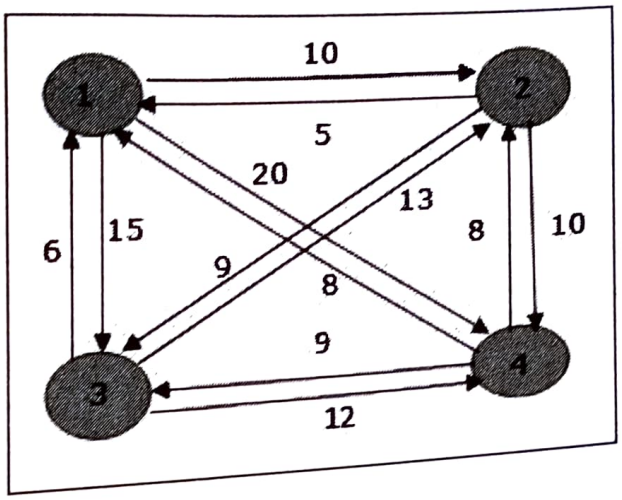
LAB SESSION 08:

Date of the Session: 23/09/21 **Time of the Session:** 7 to _____

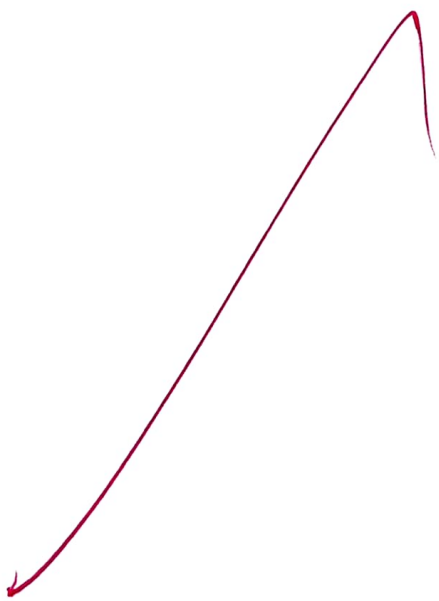
Pre-Lab:

Your father brought you a ticket to world tour. You have a choice to go to three places, your father knows the places you wanted to travel so he made a graph with the measuring distances from home. Now you start from your place (1: Source) to other places as shown in the graph below apply TSP to find shortest path to visit all places and return to home. (Ex: 2: London, 3: Paris, 4: Singapore)

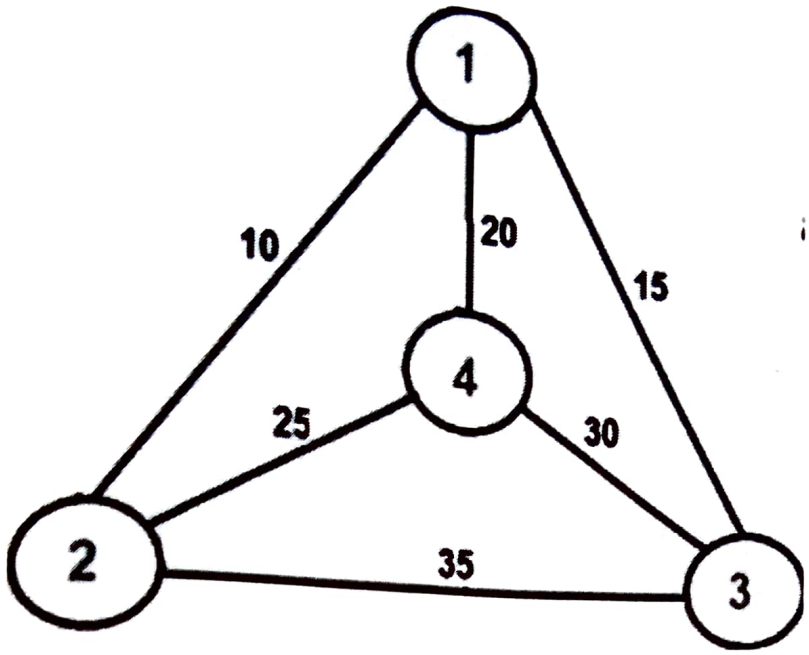



```
from sys import maxsize
from itertools import permutations

V=4

def travellingSalesmanproblem(graph,s):
    vertex=[ ]
    for i in range(V):
        if i!=s:
            vertex.append(i)
    min-path = max.size
    next-permutation=permutations(vertex)
    for i in range next-permutation:
    current-pathweight=0
```

```
    k=s
    for j in p:
    current_path_weight += graph[k][j]
        k=j
    current_path_weight = graph[k][j]
    minpath = min(min_path, current_path_weight)
    return min_path

If __name__ == "__main__":
    graph = [[20, 10, 15, 20], [10, 10, 35, 25]
             [15, 35, 0, 30] [20, 25, 30, 0]]
    print("travelling salesman problem", graph))
```



node as the ... between the places. Here you will consider the 1st point and ending point, Since the route is cyclic you can take any node as starting point and ending point. Find the minimum cost route and remember the Hamiltonian cycle.




```
from sys import maxsize
from itertools import permutations
V=4
def travellingSalesmanProblem(graph,s):
    vertex=[]
    for i in range(V):
        if i!=s:
            vertex.append(i)
    min_path=maxsize
    next_permutation=permutations(vertex)
    for i in next_permutation:
        k=s
        for j in i:
            current_pathweight+=graph[k][j]
            k=j
```

```
current_path_weight += graph[k][i]
min_path = min(min_path, current_path, weight)
return min_path

if __name__ == '__main__':
    graph = [[0, 10, 15, 20], [10, 0, 35, 25],
             [15, 35, 0, 30], [20, 25, 30, 0]]
    s = 0
    print(travellingSalesmanProblem(graph, s))
```

Today, ...ng to visit N cities (numbered 1 through N). There is a direct way to travel between each pair of cities. Each city has a specific temperature; let us denote the temperature in the i-th city by Ci. Your friend has a fixed temperature tolerance D with the following meaning: for each pair of cities an and b, he may travel from city a directly to city b only if |Ca−Cb|≤D, otherwise he would catch a heavy flu because of the sudden change in temperature.

...alesperson — he always travels to new cities

Your friend starts from city 1. Is he able to visit all N cities in such a way that each city is visited exactly once?

Notes:

1. Your friend is not able to travel through a city without visiting it.
2. City 1 is visited at the beginning.

It is not necessary to be able to travel directly to city 1 from the last city Your friend's visits.

Input

2
5 3
3 2 1 4 5
5 4
10 1 3 2 9

Output

Yes
No

```
import java.util.*;
public class Main
{
public static void main(String args[])
{
Scanner sc=new Scanner(System.in);
int t=sc.nextInt();
while(t!=0)
{
int n=sc.nextInt();
int d=sc.nextInt();
int c=0;
int arr[]=new int[n];
for(int i=0;i<n;i++)
If (arr[i]-arr[i+1]<=d)
c++;
```

```
if(c==n-1)
  System.out.println("YES");
else
System.out.println("NO");
}}
```

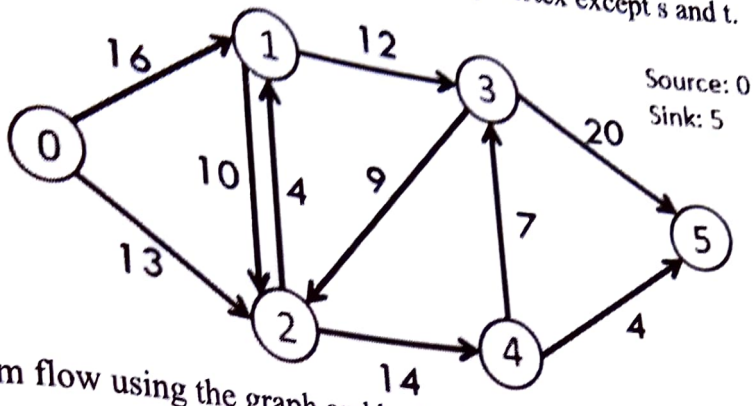
o/p:

YES

NO

In-Lab:

1) Emma has a graph which represents a flow network where every edge has a capacity. Also given two vertices source s and sink t in the graph, fin the maximum possible flow from s to t with following constraints:
   a. Flow on an edge does not exceed the given capacity of the edge.
   b. Incoming flow is equal to outgoing flow for every vertex except s and t.



Find the Maximum flow using the graph and by implementing the code.

```
public class Main{
    static class Graph{
    int vertices.
    int graph[][].

public Graph(int vertex, int[][] graph)
        {
    this.vertices=vertices.
    this.graph= graph;

    }
public int findMaxFlow(int sink)
        {

int [][] residualGraph,
    new int [vertices][vertices])

for(int i=0; i<vertices; i++)
        {

residualGraph[i][j]=
    graph[i][j];
```

```
public static void main(String args[])
    {
    int vertices=6;
int graph[] ={{ 0,16,13, 0, 0, 0},
              { 0, 0, 10,12, 0, 0},
              { 0, 4, 0, 0, 14, 0},
              { 0, 0, 9, 0, 0, 20},
              { 0, 0, 0, 7, 0, 4},
              { 0, 0, 0, 0, 0, 0}
              };

    Graph g=new Graph(vertices graph)
        int source=0;
        int dest=5;
int maxflow = g.findmaxFind(source,dest)
        System.out.println("source +dest);
        }}
```

o/p: maxflow is 23

2) For the above given graph Emma wants an s-t to cut that requires the source s and the sink t to be in different subsets, and it consists of edges going from the source's side to the sink's side. So, she wants you to find the minimum capacity s-t cut of the given network. Expected output is all the edges of minimum cut.

```
from collections import defaultdict
class Graph:
    def __init__(self,graph):
        self.graph=graph
        self.ROW=graph= [i[:] for in graph]
        self.Row=len(graph)
        self.col= len(graph[0])
    def Bfs(self,s,t,parent)
        visited = [fals] =(self Row)
        queue=[]
    queue.append(s)
    visited [s] =True
    while queue;
        v= queue.pop(0)
    for ind,val in enterotely(self.graph[v])
        if visitedLine == fals and
                val>0;
    queue.append(ind)
    visited End : True
    parent[ind]=0
    return True.
```

```
if visited[t]:
    return visited[t]
else:
    return False
parent = [-1] * (self.row)
max_flow = 0
while self.BFS(source, sink, parent):
    path_flow = float("INF")
    s = sink
    while(s != source)
    path_flow = min(path_flow, self.graph,
                    (parent[s][s])
    s = parent[s]
    max_flow += path_flow
    v = sink
    while (v != source):
        u = parent[v]
        self.graph[u][v] = path_flow
        self.graph[v][u] = path_flow
        v = parent[v]
print(str[i] + "_" + str[j])
```

11) **Hogwarts has yet again declared The Triwizard tournament. Harry must pass this round to get to the next one. Each participant will be given a graph with vertices as shown below. Each vertex is a dungeon, and a golden egg is placed in the root dungeon i.e., the root vertex. Help harry find the dungeon with the golden egg using traversing or searching tree or graph data structure. (P.S: To pass this round Harry must write a code).**

```
public class main
{
public static void main(String args[])
    {
BinaryTree bt = BinaryTree.create();
 bt.inorder();
  }}

class BinaryTree{
  static class TreeNode
    {
  int data;
   TreeNode left,right;
 TreeNode(int value){
     this.data=value;
      left=right=null;
  }}
  TreeNode root;
public void inodes(TreeNode node)
  {
 if(node==null)
  return;
 }
```

```
public static BinaryTree create()
{
    BinaryTree tree=new BinaryTree();
    tree.root=root;
    tree.root.left= New TreeNode("20");
    tree.root.left.Right=New TreeNode("50");
    return tree;
}
```