Ministerul Educaţiei, Culturii și Cercetării
Universitatea Tehnică a Moldovei

Departamentul Ingineria Software și Automatică

RAPORT

Lucrarea de laborator nr. 3
la MATEMATICI SPECIALE

Tema: ALGORITMI DE DETERMINARE A DRUMULUI MINIM ȘI MAXIM

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

A verificat: Lisnic Inga

Chișinău – 2021

Cuprins

1. Scopul și obiectivele lucrării

- Studierea algoritmilor de determinare a drumurilor minime și maxime într-un graf.
- Elaborarea programelor de determinare a drumului minim și maxime într-un graf ponderat.

2. Sarcina lucrării

- Elaborarea procedurii de introducere a unui graf ponderat;
- Elaborarea procedurii de determinare a drumului minim;
- Elaborarea procedurii de determinare a drumului maxim;
- Realizarea uni program cu următoarele funcţii:
  - ❖ introducerea grafului ponderat cu posibilităţi:
    - ✓ de analiză sintactică şi semantică
    - ✓ de corectare a informaţiei;
  - ❖ determinarea drumului minim;
  - ❖ determinarea drumuluimaxim;
  - ❖ extragerea informaţiei la display şi printer.

3. Întrebări de control

1. Ce se numeşte graf ponderat?

Un graf ponderat este un grafic în care fiecare arc primeşte o pondere sau o etichetă numerică de obicei pozitivă.

2. Definiţi noţiunea de distanţă.

Drumul care uneşte două vârfuri concrete şi are lungimea cea mai mare se numeşte distanţă.

3. Descrieţi etapele principale ale algoritmului Ford.

Permite determinarea drumului minim (maxim) care începe cu un vârf iniţial xi până la oricare vârf al grafului G. Dacă prin Lij se va nota ponderea arcului (xi, xj) atunci algoritmul conţine următorii paşi:

1) Fiecărui vârf xj al grafului G se va ataşa un număr foarte mare Hj(∞)(sau foarte mic în cazul drumului **maxim**). Vârfului iniţial i se va ataşa Ho = 0;
2) Se vor calcula diferenţele Hj - Hi pentru fiecare arc (xi, xj). Sunt posibile trei cazuri:

a) Hj - Hi < Lij,

b) Hj - Hi = Lij,

c) Hj - Hi > Lij.

Cazul "c" permite micşorarea distanţei dintre vârful iniţial şi xj din care cauză se va realiza Hj = Hi + Lij.

Iar cazul "a" (se utilizeaza la drumul **maxim**) permite marirea distanţei dintre vârful iniţial şi xj, adica Hj prin modificarea: Hj = Hi + Pij.

Pasul 2 se va repeta atâta timp cât vor mai exista arce pentru care are loc inegalitatea “c”. La terminare, etichetele H_i vor defini distanţa de la vârful iniţial până la vârful dat x_i.

3) Acest pas presupune stabilirea secvenţei de vârfuri care va forma drumul minim. Se va pleca de la vârful final xj spre cel iniţial. Predecesorul lui xj va fi considerat vârful xi pentru care va avea loc Hj - Hi = Lij. Dacă vor exista câteva arce pentru care are loc această relaţie se va alege la opţiune.

4. Care sunt momentele principale în algoritmul Bellman-Kalaba?

Permite determinarea drumului minim dintre oricare vârf al grafului până la un vârf, numit vârf final.

Etapa iniţială presupune ataşarea grafului dat G a unei matrice ponderate de adiacenţă, care se va forma în conformitate cu următoarele:

1. $M(i,j) = L_{ij}$, dacă există arcul (x_i, x_j) de pondere L_{ij};
2. $M(i,j) = \infty$, unde ∞ este un număr foarte mare (de tip întreg maximal pentru calculatorul dat)(la calcularea **drumului maxmi** acest număr este unul foarte mic), dacă arcul (x_i, x_j) este lipsă;
3. $M(i,j) = 0$, dacă $i = j$.

La etapa a doua se va elabora un vector V_0 în felul următor:

1. $V_{0(i)} = L_{in}$, dacă există arcul (x_i, x_n), unde x_n este vârful final pentru care se caută drumul minim, L_{in} este ponderea acestui arc;
2. $V_{0(i)} = \infty$, dacă arcul (x_i, x_n) este lipsă;
3. $V_{0(i)} = 0$, dacă $i = j$.

Algoritmul constă în calcularea iterativă a vectorului V în conformitate cu următorul procedeu:

1. $V_{k(i)} = min(max$ *-cand se calculeaza drumul* ***maxim***$)\{V_{k-1}; L_{ij}+V_{k-1(j)}\}$, unde $i = 1, 2,\ldots, n - 1, j = 1, 2,..., n;\ i<>j;$
2. $V_{k(n)} = 0$.

Când se va ajunge la $V_k = V_{k-1}$ - STOP.

Componenta cu numărul i a vectorului V_k cu valoarea diferită de zero ne va da valoarea minimă a drumului care leagă vârful i cu vârful n.

5. Prin ce se deosebeşte algoritmul Ford de algoritmul Bellman-Kalaba?

Algoritmul Ford se deosebeşte de algoritmul Bellman-Kalaba pri:

- Utilizarea etichetelor pe când algoritmul Bellman-Kalaba utilizează o matrice şi mai mulţi vectori.
- Metoda de utilizare a acestor lucruri specifice.

6. Cum se va stabili succesiunea vârfurilor care formează drumul minim, maxim ?

Daca am utilizat algoritmul ford indiferent că avem drumul minim sau maxim v-om începe de la ultimul vârf. Din eticheta acestui vârf v-om scădea ponderea arcelor ce vin la el, iar daca rezultatul obținut este egal cu eticheta vârfului de incidență pentru arcul dat v-om ști că acest vârf se află în fața ultimului vârf. V-om repeta această procedură pentru toate vârfurile care au fost descoperite astfel în cele din urmă obținând un drum sau mai multe.

Dacă am utilizat algoritmul bellman-kalaba v-om opera cu ultimul vector, în prima poziție este lungimea drumului dat, v-om parcurge primul rand al matricii si vectorul concomitent daca valoare ponderii i + vectoru i = mărimea drumului, atunci acest vârf se află după primul vârf, această procedură se repetă coborândune câte un rand în jos și utilizând mărimea drumului care se află la poziția vârfului dat.

Însă cel mai bine această logică poate fi observată în codul programului.

4. Codul programului

```
from collections import defaultdict
import networkx as nx
import matplotlib.pyplot as plt
import json
import random
from prettytable import PrettyTable
import math
import numpy as np


class GRAF:
    def __init__(self):
        self.graf = defaultdict(list)
        self.drumMinim = int()
        self.drumMaxim = int()
        self.minListe = []
        self.maxListe = []
    def adaugaArc(self, initial, terminal, ponderea):
        self.graf[initial].append((terminal, ponderea))

    def citirea(self):
        self.graf = defaultdict(list)
        print("citirea listei de adiacenta")
        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
            print("ponderea este")
            ponderea = input()
            if ponderea == "q":
                break
```

```
            self.adaugaArc(int(initial), int(terminal), int(ponderea))
            i += 1
        self.curatare()

    def curatare(self):
        for v in [*self.graf]:
            self.graf[v].sort()
            self.graf[v] = list(dict(self.graf[v]).items())
            self.graf[v] = [(varf, pond)
                            for (varf, pond) in self.graf[v] if varf !=
v]
            for c in self.graf[v]:
                if c[0] not in [*self.graf]:
                    self.graf[c[0]] = []
        self.graf = dict(sorted(self.graf.items()))

    def afiseazaLista(self):
        print("lista de adiacenta")
        for k in [*self.graf]:
            print(k, "-", end=" ")
            for v in self.graf[k]:
                print(str(v[0])+"("+str(v[1])+")", end=", ")
            print("0")

    def salveaza(self):
        print("denumirea fisierului")
        f = input()
        json.dump(self.graf, open(f+".json", 'w'))

    def impota(self):
        print("denumirea fisierului")
        f = input()
        self.graf = json.load(open(f+".json"))
        # self.graf = {int(k): [int(i) for i in v] for k, v in self.graf
.items()}
        temp = defaultdict(list)
        for key in [*self.graf]:
            for pereche in self.graf[key]:
                temp[int(key)].append(tuple(pereche))
```

```
        self.graf = temp
        self.curatare()

    def determinaFordMinim(self):
        ford = PrettyTable()
        ford.field_names = ["i, j", "L ij",
                            "Hj - Hi = L ij", "Distanta", "Eticheta"]
        H = defaultdict(int)
        for k in [*self.graf]:
            H[k] = math.inf
        for k in [*self.graf]:
            H[k] = 0
            break
        for k in [*self.graf]:
            for v in self.graf[k]:
                line = []
                line.append(str(k)+", "+str(v[0]))
                line.append(v[1])
                eq = "H"+str(v[0])+" - H"+str(k)+" = "
                eqn = str(H[v[0]])+" - "+str(H[k])+" = "
                rs = str(H[v[0]]-H[k])
                line.append(eq + eqn + rs + str((" < ", " > ")
                                                [H[v[0]]-H[k] > v[1]])
+str(v[1]))
                if H[v[0]] - H[k] > v[1]:
                    distanta = str(H[k])+"+"+str(v[1])
                    eticheta = "H"+str(v[0])+" = " + str(H[k]+v[1])
                else:
                    distanta = "--------"
                    eticheta = "neschimbata"
                line.append(distanta)
                line.append(eticheta)
                if H[v[0]]-H[k] > v[1]:
                    H[v[0]] = H[k]+v[1]
                ford.add_row(line)
        print(ford.get_string(title="Ford drum minim"))
        # tabelulH = PrettyTable()
        # tabelulH.field_names = ["H", "val"]
        # H = dict(sorted(H.items()))
```

```
        # for k in [*H]:
        #     tabelulH.add_row(["H"+str(k), str(H[k])])
        # print(tabelulH)
        self.drumMinim = int(H[list(H).pop()])
        self.determinaFordInput(H)
    def determinaFordMaxim(self):
        ford = PrettyTable()
        ford.field_names = ["i, j", "L ij",
                            "Hj - Hi = L ij", "Distanta", "Eticheta"]
        H = defaultdict(int)
        for k in [*self.graf]:
            H[k] = -math.inf
        for k in [*self.graf]:
            H[k] = 0
            break
        for k in [*self.graf]:
            for v in self.graf[k]:
                line = []
                line.append(str(k)+", "+str(v[0]))
                line.append(v[1])
                eq = "H"+str(v[0])+" - H"+str(k)+" = "
                eqn = str(H[v[0]])+" - "+str(H[k])+" = "
                rs = str(H[v[0]]-H[k])
                line.append(eq + eqn + rs + str((" < ", " > ")
                                                [H[v[0]]-H[k] > v[1]])
+str(v[1]))
                if H[v[0]] - H[k] < v[1]:
                    distanta = str(H[k])+"+"+str(v[1])
                    eticheta = "H"+str(v[0])+" = " + str(H[k]+v[1])
                else:
                    distanta = "--------"
                    eticheta = "neschimbata"
                line.append(distanta)
                line.append(eticheta)
                if H[v[0]]-H[k] < v[1]:
                    H[v[0]] = H[k]+v[1]
                ford.add_row(line)
        print(ford.get_string(title="Ford drum maxim"))
        self.drumMaxim = int(H[list(H).pop()])
```

```
        self.determinaFordInput(H)
    def determinaFordInput(self,etichete):
        grafInversat = defaultdict(list)
        for k in [*self.graf]:
            for v in self.graf[k]:
                grafInversat[v[0]].append((k,v[1]))
        # print(grafInversat)
        start = list(self.graf).pop(0)
        finish = list(self.graf).pop()
        drumuri = [[finish]]
        temp = []
        drumuriValide =[]
        while drumuri:
            for drum in drumuri:
                if drum[0]==finish and drum[len(drum)-1]==start:
                    drumF = []
                    for f in drum:
                        drumF.insert(0,f)
                    drumuriValide.append(drumF)
                for v in grafInversat[drum[len(drum)-1]]:
                    if etichete[list(etichete).pop()]==self.drumMinim:
                        if etichete[drum[len(drum)-1]] -v[1] =
=etichete[v[0]]:
                            temp.append(drum+[v[0]])
                    else:
                        # print(  etichete[drum[len(drum)-1]] ,etichete[
v[0]] ,v)
                        if    etichete[drum[len(drum)-1 ]] - etichete[v
[0]]  ==v[1]:
                            temp.append(drum+[v[0]])
                            # print(v)
            if not temp:
                break
            drumuri=temp
            temp = []

        if etichete[list(etichete).pop()]==self.drumMinim:
            self.minListe=drumuriValide
            self.afiseazaDrumuri("minim")
```

```
        if etichete[list(etichete).pop()]==self.drumMaxim:
            self.maxListe=drumuriValide
            self.afiseazaDrumuri("maxim")
    def determinaBellmanKalabaMinim(self):
        kalaba = PrettyTable()
        kalaba.field_names = [""]+[*self.graf]
        n = list(self.graf).pop()+1
        bk = np.zeros([n, n])
        for i in range(0, n):
            for j in range(0, n):
                bk[i][j] = -1
        varfuri = sorted(list(self.graf))
        for i in varfuri:
            for j in varfuri:
                bk[i][j] = math.inf
                if i == j:
                    bk[i][j] = 0
        for k in [*self.graf]:
            for v in self.graf[k]:
                bk[k][v[0]] = v[1]
        m = n+1
        bk.resize([m, n])
        for i in range(0, n):
            bk[m-1][i] = bk[i][n-1]

        while not (bk[m-2] == bk[m-1]).all():
            m += 1
            bk.resize([m, n])
            for i in varfuri:
                temp = math.inf
                for j in varfuri:
                    if i != j:
                        if bk[m-2][j]+bk[i][j] < temp:
                            temp = bk[m-2][j]+bk[i][j]
                            bk[m-1][i] = temp

        for i in varfuri:
            lista = [i]
            for j in varfuri:
```

```
                if math.isinf(bk[i][j]):
                    lista.append(bk[i][j])
                else:
                    lista.append(int(bk[i][j]))
            kalaba.add_row(lista)
        for i in range(n, m):
            lista = []
            lista.append("V"+str(i-n+1))
            for j in varfuri:
                if math.isinf(bk[i][j]):
                    lista.append(bk[i][j])
                else:
                    lista.append(int(bk[i][j]))

            kalaba.add_row(lista)
        print(kalaba.get_string(title="Bellman Kalaba drum minim"))
        self.drumMinim = int(bk[m-1][list(varfuri).pop(0)])
        self.determinaKalabaInput(bk[m-1])

    def determinaBellmanKalabaMaxim(self):
        kalaba = PrettyTable()
        kalaba.field_names = [""]+[*self.graf]
        n = list(self.graf).pop()+1
        bk = np.zeros([n, n])
        for i in range(0, n):
            for j in range(0, n):
                bk[i][j] = -1
        varfuri = sorted(list(self.graf))
        for i in varfuri:
            for j in varfuri:
                bk[i][j] = -math.inf
                if i == j:
                    bk[i][j] = 0
        for k in [*self.graf]:
            for v in self.graf[k]:
                bk[k][v[0]] = v[1]
        m = n+1
        bk.resize([m, n])
        for i in range(0, n):
```

```
            bk[m-1][i] = bk[i][n-1]

        while not (bk[m-2] == bk[m-1]).all():
            m += 1
            bk.resize([m, n])
            for i in varfuri:
                temp = -math.inf
                for j in varfuri:
                    if i != j:
                        if bk[m-2][j]+bk[i][j] > temp:
                            temp = bk[m-2][j]+bk[i][j]
                            bk[m-1][i] = temp

        for i in varfuri:
            lista = [i]
            for j in varfuri:
                if math.isinf(bk[i][j]):
                    lista.append(bk[i][j])
                else:
                    lista.append(int(bk[i][j]))
            kalaba.add_row(lista)
        for i in range(n, m):
            lista = []
            lista.append("V"+str(i-n+1))
            for j in varfuri:
                if math.isinf(bk[i][j]):
                    lista.append(bk[i][j])
                else:
                    lista.append(int(bk[i][j]))

            kalaba.add_row(lista)
        print(kalaba.get_string(title="Bellman Kalaba drum maxim"))
        self.drumMaxim = int(bk[m-1][list(varfuri).pop(0)])
        self.determinaKalabaInput(bk[m-1])


    def determinaKalabaInput(self,rand):
        start = list(self.graf).pop(0)
        finish = list(self.graf).pop()
```

```
        drumuri = [[start]]
        temp = []
        drumuriValide =[]
        while drumuri:
            for drum in drumuri:
                if drum[0]==start and drum[len(drum)-1]==finish:
                    drumuriValide.append(drum)
                for v in self.graf[drum[len(drum)-1]]:
                    if rand[drum[len(drum)-1]]==rand[v[0]]+v[1]:
                        temp.append(drum+[v[0]])
            if not temp:
                break
            drumuri=temp
            temp = []
        if rand[start]==self.drumMinim:
            self.minListe=drumuriValide
            self.afiseazaDrumuri("minim")
        if rand[start]==self.drumMaxim:
            self.maxListe=drumuriValide
            self.afiseazaDrumuri("maxim")
    def afiseazaDrumuri(self,drum):
        print("D",drum,"= ", (self.drumMinim,self.drumMaxim)
[drum=="maxim"])
        for l in (self.minListe,self.maxListe)[drum=="maxim"]:
            pref = list(self.graf).pop(0)
            weight = []
            for v in l:
                if v != pref:
                    print(pref, end="")
                    weight = [item for item in self.graf[pref] if v == i
tem[0]]
                    print(" =("+str(weight.pop()[1]) + ")=>", end=" ")
                pref = v
            print(pref)

    def determinareaCatalinHimselfDrumuluiMaxim(self):
        lista = []
        lista2 = []
        start = list(self.graf).pop(0)
```

```
        finis = list(self.graf).pop()
        lista.append([start])
        # for v in self.graf[start]:
        #     lista.append([v[0]])
        listePosibile = []
        while lista:
            for l in lista:
                for v in self.graf[l[len(l)-1]]:
                    lista2.append(l+[v[0]])
            lista = lista2
            # print(lista)
            for l in lista:
                if (l[0] == start and l[len(l)-1] == finis):
                    listePosibile.append(l)
            lista2 = []
        listeValide = []
        # print(listePosibile)
        for l in listePosibile:
            suma = 0
            pref = start
            weight = 0
            for v in l:
                if v != start:
                    weight = [item for item in self.graf[pref] if v in i
tem]
                    suma += weight.pop()[1]
                # print("varf", v, "suma", suma, "weight", weight)
                pref = v
            if suma == self.drumMaxim:
                listeValide.append(l)
        print("D maxim = ", self.drumMaxim)
        for l in listeValide:
            pref = start
            for v in l:
                if v != pref:
                    print(pref, end="")
                    weight = [item for item in self.graf[pref] if v in i
tem]
                    print(" =("+str(weight.pop()[1]) + ")=>", end=" ")
```

```
                pref = v
            print(pref)
        # print(listeValide)

    def deseneazaGraful(self):
        g = nx.DiGraph()
        for i in [*self.graf]:
            for j in self.graf[i]:
                g.add_edge(i, j[0], weight=j[1])
        pos = nx.circular_layout(g)
        edge_labels = {(u, v): d['weight'] for u, v, d in g.edges(data=T
rue)}
        nx.draw(g, pos, with_labels=True, node_size=1700, font_size=40)
        nx.draw_networkx_edge_labels(g, pos, edge_labels=edge_labels,fon
t_size=20)
        plt.savefig('output.png')
        plt.show()

    def removeVertex(self):
        print(self.graf)
        print("varful pe care doriti sa il stergeti :")
        v = int(input())
        for i in [*self.graf]:
            self.graf[i] = [item for item in self.graf[i] if item[0] !=
v]
        self.graf.pop(v, None)

    def removeEdge(self):
        print(self.graf)
        print("varful din care iese muchia :")
        e = int(input())
        print("varful in care intra muchia :")
        i = int(input())
        self.graf[e] = [item for item in self.graf[e] if item[0] != i]

    def addEdge(self):
        print(self.graf)
        print("puteti adauga orice muchie (chiar cu varfuri noi)")
        print("varful din care iese muchia :")
```

```
        e = int(input())
        print("varful in care intra muchia :")
        i = int(input())
        print("ponderea :")
        p = int(input())
        self.adaugaArc(e, i, p)

    def edit(self):
        print("puteti :\nsterge un varf - v\nsterge o muchie - m\nadauga
o muchie - a")
        o = input()
        if o == "v":
            self.removeVertex()
        elif o == "m":
            self.removeEdge()
        elif o == "a":
            self.addEdge()
        self.curatare()

    def START(self):
        print("program la msp")
        while True:
            print("( q ) - pentru a iesi")
            print(
                "( c ) - pentru a citi din memorie lista de adiacenta (i
n 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 adiace
nta")
            print("( 2 ) - pentru a afisa lista")
            print("( 3 ) - pentru a afisa forma grafica")
            print("( 4 ) - FORD minim")
            print("( 5 ) - Kalaba minim")
            print("( 6 ) - FORD maxim")
            print("( 7 ) - Kalaba maxim")
            print("( 8 ) - EDITARE")
            print("( 9 ) - pentru a genera un graf intamplator")
```

```
            o = input()
            if o == "q":
                break
            elif o == "c":
                self.impota()
                self.determinaFordMinim()
                self.determinaBellmanKalabaMinim()
                self.determinaFordMaxim()
                self.determinaBellmanKalabaMaxim()
                self.deseneazaGraful()
                # self.determinareaCatalinHimselfDrumuluiMaxim()
            elif o == "1":
                self.citirea()
            elif o == "s":
                self.salveaza()
            elif o == "2":
                self.afiseazaLista()
            elif o == "3":
                self.deseneazaGraful()
            elif o == "4":
                self.determinaFordMinim()
            elif o == "5":
                self.determinaBellmanKalabaMinim()
            elif o == "6":
                self.determinaFordMaxim()
            elif o == "7":
                self.determinaBellmanKalabaMaxim()
            elif o == "8":
                self.edit()
            elif o == "9":
                self.graf = defaultdict(list)
                for i in range(0, random.randint(7, 15)):
                    self.adaugaArc(random.randint(1, 7), random.randint(
                        1, 7), random.randint(1, 10))
                self.curatare()
                self.afiseazaLista()
            elif o == "h":
                print("graf:",self.graf)
                print("drum minim:",self.drumMinim)
```

```
                print("drumurile minime:",self.minListe)
                print("drum maxim:",self.drumMaxim)
                print("drumurile maxim:",self.maxListe)


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

5. Testarea programului

```
( q ) - pentru a iesi
( c ) - pentru a citi din memorie lista de adiacenta (in
( s ) - pentru a scrie in memorie lista de adiacenta

( 1 ) - pentru a citi de la tastatura lista de adiacenta
( 2 ) - pentru a afisa lista
( 3 ) - pentru a afisa forma grafica
( 4 ) - FORD minim
( 5 ) - Kalaba minim
( 6 ) - FORD maxim
( 7 ) - Kalaba maxim
( 8 ) - EDITARE
( 9 ) - pentru a genera un graf intamplator
```

- Meniul principal.
- Citirea – se citește vârful inițial, vârful terminal iar apoi pondera.
- Putem analiza graful afișându-i lista de adiacență cu ponderea în paranteze.

```
lista de adiacenta
1 - 2(2), 4(1), 0
2 - 3(3), 6(3), 0
3 - 4(4), 6(4), 0
4 - 5(3), 0
5 - 6(1), 0
6 - 0
```

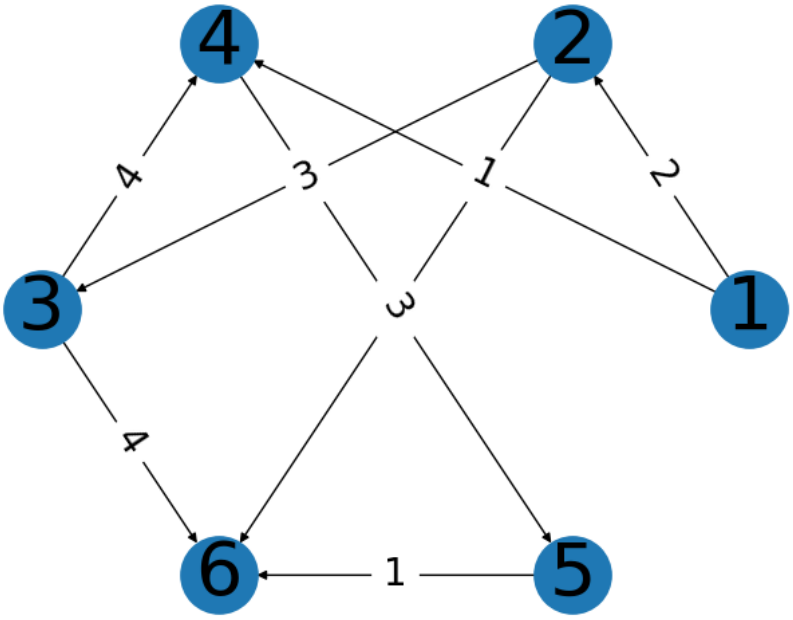
- Corectarea grafului poate fi realizata cu optiunea 8 din meniu, care mai apoi ne ofera 3 posibilități: ștergerea unui vârf sau muchie sau adaugarea unei muchii noi. Aceste opțiuni sunt suficiente pentru a edita orice aspect al grafului.

```
sterge un varf - v
sterge o muchie - m
adauga o muchie - a
```

```
pentru a termina tastati ( q )
aveti muchia 1 cu extremitatea initiala
1
si extremitatea terminala
2
ponderea este
2
```

- Exemple care au fost rezolvate la seminar pentru drumul minim:

```
lista de adiacenta
1 - 2(2), 4(1), 0
2 - 3(3), 6(3), 0
3 - 4(4), 6(4), 0
4 - 5(3), 0
5 - 6(1), 0
6 - 0
```



Ford drum minim

i, j	L ij	Hj - Hi = L ij	Distanta	Eticheta
1, 2	2	H2 - H1 = inf - 0 = inf > 2	0+2	H2 = 2
1, 4	1	H4 - H1 = inf - 0 = inf > 1	0+1	H4 = 1
2, 3	3	H3 - H2 = inf - 2 = inf > 3	2+3	H3 = 5
2, 6	3	H6 - H2 = inf - 2 = inf > 3	2+3	H6 = 5
3, 4	4	H4 - H3 = 1 - 5 = -4 < 4	--------	neschimbata
3, 6	4	H6 - H3 = 5 - 5 = 0 < 4	--------	neschimbata
4, 5	3	H5 - H4 = inf - 1 = inf > 3	1+3	H5 = 4
5, 6	1	H6 - H5 = 5 - 4 = 1 < 1	--------	neschimbata

Bellman Kalaba drum minim

	1	2	3	4	5	6
1	0	2	inf	1	inf	inf
2	inf	0	3	inf	inf	3
3	inf	inf	0	4	inf	4
4	inf	inf	inf	0	3	inf
5	inf	inf	inf	inf	0	1
6	inf	inf	inf	inf	inf	0
V1	inf	3	4	inf	1	0
V2	5	3	4	4	1	0
V3	5	3	4	4	1	0

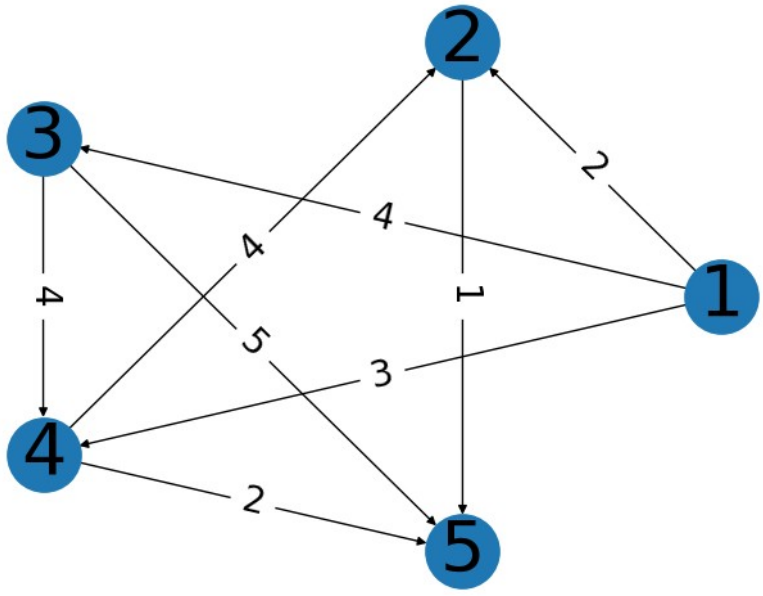
```
D minim =  5
1 =(2)=> 2 =(3)=> 6
1 =(1)=> 4 =(3)=> 5 =(1)=> 6
```

```
lista de adiacenta
1 - 2(2), 3(4), 4(3), 0
2 - 5(1), 0
3 - 4(4), 5(5), 0
4 - 2(4), 5(2), 0
5 - 0
```

Bellman Kalaba drum minim

	1	2	3	4	5
1	0	2	4	3	inf
2	inf	0	inf	inf	1
3	inf	inf	0	4	5
4	inf	4	inf	0	2
5	inf	inf	inf	inf	0
V1	inf	1	5	2	0
V2	3	1	5	2	0
V3	3	1	5	2	0

```
D minim =  3
1 =(2)=> 2 =(1)=> 5
```



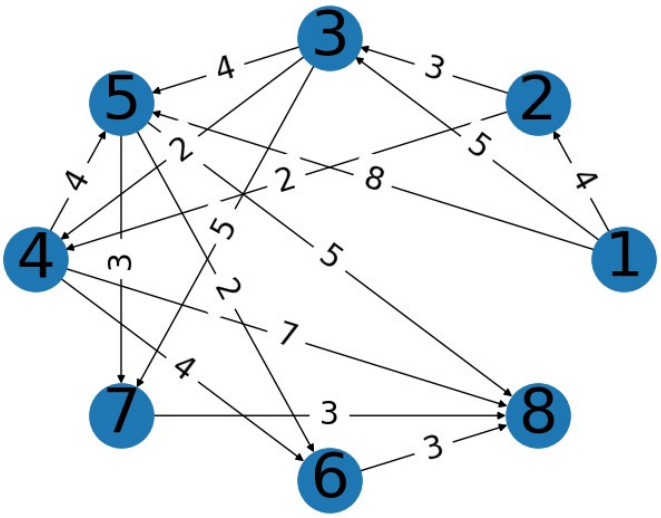
Ford drum minim

i, j	L ij	Hj - Hi = L ij	Distanta	Eticheta
1, 2	2	H2 - H1 = inf - 0 = inf > 2	0+2	H2 = 2
1, 3	4	H3 - H1 = inf - 0 = inf > 4	0+4	H3 = 4
1, 4	3	H4 - H1 = inf - 0 = inf > 3	0+3	H4 = 3
2, 5	1	H5 - H2 = inf - 2 = inf > 1	2+1	H5 = 3
3, 4	4	H4 - H3 = 3 - 4 = -1 < 4	--------	neschimbata
3, 5	5	H5 - H3 = 3 - 4 = -1 < 5	--------	neschimbata
4, 2	4	H2 - H4 = 2 - 3 = -1 < 4	--------	neschimbata
4, 5	2	H5 - H4 = 3 - 3 = 0 < 2	--------	neschimbata

```
D minim =  3
1 =(2)=> 2 =(1)=> 5
```

- Exemple care au fost rezolvate în teoria de pe else pentru drumul maxim:

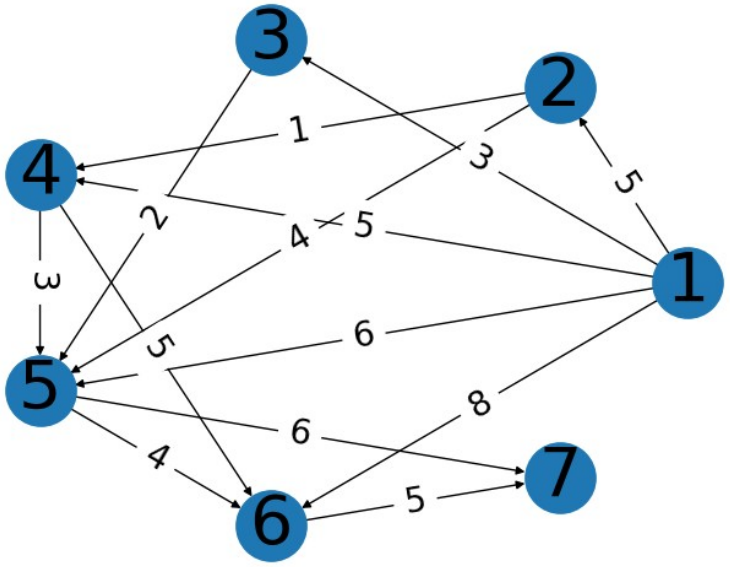
```
lista de adiacenta
1 - 2(4), 3(5), 5(8), 0
2 - 3(3), 4(2), 0
3 - 4(2), 5(4), 7(5), 0
4 - 5(4), 6(4), 8(7), 0
5 - 6(2), 7(3), 8(5), 0
6 - 8(3), 0
7 - 8(3), 0
8 - 0
```



Ford drum maxim				
i, j	L ij	Hj - Hi = L ij	Distanta	Eticheta
1, 2	4	H2 - H1 = -inf - 0 = -inf < 4	0+4	H2 = 4
1, 3	5	H3 - H1 = -inf - 0 = -inf < 5	0+5	H3 = 5
1, 5	8	H5 - H1 = -inf - 0 = -inf < 8	0+8	H5 = 8
2, 3	3	H3 - H2 = 5 - 4 = 1 < 3	4+3	H3 = 7
2, 4	2	H4 - H2 = -inf - 4 = -inf < 2	4+2	H4 = 6
3, 4	2	H4 - H3 = 6 - 7 = -1 < 2	7+2	H4 = 9
3, 5	4	H5 - H3 = 8 - 7 = 1 < 4	7+4	H5 = 11
3, 7	5	H7 - H3 = -inf - 7 = -inf < 5	7+5	H7 = 12
4, 5	4	H5 - H4 = 11 - 9 = 2 < 4	9+4	H5 = 13
4, 6	4	H6 - H4 = -inf - 9 = -inf < 4	9+4	H6 = 13
4, 8	7	H8 - H4 = -inf - 9 = -inf < 7	9+7	H8 = 16
5, 6	2	H6 - H5 = 13 - 13 = 0 < 2	13+2	H6 = 15
5, 7	3	H7 - H5 = 12 - 13 = -1 < 3	13+3	H7 = 16
5, 8	5	H8 - H5 = 16 - 13 = 3 < 5	13+5	H8 = 18
6, 8	3	H8 - H6 = 18 - 15 = 3 < 3	--------	neschimbata
7, 8	3	H8 - H7 = 18 - 16 = 2 < 3	16+3	H8 = 19

```
D maxim =  19
1 =(4)=> 2 =(3)=> 3 =(2)=> 4 =(4)=> 5 =(3)=> 7 =(3)=> 8
```

```
lista de adiacenta
1 - 2(5), 3(3), 4(5), 5(6), 6(8), 0
2 - 4(1), 5(4), 0
3 - 5(2), 0
4 - 5(3), 6(5), 0
5 - 6(4), 7(6), 0
6 - 7(5), 0
7 - 0
```



6. Concluzii

- ✓ Datorită efectuării acestui laborator am studiat algoritmul Ford și Bellman-Kalaba.
- ✓ Am studiat variațiile acestor algoritmi pentru drumul minim și maxim.
- ✓ Implementând acești algoritmi am elaborat un program care în urma unor teste s-a dovedit că operează corect.
- ✓ Deși la început mi se părea dificil stabilirea vârfurilor care formează drumul minim sau maxim în cele din urmă am reușit să scriu aceste funcții.
- ✓ Operarea cu fișiere este foarte convenabilă fiind că îmi salvează mult timp pe care l-aș fi utilizat introducând datele manual și este superioară încorporării datelor în codul programului.
- ✓ Un progam care are elemente de grafică este mult mai plăcut de utilizat, acesta fiind motivul pentru care am utilizat librăriile networkx și matplotlib.