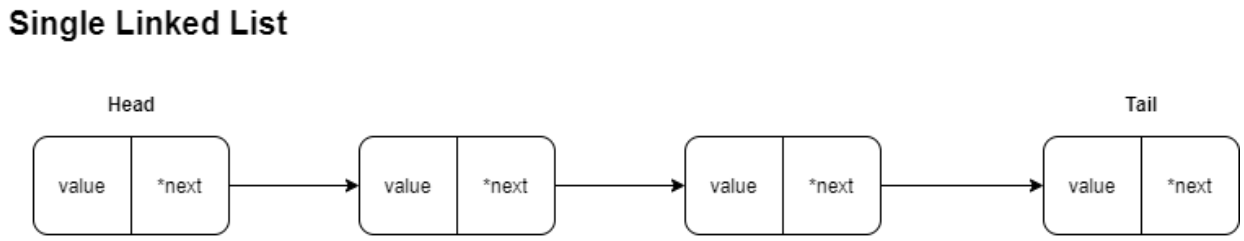
Bryan T / Bryan Tamin

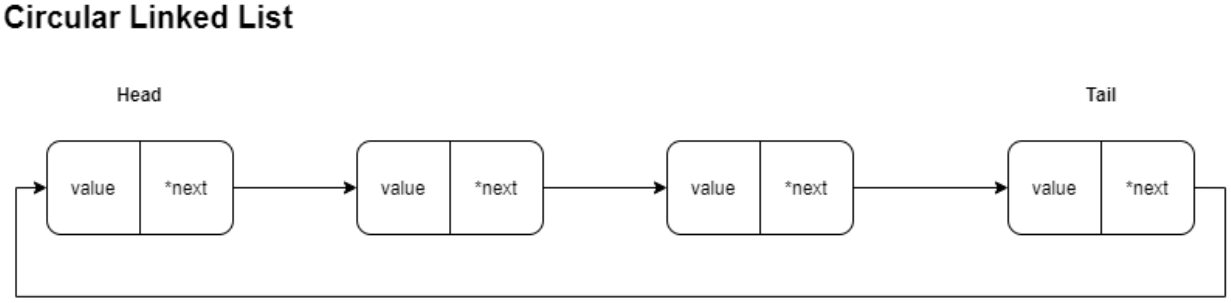
Linked List [25%]

1. Explain single, double, and circular linked list in a graphical view! (.pdf, .png)



Each node in single linked list has value and just one pointer, that point to the next node. so there is no going back here, the next node cannot call the previous one

Each node in double linked list has value and two pointer which is *prev and *next, the *prev pointer will point the node before it and *next will point the node after it. So in this linked list we can move 2 ways, after we go to the next node we can still go back to the previous node.



Each node in circular linked list has value and just one pointer, that point to the next node, but different from the single linked list, in circular usually the tail point to the head of the linked list, so we will visit the node that we have visited before.

2. What are the main differences between Linked List and Array? (.pdf)

1. Linked list is more dynamic than array, in linked list we can add or delete node easily but in array once the array is full we cannot add any value before we delete something, because **array is static and have fixed size meanwhile linked list size can vary**.

2. In linked list we use Node and pointer (there is head and tail) to go a certain node we have to traverse each node from head until that node, but in array we use index so when we want to display the value of the exact point of array (indexing) it is easier.

3. Explain Floyd's algorithm and implementation including its pseudocode! (.pdf)

Floyd’s cycle detection algorithm is an algorithm to detect whether a linked list has a cycle or not.

How it works:

We will place 2 temp variable at the head of the linked list, let it be temp1 and temp2. temp1 is moving 1 node each and temp2 moves 2 node each loop and when temp1 and temp2 adress is the same that means the linked list has a cycle in it.

Simulation:

temp1 = 1, temp2 = 1

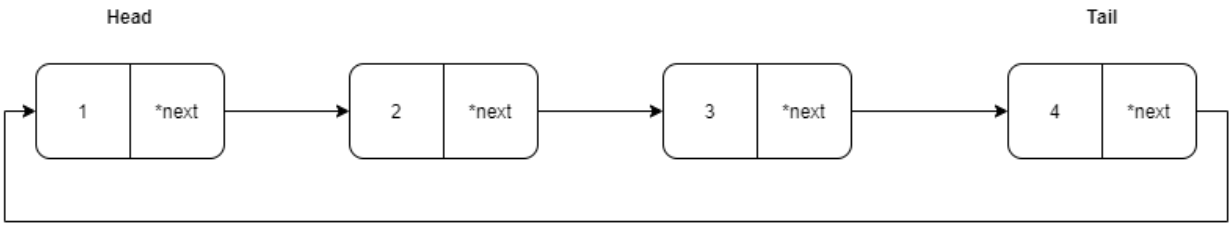
temp1 = 2, temp2 = 3

temp1 = 3, temp2 = 1

temp1 = 4, temp2 = 3

temp1 = 1, temp2 = 1

temp1 == temp2 so this linked list has a cycle.



Pseudocode:

START

```
Struct Node{
      int value
      int flag = 0
      Node *next
}
isCycle = False
Node *temp1 = *temp2 = head
While(temp1 != head->next && head->next->flag < 2){
      If(temp1 == temp2 && temp1->flag >= 1) {
            isCycle = True
            break
      }
      temp1->flag++
      temp2->flag++
      temp1 = temp1->next
      temp2 = temp2->next
      temp2 = temp2->next
}
If(isCycle == True)print(“This Linked List has a cycle”)
Else print(“This Linked List does not have a cycle”)
```

END

Stack and Queue [20%]

1. What are the main differences between Stack and Queue? (.pdf)

The difference between stack and queue is in stack it has a rule that First In Last Out (FILO) this means when we wanted to pop a stack the one that we can pop is the last one that entered the stack, meanwhile queue has a different rule that is First In First Out (FIFO) this means when we wanted to pop something in queue the one that we will pop is the first node that we entered.

2. Explain prefix, infix, and postfix notation and its implementation using stack! (.pdf)

Prefix is a writing method that write the operator before the number example : * 2 3 = 6
when we are using stack, we will read from left to right, and when we reach a condition of getting 2 numbers in a row we will pop 3 times and push the result to the stack.

Infix is a common writing method that write the operator between the number example: 2 * 3 = 6
when we are using stack, we should convert the infix to postfix first so then we can use it in stack more effectively.

Postfix is a writing method that write the operator after the number example: 2 3 * = 6
when we are using stack we will read the stack from left to right and when we get an operand then we push it into the stack, when we get an operator we pop 2 times and push the result into the stack.

Hashing and Hash Tables [20%]

1. Explain what is a hashtable, hash function, and collision! (.pdf)

Hashtable is a table to store keys that has been hashed, this table can be searched with O(1) time using hash function to form the address from the key.

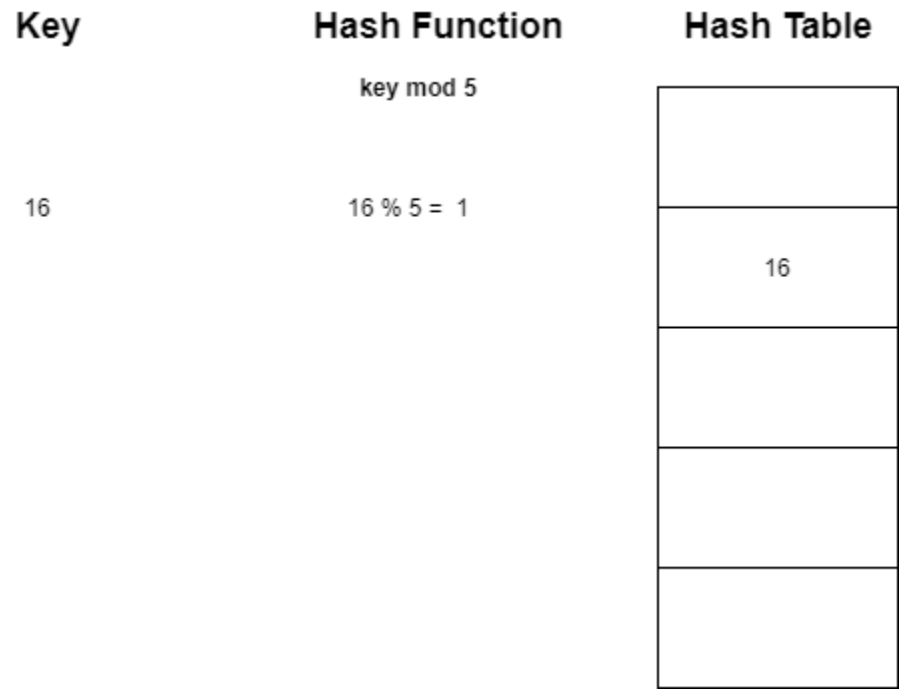
Hash Function is a function that will return address in int when given key, this address will be then used to store the key in hashtable.

Collision is occurred when 2 different keys has same address in the hashtables, when collision happens we should use either open address or chaining to solve the same address problem.

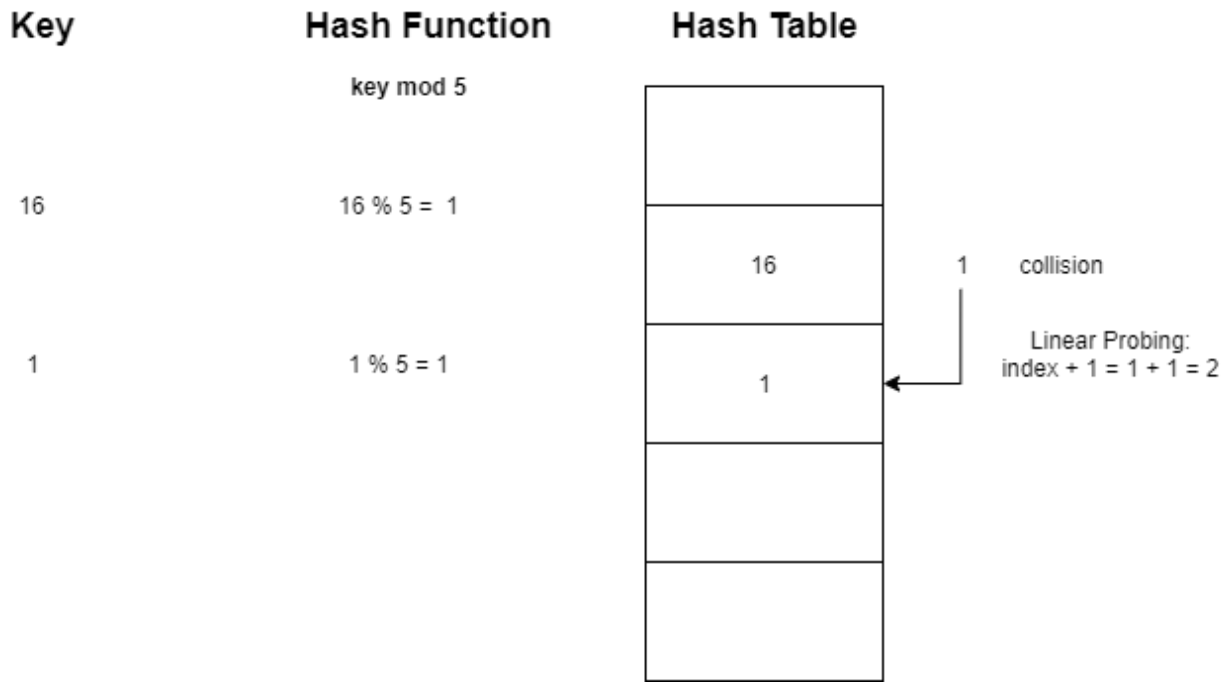
2. Explain 2 methods for collision handling and simulate the process in a graphical view! (.pdf, .png)

Open Address has many method and here I picked linear probing method, linear probing method will gives (index+1) until it found an empty spot if collision happens (hashtables address already has value).

Example for linear probing:

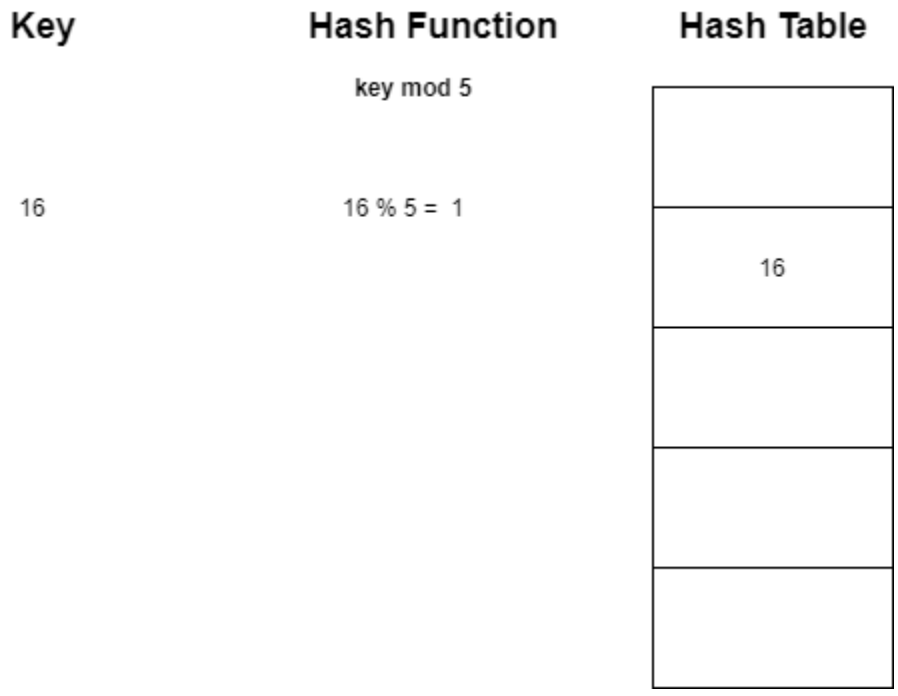


First we give the key to the hash function then it produce the index, 16 % 5 = 1 because the hash tables for index 1 is still empty so no collision happened and key 16 stored in index 1.



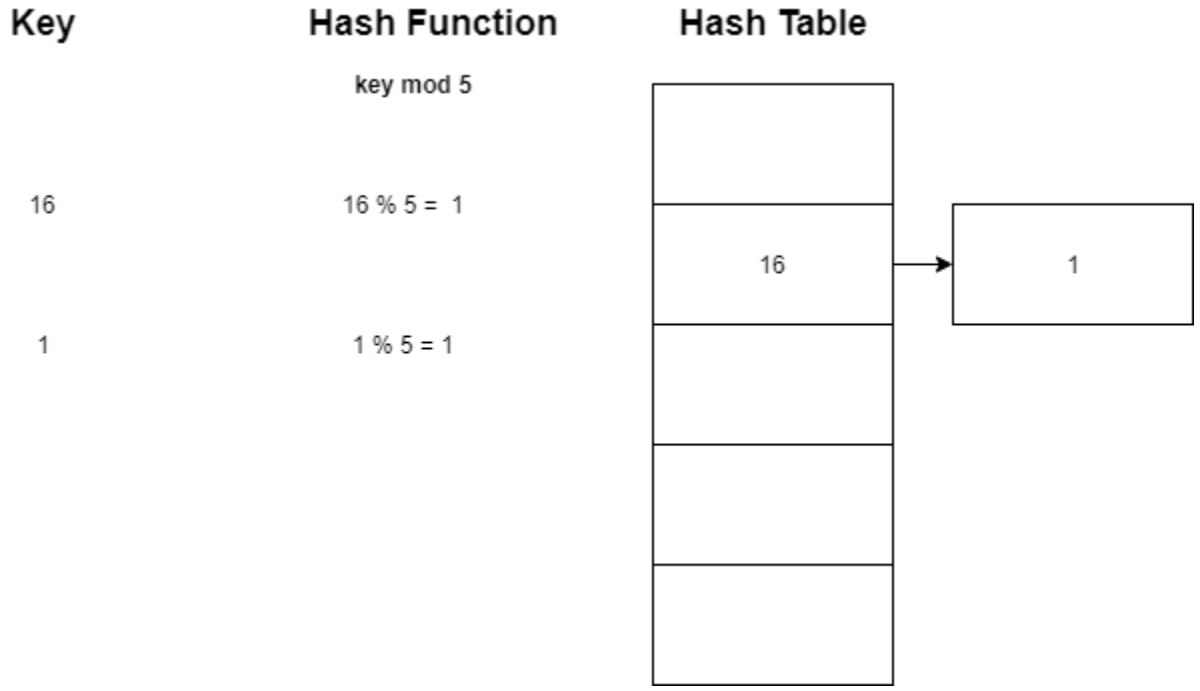
When we want to input key 1, hash function give address of 1 because 1 % 5 = 1, and because in index 1 there is already a key stored, collision happened, and because we are using linear probing so we use index+1 which is 1 + 1 = 2, we check whether index 2 is empty, because index 2 is empty so we put key 1 in index 2.

Example for Chaining:



Same as the previous one

First we give the key to the hash function then it produce the index, 16 % 5 = 1 because the hash tables for index 1 is still empty so no collision happened and key 16 stored in index 1.

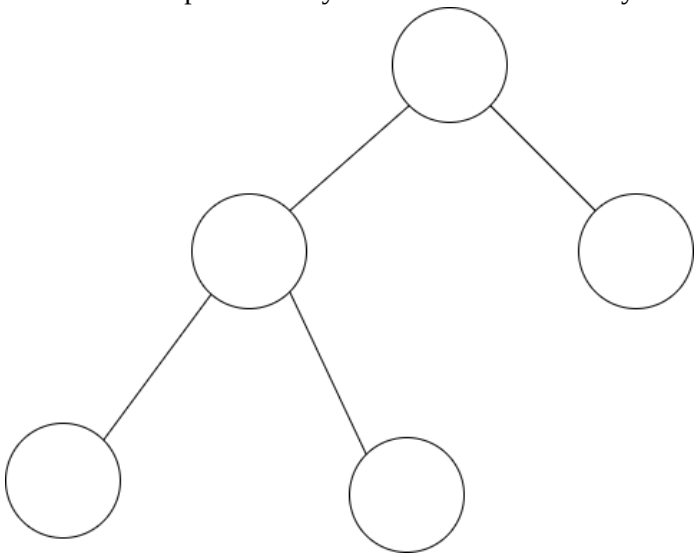


When collision happened this time we create an extension from index 1 so that 1 is remain in index 1 but with a linked list. The head is at 16 and the tail is at 1.

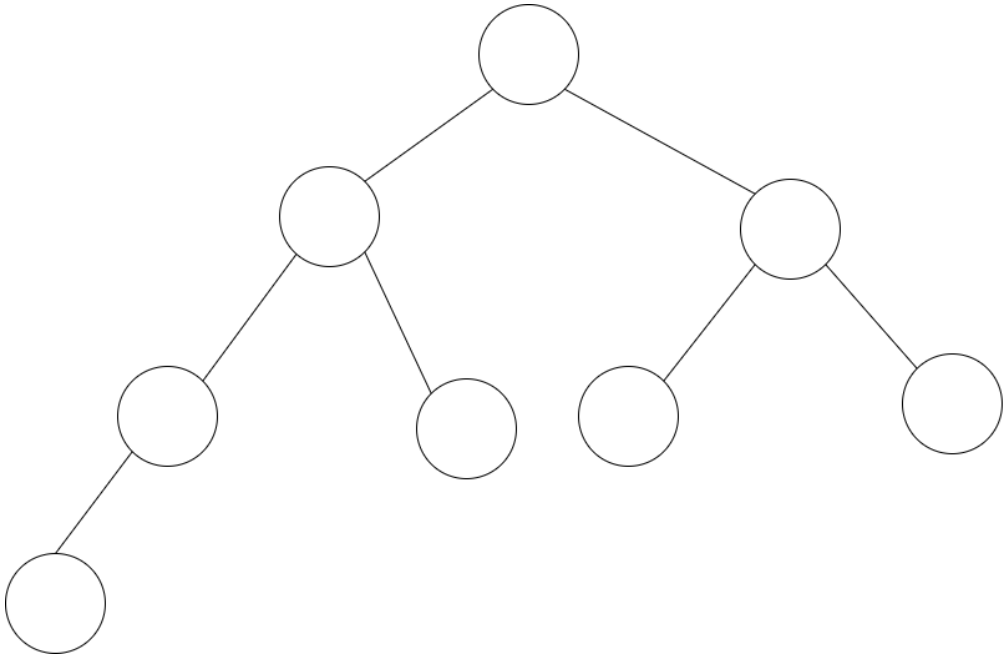
Binary Search Tree [35%]

1. Explain 5 types of Binary Tree and draw each of them! (.pdf)

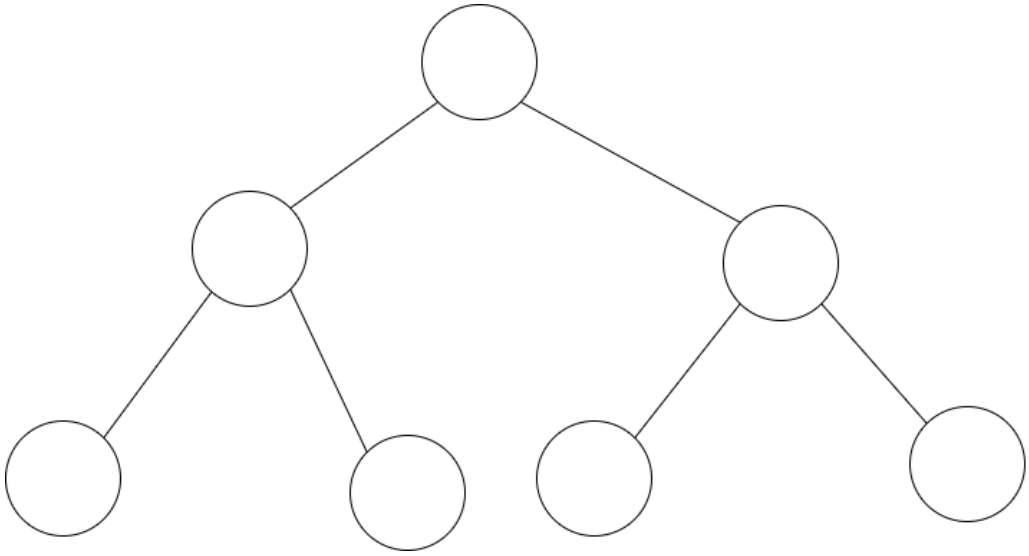
1. **Full Binary Tree**
   This Tree is a special binary tree that each nodes only has zero or two children.



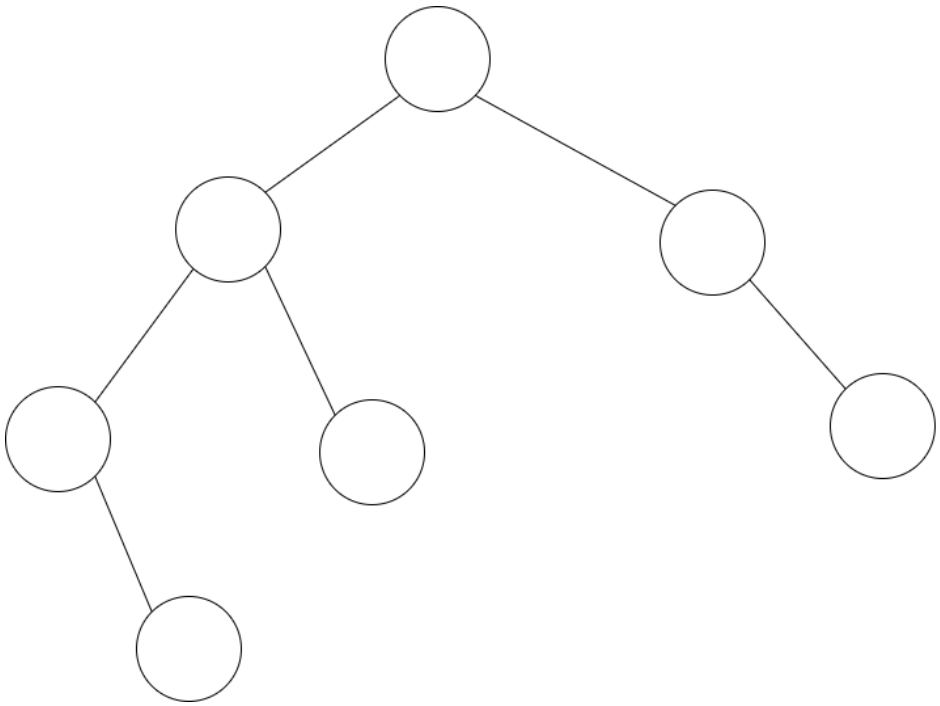
2. **Complete Binary Tree**
   this tree is a special binary tree that each level is filled with nodes except the lowest level (leaf). And in the lowest level of the binary tree every node possibly should reside on the left side of the tree.



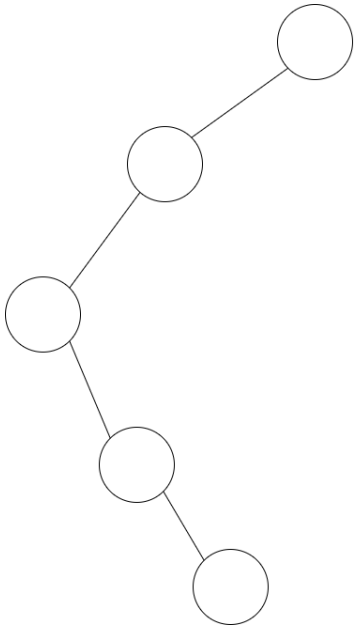
3. **Perfect Binary Tree**
   this tree is a special binary tree which each node have two children, except the leaf. A perfect binary tree that has ‘h’ height has 2h-1 node.



4. **Balanced Binary Tree**
   this tree is a special binary tree which its height is O(logN) where N is the number of nodes. In this tree the height of the left and the right subtree can be vary at most one.

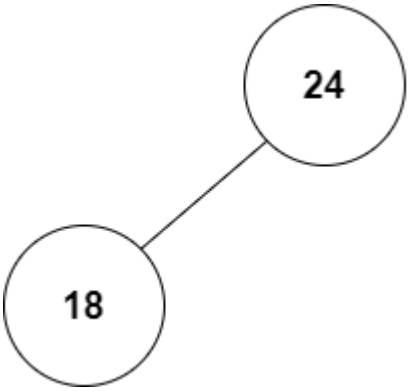


5. **Degenerate Binary Tree**
   this tree is a special binary tree which each node only has one child. This binary tree is similar to linked list.

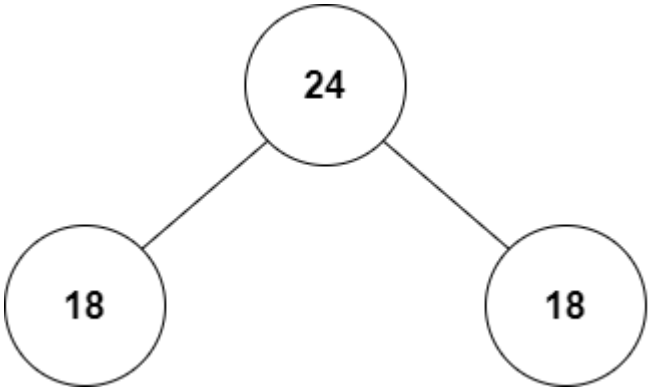


2. Simulate and explain clearly step by step the process of insertion: 24, 18, 55! (.pdf, .png)
First we check if there is head or no, because there is no head then 24 will be the head

After that we insert 18, we check if there is head, because there is already 24 as head then we check whether 18 < 24 because it is true then we check whether 24's left child is empty, because it is empty then 18 will be the left child of 24.

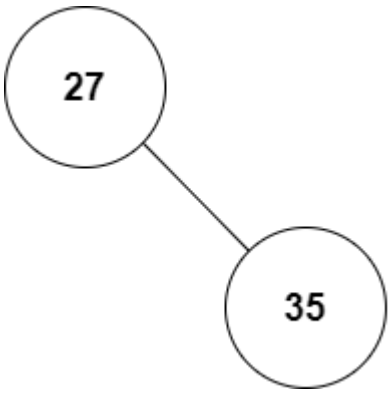


After that we insert 55, first we check whether there is head, because there is 24 as head already then we check whether 55 < 24 because it is false, next we check wheter 55 > 24 because it is true, we check wheter 24's right child is empty because it is empty we place 55 as right child of 24.



3. Simulate and explain clearly step by step the process of deletion: 27, 35, 42! (.pdf, .png)
First we insert 27, we check whether the BST already has head or not, because it does not have head then we place 27 as the head.

After that we insert 35, we check if there is head, because there is already 27 as head then we check whether 35 < 27 because it is false then we check 35 > 27 because it is true then we check whether 27's right child is empty, because it is empty we place 35 as 27's right child.



After that we insert 42. We check if there is head, because there is already 27 as head then we check whether 42 < 27 because it is false we then check 42 > 27 because it is true we check whether the slot for right child is empty or not, because there is already 35 as the right child of 27 then we move to 35, first we check whether 42 < 35 because it is false then we check 42 > 35 because it is true then we place it as 35's right child.