Data Structure

A data structure is a way of organizing and storing the data so operations like updating, searching ,deleting can be done efficiently

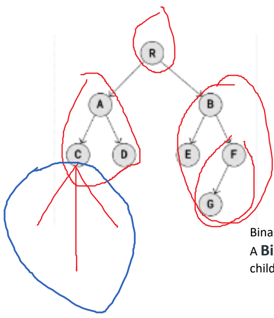
Why efficiency?

- Time complexity
- Memory usage
- CPU operations

Common Data Structures

Array	Sequential data
Linked list	Dynamic data
Stack	Undo/back operations (LIFO)
Queue	Scheduling (FiFo)
Hash Table	Quick lookup
Trees	Hierarchal data
Graphs	Network

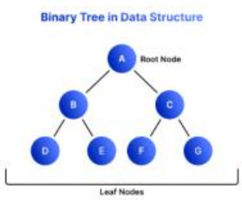
Tree Data Structure : **Tree can be defined as a collection of entities(Nodes) liked together to simulate hierarchy**

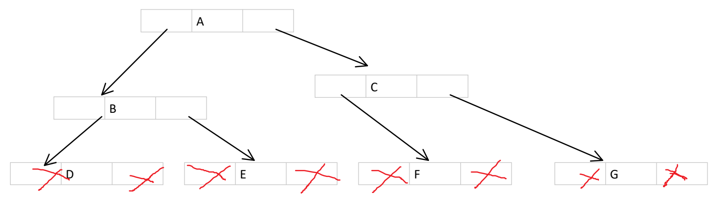


Root : top most node in a tree doesn't have any parent R
Node : its also known as element (it can be able to store information or data , R,A,B,C,D,E,F,G
Parent Node: a node that is an immediate predecessor of another node ...parent of E=B
Child node : a node that is an immediate successor of another node B= E,F
Leaf node :node that doesn't have any children , C,D,E,G,
Non leaf node : the node which contains child node ,R,A,B,F
Path: sequence of consecutive edges from source node to destination node
Ancestor: any predecessor node on the path from root to the node
Descendant: any successor node on the path from that node to leaf node
Siblings : the node that can share the same parent
Level of node :the number of edges in the path from the root to that node . Root node level is always 0
Levl=height

Binary tree

A **Binary Tree Data Structure** is a hierarchical data structure in which each node has at most two children, referred to as the left child and the right child.





Maximum number of children nodes 2
No rule for ordering to insert elements
Depth can be grow large
Not necessarily balanced
Search : O(n)
Insert/delet o(n)

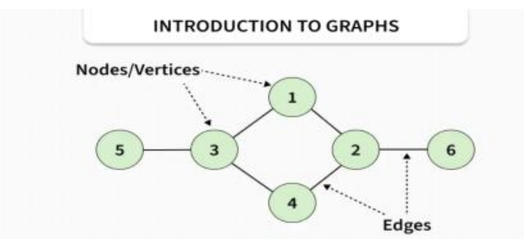
Trees are always non-linear data structures:
Data is in a tree is not stored in sequentially ,instead it is organized across multiple levels , forming a hierarchical structure . Because of this arrangement a tree is classified in to linear data structure

Representation of a node using collection of nodes .each node can be represented with help of class or structs

```
Class node :
    Def __init__(self,x):
        Self.data=x
        Self.chikdren =[]
```

Graphs data structure:

A graph is a non-linear data structure made up of vertices (node) and edges (connection) that represent relationship between the objects .



Components :

Vertices :vertices are the fundamental units of the graph

Edges: Edges are drawn or used to connect two nodes of the graph .

Types of Graphs:

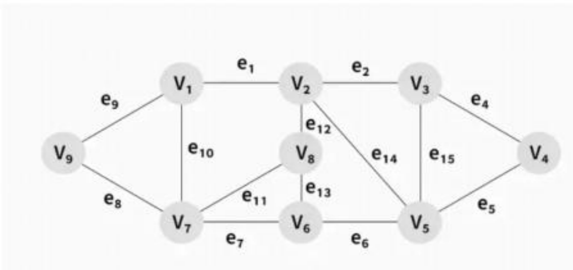
Based on the different properties we can divide the graphs

Such as edges,directions,connectivity and more …..

Based on weight :

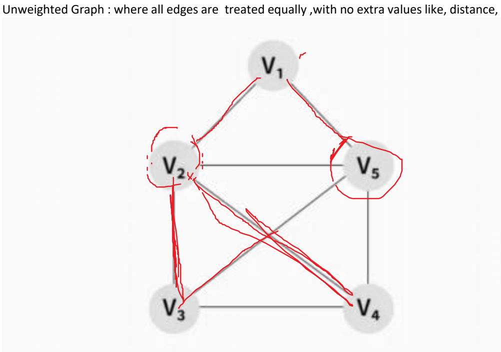
Weighed graphs and unweighted graphs

1. Weighted graph :A weighted graph is a graph where each edge has a number(weight) that represents distance,time,cost.



Example; travelling from source to destination (we are having distance in kM, time in mi/hr)

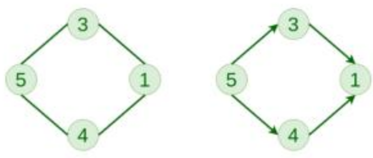
2. Unweighted Graph : where all edges are treated equally ,with no extra values like, distance, cost



Based on edges we have divided in to directed and undirected graph :

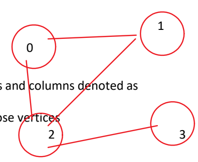
Undirected graph: a graph in which edges do not have any direction the nodes are ordered pairs in the definition of every edge

Directed graph : a graph in which edges are having direction



Undirected Graph Directed Graph

Representation of graphs



Adjacency matrix : graph is stored in the form of 2D matrix where rows and columns denoted as columns .

Each entry in the matrix represents the weight of the edge between those vertices

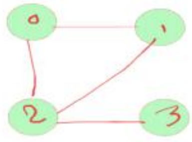
Matrix[i][j] =1 if there is an edge between vertex I and vertex j

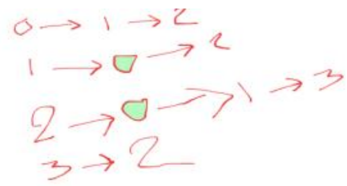
Matrix[i][j]=0 if there is no edge

	0	1	2	3
0	0	1	1	0
1	1	0	1	0
2	1	1	0	1
3	0	0	1	0

Adjacency list representation of a graph

This graph represented as collection of array list .there is an array of pointer which points to the edges connected to the vertex





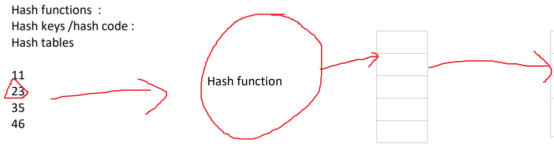
[0,1,2]
[1,0,2]
[2,0,1,3]
[3,2]

Applications:
1. Social networks
2. Computer networks
3. Transportation network
4. Neural network;-
5. Compliers
6. Network optimizations

Hashing Table:
It is a technique in data structure to improve efficiency in store and retrieve the data
-- the data is stored in the form of key-value pair

Elements : values
Hash functions :
Hash keys /hash code :
Hash tables

11
23
35
46



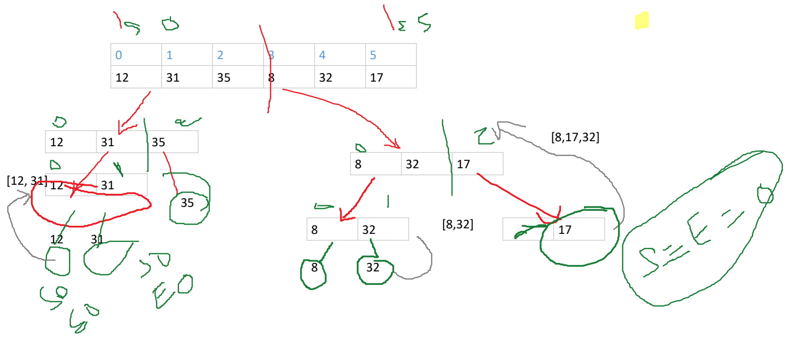
index	value
0	11
1	23
2	35

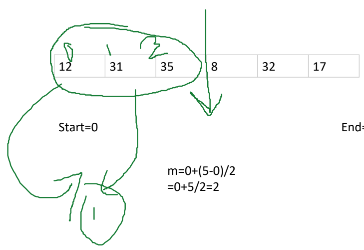
Hash key/hash code

Applications:
Login systems ,
Dictionary loops
Caching
Compliers
Searching phone contact

Merge sort:
1 divide the arrya(mid)
2 divide the Same process until single element is remains
3. Merge parts to create sorted array

0	1	2	3	4	5
12	31	35	8	32	17





Start=0

End=5

m=0+(5-0)/2
=0+5/2=2

12>8
12>17

[12,31,35] [8,17,32]

12	31	35	8	17	32

8	12	17	31	32	35

TEMP:

Pseudo code:

Mergesort(A,left, right)

If(left < right
    mid=left+(right-left)/2
    Mergesort(A,left ,mid) ...left
    Mergesort(A,mid+1,right) --right

Merge(A,Left , mid,right)

Merge (A, left, mid ,right)

create an empty Array temp
i= left
j=mid+1
k=0

while i <=mid and j< =right)

    if A[i] <= A[j]
    temp[k]=A[i]
    i=i+1;

      else :
    temp[k]=A[i]
    j=j+
k=k+

while i<=mid
    temp[k]=A[i]
    i=i++
    k=k+1

while j<= right;
    temp[k]=A[i]

j=j+1
k=k+1

for f=0 to k-1
A[left+ f] =temp[x]

Quick Sort :

10	**15**	**1**	**2**	**9**	**16**	**11**

PIVOT -->10

PARTITION 1 VALUES < PIVIOT	PIVIOT	PARTITION 2 VALUES > PIVOT

2	1	9	10	15	11	16

0---->2
4----> 6

PIVOT= 7

7	6	10	5	9	2	1	15	7
Start								end

7<=pivot=7
6<=pivot=7
10<=pivot=7 false

7	6	10	5	9	2	1	15	7
		Start						end

7> pivot=7 false
Swap end value start with end

7	6	7	5	9	2	1	15	10
		Start						end

7<=pivot=7 if true move start point
5<=piovt=7 if true
9<=pivot=7 false

7	6	7	5	9	2	1	15	10
				Start		end		

10> pivot=7 true
15> pivot=7
1>piovt=7 false

7	6	7	5	1	2	9	15	10
				Start		end		

7	6	7	5	1	2	9	15	10
						start end		
7	6	7	5	1	2	9	15	10
					end	start		

1<=pivot=7
2<=pivot=7
9<=pivot=7 false

9> 7
2>7 we canot do the swap because
starting index is greater than the ending index.
Swap pivot element with end value

2	6	7	5	1	7	9	15	10
						start		

Tim complex = O(n^2)
=O(log n)

```
Partition(A, lb ,ub)

        Pivot =A[lb]
        start=lb;
        end =ub ;

while(start <end)
        {
                while( A[start] <= pivot)
                        {
                        start ++ ;
                        }
        while ( A[end]  > pivot )
                {
                end - - ;
                }
if (start  <  end)
        {
        swap(A[start] ,A[end])
        }
}

swap(A[lb] ,A [end]) ; return end

Quicksort( A, Lb, Ub)
        if( lb < ub)
{
        loc=partion(A, lb,bb)
quicksort(A, lb, loc-1);
quicksort (A, loc+1, ub);

}
```