Ministerul Educaţiei, Culturii și Cercetării
Universitatea Tehnică a Moldovei

Departamentul Ingineria Software și Automatică

RAPORT

Lucrarea de laborator nr. 2
la MATEMATICI SPECIALE

Tema: ALGORITMUL DE CĂUTARE ÎN LĂRGIME

A efectuat: Cătălin Pleșu
st. gr. TI-206

A verificat: Lisnic Inga

Chișinău – 2021

Cuprins

1. Scopul și obiectivele lucrării

- Studierea algoritmului de căutare în lărgime;
- Elaborarea programului de căutare în lărgime.

2. Sarcina lucrării

- Elaboraţi procedura care va realiza algoritmul de parcurgere a grafului în lărgime;
- Folosind procedurile din lucrările precedente, elaboraţi programul care va permite:
  - introducerea grafului în calculator;
  - parcurgerea grafului în lărgime;
  - extragerea datelor la display şi printer.

3. Considerații teoretice – algoritmul de căutare în lățime

Parcurgerea grafului în lărgime, ca şi parcurgerea în adâncime, va garanta vizitarea fiecărui vârf al grafului exact o singură dată, însă principiul va fi altul. După vizitarea vârfului iniţial, de la care va începe căutarea în lărgime, vor fi vizitate toate vârfurile adiacente cu vârful dat, apoi toate vârfurile adiacente cu aceste ultime vârfuri ş.a.m.d. până vor fi vizitate toate vârfurile grafului. Evident, este necesar ca graful să fie conex. Această modalitate de parcurgere a grafului (în lărgime sau postordine), care mai este adesea numită parcurgere în ordine orizontală, realizează parcurgerea vârfurilor de la stânga la dreapta, nivel după nivel.

Vom nota că procedura parcurgerii grafului în lărgime permite să realizăm arborele de căutare şi în acelaşi timp să construim acest arbore. Cu alte cuvinte, se va rezolva problema determinării unei rezolvări sub forma vectorului (a_1, a_2,...) de lungime necunoscută, dacă este cunoscut că există o rezolvare finită a problemei.

Algoritmul pentru cazul general este analogic cu cel pentru un graf în formă de arbore cu o mică modificare care constă în aceea că fiecare vârf vizitat va fi marcat pentru a exclude ciclarea algoritmului.

4. Întrebări de control

- În ce constă parcurgerea arborelui şi a grafului în lărgime?
    - Parcurgerea grafului sau arborelui este un proces care consă în vizitarea fiecărui vârf o singruă dată. Parcurgerea în lățime presupune vizitarea primului vârf și a vârfurilor adiacente lui, api vizitarea vârfurilor adiacente pentru aceste vârfuri adiacente în ordine, ș.a.m.d. până vor fi vizitatea toate vârfurile.
- Care este diferența dintre parcurgerea în lărgime a unui arbore şi a unui graf arbitrar?
    - Este necesar ca graful să fie conex. Spre deosebire de copaci, graficele pot conține cicluri, așa că putem ajunge din nou la același vârf. Pentru a evita procesarea unui vârf de mai multe ori, folosim un tablou boolean vizitat.
- Ce fel de structuri de date se vor utiliza în algoritmul de căutare în lărgime?
    - Personal am utilizat coada deoarece îmi permite să înscriu elemente în ordinea vizitării lor și să extrag primul element sosit.
- Exemplificaţi algoritmul căutării în lărgime.
    - Metoda parcurgereaInLatime din singura clasă a programului.

5. Codul programului

```
from collections import defaultdict
import networkx as nx
import matplotlib.pyplot as plt
import json

# pentru coada si stiva voi folosi lista pop append si voi sterge elementu 1 cand am nevoie de
el


class GRAF:
    def __init__(self):
        self.graf = defaultdict(list)

    def adaugaArc(self, initial, terminal):
        self.graf[initial].append(terminal)

    def citirea(self):
        # <- pote fi interpretat si ca comentariu
        print("citirea listei de adiacenta")
        self.graf = defaultdict(list)
        i = 1
        while True:
            print("pentru a termina tastati ( q )")
            print("aveti muchia", i, "cu extremitatea initiala")
            initial = input()
            if initial == "q":
                break
            print("si extremitatea terminala")
            terminal = input()
            if terminal == "q":
                break
            self.adaugaArc(int(initial), int(terminal))
            i += 1
        self.curatare()

    def curatare(self):
        self.graf = dict(sorted(self.graf.items()))
        for v in [*self.graf]:
```

```
            self.graf[v].sort()
            self.graf[v] = list(dict.fromkeys(self.graf[v]))

    def afiseazaLista(self):
        print("lista de adiacenta")
        for k in [*self.graf]:
            print(k, "-", end=" ")
            for v in self.graf[k]:
                print(v, ",", end="")
            print("0")

    def parcurgereaInLatime(self, start):
        for k in [*self.graf]:
            for v in self.graf[k]:
                if not v in self.graf:
                    self.graf[v] = []
        vizitat = {}
        for k in [*self.graf]:
            vizitat[k] = False
        coada = []
        parcurgerea = []
        coada.append(start)
        vizitat[start] = True
        while coada:
            aici = coada.pop(0)
            parcurgerea.append(aici)
            for varf in self.graf[aici]:
                if not vizitat[varf]:
                    coada.append(varf)
                    vizitat[varf] = True
        capet = False
        for hz in [*vizitat]:
            if not vizitat[hz]:
                capet = True
        if capet:
            print("incepand cu varful", start,
                  "graful nu poate fi parcurs in latime")
            print(vizitat)
        else:
```

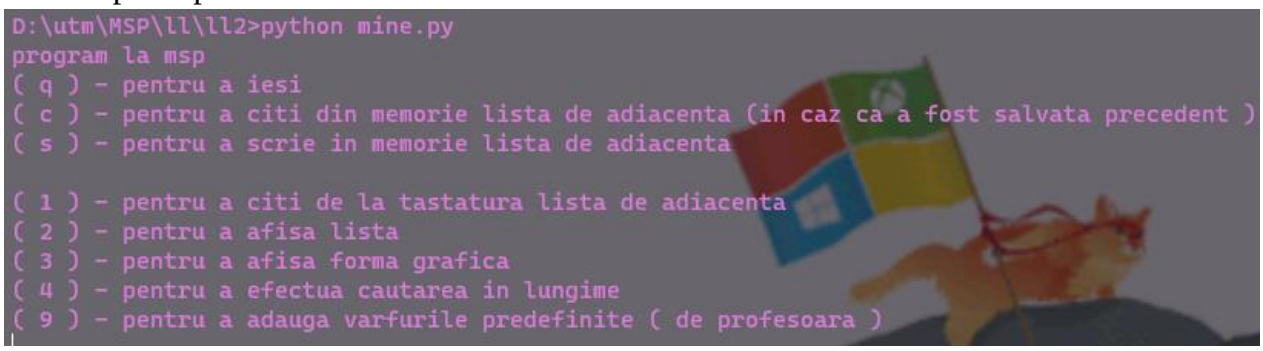
```
        print("parcurgerea in latime incepand din",start)
        for v in parcurgerea:
            print(v, end=" ")
        print()

    def parcugereaAdancime(self, start):
        for k in [*self.graf]:
            for v in self.graf[k]:
                if not v in self.graf:
                    self.graf[v] = []
        vizitat = {}
        for k in [*self.graf]:
            vizitat[k] = False
        stiva = []
        parcurgerea = []
        parcurgerea.append(start)
        stiva.append(start)
        vizitat[start] = True
        while stiva:
            aici = stiva[len(stiva)-1]
            if not vizitat[aici]:
                parcurgerea.append(aici)
                vizitat[aici] = True
            gata = 0
            for varf in self.graf[aici]:
                if not vizitat[varf]:
                    stiva.append(varf)
                    break
                else:
                    gata += 1
            if not self.graf[aici]:
                stiva.pop()
            elif gata == len(self.graf[aici]):
                stiva.pop()
        capet = False
        for hz in [*vizitat]:
            if not vizitat[hz]:
                capet = True
        if capet:
```

```
            print("incepand cu varful", start,
                  "graful nu poate fi parcurs in adancime")
            print(vizitat)
        else:
            print("parcurgerea in adancime incepand din",start)
            for v in parcurgerea:
                print(v, end=" ")
            print()

    def salveaza(self):
        json.dump(self.graf, open("a.txt", 'w'))

    def impota(self):
        self.graf = json.load(open("a.txt"))
        self.graf = {int(k): [int(i) for i in v] for k, v in self.graf.items()}

    def deseneazaGraful(self):
        g = nx.DiGraph()
        for i in [*self.graf]:
            for j in self.graf[i]:
                g.add_edge(i, j)
        nx.draw(g, with_labels=True)
        plt.draw()
        plt.show()

    def START(self):
        print("program la msp")
        while True:
            print("( q ) - pentru a iesi")
            print(
                "( c ) - pentru a citi din memorie lista de adiacenta (in caz ca a fost salvata
precedent )")
            print("( s ) - pentru a scrie in memorie lista de adiacenta")
            print("")
            print("( 1 ) - pentru a citi de la tastatura lista de adiacenta")
            print("( 2 ) - pentru a afisa lista")
            print("( 3 ) - pentru a afisa forma grafica")
            print("( 4 ) - pentru a efectua cautarea in lungime")
            print("( 9 ) - pentru a adauga varfurile predefinite")
```

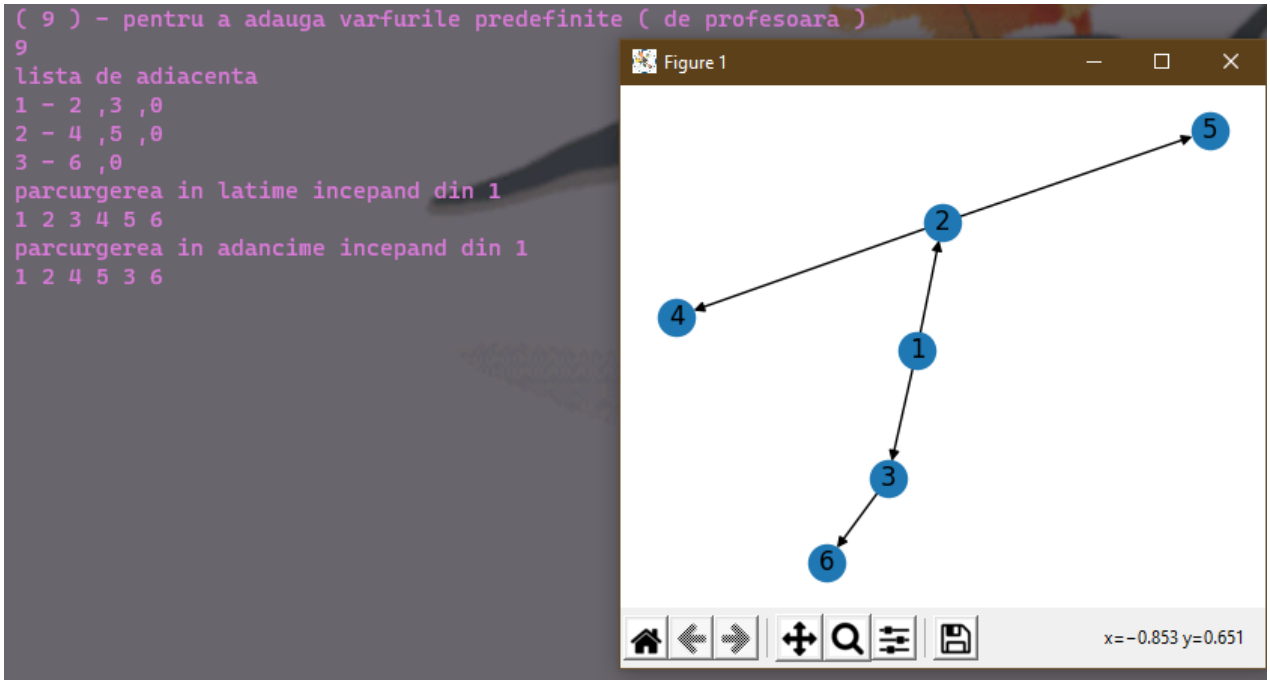
```
            o = input()
            if o == "q":
                break
            elif o == "c":
                self.impota()
            elif o == "1":
                self.citirea()
            elif o == "s":
                self.salveaza()
            elif o == "2":
                self.afiseazaLista()
            elif o == "3":
                self.deseneazaGraful()
            elif o == "4":
                print(self.graf)
                print("dati varful pentru care doriti sa efectuati cautarea in latime")
                v = input()
                self.parcurgereaInLatime(int(v))
            elif o == "9":
                self.adaugaArc(1, 2)
                self.adaugaArc(1, 3)
                self.adaugaArc(2, 4)
                self.adaugaArc(2, 5)
                self.adaugaArc(3, 6)
                self.afiseazaLista()
                self.parcurgereaInLatime(1)
                self.parcugereaAdancime(1)
                self.deseneazaGraful()


graf = GRAF()
graf.START()
```

6. Testarea programului

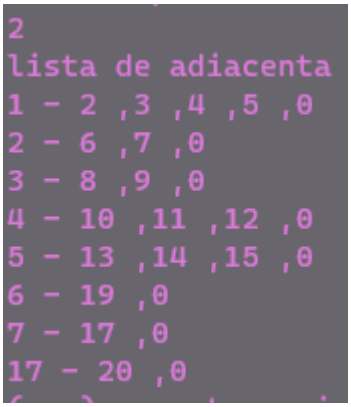
a. Meniul principal :



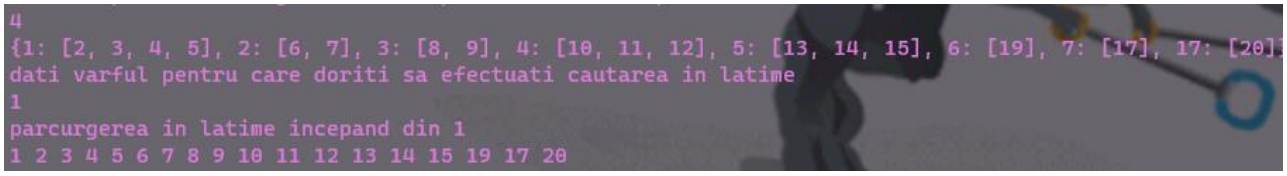
b. Testarea pe arborele prezentat de doamna Lisnic Inga la laborator :



c. Apoi am introdus un graf arbitrar pentru a testa programul pe el :

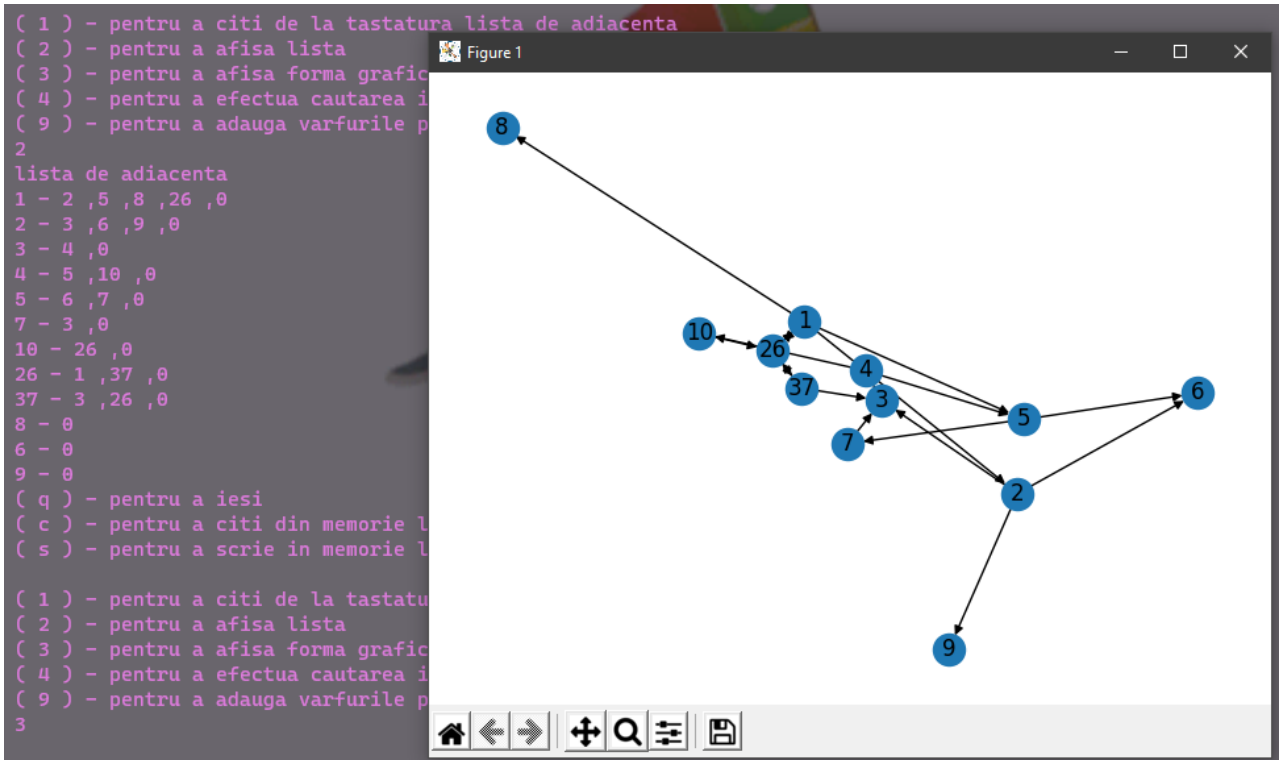


d. Am încercat să parcurg graful în lățime iar rezultatul este urmatorul :



Însă acest graf nu este suficient de complex așa că voi mai crea câteva exemple.

e. Aceast graf conține niște muchii mai haotice :



iar parcurgere lui în lățime este următoarea :

7. Concluzii

f. Datorită efecutării acestui program am însușit algoritmii de parcurgere a grafului și arborelui în lațime cât și în adâncime.
g. Am elaborat programul care să îndeplinească algoritmii respectivi.
h. M-am inițiat în OOP fiind că am utilizat o clasă care conține toate metodele din acest program, totuși probabil nu cea mai bună practică.
i. Mi-am aprofundat cunoștințele în python, aflând despre lisete, seturi și tuple.
j. Am început să lucrez mai eficient.
k. Am utilizat cunoștințe obținute la efectuarea laboratorului precedent.