**Ministerul Educaţiei Tineretului şi Sportului al Republicii Moldova**

**Universitatea Tehnică a Moldovei**

**RAPORT**

**LA MATEMATCA DISCRETA**

Tema: ALGORITMUL DETERMINĂRII GRAFULUI DE ACOPERIRE

Lucrarea de Laborator nr. 6

A efectuat Studentul grupei \_SI-212\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_Vozian Vladimir\_\_\_\_

*semnătura nume, prenume*

A verificat \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*nume, prenume profesor*

**Chisinău 2021**

**SCOPUL LUCRĂRII:**

### Studierea algoritmului de determinare a grafului de acoperire şi elaborarea programelor care vor realiza acest algoritm.

1. **SARCINA DE BAZĂ**
2. Elaboraţi organigrama algoritmului şi programul procedurii de determinare a grafului de acoperire cu posibilităţi de pornire a procedurii din oricare vârf al grafului.
3. Utilizând procedurile de introducere a grafului în memoria CE din lucrarea Nr. 1, elaboraţi un program cu următoarele facilităţi:

* introducerea grafului care este dat sub formă de matrice de incidenţă, adiacenţă sau listă de adiacenţă;
* determinarea grafului de acoperire, pornind de la un vârf arbitrar;
* extragerea informaţiei la display sau imprimantă în oricare din formele numite.

import matplotlib.pyplot as plt

import pandas as pd

import networkx as nx

from networkx.algorithms import tree

import numpy as np

input\_data = pd.read\_csv('test.csv', index\_col=0)

G = nx.Graph(input\_data.values)

pos=nx.circular\_layout(G)

nx.draw\_networkx(G,pos)

labels = nx.get\_edge\_attributes(G,'weight')

nx.draw\_networkx\_edge\_labels(G,pos,edge\_labels=labels)

plt.show()

nodes = G.nodes

all\_edges = G.edges

def Adjacency\_matrix():

    print("Matricea de adiacenta")

    G.add\_nodes\_from(nodes)

    G.add\_edges\_from(all\_edges)

    A = np.array(nx.adjacency\_matrix(G,nodelist=sorted(G.nodes()),weight="weight").todense())

    print(A)

def Graph\_list():

    class Graph:

        def \_\_init\_\_(self,Nodes):

            self.nodes = Nodes

            self.adj\_list = {}

            for node in self.nodes:

                self.adj\_list[node] = []

        def add\_edge(self,u,v):

            self.adj\_list[u].append(v)

            self.adj\_list[v].append(u)

        def adds\_zero(self):

            for node in self.nodes:

                self.adj\_list[node].append("0")

        def print\_adj\_list(self):

            for node in self.nodes:

                print(node,"->",self.adj\_list[node])

    print("Lista de adiacenta")

    graph = Graph(nodes)

    for u,v in all\_edges:

        graph.add\_edge(u,v)

    graph.adds\_zero()

    graph.print\_adj\_list()

def MST():

    mst = tree.minimum\_spanning\_edges(G,weight="weight", algorithm="prim", data=False)

    edgelist = list(mst)

    sorted(sorted(weight) for weight in edgelist)

    H = nx.DiGraph()

    H.add\_edges\_from(edgelist)

    nx.draw(H,pos=nx.circular\_layout(H) ,with\_labels = True,arrows=True)

    plt.show()

def Menu():

    while True :

        choice = input("""

        1: Graful de acoperire

        2: Matricea

        3: Lista

        0: Iesire

        Alegeti o optiunea:   """)

        if choice == "1":

            MST()

        elif choice == "2":

            Adjacency\_matrix()

        elif choice =="3":

            Graph\_list()

        elif choice == "0":

            quit()

        else :

            print("Alegeti va rog variantele din meniu (1/2/3/0)")

Menu()

## Exemple de executie a programului: Graful a fost introdus de la tastatură dar citit din fișier.

## 

## 

## 

## Deci graful de acoperire:

## 

## 

## Matricea de adiacență:

## 

## Lista de adiacență:

## 

## Concluzii:

Lucrarea dată era efectuată in python, sa exersat asupra cunoștiințelor determinării grafului de acoperire cu efectuarea lui prin program, am folosit o bibliotecă importată care automat determină și calculează graful de acoperire și l-am implementat in biblioteca pentru afișarea grafului.