Topic.....Date.....

Name : Sandesh Shrestha
Roll : 2K21/CO/417
Batch : A6

Discrete Mathematics

assignment 2

Topic... discrete assignment 2 ...Date.....

1) Explain Depth first and breadth first search algorithm with example.

Breadth First Search algorithm:

→ Breadth First Search algorithm is a graph traversal algorithm that starts traversing the graph from the root node and explores all the neighbouring nodes. Then, it selects the nearest node and explores all the unexplored nodes. While using BFS, any node can be considered as starting node. BFS puts every vertex/node of graph into 2 categories - visited and non-visited. It selects a node, marks it visited, puts it into queue, then visits all adjacent nodes and puts in queue if unvisited and so on.

→ It is used to determine shortest path and minimum spanning tree.

→ used in web crawlers to create web page indexes.

→ used to find neighbouring locations from a given source location.

Algorithm:

input: graph G = (V, E) and source nodes in V

step1: Mark all the nodes v in V as unvisited

step2: Mark source as visited

step3: Enqueue the source node Q(s).

step4: while (Q is not empty) {

integer u = dequeue Q or [u = front(q) and q.pop]

print u;

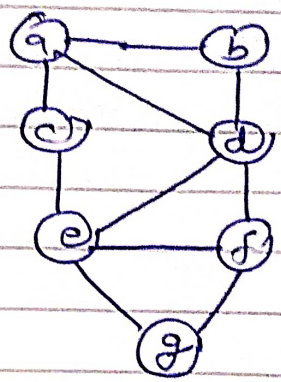
~~step5:~~ for (each unvisited neighbour v of u) {

Teacher's Sign.....

```
        mark v as visited;
            enqueue (Q, v);
        }
    }
```

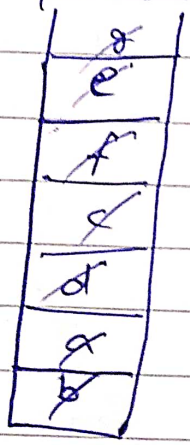
example:

start / source node : (b)

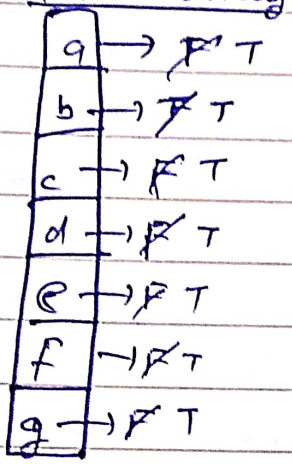


step 1:

queue



visited array



BFS output:

b a d c f e g

Depth First search algorithm

Depth First search is recursive algorithm to search all vertices of a tree data structure of a graph. It starts with initial node of graph G and goes deeper until we find goal node or node with no children. We ~~can~~ use stack data structure to implement DFS. It starts at a selected node and goes in / explores as far as possible along each branch before backtracking.

So, the basic idea is to start from root or any arbitary node and mark it visited then move to its adjacent node which is unmarked

Teacher's Sign .....

Topic.....Date.....

and continue this loop until there is no unmarked adjacent node. Then backtrack and check for other unmarked nodes and traverse them. Finally, print all nodes in path.

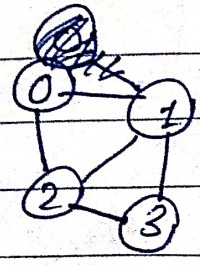
Algorithm:

- put any one vertices to stack
- pop it and print it
- add adjacent connected unvisited node to stack
  - repeat

function for DFS:

```
void DFS(int start){
    visited[start] = true;
    cout << start;
for(int j = 0; j < 4; j++){
  if(a[start][j] == 1 && visited[j] == false){
        DFS(j);
    }
  }
}
```

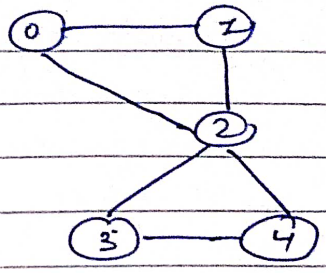
```
bool visited[4] = {false, false, false, false};
int a[4][4] = {
                {0, 1, 0, 0};
                {0, 0, 1, 1}  ⇒
                {1, 0, 0, 0}
                {0, 1, 1, 0}
              };
```



Teacher's Sign .....

Topic.....Date.....

example:



start node = 0

~~0 1 2~~

~~0 1 2 4 3~~ 2

0 2 4 3 1

DFS

visited

0 → ~~F~~ T

1 → ~~F~~ T

2 → ~~F~~ T

3 → ~~F~~ T

4 → ~~F~~ T

2) Explain inorder, preorder and postorder tree traversal algorithms with examples.

⇒ Unlike linear data structures having only one way to traverse them, trees can be traversed in different ways.

i) inorder traversal of a tree

Algorithm (Recursive)

~~Write~~ If root is not null {

inorder (tree → left)

print the root data

inorder (tree → right)

}

Steps: Traverse left sub-tree in inorder

visit the root

Traverse right sub-tree in inorder

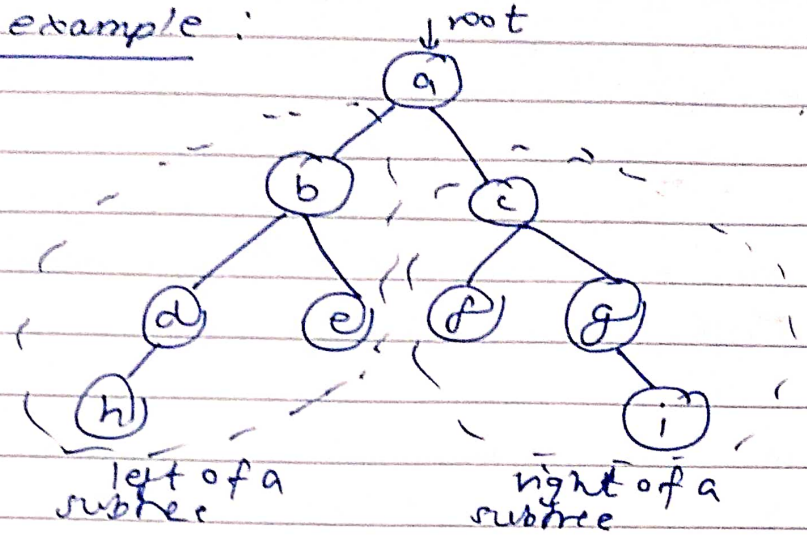
Complexity: O(n) where n is size of binary tree

Teacher's Sign .....

Topic.....Date.....

Inorder (left Root Right)

example :



Inorder : h, d, b, e, a, f, c, g, i

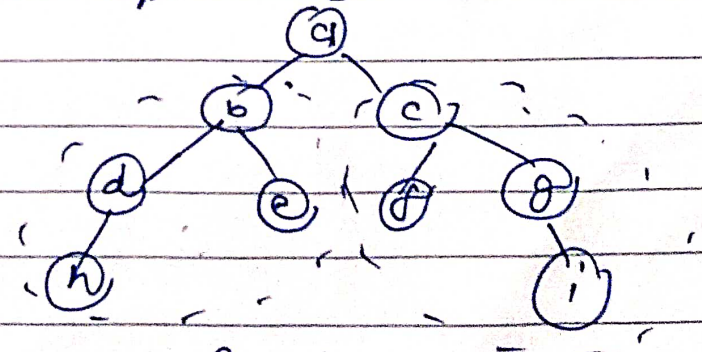
ii) preorder (Root→left→right)

In preorder traversal, first root node is visited then left subtree and after that right subtree is visited.

Algorithm

```
If root is not null {
    print root data;
    preorder (root→left);
    preorder (root→right);
}
```

preorder example:



a b d h e c f g i

Teacher's Sign .....

Topic.....Date.....

~~Iterative algorithm~~

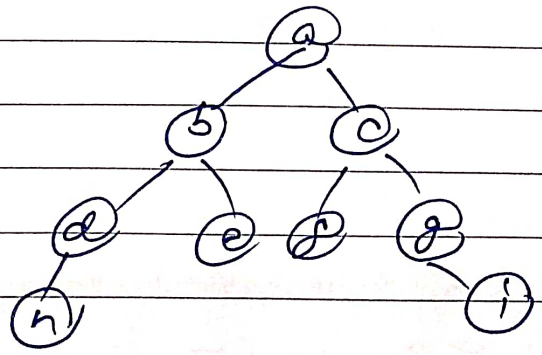
(c) iii) postorder traversal (left→right→root)

The postorder traversal follows the left right root form. Here, left ~~root~~ subtree is evaluated first then right subtree then root.

algorithm:

1) if tree root is not null {
    postorder(root→left)
    postorder(root→right)
    print root data.
}

postorder example:



postorder: h d e b f i g c a

Teacher's Sign

Topic..... Date.....

4) Explain Warshall's algorithms with example.

→ Warshall's algorithm is an efficient and effective way to find the transitive closure of a relation by using adjacency matrix.

Warshall algorithm (matrix generation)

$R^k(i,j) = R^{k-1}(i,j)$ ✱

Rule 1: in row i and column j, if element is 1 in R(k-1) then in R(k) it will remain 1.

Rule 2: in row i and column j, if element is 0 in R(k-1) then elements in R(k) will be changed to 1 iff element in column j and row k and in column k and row i are both 1 in R(k-1).

$R^{k-1} =$

	j	k
k	1	
i	0	1

→

$R^k =$

	j	k
k	1	1
i	1	1

example: By using Warshall's algorithm, find the transitive closure of relation $R = \{(2,1), (2,3), (3,1), (3,4), (4,1), (4,3)\}$ on set $A = \{1,2,3,4\}$.

So, m_{ij}: $P_0 =$

	1	2	3	4
1	0	0	0	0
2	1	0	1	0
3	1	0	0	1
4	1	0	1	0

initial matrix

I) in row 1 and column 1 of P_0

C R

{2,3,4} ϕ $C \times R = \{\}$ so no new changes

Teacher's Sign .....

Topic.....Date.....

$$\therefore P_2 = P_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

II) in ~~row II and~~ column II and row II of P_1

C = ϕ, R = $\{1, 3\}$ $C \times R = \phi$ so no changes

$P_2 = P_1$

III) in column III and row III of P_2

C = $\{2, 4\}$, R = $\{1, 4\}$ $C \times R = \{(2,1), (4,1), (4,4), (2,4)\}$

$$P_3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \end{array}$$

IV) in column IV and row IV of P_3

C = $\{2, 3, 4\}$, R = $\{1, 3, 4\}$ $C \times R = \{(2,1), (2,3), (2,4), (3,1), (3,3), (3,4), (4,1), (4,3), (4,4)\}$

$$P_4 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \end{array}$$

transitive closure

#

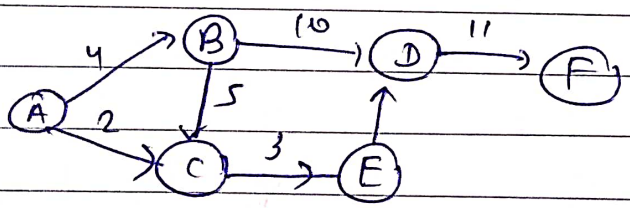
$\therefore R^+ = \{(2,1), (2,3), (2,4), (3,1), (3,3), (3,4), (4,1), (4,3), (4,4)\}$ which is required transitive closure.

Topic..... Date.....

Q) Explain shortest path and minimal spanning trees with example.

→ shortest path algorithm — Dijkstra algorithm

Dijkstra algorithm is a single source shortest path algorithm. Here, single source means that only one source is given and we find to find the shortest path from source to all nodes.
Shortest path in a graph is the path between 2 nodes (or vertices) such that the sum of weights of its constituent edges is minimum.

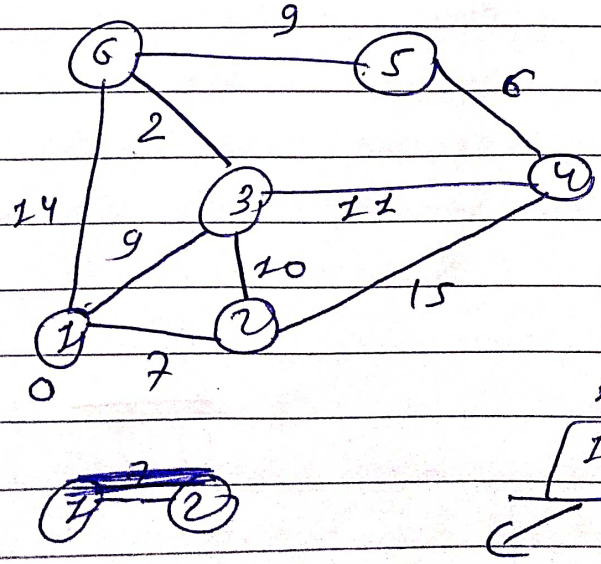


A to F
shortest path: ACEDF
weight : 20

Relaxation

if $d(u) + c(u,v) < d(v)$
$d(v) = d(u) + c(u,v)$

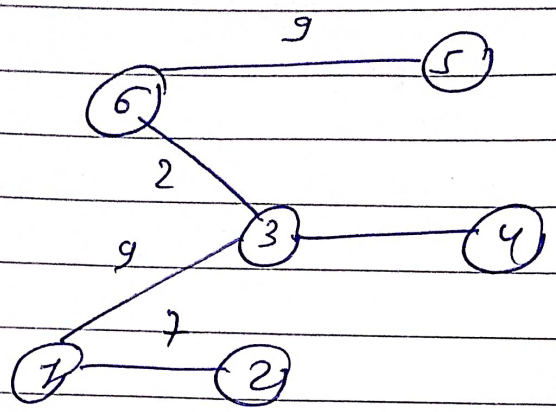
example:



Source	destination				
1	2	3	4	5	6
	∞	∞	∞	∞	∞
1,2	7	9	∞	∞	14
1,2,3	7	9	22	∞	14
1,2,3,6	7	9	22	∞	11
1,2,3,6,	7	9	20	20	11
1,2,3,6,4,5	7	9	20	20	11

shortest paths are

Teacher's Sign .....



shortest paths by Dijkstra's algorithm

sequence: 1, 2, 3, 6, 4, 5

Minimum spanning tree in a connected weighted graph is a spanning tree that has the smallest possible sum of weights of its edges. A spanning tree is a subgraph ~~of~~ that is a tree containing every nodes of graph and one less edge than no. of nodes [(n-1) edges].

→ minimum spanning tree algorithms are:
(i) prim's algorithm (ii) Kruskal's algorithm

* Prim's algo: It is a greedy algorithm that is used to find the minimum spanning tree from graph. It finds the ~~edges~~ subsets of edges that include every vertex of graph such that sum of weights of edges can be minimized. It starts with single node and explores all the adjacent nodes with all connecting edges at every step. The edges with minimal weights causing no cycles are selected.

steps: i) first initiate/initialize ~~empt~~ MST with randomly chosen vertex.

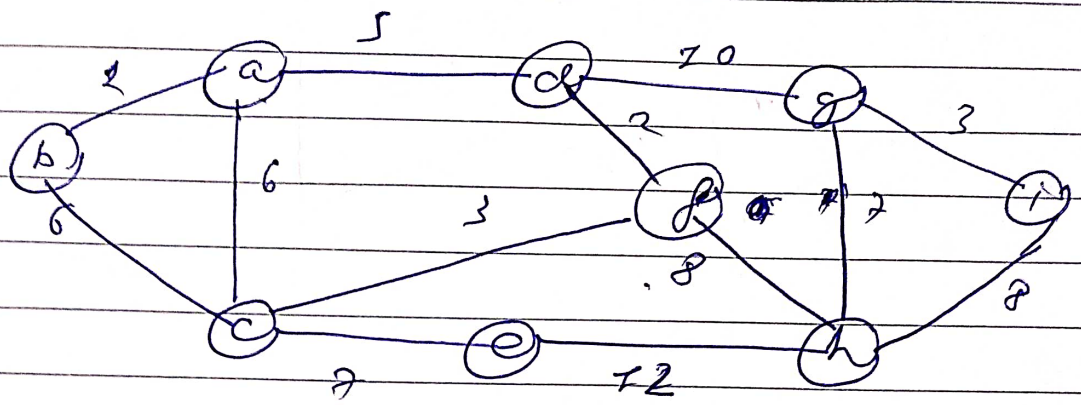
ii) find all the edges that connect tree in above step with new vertices and from edges found select

Teacher's Sign .....

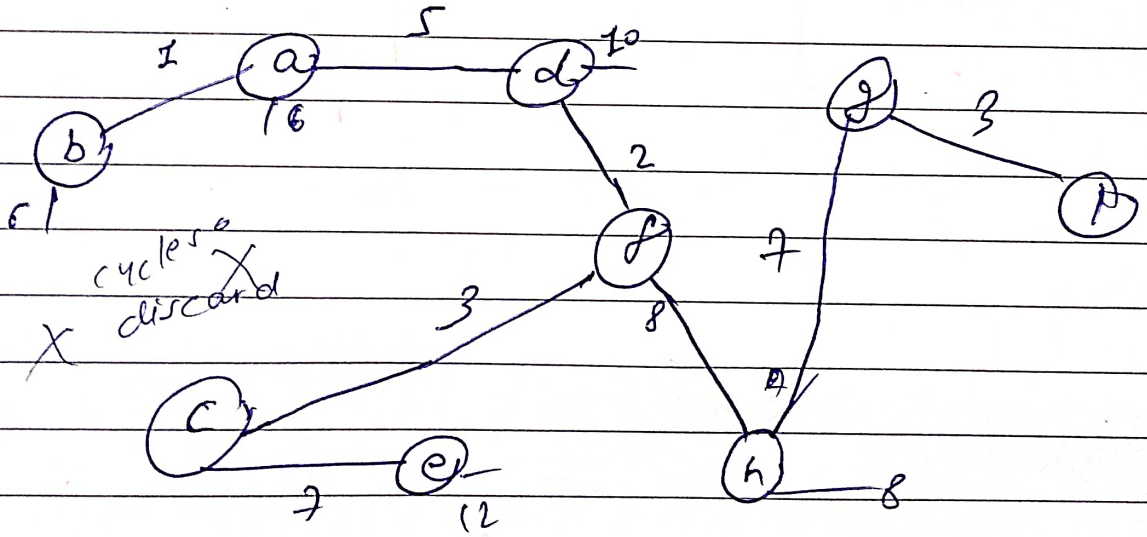
Topic.....Date.....

prim's algo example:

→ min. cost spanning tree



9 vertices
so, 8 edges
in ~~mst~~ spanning tree

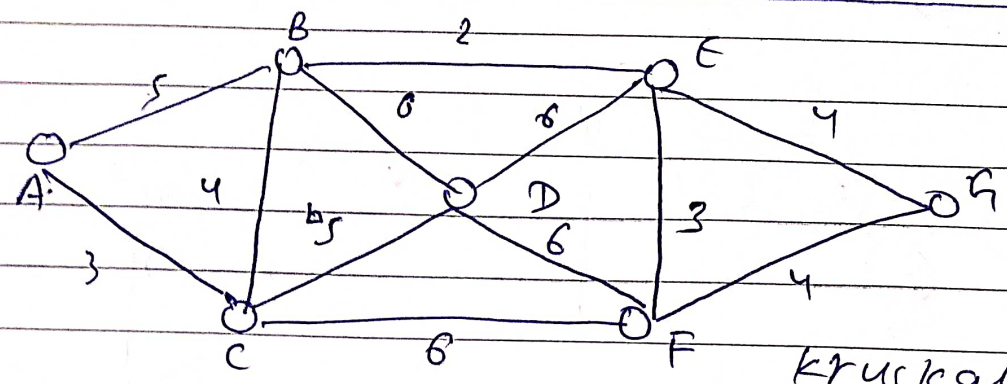


minimum spanning tree

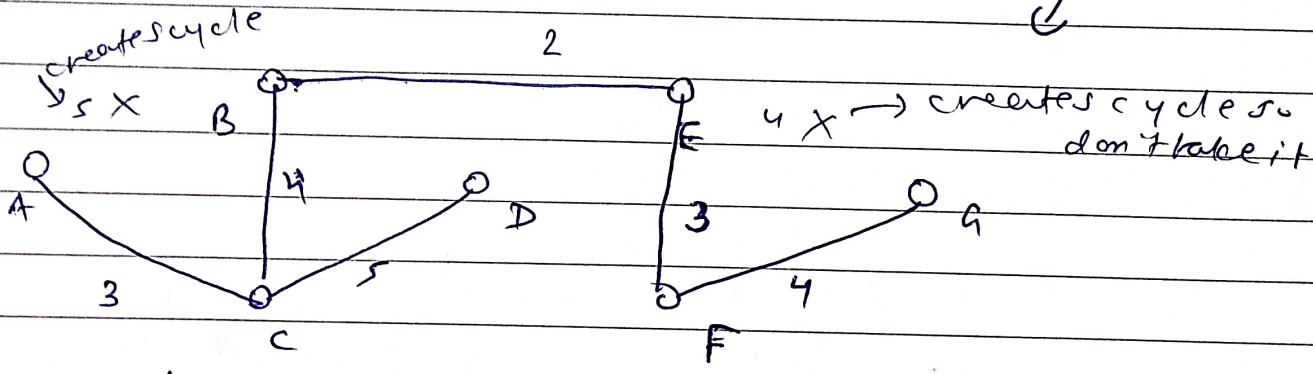
Topic.....Date.....

minimum edge and add it to tree.

iii) Repeat step 2 until mst is formed.



Kruskal example



minimum spanning tree weight : 21

* Kruskal's algo

In this algo, we start from edges ~~and~~ with lowest weight and keep adding edges until the goal is reached.

Steps:

i) first sort all edges from low weight to high.

ii) Now, take edge with lowest weight and add it to spanning tree. If edge to be added creates cycle then reject the edge.

iii) continue to add the edges until we reach all vertices and a mst is created.

Teacher's Sign .....