IARE '24 DISCORD TEAM

II-II

DAA MODULE 2 SOLUTIONS

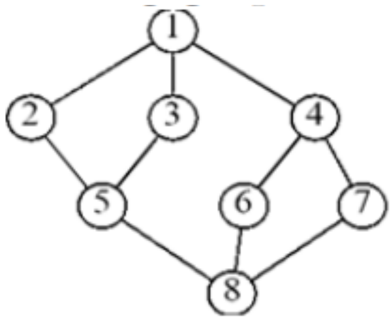
UJJWAL • ABHIRAMI • VISHAL

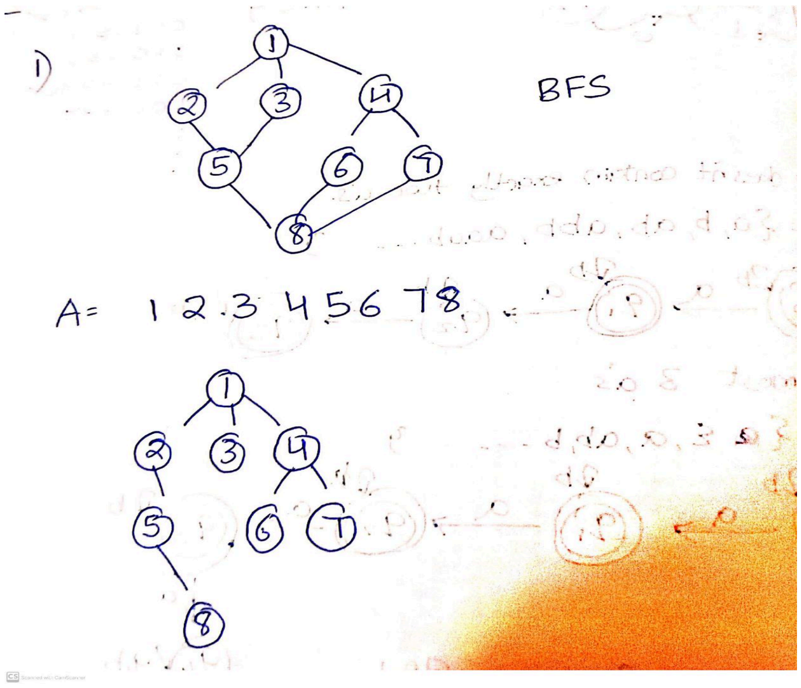
SEARCHING AND TRAVERSAL TECHNIQUES

DAA MODULE 2

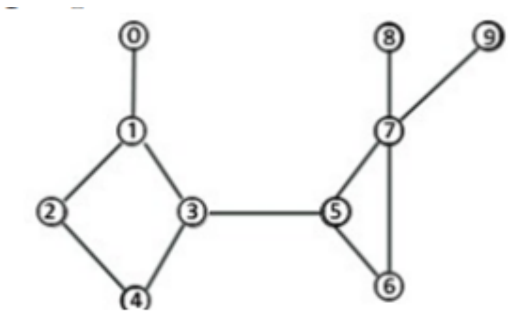
PART-A

1)Build a BFS traversal tree of the following graph.



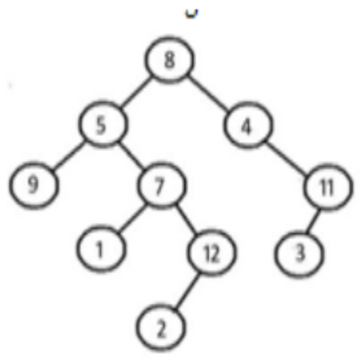


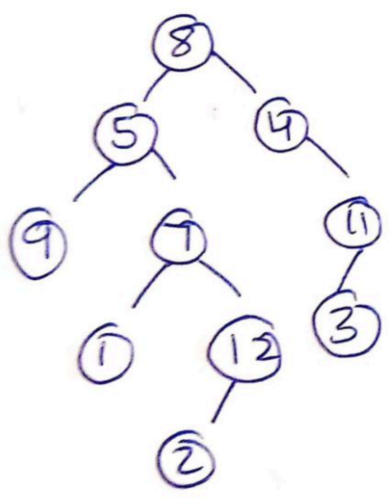
2) Identify the articulation points from the following graph



[1,3,5,7] Verified ✅

3) Organise Order, pre order, post order traversal of the following tree





Inorder (LVR):

9 5 1 7 2 12 8 4 3 11

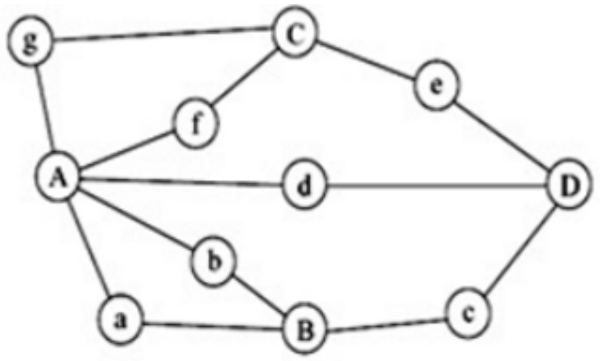
Preorder (VLR):

8 5 9 7 1 12 2 4 11 3

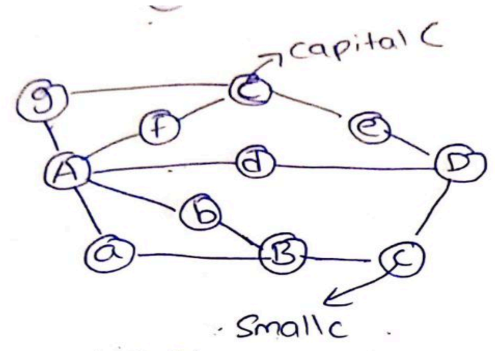
Postorder (LRV):

9 1 2 12 7 5 3 11 4 8

4)Construct DFS and BFS traversal trees of following graph

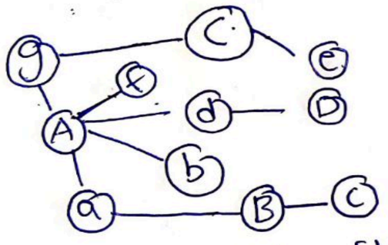


4)

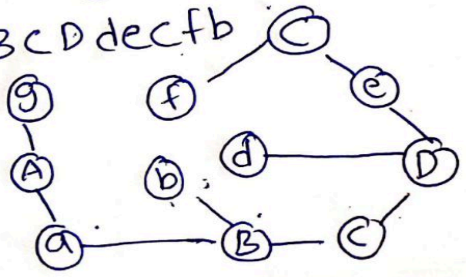


DFS and BFS

BFS:- gACabdfeBDc



DFS:- gAaBCDdecfb



PART-B

1)Explain the BFS algorithm with an example.

Breadth-first search 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 for traversal, any node in the graph can be considered as the root node.

Algorithm

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Enqueue the starting node A and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until QUEUE is empty

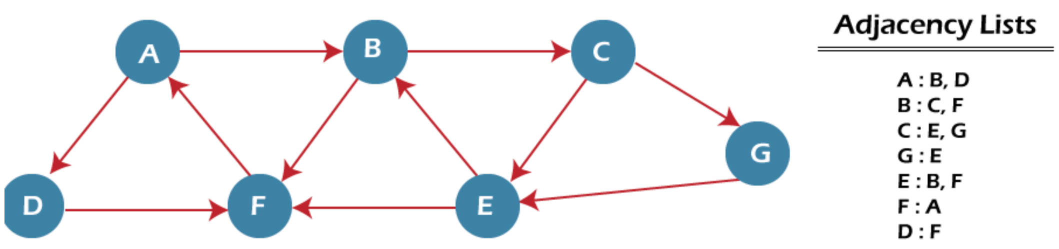
Step 4: Dequeue a node N. Process it and set its STATUS = 3 (processed state).

Step 5: Enqueue all the neighbours of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state)

[END OF LOOP]

Step 6: EXIT

Example of BFS algorithm

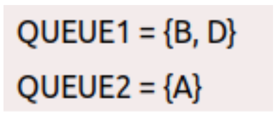


Step 1 - First, add A to queue1 and NULL to queue2.

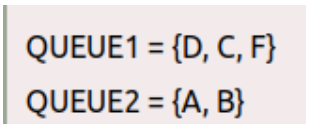
QUEUE1 = {A}

QUEUE2 = {NULL}

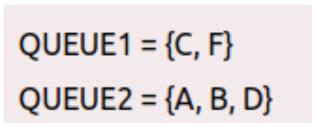
Step 2 -Now, delete node A from queue1 and add it into queue2. Insert all neighbours of node A to queue1.


QUEUE1 = {B, D}
QUEUE2 = {A}

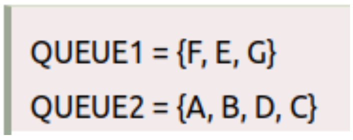
Step 3 - Now, delete node B from queue1 and add it into queue2. Insert all neighbours of node B to queue1.


QUEUE1 = {D, C, F}
QUEUE2 = {A, B}

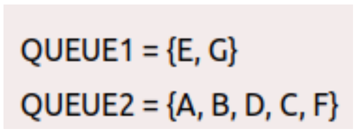
Step 4 - Now, delete node D from queue1 and add it into queue2. Insert all neighbours of node D to queue1. The only neighbour of Node D is F since it is already inserted, so it will not be inserted again.


QUEUE1 = {C, F}
QUEUE2 = {A, B, D}

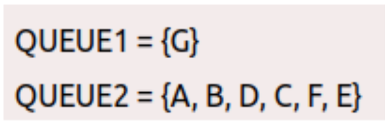
Step 5 - Delete node C from queue1 and add it into queue2. Insert all neighbours of node C to queue1.


QUEUE1 = {F, E, G}
QUEUE2 = {A, B, D, C}

Step 6 - Delete node F from queue1 and add it into queue2. Insert all neighbours of node F to queue1. Since all the neighbours of node F are already present, we will not insert them again.


QUEUE1 = {E, G}
QUEUE2 = {A, B, D, C, F}

Step 7 - Delete node E from queue1. Since all of its neighbours have already been added, we will not insert them again. Now, all the nodes are visited, and the target node E is encountered into queue2.


QUEUE1 = {G}
QUEUE2 = {A, B, D, C, F, E}

Complexity of BFS algorithm

Time complexity of BFS depends upon the data structure used to represent the graph. The time complexity of the BFS algorithm is **O(V+E)**, since in the worst case, the BFS algorithm explores every node and edge. In a graph, the number of vertices is O(V), whereas the number of edges is O(E).

The space complexity of BFS can be expressed as **O(V)**, where V is the number of vertices.

2) Explain depth first search algorithm with example

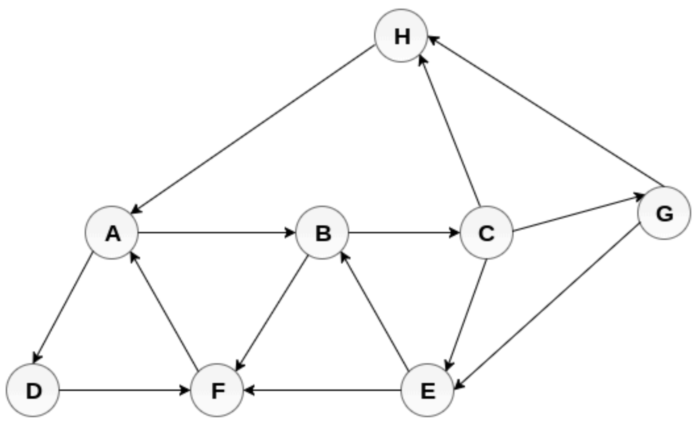
Depth first search (DFS) algorithm starts with the initial node of the graph G, and then goes deeper and deeper until we find the goal node or the node which has no children. The algorithm then backtracks from the dead end towards the most recent node that is yet to be completely unexplored.

Algorithm

- **Step 1:** SET STATUS = 1 (ready state) for each node in G
- **Step 2:** Push the starting node A on the stack and set its STATUS = 2 (waiting state)
- **Step 3:** Repeat Steps 4 and 5 until STACK is empty
- **Step 4:** Pop the top node N. Process it and set its STATUS = 3 (processed state)
- **Step 5:** Push on the stack all the neighbours of N that are in the ready state (whose STATUS = 1) and set their
  STATUS = 2 (waiting state)
  [END OF LOOP]
- **Step 6:** EXIT

Example :

Consider the graph G along with its adjacency list, given in the figure below. Calculate the order to print all the nodes of the graph starting from node H, by using depth first search (DFS) algorithm.



Adjacency Lists

A :	B, D
B :	C, F
C :	E, G, H
G :	E, H
E :	B, F
F :	A
D :	F
H :	A

Solution :

Push H onto the stack

STACK : H

POP the top element of the stack i.e. H, print it and push all the neighbours of H onto the stack that are in ready state.

Print H

STACK : A

Pop the top element of the stack i.e. A, print it and push all the neighbours of A onto the stack that are in ready state.

Print A

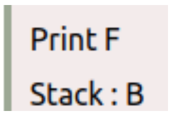
Stack : B, D

Pop the top element of the stack i.e. D, print it and push all the neighbours of D onto the stack that are in ready state.

Print D

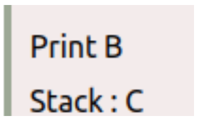
Stack : B, F

Pop the top element of the stack i.e. F, print it and push all the neighbours of F onto the stack that are in ready state.


Print F

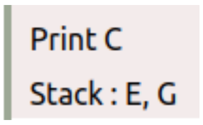
Stack : B

Pop the top of the stack i.e. B and push all the neighbours


Print B

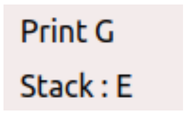
Stack : C

Pop the top of the stack i.e. C and push all the neighbours.


Print C

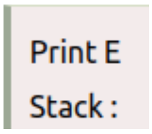
Stack : E, G

Pop the top of the stack i.e. G and push all its neighbours.


Print G

Stack : E

Pop the top of the stack i.e. E and push all its neighbours.


Print E

Stack :

Stack is empty now

The printing sequence of the graph will be :

H → A → D → F → B → C → G → E

3) Explain iterative versions of binary tree traversal concept of tree traversal algorithms (inorder, algorithms and Contrast preorder and post order).

Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, trees can be traversed in different ways. Following are the generally used ways for traversing trees.

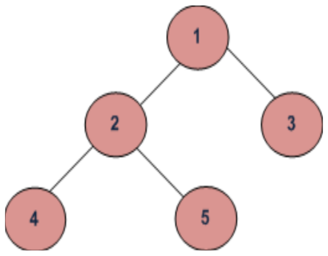
1) DFS
2) BFS

In DFS, we have:

(a) Inorder (Left, Root, Right)

(b) Preorder (Root, Left, Right)

(c) Postorder (Left, Right, Root)



Inorder Traversal:

Algorithm Inorder(tree)

1. Traverse the left subtree, i.e., call Inorder(left-subtree)
2. Visit the root.
3. Traverse the right subtree, i.e., call Inorder(right-subtree)

Uses of Inorder: In the case of binary search trees (BST), Inorder traversal gives nodes in non-decreasing order.

Example: In order traversal for the above-given figure is 4 2 5 1 3.

Preorder Traversal

Algorithm Preorder(tree)

1. Visit the root.
2. Traverse the left subtree, i.e., call Preorder(left-subtree)
3. Traverse the right subtree, i.e., call Preorder(right-subtree)

Uses of Preorder

Preorder traversal is used to create a copy of the tree. Preorder traversal is also used to get prefix expressions on an expression tree. Preorder traversal for the above-given figure is 1 2 4 5 3.

Postorder Traversal (Practice):

Algorithm Postorder(tree)

1. Traverse the left subtree, i.e., call Postorder(left-subtree)

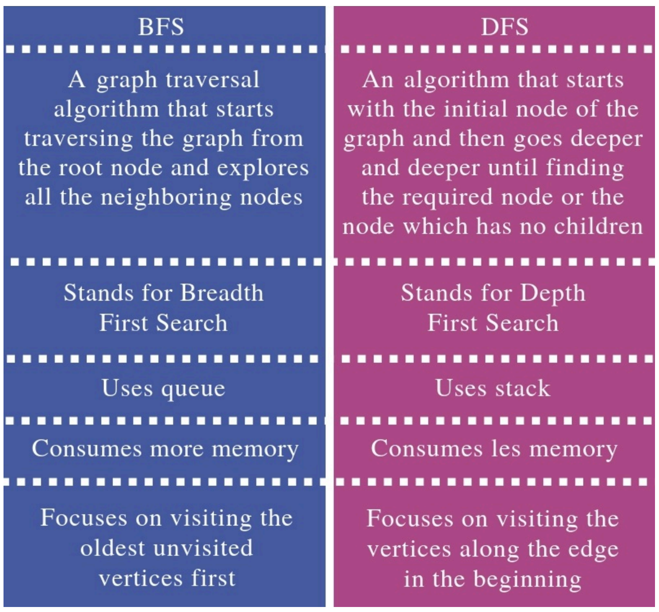
2. Traverse the right subtree, i.e., call Postorder(right-subtree)

3. Visit the root.

Uses of Postorder : Postorder traversal is used to delete the tree.Postorder traversal is also useful to get the postfix expression of an expression tree.

Example: Postorder traversal for the above-given figure is 4 5 2 3 1.

4) Compare the approaches of BFS and DFS methods and derive the time complexity of both methods for the inputs of adjacency list and adjacency matrix separately.



BFS	DFS
A graph traversal algorithm that starts traversing the graph from the root node and explores all the neighboring nodes	An algorithm that starts with the initial node of the graph and then goes deeper and deeper until finding the required node or the node which has no children
Stands for Breadth First Search	Stands for Depth First Search
Uses queue	Uses stack
Consumes more memory	Consumes les memory
Focuses on visiting the oldest unvisited vertices first	Focuses on visiting the vertices along the edge in the beginning

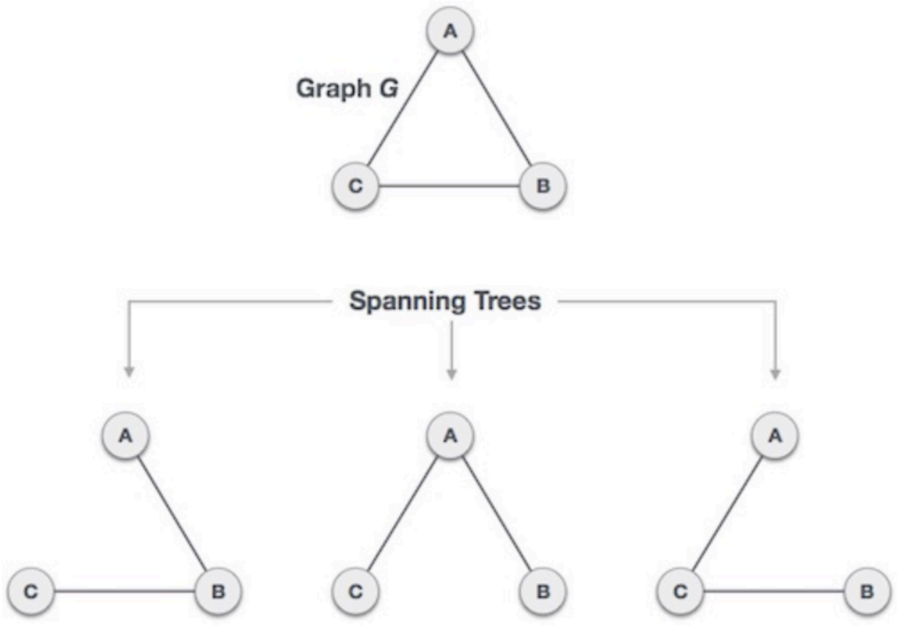
5) Explain BFS and spanning trees in detail.

BFS explanation is already in PART(B) →1

SPANNING TREE:

A spanning tree is a subset of Graph G, which has all the vertices covered with a minimum possible number of edges. Hence, a spanning tree does not have cycles and it cannot be disconnected..

By this definition, we can draw a conclusion that every connected and undirected Graph G has at least one spanning tree. A disconnected graph does not have any spanning tree, as it cannot be spanned to all its vertices.



A complete undirected graph can have a maximum $\mathbf{n^{n-2}}$ number of spanning trees, where **n** is the number of nodes. In the above addressed example, **n is 3,** hence $\mathbf{3^{3-2} = 3}$ spanning trees are possible.

General Properties of Spanning Tree

We now understand that one graph can have more than one spanning tree. Following are a few properties of the spanning tree connected to graph G –

- A connected graph G can have more than one spanning tree.
- All possible spanning trees of graph G, have the same number of edges and vertices.
- The spanning tree does not have any cycle (loops).

- Removing one edge from the spanning tree will make the graph disconnected, i.e. the spanning tree is **minimally connected**.
- Adding one edge to the spanning tree will create a circuit or loop, i.e. the spanning tree is **maximally acyclic**.

Application of Spanning Tree

Spanning tree is basically used to find a minimum path to connect all nodes in a graph. Common application of spanning trees are –

- **Civil Network Planning**
- **Computer Network Routing Protocol**
- **Cluster Analysis**

6) Explain weighting rule for finding UNION of sets and collapsing rule

Weighting rule for Union:

If the number of nodes in the tree with root i is less than the number in the tree with the root j, then make ‘j’ the parent of i; otherwise make ‘i’ the parent of j. To implement the weighting rule we need to know how many nodes are there in every tree. To do this we maintain a “count” field in the root of every tree. If ‘i’ is the root then count[i] equals the number of nodes in the tree with roots ‘i’. Since all nodes other than roots have positive numbers in parent (P) field, we can maintain count in P field of the root as negative number.

```
Algorithm WeightedUnion(i,j)
//Union sets with roots i and j, i j using the weighted rule
//P[i]=-count[i] andp[j]=-count[j]
{
temp:=P[i]+P[j];
if (P[i]>P[j])then {
// i has fewer nodes P[i]:=j;
P[j]:=temp;
}
else {
// j has fewer nodes P[j]:=i;
}
}
```

7) How to construct a binary tree from inorder and preorder traversals.

Let us consider the below traversals:

Inorder sequence: D B E A F C

Preorder sequence: A B D E C F

In a Preorder sequence, the leftmost element is the root of the tree. So we know 'A' is the root for given sequences. By searching 'A' in the Inorder sequence, we can find out all elements on the left side of 'A' are in the left subtree, and elements on right in the right subtree. So we know the below structure now.

```
         A
       /   \
     /       \
   D B E      F C
```

We recursively follow the above steps and get the following tree.

```
         A
       /   \
     /       \
    B         C
   / \        /
 /     \    /
D       E  F
```

(not much info here, better to go check out youtube)

5.7 Construct Binary Tree from Preorder and Inorder traversal with examp...

8) Explain about DFS and spanning trees.

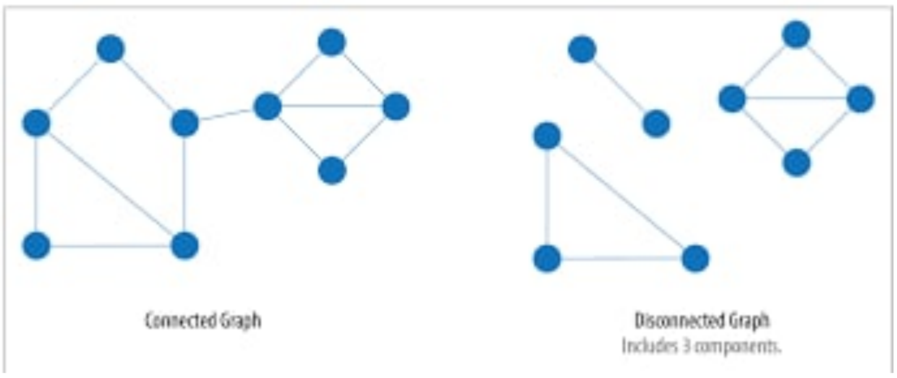
refer to PART(B) → 2 and PART(B) →5

9) Illustrate how to identify given graph is connected or not

In an undirected graph G, two vertices u and v are called connected if G contains a path from u to v. Otherwise, they are called disconnected.

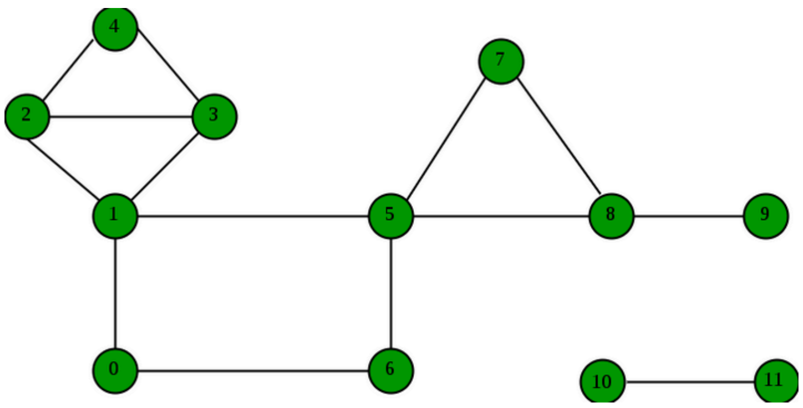
If the two vertices are additionally connected by a path of length 1, i.e. by a single edge, the vertices are called adjacent.

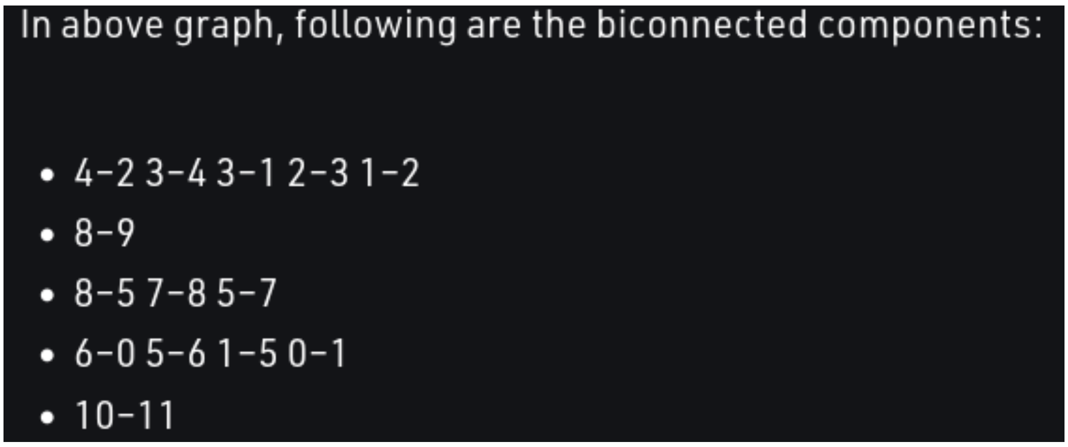
A graph is said to be connected if every pair of vertices in the graph is connected. This means that there is a path between every pair of vertices. An undirected graph that is not connected is called disconnected. An undirected graph G is therefore disconnected if there exist two vertices in G such that no path in G has these vertices as endpoints. A graph with just one vertex is connected. An edgeless graph with two or more vertices is disconnected.



10) Explain the concept of biconnected component with example

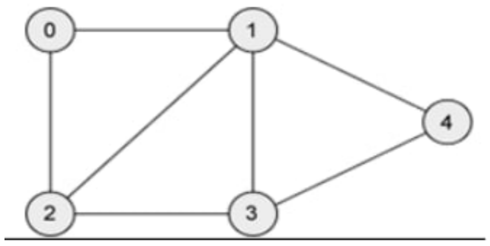
A biconnected component of a graph is a connected subgraph that cannot be broken into disconnected pieces by deleting any single node (and its incident links).




In above graph, following are the biconnected components:

- 4–2 3–4 3–1 2–3 1–2
- 8–9
- 8–5 7–8 5–7
- 6–0 5–6 1–5 0–1
- 10–11

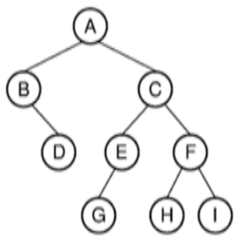
11) Develop a program to print all the nodes reachable from a given starting node in a bigraph using the BFS method.



Python Program for BFS Implementation of the above graph:

```
graph = {
  '4' : ['1','3'],
  '1' : ['0', '4', '2', '3'],
  '3' : ['2', '1', '4'],
  '2' : ['0', '1', '3'],
  '0' : ['1', '2'],
}

visited = [] #List for visited nodes.
queue = []   #Initialise a queue

def bfs(visited, graph, node): #function for BFS
  visited.append(node)
  queue.append(node)

  while queue:          # Creating loop to visit each node
    m = queue.pop(0)
    print (m, end = " ")
    for neighbour in graph[m]:
      if neighbour not in visited:
        visited.append(neighbour)
        queue.append(neighbour)

print("Following is the Breadth-First Search")
bfs(visited, graph, '5')    # function calling
```

12) Develop a program to perform various tree traversal algorithms for a given tree.



Python Code for Tree Traversal Algorithms

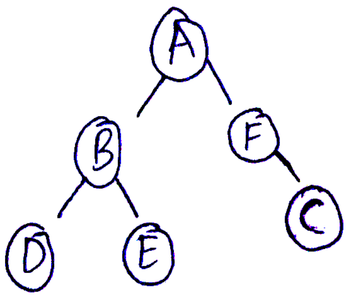
```
class Node:
```

```python
    def __init__(self, key):
        self.left = None
        self.right = None
        self.val = key

def printInorder(root):
    if root:
        printInorder(root.left)
        print(root.val)
        printInorder(root.right)

def printPostorder(root):
    if root:
        printPostorder(root.left)
        printPostorder(root.right)
        print(root.val)

def printPreorder(root):
    if root:
        print(root.val)
        printPreorder(root.left)
        printPreorder(root.right)

# Driver code
root = Node("A")
root.left = Node("B")
root.right = Node("C")
root.left.right = Node("D")
root.right.left = Node("E")
root.right.left.left = Node("G")
root.right.right = Node("F")
root.right.right.left = Node("H")
root.right.right.right = Node("I")

print("Preorder traversal of binary tree is")
printPreorder(root)

print("Inorder traversal of binary tree is")
printInorder(root)

print("Postorder traversal of binary tree is")
printPostorder(root)
```

13) Construct a binary tree from the following Inorder Sequence: D B E A F C and Preorder sequence: A B D E C F.

DBEAFC
[LVR]

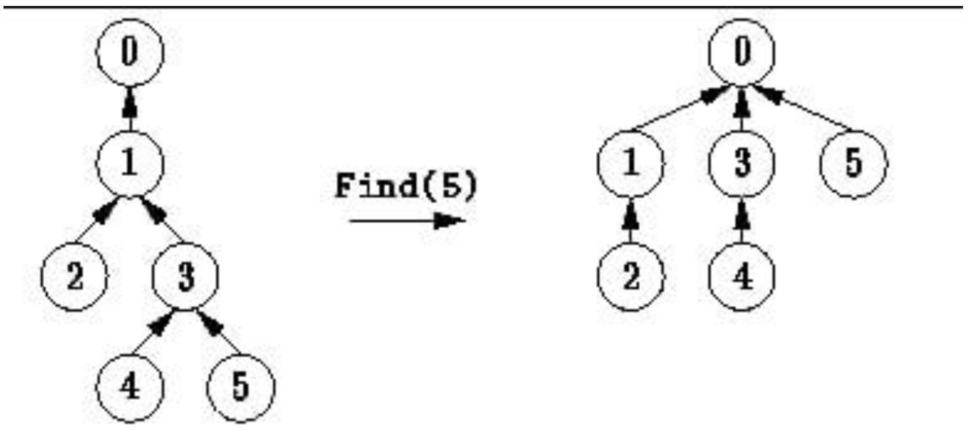
A BDEFC
[VLR]



14) Illustrate the advantage of collapse find over simple find with an example.

If, having found the root, we replace the parent pointer of the given node with a pointer to the root, the next time we do a Find it will be more efficient. In fact, we can go one step further and replace the parent pointer of every node along the search path to the root. This is called a collapsing find operation.

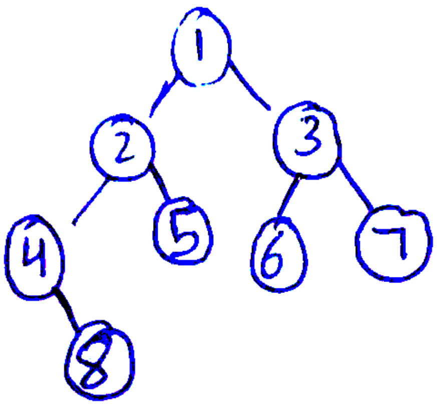
Effect of a collapsing find operation. After the find, all the nodes along the search path are attached directly to the root. I.e., they have had their depths decreased to one. As a side-effect, any node which is in the subtree of a node along the search path may have its depth decreased by the collapsing find operation. The depth of a node is never increased by the find operation. Eventually, if we do enough collapsing find operations, it is possible to obtain a tree of height one in which all the non-root nodes point directly at the root.



15) Construct a binary tree from the following Inorder sequence: 4, 8, 2, 5, 1, 6, 3, 7 and Postorder sequence: 8, 4, 5, 2, 6, 7, 3, 1.

Inorder sequence: 4, 8, 2, 5, 1, 6, 3, 7

Postorder sequence: 8, 4, 5, 2, 6, 7, 3, 1

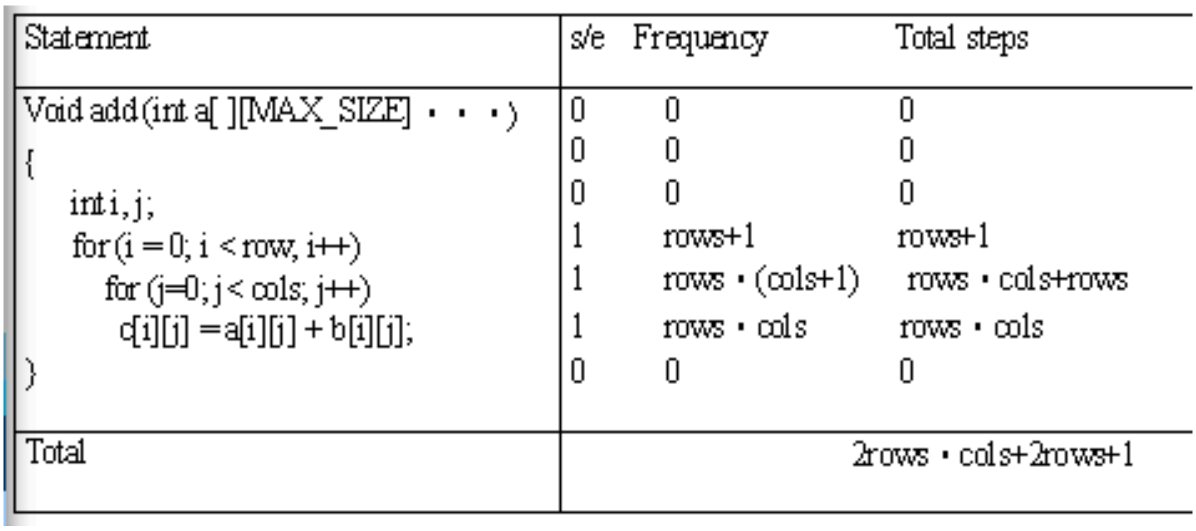


16) Explain step count method and analyse the time complexity when two nxn matrices are added.

Step Count Method

The step count method is one of the methods to analyse the algorithm. In this method, we count the number of times one instruction is executing. From that we will try to find the complexity of the algorithm.

Given below is the step count table of general matrix addition algorithm:



Statement	s/e	Frequency	Total steps
Void add (int a[ ][MAX_SIZE] · · ·)	0	0	0
{	0	0	0
int i, j;	0	0	0
for (i = 0; i < row, i++)	1	rows+1	rows+1
for (j=0; j < cols; j++)	1	rows · (cols+1)	rows · cols+rows
c[i][j] = a[i][j] + b[i][j];	1	rows · cols	rows · cols
}	0	0	0
Total			2rows · cols+2rows+1

Here, the number of rows and columns are equal to n. Therefore, the total time complexity for matrix addition is as follows:

Time Complexity of nxn Matrix Addition: **$2n^2 + 2n + 1$**

Time Complexity: **n^2**

18) What is meant by divide and conquer? Give the recurrence relation for divide and conquer.

The divide-and-conquer technique involves taking a large-scale problem and dividing it into similar sub-problems of a smaller scale, and recursively solving each of these sub-problems. Generally, a problem is divided into sub-problems repeatedly until the resulting sub-problems are very easy to solve.
This type of algorithm is so called because it divides a problem into several levels of sub-problems, and conquers the problem by combining the solutions at the various levels to form the overall solution to the problem. The control abstraction for divide and conquer technique is DANDC(P),where P is the problem to be solved.

```
DANDC (P)  {
if SMALL (P) then return S (p);
else {
divide p into smaller instances p1, p2, .... Pk, k>=1;
apply DANDC to each of these sub problems;
return (COMBINE (DANDC (p1) , DANDC (p2),...., DANDC (pk));  }
}
```

If the sizes of the two subproblems are approximately equal then the computing time of DANDC is given by the recurrence relation:

$$T(n) = \begin{cases} g(n) & n \text{ small} \\ 2\,T(n/2) + f(n) & \text{otherwise} \end{cases}$$

Where, T (n) is the time for DANDC on 'n' inputs
g (n) is the time to complete the answer directly for small inputs and
f (n) is the time for Divide and Combine

19) Explain Control abstraction of divide and conquer.

A control abstraction is a procedure whose flow of control is clear but whose primary operations are specified by other procedures whose precise meanings are left undefined. The control abstraction for divide and conquer technique is DANDC(P), where P is the problem to be solved.

```
DANDC (P)  {
if SMALL (P) then return S (p);
else {
divide p into smaller instances p1, p2, .... Pk, k>=1;
apply DANDC to each of these sub problems;
return (COMBINE (DANDC (p1) , DANDC (p2),...., DANDC (pk));  }
}
```

If the sizes of the two subproblems are approximately equal then the computing time of DANDC is given by the recurrence relation:

$$T(n) = \begin{cases} g(n) & n \text{ small} \\ 2\,T(n/2) + f(n) & \text{otherwise} \end{cases}$$

Where, T (n) is the time for DANDC on 'n' inputs
g (n) is the time to complete the answer directly for small inputs and
f (n) is the time for Divide and Combine

20) Find out any two drawbacks of the binary search algorithm.

Binary search is an efficient algorithm for finding an item from a sorted list of items. It works by repeatedly dividing in half the portion of the list that could contain the item, until you've narrowed down the possible locations to just one.
The drawbacks of Binary Search algorithm are as follows:

- It employs a recursive approach which requires more stack space.
- Programming binary search algorithms is error prone and difficult.
- The interaction of binary search with memory hierarchy i.e. caching is poor. (because of its random access nature)

21) Find out the drawbacks of the Merge Sort algorithm.

Merge Sort is based on the divide-and-conquer strategy. Merge sort continuously cuts down a list into multiple sublists until each has only one item, then merges those sublists into a sorted list.
The drawbacks of Merge Sort Algorithm are as follows:

- For small datasets, merge sort is slower than other sorting algorithms.
- For the temporary array, mergesort requires an additional space of O(n).
- Even if the array is sorted, the merge sort goes through the entire process.