1.1 Explain in detail about red black tree with an example. Unit 3 Pg 17

Red Black Trees :-

Red black tree is a binary search tree in which every node is colored with either red or black. The red-black tree satisfies all the properties of binary search tree.

1) Root property :- The root is black.

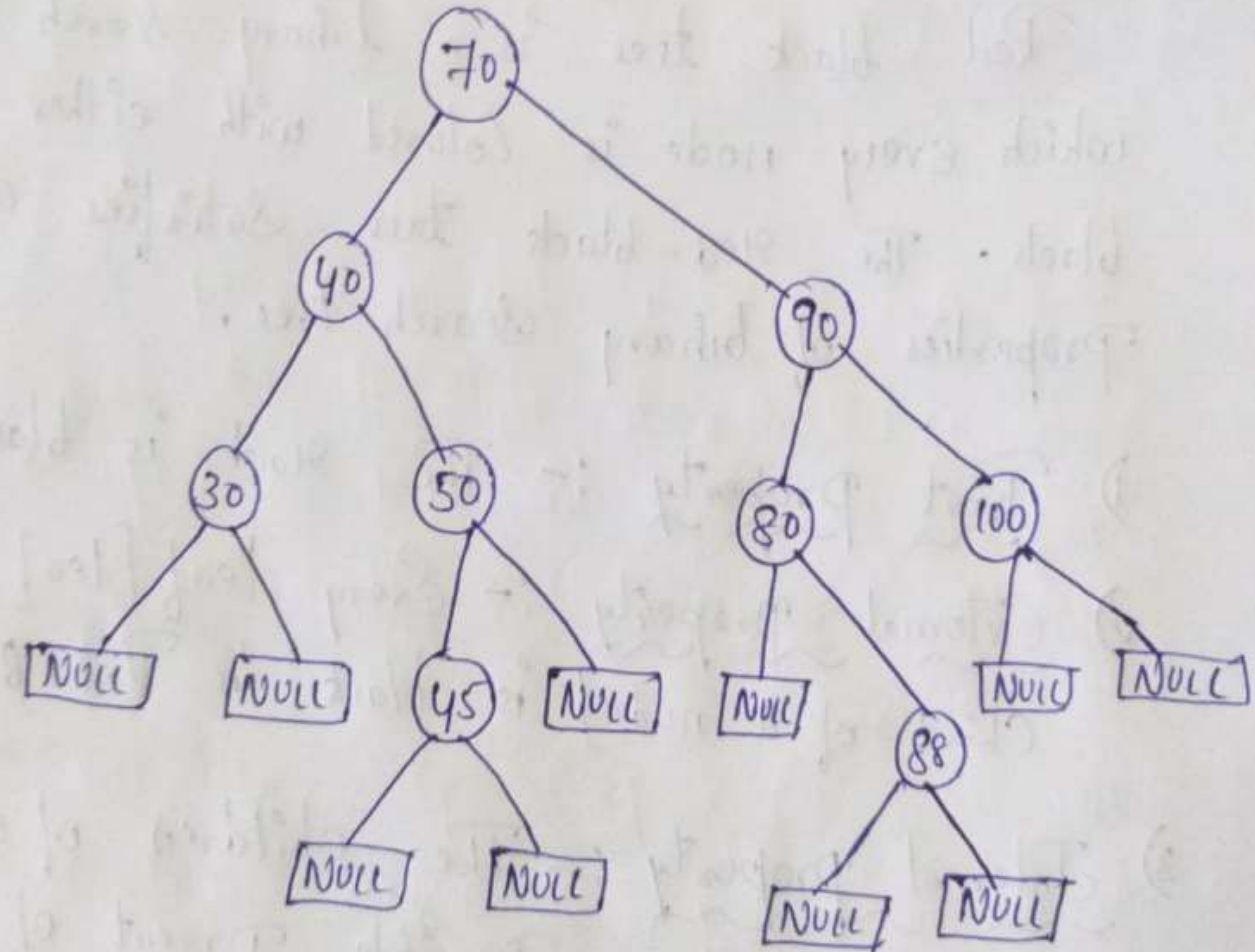
2) External property :- Every leaf [leaf is a Null child of a node] is black in Red-Black tree.

3) Internal property :- The children of a red node are black. Hence possible parent of red node is a black node.

4) Depth property :- All the leaves have the same black depth.

5) Path Property :- Every simple path from root to leaf node contains same number of black nodes.

Representation of Red-Black tree



1. It is a binary search tree.
2. The root node is black.
3. The children of red node are black.
4. No root - to external node path has two consecutive red nodes [e.g. 70 — 90 — 80 — 88 — NULL].

(Cont...d)

1.2 What is a graph? Explain various representations of graphs. Unit 4 Pg 1

Graph :

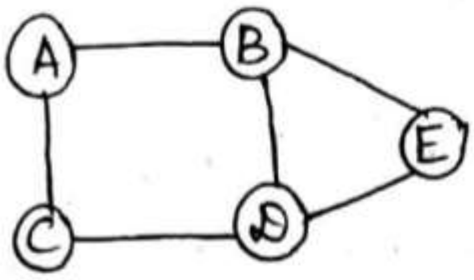
A graph is a non-linear datastructure consisting of collection of nodes and edges. The nodes are referred to as Vertices and edges are lines that connects pair of Vertices.

Generally, a graph G is represented as $G=(V,E)$.

$V \rightarrow$ Set of vertices

$E \rightarrow$ set of Edges

Eg:.



$\rightarrow$ fig (i)

Here, $G=(V,E)$

$V=\{A,B,C,D,E\}$

$E=\{(A,B),(A,C),(B,E),(C,D),(D,E),(B,D)\}$

A graph may have cycles.

Graph Representation:

Graphs are generally represented in the following scheme as,

1. Adjacency matrix representation
2. Adjacency list representation
3. Incidence matrix representation.

1. Adjacency matrix representation:

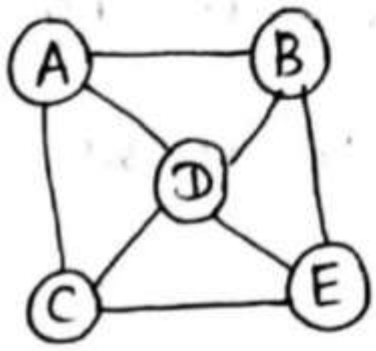
One simple way to represent a graph is to use 2-dimensional array., ie., using a matrix of size V×V, V-rows and V-columns. Here both rows and columns represents vertices and the value of matrix is either 0 or 1.

1 - if there exists an edge b/w vertices
0 - if there is no edge between vertices.

Adjacency matrix is given for both directed and undirected graphs respectively.

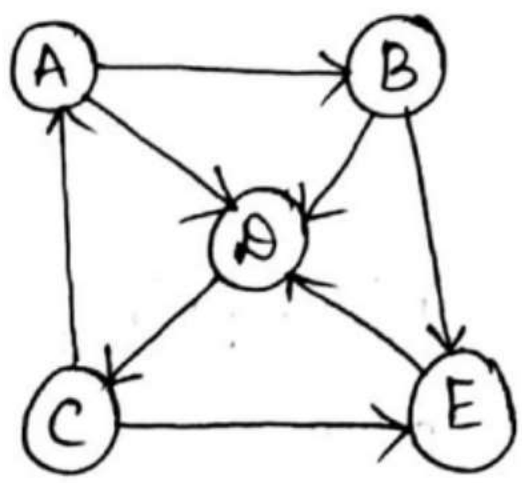
Eg:

Undirected graph



	A	B	C	D	E
A	0	1	1	1	0
B	[redacted]	0	0	1	[redacted]
C	1	0	0	1	1
D	1	1	1	0	1
E	0	1	1	1	0

Directed graph



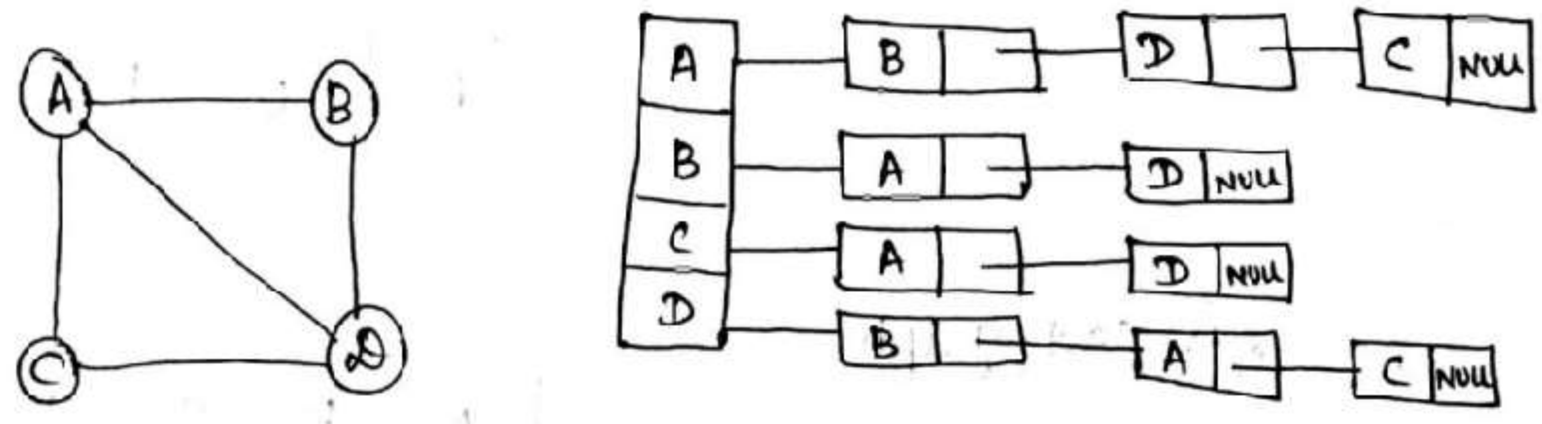
	A	B	C	D	E
A	0	1	0	1	0
B	0	0	0	1	1
C	1	0	0	0	1
D	0	0	1	0	0
E	0	0	0	1	0

In directed graph, the cell value AB is 1 because there is an edge from A to B, but on the other side cell value of BA is 0.

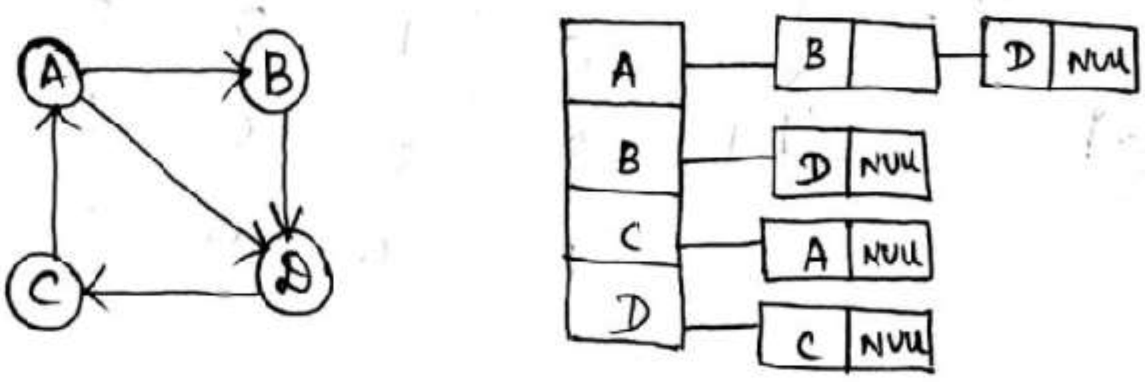
2. <u>Adjacency List representation</u>:

In this representation, graph is stored as a linked structure. If a node has any adjacency to any other node, then the adjacent node will be linked with its predecessor by storing it's address in the link field of predecessor node.

<u>Undirected graph</u>:



<u>Directed graph</u>:



3. <u>Incidence Matrix representation</u>:

In this representation, a graph $G(V, E)$ with V vertices and E edges can be represented using a matrix of size $V \times E$ i.e., V rows & E columns.

This matrix is filled with either 0, 1, or -1.

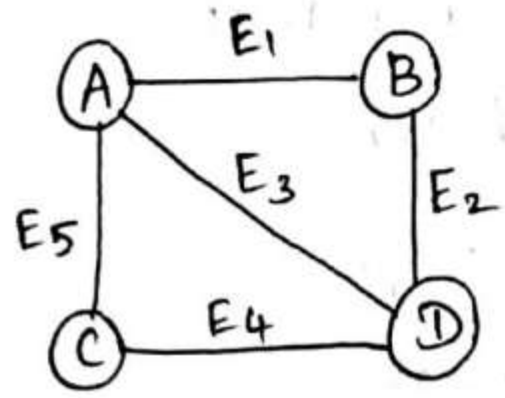
0 → There is no edge between Vertices for both directed & undirected graphs

1 → There is an edge between Vertices for both directed & undirected graphs.

-1 → If there is an incoming edge from the row Vertex to the column Vertex, in directed graphs.

For self-loop edges, only <u>1</u> is used to represent both incoming and outgoing edge.

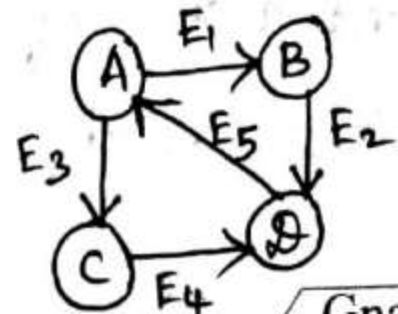
<u>undirected graph:</u>



	E_1	E_2	E_3	E_4	E_5
A	1	0	1	0	1
B	1	1	0	0	0
C	0	0	0	1	1
D	0	1	1	1	0

<u>Directed graph</u>:

	E_1	E_2	E_3	E_4	E_5
A	1	0	1	0	-1
B	-1	1	0	0	0
C	0	0	-1	1	0
D	0	-1	0	-1	1



1.3 Explain BFS and DFS algorithm Unit 4 Pg 11

There are two major graph traversal techniques such as,

* Depth First Search
* Breadth First Search.

<u>Depth First Search</u>:

It produces spanning tree as final result. spanning tree is a graph without any loops. In dfs, starting at some vertex v, we process v then recursively traverse all vertices adjacent to v. To do this we mark a vertex v as visited once we visit it. We use stack data structure with maximum size of vertices in the graph to implement DFS traversal.

The graph is represented using adjacency list method. Hence the node of the adjacency list is defined as,

```
Struct node
{
  Struct node *next;
  int Vertex;
};
typedef struct node *GNODE;
```

The graph array is,

```
GNODE graph[20];
```

The visited array is,

```
int visited[20];
```

DFS(int i):

```
Void DFS(int i) {
        Declare a Var p of type GNODE.
         Print %d
         Assign graph[i] to p.
        Set visited[i]=1
          while ( P!=NULL) {
             Assign Vertex of p to i
            if ( visited[i]!=1)
            { DFS(i); }
             Assign next of P to P
     } }
```

s

Breadth First Search:

BFS traversal produces a spanning tree as a final result. We use Queue data structure with maximum size of number of vertices in the graph to implement BFS traversal. Here, a node is selected randomly as the start position. Starting from that node, all of its unvisited nodes are visited. This Process is done recursively until all the nodes are visited.

```
int Visited [20];
```

The queue array with front and rear Variables are,

```
int queue [99], front=-1, rear=-1;
```

BFS (int V):

```
Void BFS(int V)
 {  declare a var W.
 Call: InsertQueue (V0)
      { V= dequeue(V);
         Print the Vertex value }
     Set visited [V]=1.
GNODE g=graph[v];
      for (; g!=NULL; g=g->next)
  {
  assign Vertex of g to W
        if(visited [W]==0) { Call: InsertQueue(w); visited[W]=1;
        Print W; }}} print("\n"); }
```

1.4 Describe representation of graphs in Data structures Unit 4 Pg 6

Graph Representation:

Graphs are generally represented in the following scheme as,

1. Adjacency matrix representation
2. Adjacency list representation
3. Incidence matrix representation.

1. Adjacency matrix representation:

One simple way to represent a graph is to use 2-dimensional array., ie., using a matrix of size V×V, V-rows and V-columns.

Here both rows and columns represents vertices and the value of matrix is either 0 or 1.

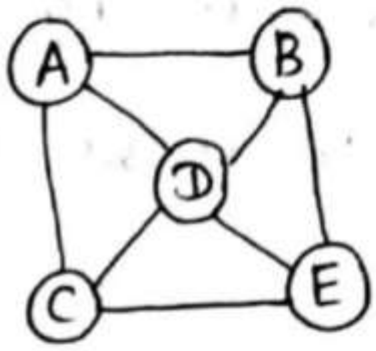
1 - if there exists an edge b/w vertices

0 - if there is no edge between vertices.

Adjacency matrix is given for both directed and undirected graphs respectively.

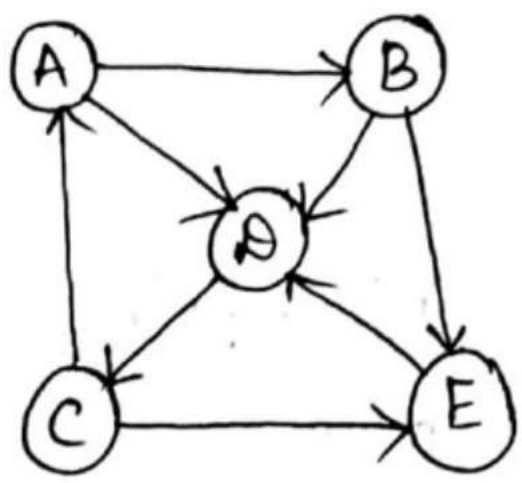
Eg:

Undirected graph



	A	B	C	D	E
A	0	1	1	1	0
B	[illegible]	0	0	1	[illegible]
C	1	0	0	1	1
D	1	1	1	0	1
E	0	1	1	1	0

Directed graph



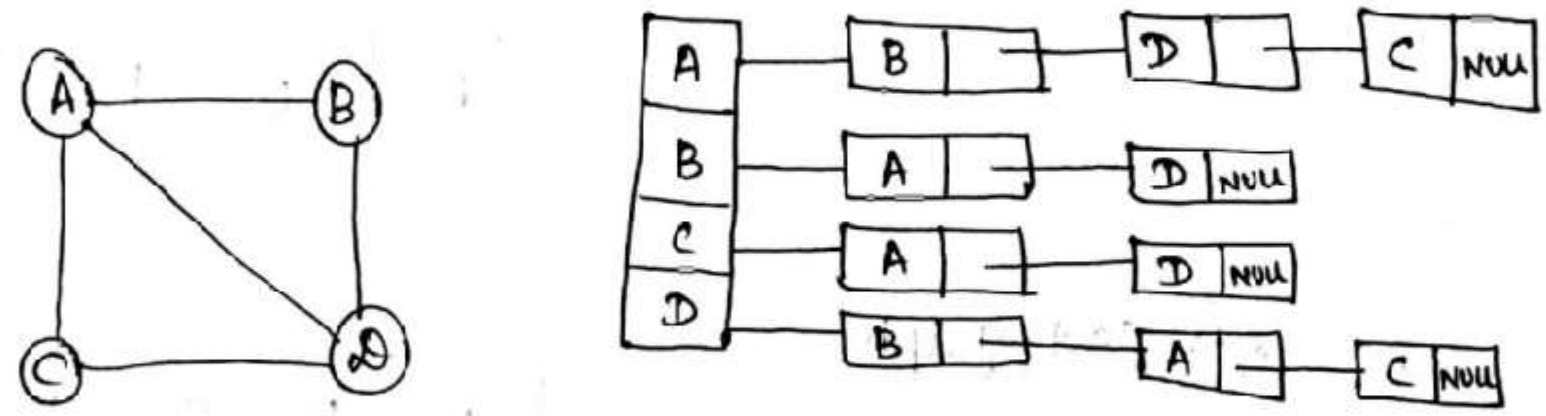
	A	B	C	D	E
A	0	1	0	1	0
B	0	0	0	1	1
C	1	0	0	0	1
D	0	0	1	0	0
E	0	0	0	1	0

In directed graph, the cell value AB is 1 because there is an edge from A to B, but on the other side cell value of BA is 0.

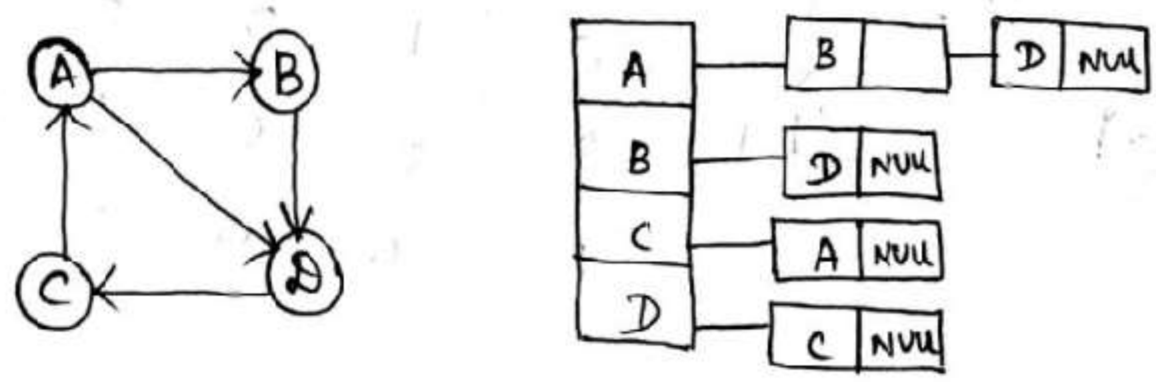
2. Adjacency List representation:

In this representation, graph is stored as a linked structure. If a node has any adjacency to any other node, then the adjacent node will be linked with its predecessor by storing it's address in the link field of predecessor node.

Undirected graph:



Directed graph:



3. Incidence Matrix representation:

In this representation, a graph G(V, E) with V vertices and E edges can be represented using a matrix of size V×E i.e., V rows & E columns.

This matrix is filled with either 0, 1, or -1.

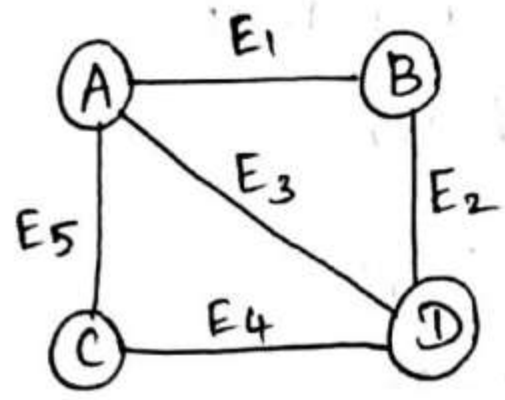
0 → There is no edge between vertices for both directed & undirected graphs

1 → There is an edge between vertices for both directed & undirected graphs.

-1 → If there is an incoming edge from the row vertex to the column vertex, in directed graphs.

For self-loop edges, only <u>1</u> is used to represent both incoming and outgoing edge.

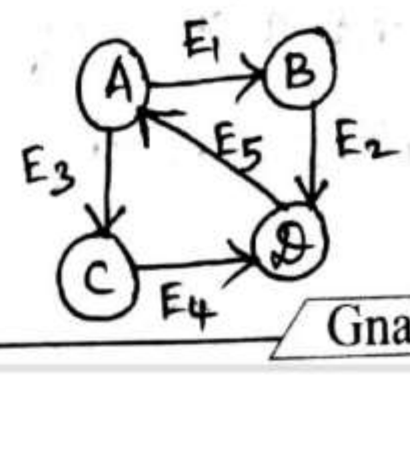
<u>undirected graph:</u>



	E_1	E_2	E_3	E_4	E_5
A	1	0	1	0	1
B	1	1	0	0	0
C	0	0	0	1	1
D	0	1	1	1	0

<u>Directed graph:</u>

	E_1	E_2	E_3	E_4	E_5
A	1	0	1	0	-1
B	-1	1	0	0	0
C	0	0	-1	1	0
D	0	-1	0	-1	1



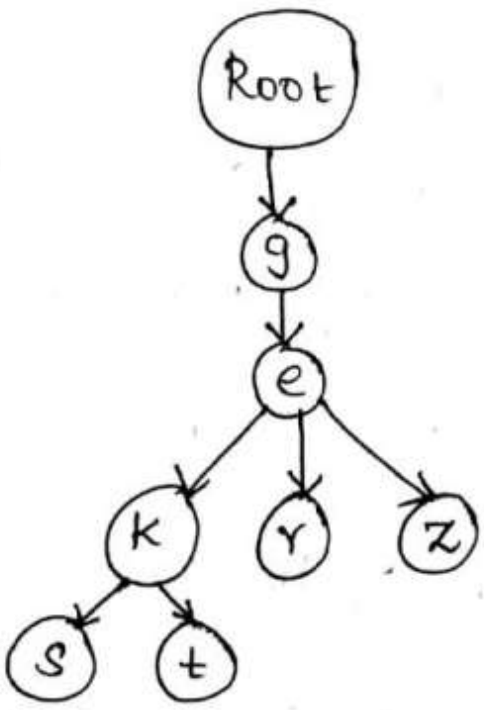
1.5 Write a note on TRIES Unit 5 Pg 14

Trie :-

A tree like data-Structure wherein the nodes of the tree Store the entire alphabet and strings/words can be retrieved by traversing down a branch path of the tree.

It is an efficient information retrieval data Structure. Search Complexity Can be brought to optimal limit (key length). Thus, it is O(M) time, where M is the maximum String length.

Eg:



Every node has multiple branches. Each branch represents a possible character of keys. The last node of every key should be marked as end of word node.

Implementation:

If two strings have a common prefix of length n then these two will have a common path in the trie tree till the length n.

```
typedef struct trie_node
{
  bool NotLeaf;
  trie_node *pchildren[NR];
  Var_type word[20];
} node;
```

where,

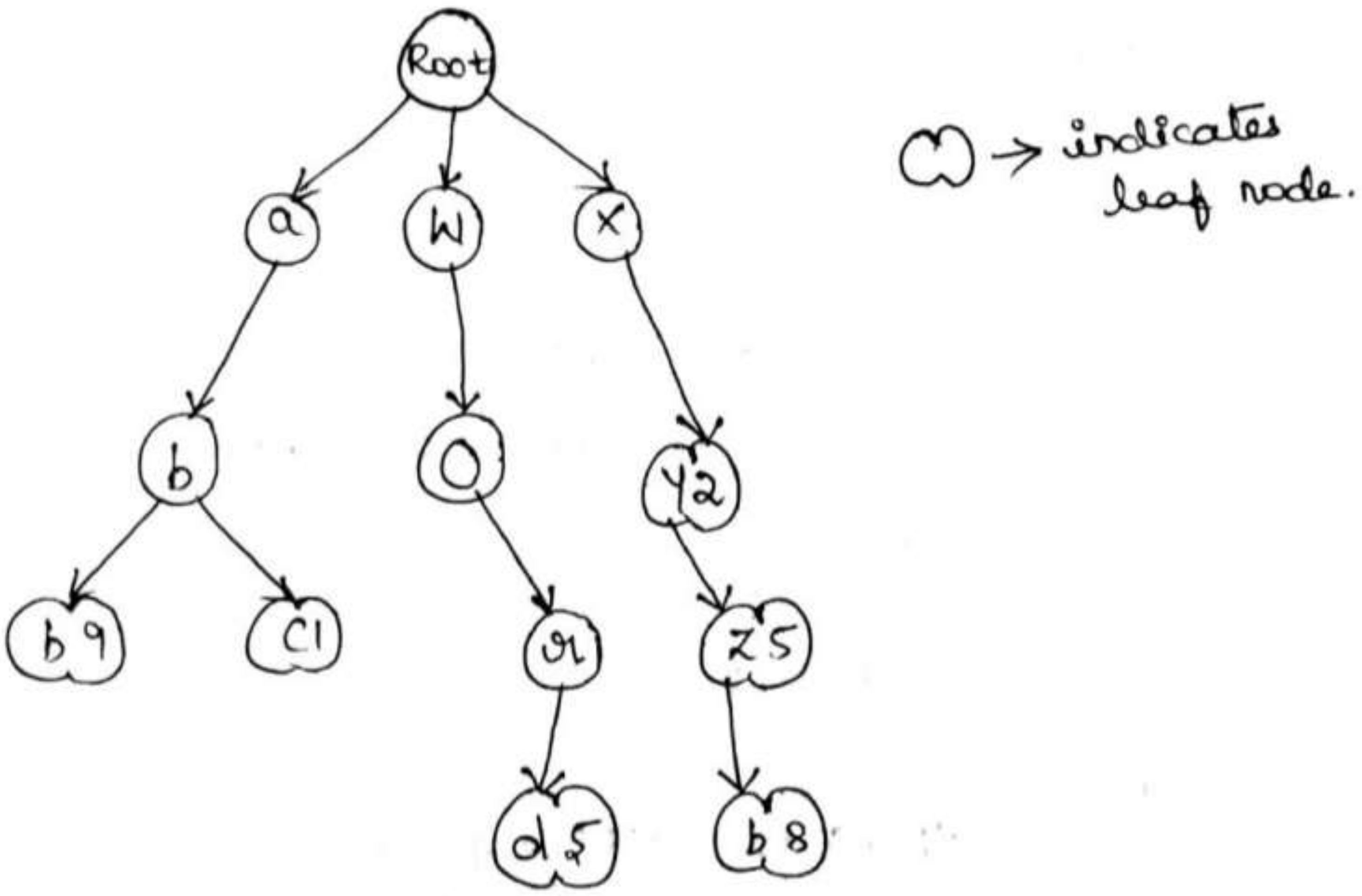
```
#define NR 27 (26 letters + blank)
Var_type -> char (Set of characters).
```

Operations:

* Insertion
* Deletion
* Searching
* Traversing.

Trie data structure for key, value pairs,

("abc", 1), ("xy", 2), ("xyz", 5), ("abb", 9), ("xyzb", 8), ("word", 5).



1.6 Explain about Boyer-Moore algorithm in detail. Unit 5 Pg 4

Boyer- Moore Algorithm:

This is the most efficient String-matching algorithm because it works fastest when the alphabet is moderately sized and the pattern is relatively long.

The algorithm scans the characters of the pattern from right to left beginning with rightmost character. During the testing of a possible placement of pattern P against text T, a mismatch of text character T[i] = c with the corresponding pattern character P[j] is handled as follows: If c is not contained anywhere in P, then shift the pattern P completely past T[i]. Otherwise, shift P until an occurrence of character c in P gets aligned with T[i].

This technique likely to avoid lot of needless comparisons by significantly shifting pattern relative to text. For deciding the possible shifts, it uses two preprocessing strategies simultaneously, to reduce the search.

* Bad character Heuristics
* Good Suffix Heuristics.

Bad character Heuristics :-

The idea is simple that the character of the text which doesn't match with the current character of the pattern is called the bad character.

Upon mismatch, we shift the pattern until,

a) The mismatch becomes a match

b) Pattern P move past the mismatched character.

Case 1: Mismatch become match.

Lookup the position of last occurrence of mismatching character in the pattern and if mismatching character exist in pattern then we will shift the pattern such that it get aligned to the mismatching character in text T.

Eg:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
G	C	A	A	T	G	C	C	T	A	T	G	T	G	A
T	A	T	G	T	G									

Case 2: pattern move past the mismatch character.

Lookup the position of last occurrence of mismatching character in pattern and if character does not exist we will shift pattern past the mismatching character.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
G	C	A	A	T	G	C	C	T	A	T	G	T	G	A
		T	A	T	G	T	G							

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
G	C	A	A	T	G	C	C	T	A	T	G	T	G	A
								T	A	T	G	T	G	

Good Suffix Heuristics:

Let t be substring of text T which is matched with substring of pattern p. Now, shift pattern until:

1) Another occurrence of t in P matched with t in T.
2) A prefix of P, matches with suffix to 't'
3) P moves past it.

Case 1: Another occurrence of t in p marched with t in T.

Pattern p might Contain few more occurrences of t. In such Case, we will try to shift the pattern to align that occurrence with t in text T.

Eg:

0	1	2	3	4	5	6	7	8	9	10
A	B	A	A	B	A	B	A	C	B	A
C	$\boxed{A}$	$\boxed{B}$	A	B						

Case 2: A prefix of P, which matches with suffix of t in T.

Sometimes, there is no occurrence of t in P at all. In Such cases we can Search for some suffix of t matching with some prefix of P and try to align them by shifting P.

Eg:

0	1	2	3	4	5	6	7	8	9	10
A	A	B	$\boxed{A}$	$\boxed{B}$	A	B	A	C	B	A
$\boxed{A}$	$\boxed{B}$	B	A	B						

Case 3: p moves past t.

If two cases above are not satisfied, we will shift the pattern past the t.

Eg:

0	1	2	3	4	5	6	7	8	9	10
A	A	C	A	B	A	B	A	C	B	A
			↓	↓						
C	B	A	A	B						

0	1	2	3	4	5	6	7	8	9	10
A	B	A	A	B	A	B	A	C	B	A
					C	B	A	A	B.	

Here, we can never find any perfect match before index 4, so we will shift the p past the t.

2.1 Write short note on splay tree rotations and with an example. Unit 3 Pg 34/42

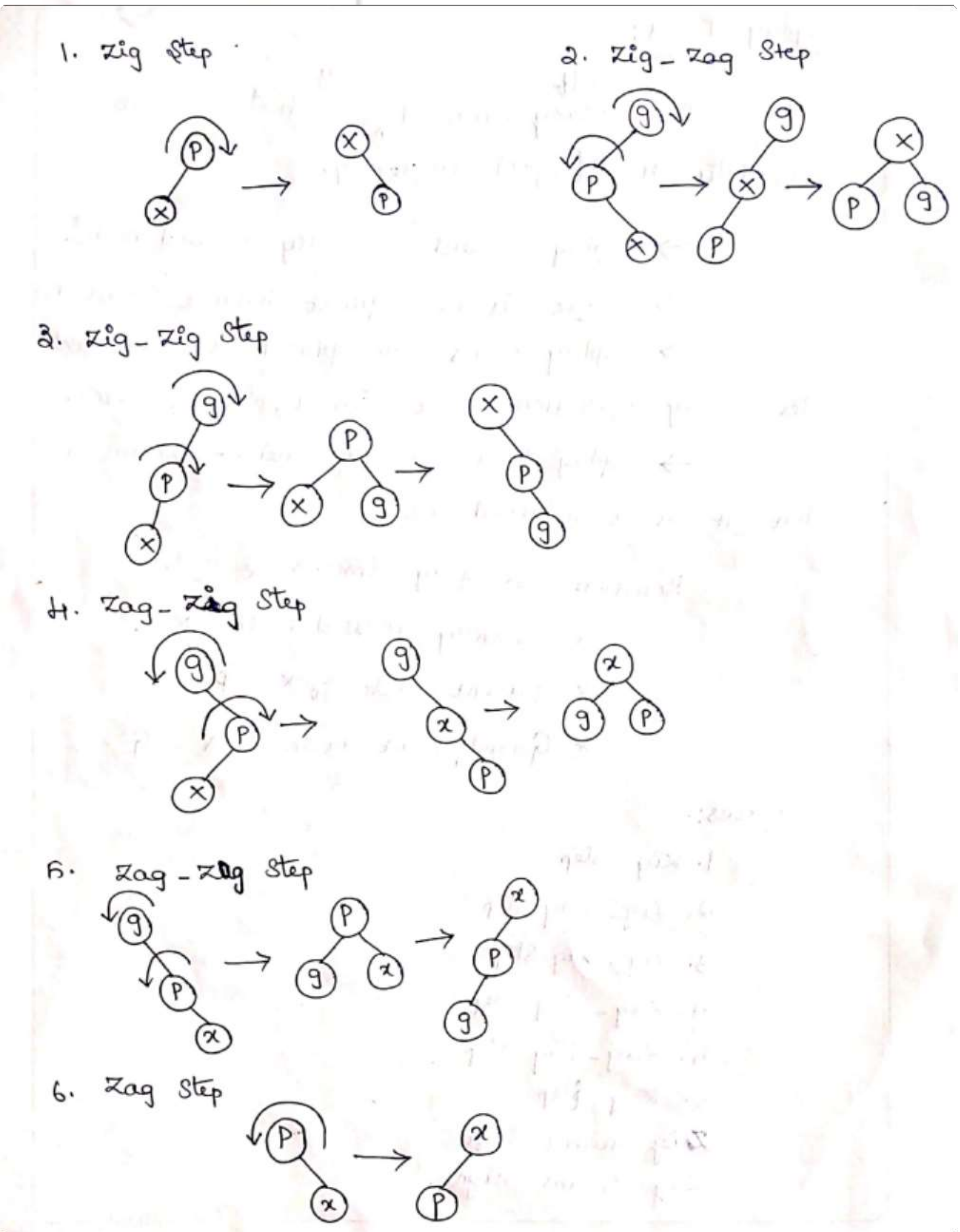
Rotation in Splay involves 3 nodes,

* recently accessed node – x
* parent node of x – P
* Grand parent node of x – G

Cases:-

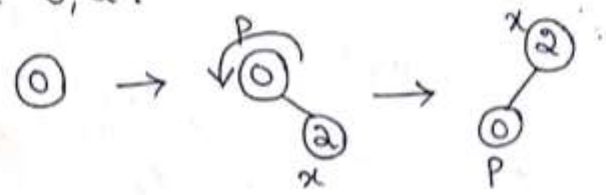
1. Zig step
2. Zig- Zag step
3. Zig- zig step
4. Zag- Zag step
5. Zag- Zig step
6. Zag step

Zig means left
Zag means right.

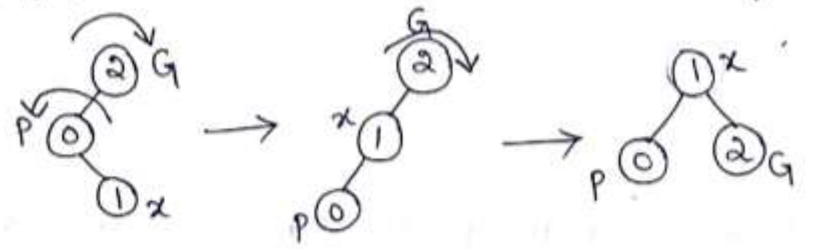


Insertion:

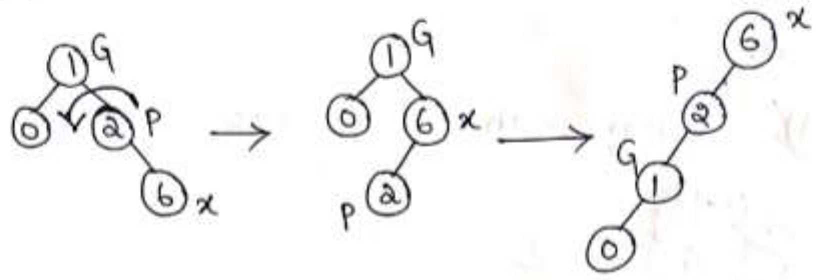
Insert 0, 2 :



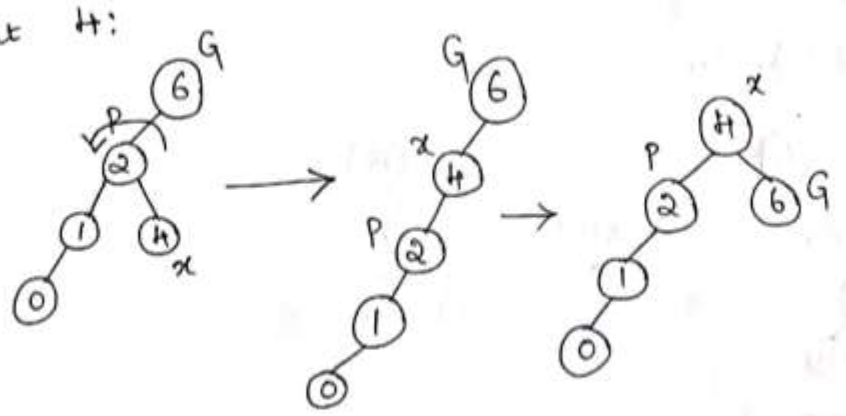
Insert 1 :



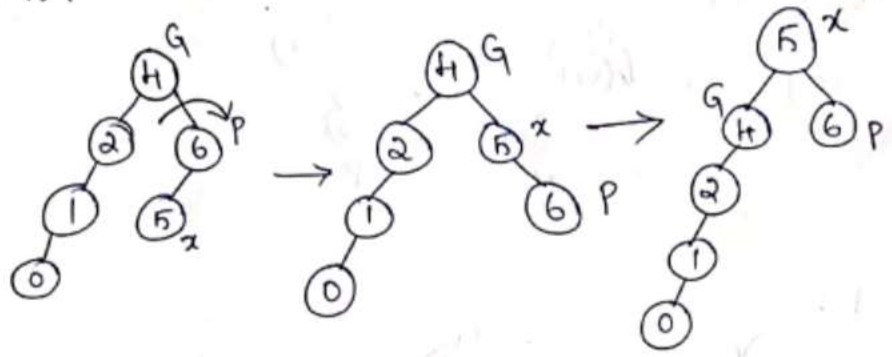
Insert 6 :



Insert 4 :



Insert 5 :



2.2 Implement Depth First Search (DFS) algorithm. Unit 4 Pg 11

There are two major graph traversal techniques such as,

* Depth First Search
* Breadth First Search.

<u>Depth First Search</u> :

It produces spanning tree as final result. spanning tree is a graph without any loops. In dfs, starting at some vertex v, we process v then recursively traverse all vertices adjacent to v. To do this we, mark a vertex v as visited once we visit it. We use stack data structure with maximum size of vertices in the graph to implement DFS traversal.

The graph is represented using adjacency list method. Hence the node of the adjacency list is defined as,

```
Struct node
{
  Struct node *next;
  int Vertex;
};
typedef Struct node *GNODE;
```

The graph array is,

```
GNODE graph[20];
```

The visited array is,

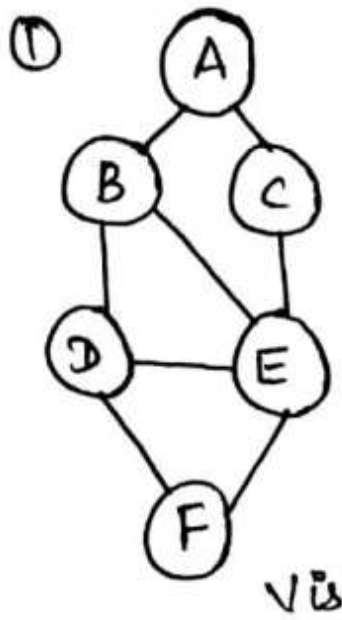
```
int visited[20];
```

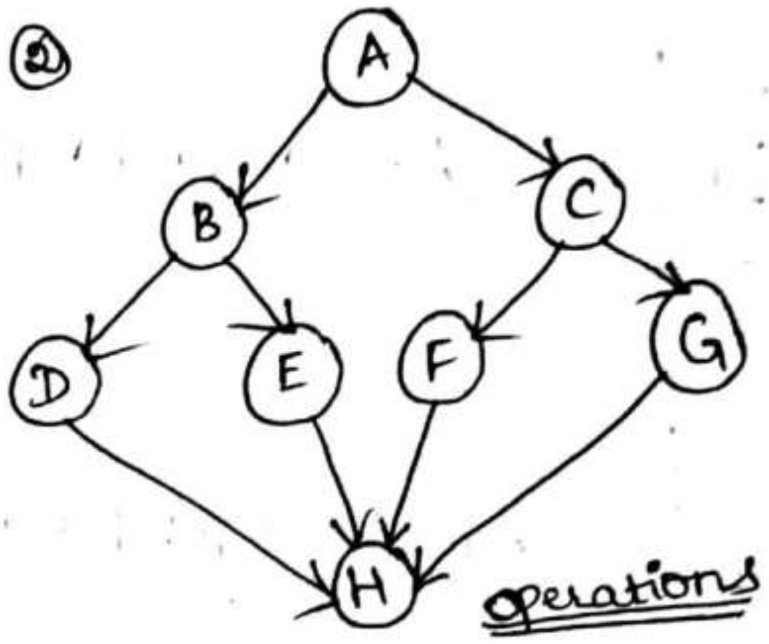
DFS(int i):

```
Void DFS(int i) {
        Declare a var p of type GNODE.
        Print %d
        Assign graph[i] to p.
        Set visited[i]=1
          while ( P!=NULL) {
             Assign Vertex of p to i
            if ( visited[i]!=1)
            {  DFS(i); }
             Assign next of P to P
        } }
```

DFS is implemented using Stack. The time complexity is $(O(|V|+|E|))$.

Eg:

① 

② 

By Solving ②

	Stack	Visited
①	A	
②	C, B	- A
③	G, F, B	- A C
④	H, F, B	- A C G
⑤	F, B	- A C G H

Operations

Initially, push A into Stack. Start from the node A and traverse to the adjacent nodes of A. Then, mark node A as visited node, and pop it from the Stack.

Now push the adjacent nodes of A into Stack.

Continue the Process until all nodes are visited once.

	Stack		Visited nodes
(6)	B	-	A C G H F
(7)	F D	-	A C G H F B
(8)	D	-	A C G H F B E
(9)		-	A C G H F B E D

The Stack becomes empty and Stop the process.

Now, the order of traversal = A → C → G → H → F → B → E → D

2.3 Explain Adjacency matrix and Adjacency List Unit 4 Pg 6

Graph Representation:

Graphs are generally represented in the following scheme as,

1. Adjacency matrix representation
2. Adjacency List representation
3. Incidence matrix representation.

1. Adjacency matrix representation:

One simple way to represent a graph is to use 2-dimensional array., ie., using a matrix of size V×V, V- rows and V- columns.

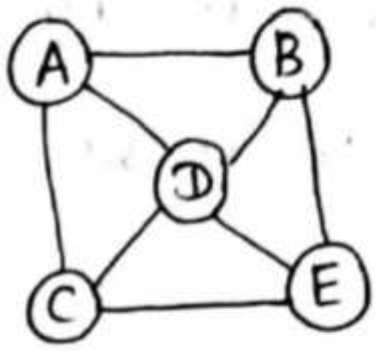
Here both rows and columns represent vertices and the value of matrix is either 0 or 1.

1 - if there exists an edge b/w vertices
0 - if there is no edge between vertices.

Adjacency matrix is given for both directed and undirected graphs respectively.

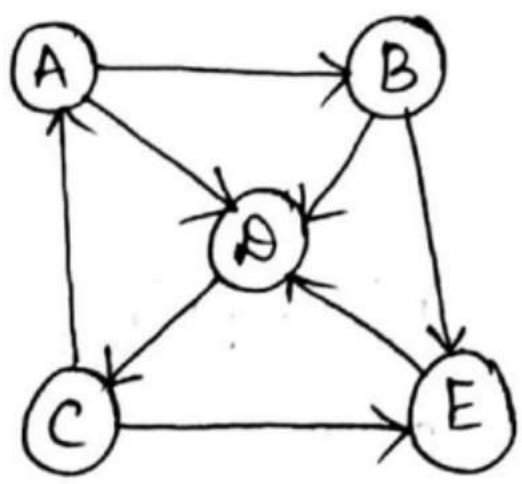
Eg:

Undirected graph



	A	B	C	D	E
A	0	1	1	1	0
B	1	0	0	1	1
C	1	0	0	1	1
D	1	1	1	0	1
E	0	1	1	1	0

Directed graph



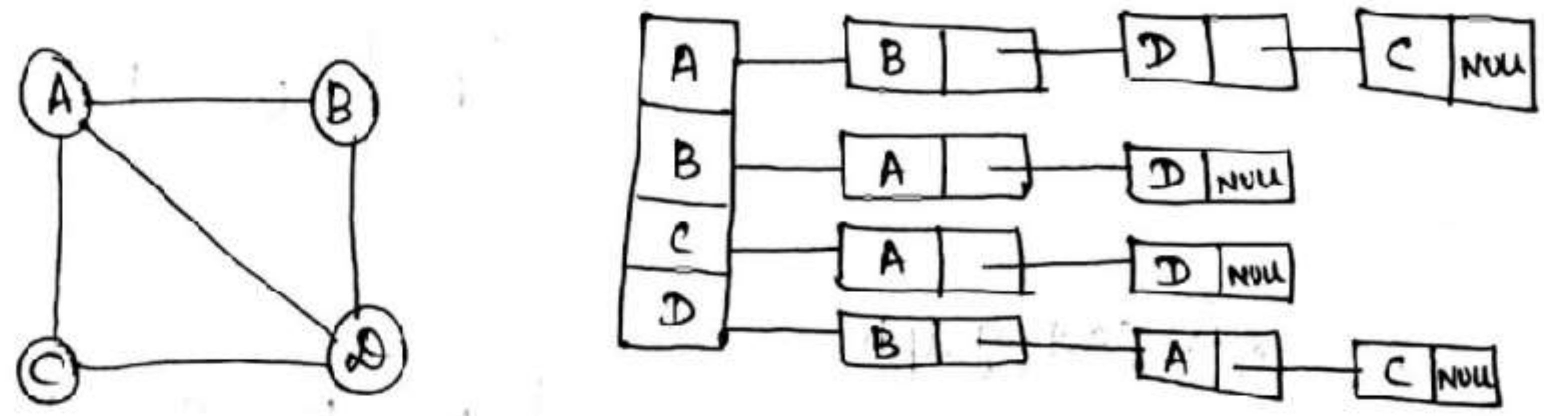
	A	B	C	D	E
A	0	1	0	1	0
B	0	0	0	1	1
C	1	0	0	0	1
D	0	0	1	0	0
E	0	0	0	1	0

In directed graph, the cell value AB is 1 because there is an edge from A to B, but on the other side cell value of BA is 0.

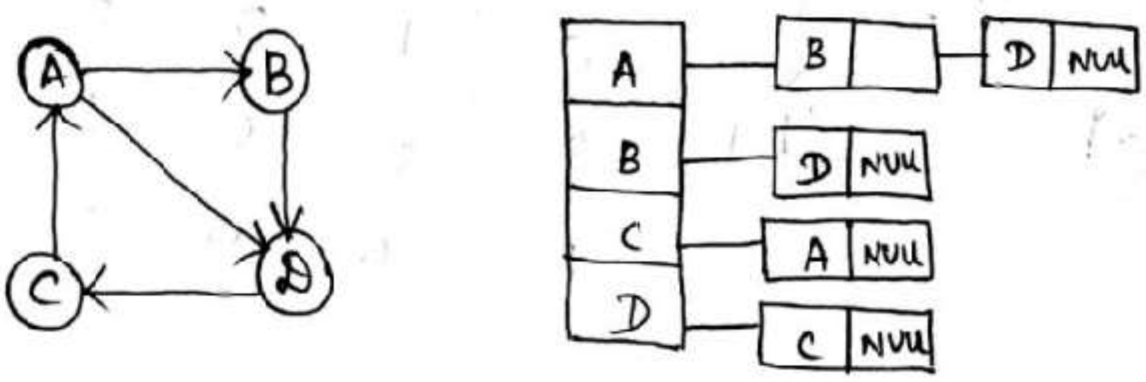
2. Adjacency List representation:

In this representation, graph is stored as a linked structure. If a node has any adjacency to any other node, then the adjacent node will be linked with its predecessor by storing it's address in the link field of predecessor node.

Undirected graph:



Directed graph:



2.4 What is Graph? Define any five types of graph. Unit 4 Pg 1

Graph :

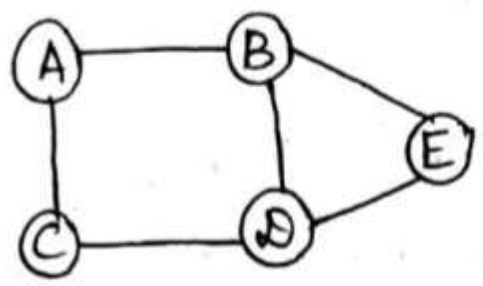
A graph is a non-linear datastructure consisting of collection of nodes and edges. The nodes are referred to as Vertices and edges are lines that connects pair of Vertices.

Generally, a graph G is represented as $G=(V,E)$.

$V \rightarrow$ Set of Vertices

$E \rightarrow$ set of Edges

Eg:



$\rightarrow$ fig (i)

Here, $G=(V,E)$

$V=\{A,B,C,D,E\}$

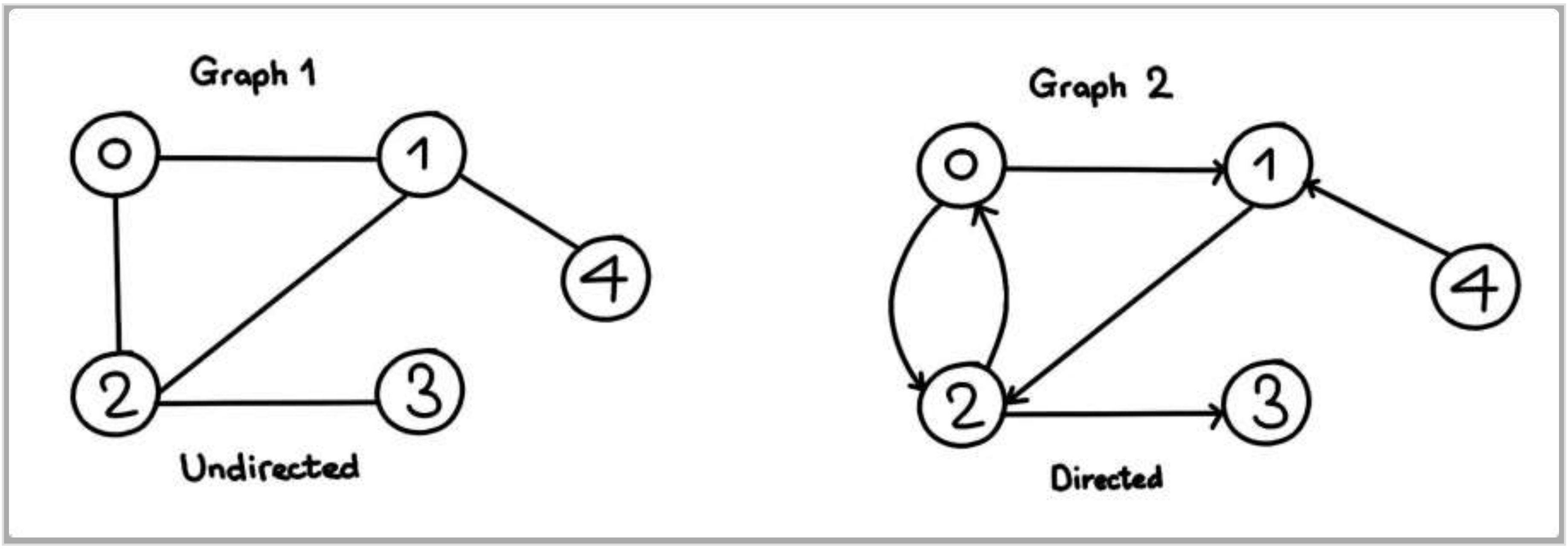
$E=\{(A,B),(A,C),(B,E),(C,D),(D,E),(B,D)\}$

A graph may have cycles.

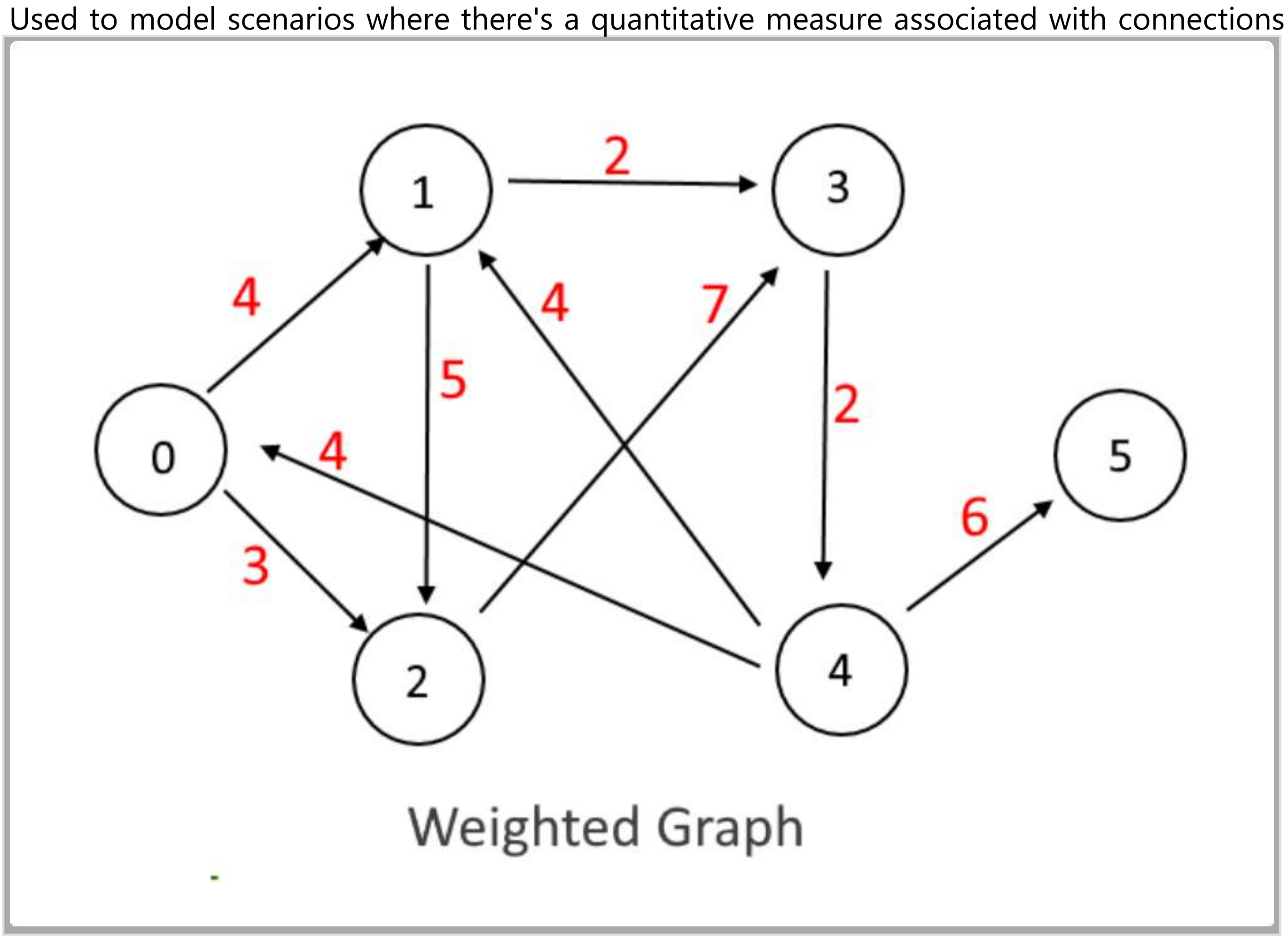
Types of graph

1. **Undirected Graph:**
   - Edges have no direction.
   - Connections between vertices are bidirectional.
   - If there's an edge from vertex A to vertex B, there's also an edge from B to A.
2. **Directed Graph (Digraph):**
   - Edges have a direction associated with them.
   - Relationships between vertices are one-way.

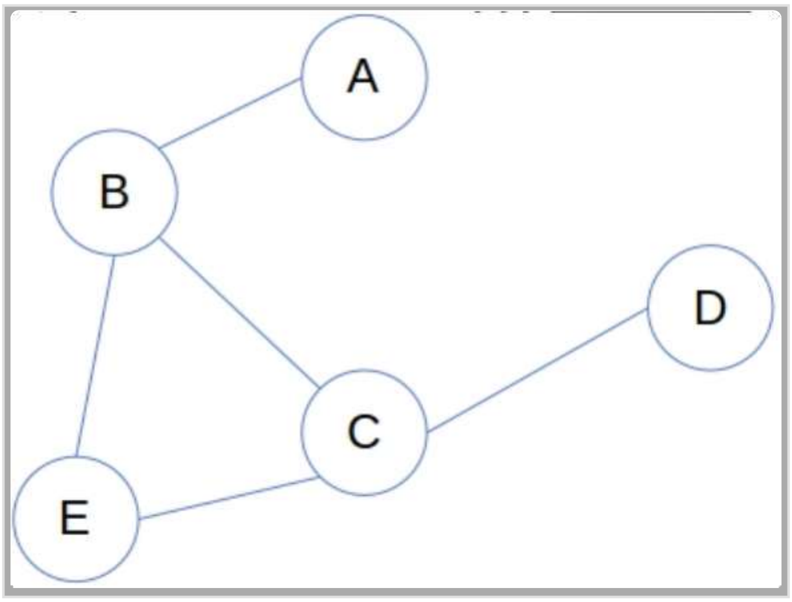
     If there's a directed edge from vertex A to vertex B, there might not be a directed edge from B to A.



3. **Weighted Graph:**
   - Each edge has an associated weight or cost.
   - Weights represent quantitative measures like distance, time, or cost.
   - Used to model scenarios where there's a quantitative measure associated with connections between vertices.

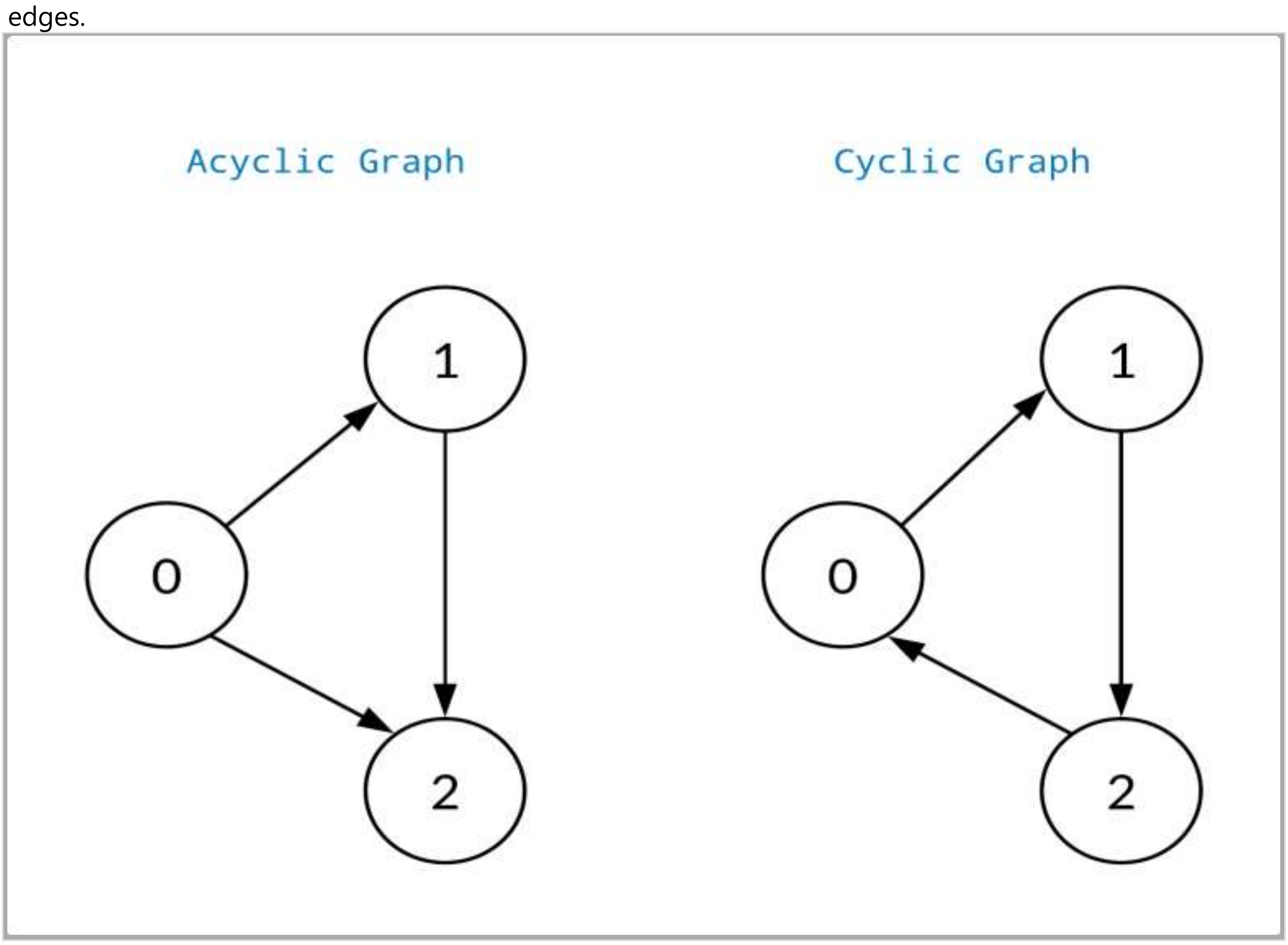


4. **Unweighted Graph:**
   - Edges have no associated weight.
   - All edges are considered equal.
   - Used when relationships between vertices are binary, without any quantitative measure.



5. Cyclic Graph and Acyclic Graph:
   - Cyclic Graph:
     - Contains at least one cycle.
     - A cycle is a path that starts and ends at the same vertex.
     - You can traverse through the graph and return to the same starting point following a sequence of edges.
   - Acyclic Graph:

     - Does not contain any cycles.
     - Also known as a directed acyclic graph (DAG).
     - There's no way to traverse through the graph and return to the same starting point following a sequence of directed edges.



2.5 Write and explain Knuth-Morris-Pratt pattern matching algorithm. Unit 5 Pg 10

Knuth - Morris - Pratt algorithm:

KMP algorithm searches for occurrences of a word "w" within a main text "S".

The naive approach doesn't work well in cases where we see many matching characters followed by a mismatching character.

Eg: Text = A A A A A A A B

Pattern = A A A A B.

When the naive approach is applied then, the inner for loop keeps looping till the last to encounter mismatch. Solving a pattern matching problem in linear time was a challenge. Hence, kmp algorithm is used which is known for linear time for exact matching.

Case 1: When all the patterns to be matched has all unique characters.

Eg:

```
0 1 2 3 4 5 6 7 8
C O D E E C O D Y
↓ ↓ ↓ ↓
C O D Y
```

Case-2:

When all the pattern or parts of pattern have common Suffix and Prefix.

For a given String, a proper prefix is prefix with whole string not allowed.

For-eg:- "ABC"

Prefixes are: "", "A", "AB", "ABC".

Proper Prefixes are: "", "A", "AB".

Suffixes are: "", "C", "BC", "ABC".

4. Example:

text (T) and a pattern (P) as follows:

```
T: ABC ABCDAB ABCDABCDABDE
P: ABCDABD
```

1. Preprocessing (Building the Failure Function):
    - For the pattern (P = "ABCDABD"), the failure function would be:

    ```
    P:  A  B  C  D  A  B  D
    F:  0  0  0  0  1  2  0
    ```

2. Matching Algorithm:
    - We start comparing characters from the beginning of the text and the pattern.
    - When a mismatch occurs at position 7 in the pattern ("D" vs "B"), we consult the failure function.
    - The failure function tells us to shift the pattern by 2 characters to the right.
    - We continue comparing characters from the text and the shifted pattern.
3. Output:
    - The algorithm finds occurrences of the pattern (P) in the text (T) at positions 0 and 15.

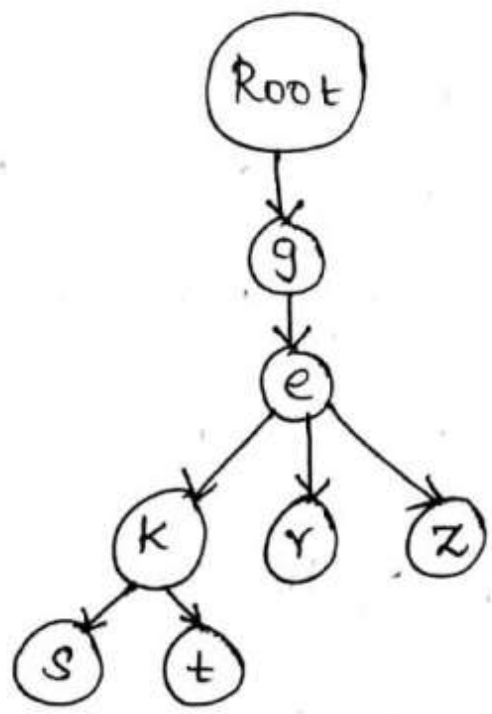
2.6 Define TRIE. List and explain the types of tries. Unit 5 Pg 14

Trie :-

A tree like data-Structure wherein the nodes of the tree Store the entire alphabet and strings/words can be retrieved by traversing down a branch path of the tree.

It is an efficient information retrieval data Structure. Search Complexity Can be brought to optimal limit (key length). Thus, it is O(M) time, where M is the maximum String length.

Eg:



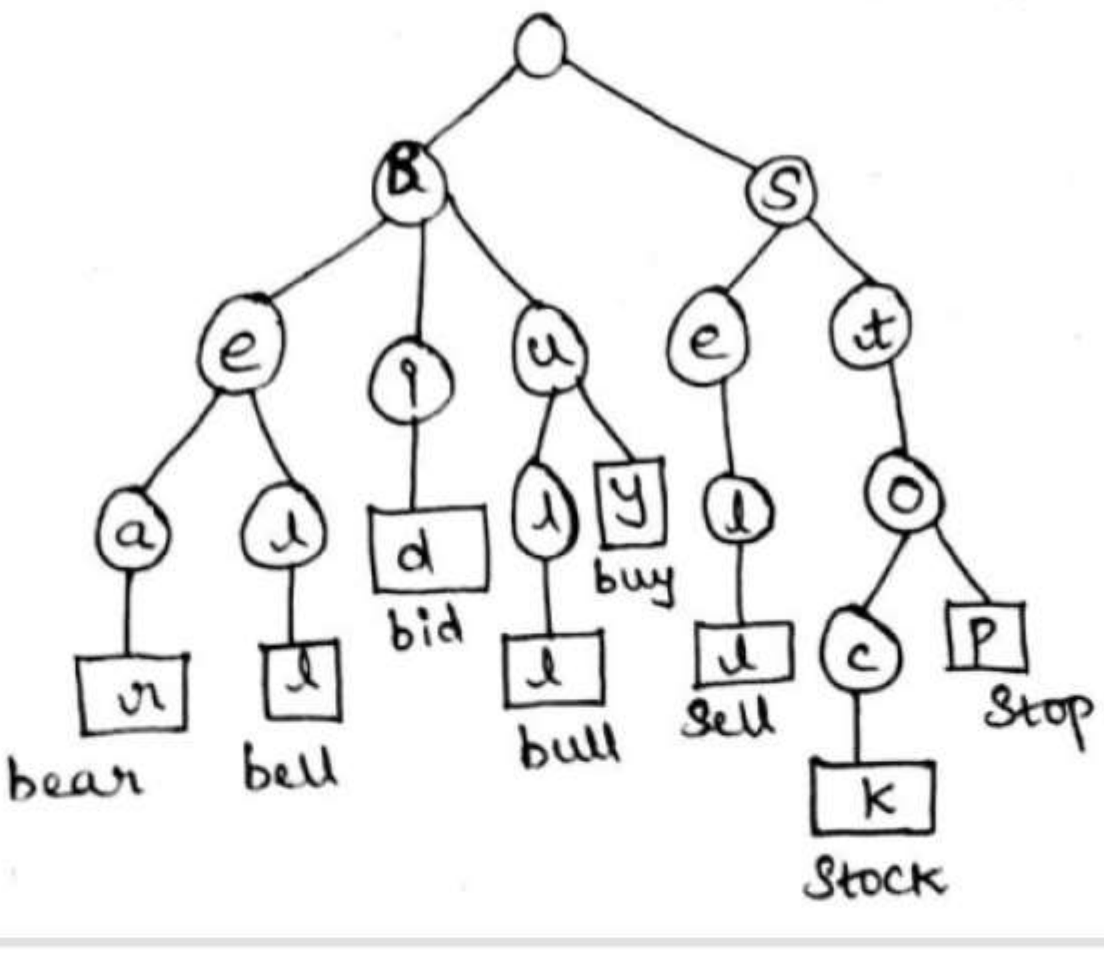
Every node has multiple branches. Each branch represents a possible character of keys. The last node of every key should be marked as end of word node.

Compressed Tries:

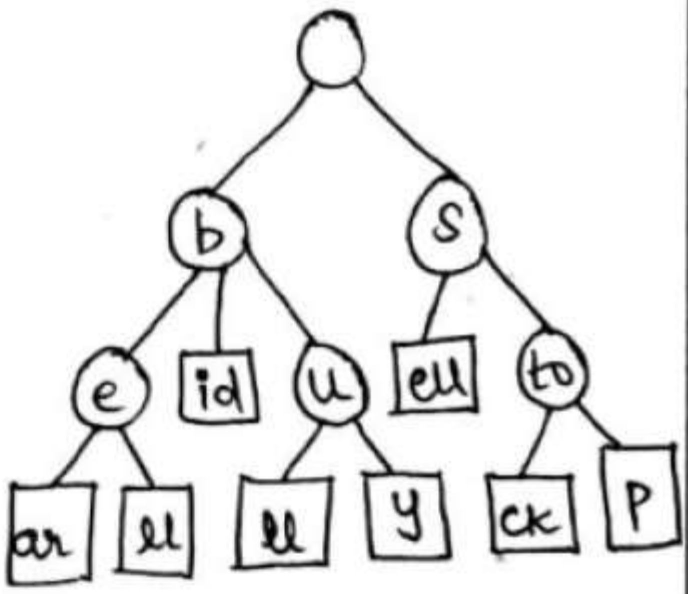
To overcome the disadvantage of Standard trie - space requirement, Compressed tries are formed by compressing the chains of redundant nodes in Standard tries.

Eg.

Standard trie



Compressed trie



Suffix Trees :-

The suffix trie of a string x is the compressed trie of all the suffixes of x. It helps in solving a lot of string related problems like pattern matching, finding distinct substrings in a given string, finding longest palindrome etc.,.

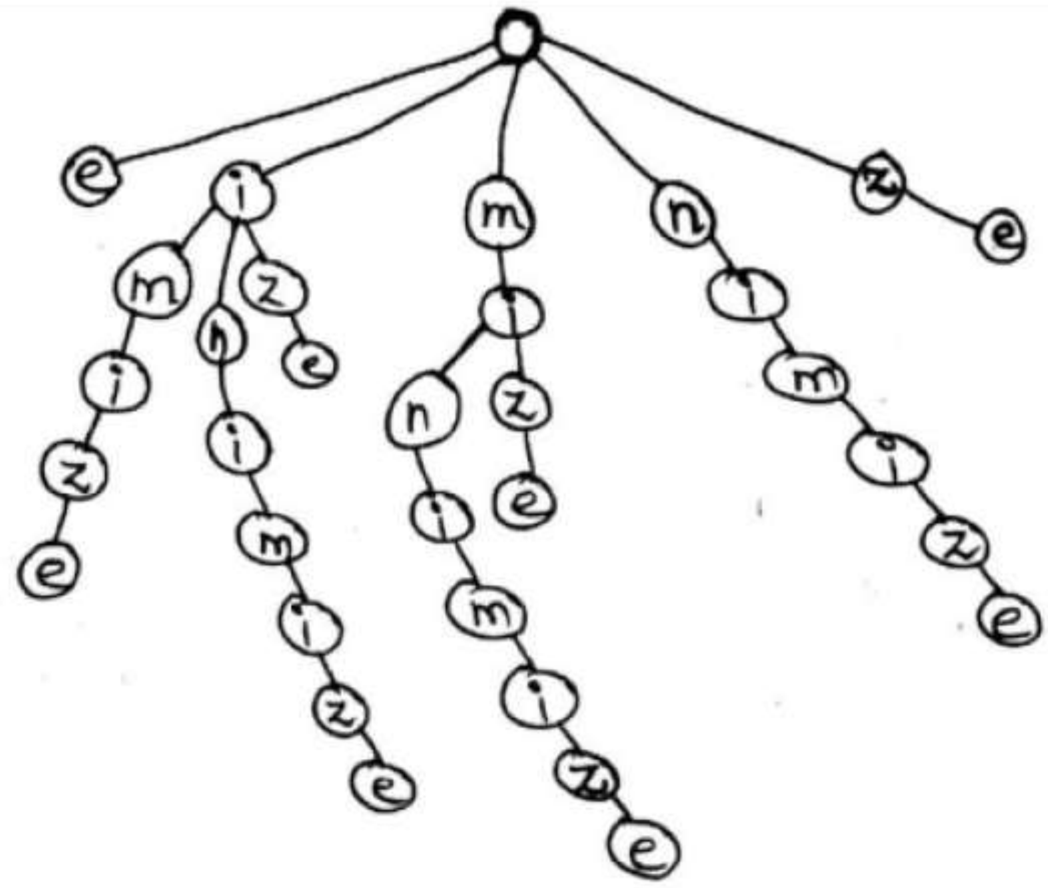
Eg:

minimize

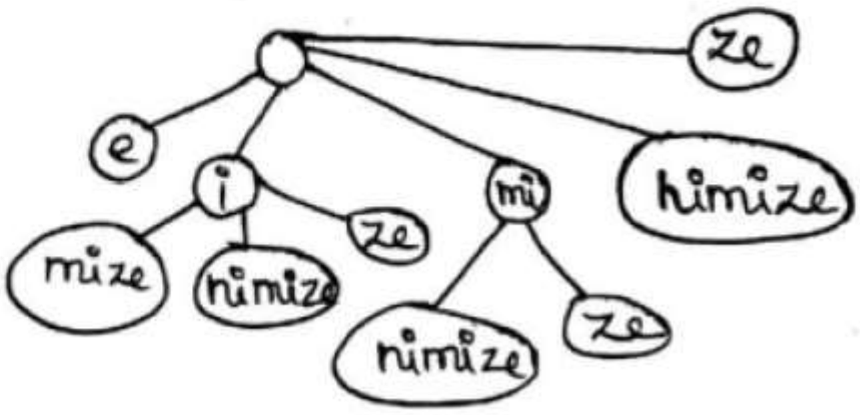
Identify all suffixes,

e
ze
ize
mize
imize
nimize
inimize
minimize

S

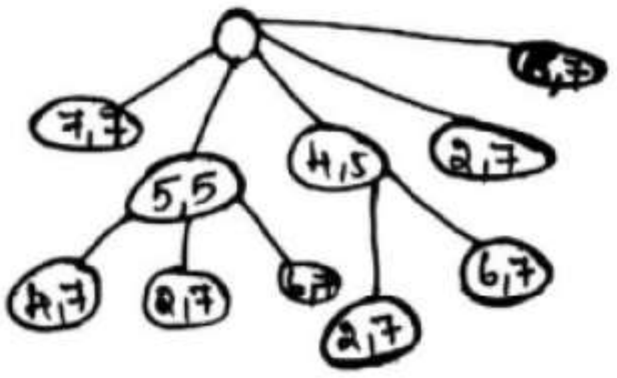


compressed tree



Represent the compressed tree using numbers.

0 1 2 3 4 5 6 7
m i n i m i z e



Hence, the suffix trie representation of a given word is done.