7) From a given vertex in a weighted connected graph, find shortest paths to other vertices using **Dijkstra's algorithm**. Write the program in Java.

```
import java.util.Scanner;

public class Sssp

{

   static int[][] cost;

   static int dist[],n;

   static int min(int m, int n)

   {

      return(( m < n) ? m: n);

   }

   static void Dijkstra(int source)

   {

      int[] s=new int[n];

      int min, w=0;

      System.arraycopy(cost[source], 0, dist, 0, n);

      //Initialize dist from source to source as 0

      //mark source vertex - estimated for its shortest path

      s[source] = 1;  dist[source] = 0;

      for(int i=0; i < n-1; i++)

      {

         //Find the nearest neighbour vertex

         min = 999;

         for(int j = 0; j < n; j++)

            if ((s[j] == 0 ) && (min > dist[j]))

            {
```

```
            min = dist[j];
            w = j;
          }
       s[w]=1;
       //Update the shortest path of neighbour of w
       for(int v=0;v<n;v++)
          if(s[v]==0 && cost[w][v]!=999)
          {
             dist[v]= min(dist[v],dist[w]+cost[w][v]);
          }
    }
}
public static void main(String[] args)
{
    int source;
    Scanner s=new Scanner(System.in);
    System.out.println("Enter the no.of vertices");
    n = s.nextInt();
    cost = new int[n][n];
    dist = new int[n];
     //Enter the cost matrix, 999 for no direct edge from i to j
    System.out.println("Enter the cost matrix");
    for(int i=0; i<n; i++)
       for (int j=0; j<n; j++)
          cost[i][j] = s.nextInt();
          System.out.println("Enter the source vertex");
    source = s.nextInt();
```

```
    Dijkstra(source);

        System.out.println(" the shortest distance is...");

    for(int i=0; i<n; i++)

       System.out.println("Cost from "+source+" to "+i+" is " + dist[i]);

  }

}
```

10. Write Java programs to
(a) Implement All-Pairs Shortest Paths problem using **Floyd's algorithm**.
(b) Implement **Travelling Sales Person problem** using Dynamic programming

10. A) Floyd's

```
 public class Floyds

{

   static void floyd(int D[][],int n)

   {

      for(int k=1;k<=n;k++)

         for(int i=1;i<=n;i++)

            for(int j=1;j<=n;j++)

               D[i][j]=min(D[i][j],D[i][k]+D[k][j]);

    }



   static int min(int a, int b)

   {

      return(a < b ? a : b);

   }



   public static void main(String[] args)

   {

      Scanner s=new Scanner(System.in);
```

```
        int n;
        System.out.println("Enter no. of Vertices");
        n = s.nextInt();
        int[][] cost=new int[n+1][n+1];
        System.out.println("Enter the cost matrix");
        for(int i=1;i<=n;i++)
           for(int j=1;j<=n;j++)
              cost[i][j]=s.nextInt();
        floyd(cost,n);

        System.out.println("All pair shortest path");
        for(int i=1;i<=n;i++)
        {
           for(int j=1;j<=n;j++)
              System.out.print(cost[i][j]+"  ");
           System.out.println();
        }
     }
}
```

10.B) TSP

```
import java.util.Scanner;
public class TSP
{    public static void main(String[] args)
     {  int c[][]=new int[10][10], tour[]=new int[10];
        Scanner in = new Scanner(System.in);
        int i, j,cost;
```

```
    System.out.println("Enter the number of cities: ");
    int n = in.nextInt();

    System.out.println("Enter the cost matrix");
    for(i=1;i<=n;i++)
      for(j=1;j<=n;j++)
           c[i][j] = in.nextInt();

    for(i=1;i<=n;i++)
          tour[i]=i;

    cost = tspdp(c, tour, 1, n);
    System.out.println("The accurate path is");
    for(i=1;i<=n;i++)
        System.out.print(tour[i]+"->");
    System.out.println("1");
    System.out.println("The accurate mincost is "+cost);
}

static int tspdp(int c[][], int tour[], int start, int n)
{   int mintour[]=new int[10], temp[]=new int[10], mincost=999, ccost, i, j, k;
    if(start == n-1)
          return (c[tour[n-1]][tour[n]] + c[tour[n]][1]);
    for(i=start+1; i<=n; i++)
    {  for(j=1; j<=n; j++)
      temp[j] = tour[j];
      temp[start+1] = tour[i];
```

```
        temp[i] = tour[start+1];
        if((c[tour[start]][tour[i]]+(ccost=tspdp(c,temp,start+1,n))) < mincost)
        {  mincost = c[tour[start]][tour[i]] + ccost;
            for(k=1; k<=n; k++)
                mintour[k] = temp[k];
        }
      }
   for(i=1; i<=n; i++)
       tour[i] = mintour[i];
   return mincost;
   }
}
```

11. Design and implement in Java to find a **subset** of a given set **S** = {Sl, S2,.....,Sn} of ***n*** positive integers whose SUM is equal to a given positive integer ***d***. For example, if S ={1, 2, 5, 6, 8} and ***d***= 9, there are two solutions {1,2,6}and {1,8}. Display a suitable message, if the given problem instance doesn't have a solution

Sum of Subsets

```
import java.util.Scanner;
public class Subset
{
  private int s[]=new int[10],x[]=new int[10],d,n;
  public void read()
  {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the number of elements:");
    n = sc.nextInt();
    System.out.println("Enter the set in increasing order");
    for(int i=1;i<=n;i++)
```

```
    s[i] = sc.nextInt();
  System.out.println("Enter the subset sum:");
  d = sc.nextInt();
}

public void check()
{
  int sum=0,i;
  for(i=1;i<=n;i++)
    sum+=s[i];
  if(sum<d || s[i]>d)
    System.out.println("No subset possible");
  else
  {
     System.out.println("Solutions are");
     sumofsub(0,1,sum);
  }
 }

 void sumofsub(int m,int k,int r)
 {
  x[k]=1;//Selecting the first weight
  if((m+s[k])==d) // Terminal condition and print
  {
    for(int i=1;i<=k;i++)
    if(x[i]==1)
      System.out.print(" "+s[i]+" ");
```

```
      System.out.println();;
   }
   else if(m+s[k]+s[k+1]<=d) //Create left subtree
         sumofsub(m+s[k],k+1,r-s[k]);

   if((m+r-s[k]>=d) && (m+s[k+1]<=d)) // create right subtree
   {
    x[k]=0; // Backtrack and consider k+1 weight
    sumofsub(m,k+1,r-s[k]);
   }
}

 public static void main(String[] args)
{
   Subset s1=new Subset();
   s1.read();
   s1.check();
}
}
```

12. Design and implement in Java to find all **Hamiltonian Cycles** in a connected undirected Graph G of *n* vertices using backtracking principle.

```
import java.util.*;
class Ham
{
private int adj[][],x[],n;
```

```
public Ham()
{
  Scanner src = new Scanner(System.in);
  System.out.println("Enter the number of nodes");
  n = src.nextInt();
  x = new int[n];
  x[0]=0;
  for (int i=1;i<n; i++)
     x[i]=-1;
  adj=new int[n][n];
  System.out.println("Enter the adjacency matrix");
  for (int i=0;i<n; i++)
    for (int j=0; j<n; j++)
      adj[i][j]=src.nextInt();
}

public void nextValue (int k)
{
  int i=0;
  while(true)
  {
    x[k]=x[k]+1;
    if (x[k]==n)
       x[k]=-1;
    if (x[k]==-1)
       return;
    if (adj[x[k-1]][x[k]]==1)
```

```
    for (i=0; i<k; i++)
      if (x[i]==x[k])
        break;
      if (i==k)
        if (k<n-1 || k==n-1 && adj[x[n-1]][0]==1)
         return;
  }
}

public void getHCycle(int k)
{
  while(true)
  {
    nextValue(k);
    if (x[k]==-1)
      return;
    if (k==n-1)
    {
     System.out.println("\nSolution : ");
     for (int i=0; i<n; i++)
     System.out.print((x[i]+1)+" ");
     System.out.println(1);
     }
    else getHCycle(k+1);
  }
}
```

```
  public static void main(String args[])
  {
   Ham obj = new Ham();
   obj.getHCycle(1);
  }
}
```