1. Write a Java program to

a. Traverse nodes in a graph using breadth first search

```
import java.io.*;
import java.util.*;
    public class BFSTraversal
    {
       private int node;
       private LinkedList<Integer> adj[];
       private Queue<Integer> que;
       BFSTraversal(int v)
       {
          node = v;
           adj = new LinkedList[node];
      for (int i=0; i<v; i++)
      {
         adj[i] = new LinkedList<>();
      }
      que = new LinkedList<Integer>();
       }
       void insertEdge(int v,int w)
       {
          adj[v].add(w);
  }
       void BFS(int n)
       {
      boolean nodes[] = new boolean[node];              int a = 0;
          nodes[n]=true;
           que.add(n);
      while (que.size() != 0)
          {
              n = que.poll();
              System.out.print(n+" ");
        for (int i = 0; i < adj[n].size(); i++)
        {
                a = adj[n].get(i);
```

```
                if (!nodes[a])                    {
                nodes[a] = true;
                que.add(a);
                 }
              }
           }
        }
        public static void main(String args[])
        {
           BFSTraversal graph = new BFSTraversal(6);
          graph.insertEdge(0, 1);
          graph.insertEdge(0, 3);
          graph.insertEdge(0, 4);
           graph.insertEdge(4, 5);
           graph.insertEdge(3, 5);
           graph.insertEdge(1, 2);
          graph.insertEdge(1, 0);
          graph.insertEdge(2, 1);
           graph.insertEdge(4, 1);
           graph.insertEdge(3, 1);
           graph.insertEdge(5, 4);
          graph.insertEdge(5, 3);
           System.out.println("Breadth First Traversal for the graph is:");
           graph.BFS(0);
    }
     }
```

```
Breadth First Traversal for the graph is:
0 1 3 4 2 5
```

2. Write a Java program to

b. Implement circular queue

```
import java.util.*;

  public class prepinsta
{

   int Queue[] = new int[100];

      int n, front, rear;

    public CircularQueue(int size)

  {
        n=size;

      front = 0;

         rear=0;
  }

    public static void enqueue(int item)

  {
      if((rear+1) % n != front)

     {
        rear = (rear+1)%n;

           Queue[rear] = item;
     }
      else

    {
        System.out.println(” No Insertion -Queue is full!”);
    }
  }

    public static int dequeue()

  {
    int item;

      if(front!=rear)
    {
      front = (front+1)%n;

       item = Queue[front];

       return item;
    }
```

```
      else
    {

         System.out.println("Can't remove element ");
    }
  }

    public static void display()

  {
      int j;

       if(front != rear)
    {

        for(j=(front+1)%n ; j<rear ; j=(j+1)%n)

      {

          System.out.println(Queue[j]);
      }

    }

       else

         System.out.println("Queue is empty cant display!");
  }

    public static void main(String args[])

  {

     System.out.print("Size of queue : ");

       Scanner sc = new Scanner (System.in);

         int size = sc.nextInt();

     CircularQueue cq = new CircularQueue(size);
     System.out.println(" element in queue are  ");
       cq.enqueue(20);
       cq.enqueue(40);
       cq.enqueue(60);
       cq.enqueue(80);
       cq.display();
       int data = cq.dequeue();
       System.out.println(" element delete is "+data);
       System.out.println(" element in queue after deletion  ");
       cq.display();



  }
}
```

Output:

```
elements in queue are

20

40

60

80

element deleted is 20

elements in queue after deletion

40

60

80
```