the logarithm of the number of keys, so that each search, insertion, or deletion operation takes logarithmic time to perform.

A treap with alphabetic key and numeric max heap order

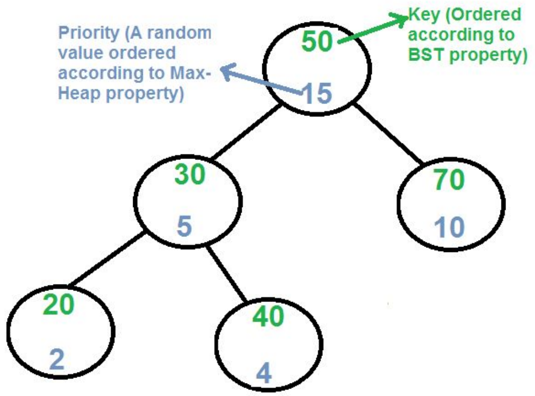


Like Red-Black and AVL Trees, Treap is a Balanced Binary Search Tree, but not guaranteed to have height as O(Log n). The idea is to use Randomization and Binary Heap property to maintain balance with high probability. The expected time complexity of search, insert and delete is O(Log n).

Every node of Treap maintains two values.

1) **Key** Follows standard BST ordering (left is smaller and right is greater)

2) **Priority** Randomly assigned value that follows Max-Heap property.


Priority (A random value ordered according to Max-Heap property)
Key (Ordered according to BST property)
50
15
30
5
70
10
20
2
40
4