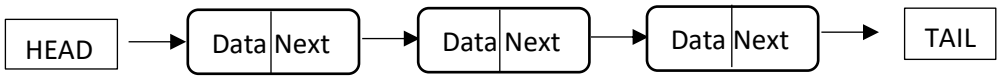
Name : Lucky Laurens

Username Discord : Luckymai

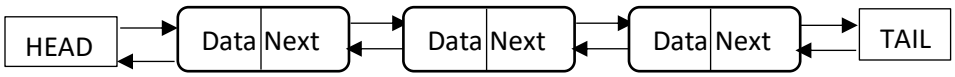
Group : OUTPLAY

Linked List

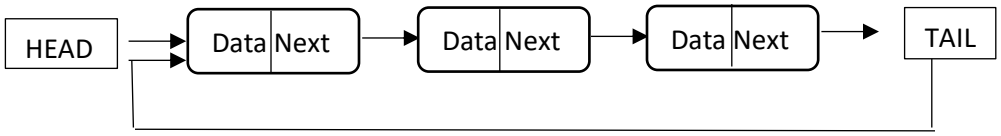
1. Single Linked List is a linked list which each Node has data and a pointer to the next Node.



Double Linked List is a linked list which each Node has data and pointers to the next and previous Node.



Circular Linked List is a linked list which each Node has data and pointer to the next Node, and the last element /node is linked to the head (first element/node).



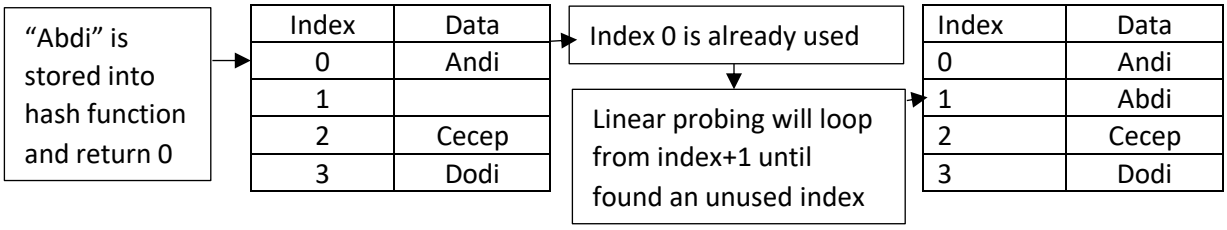
2. The main different between linked list and array is the flexibility. With array, we should do indexing, and the size is limited. But, Linked List has no index and the size is very flexible using dynamic allocation.
3. Floyd Algorithm is an algorithm to find the shortest path from a graph or tree. The implementation of floyd algorithm is to find the shortest path from Node to another Node, to determine/ detect loop in linked list, etc.

Stack and Queue

1. The main different between Stack and Queue is, Stack FILO (First in, Last Out) and Queue is FIFO (First in, First Out). The implementation of Stack is like pile of plates. When we put plates in a pile, the first plate will be in the lowermost, and the last plate will be in the topmost, so, the first in, the last out. And the implementation of Queue is like queue at cashier, the first to come, the first to be served and the last to come, the last to be served.
2. Prefix is a way of writing notation with the operator in front of two operand. Infix is a way of writing notation in a normal format. Postfix is a way of writing notation with the two operand first in front of operator. The implementation using stack is to hold the operand until we found an operator. The prefix notation using stack is read from behind, and the postfix notation using stack is read from the front. So, if we get an operand, we should hold it into a stack until we found an operator. After we found an operator, we should do the calculation of 2 number from the stack, and change the 2 numbers from the stack into the result of the calculation.

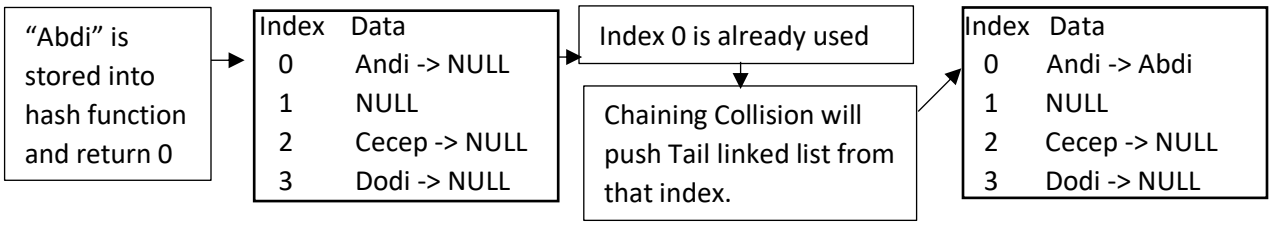
Hashing and Hash Table

1. Hashtable is a table that contains data with index that already given from the hash function. Hash function is a function to produce an unique number which will be used for index in hashtable. Collision is a way to solve the problem if the index that is already produce from hash function is already used in hashtable.
2. Linearprobing is a collision method that will search an unused index. So, when the index produced from the hash function is already used in hashtable, the linear probing collision will search an unused index which will going to be used for the new data.



When user insert a new data, for example Abdi, and the hash function will return 0 (first ASCII element), the index 0 is already used. So, the linear probing will loop from index + 1, until it found a new unused index.

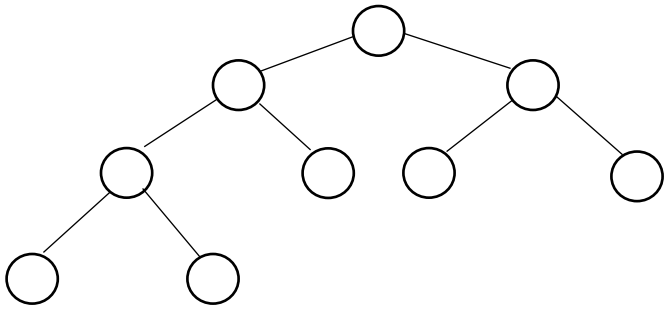
Chaining collision is a method that will use a linked list to store a new data. So, when the index produced from hash function is already used in hash table, the new data will be stored into the linked list from that index.



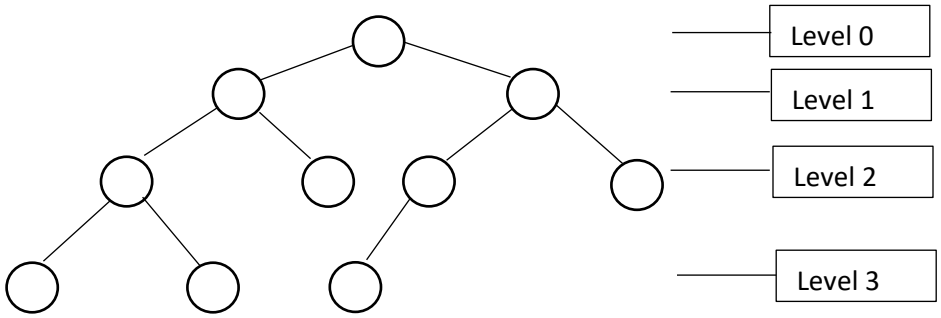
So, the hash function will return 0 from "Abdi", but in hash table, index 0 is already used. So, the chaining collision will push Tail linked list from index 0. The "Abdi" is already stored into linked list index 0.

Binary Search Tree

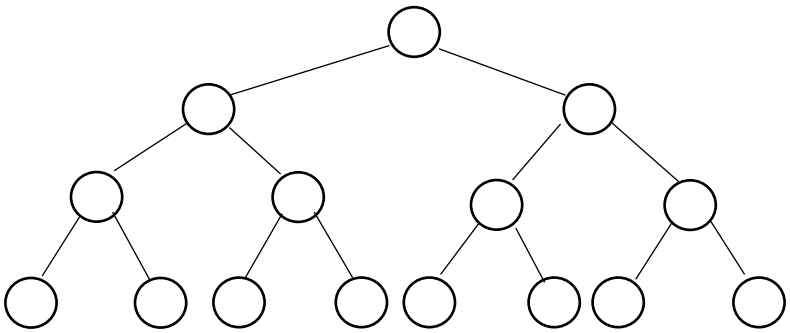
1. 5 Types of Binary Tree:
   - Full Binary Tree
     Is a Binary Tree that has either zero or two children. So, there is no parents that has only one child.



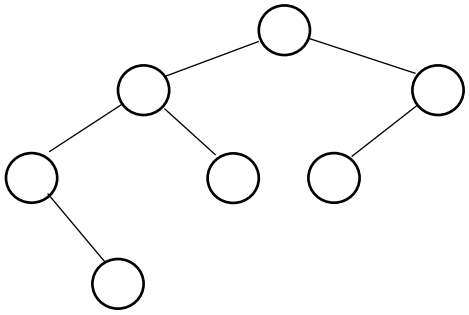
   - Complete Binary Tree
     Is a binary tree where all the tree levels all full filled (entirely) with nodes, except the lowest level.



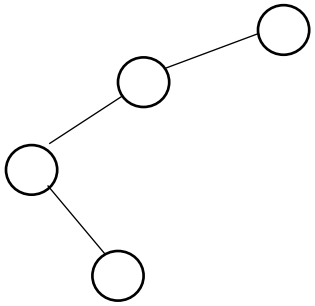
     As we can see, the level 0 – 2 are filled entirely with nodes. The only one which is not fullfilled is only the last level (lowest) which is level 3.
   - Perfect Binary Tree
     Is a binary tree that all levels should be fullfilled entirely with Nodes. All Nodes have strictly two children, except the last level of Nodes (lowest) and the lowest Nodes (leaf) is at the same level.

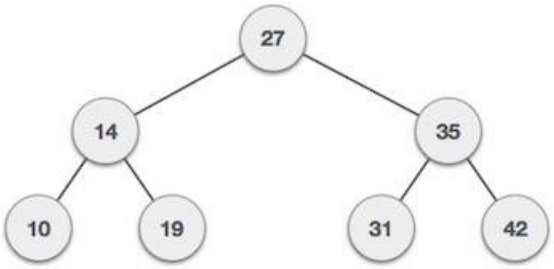


- Balanced Binary Tree
  Is a binary tree with O (logN) height, where 'N' is the number of Nodes. Or we can say that the height of the left and the right subtrees is only have at most one level difference.



  The height of the left subtrees is 3, and the height of the right subtrees is 2. The different is only one, so that is a balanced binary tree.
- Degenerate Binary Tree
  Is a binary tree that all nodes have only a single child except the leaf node.





2. Insertion of 24, 18, 55.
   The insertion of Node is using recursion until we found a NULL root.
   Insert 24 :
   1. First, 24 is less than 27. So, root = root-> left, which is 14.
   2. Then, 24 is more than 14, so root = root -> right, which is 19.
   3. Then, 24 is more than 19, so root = root-> right, which is NULL.
   4. Because the root is NULL, then we put Node 24 there, where is in the right subtree of 19.

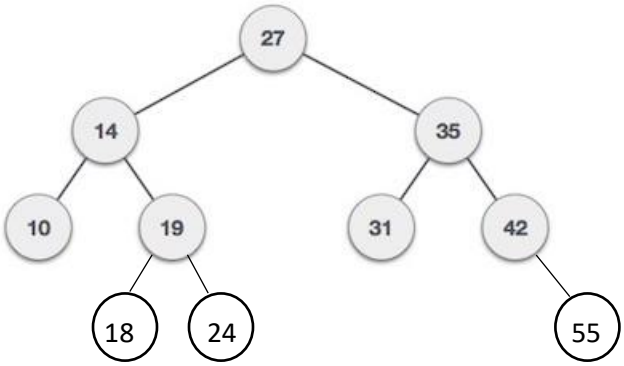
Insert 18:

1. First, 18 is less than 27, so root = root ->left, which is 14.
2. Then, 18 is more than 14, so root = root->right, which is 19.
3. Then, 18 is less than 19, so root = root ->left, which is NULL.
4. Because the root is NULL, then we put Node 18 there, where is in the left subtree of 19.

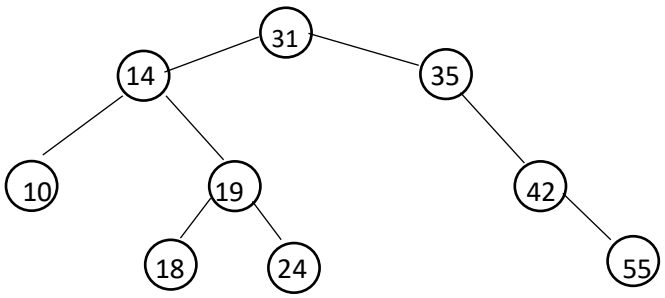
Insert 55:

1. First, 55 is more than 27, so root =root->right, which is 35.
2. Then, 55 is more than 35, so root = root->right, which is 42.
3. Then, 55 is more than 42, so root = root->right, which is NULL.
4. Because the root is NULL, then we put Node 55 there, where is in the right subtree of 42.

The final result of insertion :

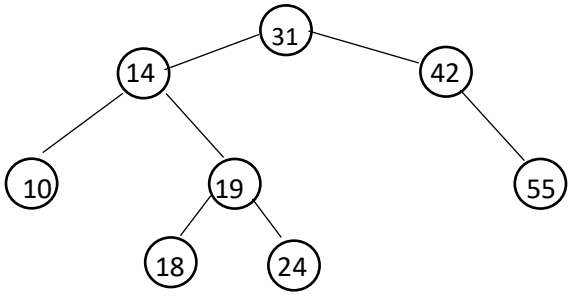


3. Deletion of 27, 35, 42.
   We use recursion to delete Node, but first is to find the Node first.
   Delete 27:
   1. Because the BaseRoot is the Node that we are going to delete, we can directly delete it.
   2. Because Node 27 has 2 children, then we should find the maximun value of left subtree, or the minimum value of right subtree.
   3. min = root -> right, which is 35, and we will loop until we find the minimum value.
   4. min = min->left, which is 31.
   5. Because min -> left = NULL, so the minimum value of right subtree of Node 27 is 31.
   6. Change the value of Node 27 to 31. Root->value = min -> value. Now, the value of root is 31.
   7. Delete min Node, because the min Node is already replace to root.
      The result of deletion 27 :



Delete 35 :

1. First, declare root = baseroot, and find the Node that we are going to delete.
2. Because 35 is more than 31 (root->value), so root = root->right, which is 35.
3. Because now root->value is the same with value that we are going to delete (right Node), we can directly delete it.
4. Because 35 (root) has only one child, which is 42, so we can directly move it.
5. Make a temp = root->right, which is 42.
6. free the root, and set root=NULL. Or we can simply directly “root = temp”.
7. If we use recursion, we should return temp, so later the the temp will be the child of the previous node.
   The result of deletion 35 :



Delete 42:

1. First, declare root = baseroot, and find the Node that we are going to delete.
2. Because 42 is more than 31 (root->value), so root = root->right, which is 42.
3. Because now root->value is the same with value that we are going to delete (right Node), we can directly delete it.
4. Because 42 (root) has only one child, which is 55, so we can directly move it.
5. Make a temp = root->right, which is 55.
6. Free the root and set root = NULL, or we can simply directly “root = temp”.
7. If we use recursion, we should return temp, so later the the temp will be the child of the previous node.
   The result of deletion 42 :