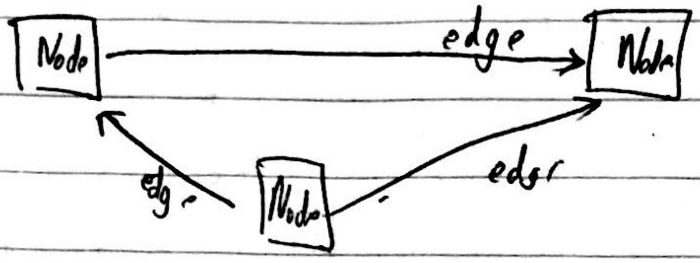
MODULE #9 Graphs

→ Nodes & edges that join nodes.

→ Nodes are objects.

→ Edges connect nodes.

→ Directed graph: edges are arrows.



Trees:

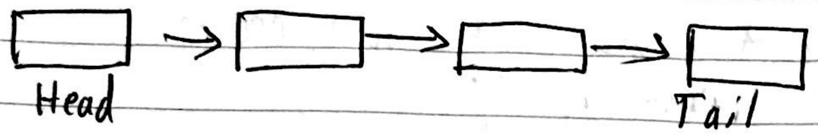
→ One node has no edge in (root)

Binary Trees:-

Tree & atmost 2 edges from any node.

```
class Node<T>{
    T data;
    Set<Node<T>> neighbours;
Node(T data){ this.data = data;}
void addNeighbor(Node<T> node) {}.
void removeNeighbor(Node<T> node) {}.
Arraylist<Node<T>> getShortestPathTo(Node<T> n) {}.
Set<Node<T>> getNeighbors(int radius) {}
static Set<Set<Node>>>
    get Cliques(Set<Node<T>> graph) {}
}
```

LINKED LISTS:-



```
class Node<T> {
  private Node<T> next;
  private T data;

 Node(T data) { this.data = data}
 Node<T> getNext() {return next;}
 T getData() { return data;}
```

```
class LinkedList<T>{
  private Node<T> head;
  private Node<T> tail;
     insertAtHead (T data) {}
     appendAtTail (T data) {}
     size () {}
     delete (T data) {}
}
```

```
1. insertAtHead (T data)
     1. create new node with `data`
     2. Set node's next to head
     3. Set head to new node.
          if tail == null => tail = node // in case list is empty.

2. appendAtTail (T data)
     1. create node w/ data
     2. If tail != null
          2.1- Set tail's next to `node` (Not empty LL)
     3. Else.
          head = node (empty Linked List)
     4. set tail to new node
```

```
3. size() {
      if head == null        ] ??
          return 0.          ]
     int n = 0;
     Node<T> node = head.
    while (node != null) {
           n++.
            node = node.next;
       }.

     return n;
  }.
```

```
4. delete(T data){.
     if (head == null)              ] Empty
        return.                     ]
  else if (head == tail ~~&&~~ head.getData()
                             .equals(data)).     } Size = 1.   ∴ head = tail
     head = null                                 }             & data is in
     tail = null                                 }             head.

elseif (head.getData().equals(data))   ] // delete at head.
     head = head.next
else.                                     General vanilla case.
```