Zadatci za vježbu

RJEŠENJA

Zadatak 1.

File Edit Format Run Options Windows Help

```
# 06_01

from matrica import *

##Rješenje zadatka se svodi na to da se u matrici susjedstva
##ispišu indeksi svih elemenata koji u zadanom retku imaju vrijednost 1
n = int(input())
m = int(input())
a = Matrica(n)
for i in range(m):
    brid = input().split()
    r, s = int(brid[0]), int(brid[1])
    a[r, s] = 1
    a[s, r] = 1

k = int(input())

for i in range(n):
    if a[k, i] == 1:
        print(i)
```

Zadatak 2.

File Edit Format Run Options Windows Help

```
# 06_02
from matrica import *

##Potrebno je u matrici susjedstva pronaći redak s najviše jedinica
n = int(input())
m = int(input())
a = Matrica(n)
for i in range(m):
    brid = input().split()
    r, s = int(brid[0]), int(brid[1])
    a[r, s] = 1
    a[s, r] = 1

naj = [0] * n
ind = 0
for i in range(n):
    for j in range(n):
        naj[i] += a[i, j]
    if naj[ind] < naj[i]:
        ind = i
print(ind)
```

Zadatak 3.

File Edit Format Run Options Windows Help

```
# 06_03

from matrica import *

##Potrebno je u matrici susjedstva provjeriti da je broj jedinica
##u svakom retku paran
n = int(input())
m = int(input())
a = Matrica(n)
for i in range(m):
    brid = input().split()
    r, s = int(brid[0]), int(brid[1])
    a[r, s] = 1
    a[s, r] = 1

eulerov = True
for i in range(n):
    broj_jedinica = 0
    for j in range(n):
        broj_jedinica += a[i, j]
    if broj_jedinica % 2 != 0:
        eulerov = False
if eulerov:
    print('Graf je Eulerov')
else:
    print('Graf nije Eulerov')
```

Zadatak 4.

File Edit Format Run Options Windows Help

```
# 06_04

from matrica import *

##Čvor će biti izoliran ako nije povezan niti s jednim drugim čvorom
##tj. u matrici susjedstva postoji barem jedan redak u kojem se nalaze samo 0
n = int(input())
m = int(input())
a = Matrica(n)
for i in range(m):
    brid = input().split()
    r, s = int(brid[0]), int(brid[1])
    a[r, s] = 1
    a[s, r] = 1

izolirani = False
for i in range(n):
    broj_jedinica = 0
    for j in range(n):
        broj_jedinica += a[i, j]
    if broj_jedinica == 0:
        izolirani = True
if izolirani:
    print('U grafu postoji izolirani čvor')
else:
    print('U grafu ne postoji izoliran čvor')
```

Zadatak 5.

File Edit Format Run Options Windows Help

```
# 06_05

from matrica import *
from Queue import *

##Graf ćemo obilaziti BFS-om. U listu ćemo zapisivati boje čvorova
##u trenutku kada čvor prvi puta posjećujemo. Ako se dogodi da kod nekog vrha
##piše jedna boja, a nekim drugim posjetom dobijemo da bi boja trebala biti
##drukčija od te koja piše, znači da graf neće biti dvoobojiv.

def dvoobojiv(graf):
    ##Radi jednostavnosti boje ćemo označavati brojevima 1 i 0, a čvorovi koji
    ##još nisu posjećeni imat će vrijednost boje -1.
    boje = [-1] * graf.n
    r = Queue()
    r.enqueue(0)
    boje[0] = 0
    dvoobojiv = True
    while not r.isEmpty():
        t = r.dequeue()
        for i in range(graf.n):
            if graf[t, i] == 1:
                ##Ako čvor još nije posjećen postavljamo mu boju
                ##na suprotnu boju od boje trenutnog čvora.
                if boje[i] == -1:
                    boje[i] = 1 - boje[t]
                    r.enqueue(i)
                ##Ako je čvor posjećen i ako mu je boja drukčija
                ##od boje koja bi trebala biti zbog trenutnog vrha,
                ##znači da graf nije dvoobojiv.
                elif boje[i] != 1 - boje[t]:
                    dvoobojiv = False
    return dvoobojiv

n = int(input())
m = int(input())
graf = Matrica(n)
for i in range(m):
    brid = input().split()
    r, s = int(brid[0]), int(brid[1])
    graf[r, s] = 1
    graf[s, r] = 1

if dvoobojiv(graf):
    print('Dvoobojiv')
else:
    print('Nije dvoobojiv')
```

Zadatak 6.

File Edit Format Run Options Windows Help

```
# 06_06

from matrica import *
from Queue import *

##Problem ćemo riješiti širinskim pretraživanjem grafa na način
##da ćemo krenuti iz pozicije na kojoj se nalazi Marica i širinski
##obilaziti sve čvorove grafa dok ne obiđemo cijeli graf. Tijekom obilaska
##u posebnu ćemo matricu pisati broj koraka koje trebamo načiniti kako
##bismo došli od pozicije na kojoj se nalazi Marica. Do trenutne pozicije
##inicijalno će vrijednosti te matrice biti jednake -1.

##Funkcija vraća indekse svih susjeda nekog polja
def susjedi(x, y, n):
    if y - 1 >= 0:
        yield x, y - 1
    if x - 1 >= 0:
        yield x - 1, y
    if x + 1 < n:
        yield x + 1, y
    if y + 1 < n:
        yield x, y + 1

def broj_koraka(graf):
    udaljenosti = Matrica(graf.n)
    for i in range(udaljenosti.n):
        for j in range(udaljenosti.n):
            udaljenosti[i, j] = -1
    r = Queue()
    for i in range(graf.n):
        for j in range(graf.n):
            if graf[i, j] == 'S':
                r.enqueue((i, j))
                udaljenosti[i, j] = 0
            elif graf[i, j] == 'K':
                kraj = i, j

    while not r.isEmpty():
        x, y = r.dequeue()
        d = udaljenosti[x, y]
        for i, j in susjedi(x, y, graf.n):
            if udaljenosti[i, j] == -1 and graf[i, j] != '-':
                udaljenosti[i, j] = d + 1
                r.enqueue((i, j))
    return udaljenosti[kraj]

n = int(input())
graf = Matrica(n)
for i in range(n):
    redak = input().split()
    for j in range(n):
        graf[i, j] = redak[j]
```

```
t = broj_koraka(graf)
if t > 0:
    print(t)
else:
    print('Zarobljena')
```

Zadatak 7.

File Edit Format Run Options Windows Help

```
# 06_07

from Queue import *

##Problem ćemo riješiti širinskim pretraživanjem grafa na način
##da ćemo krenuti iz početne konfiguracije i širiti se na sve
##dozvoljene konfiguracije te ćemo obrađene konfiguracije
##dodavati u listu nedozvoljenih konfiguracija.
##Nedozvoljene konfiguracije čuvat ćemo u skupu dok ćemo broj
##koraka u kojim smo došli do neke konfiguracije čuvati u rječniku.

##Funkcija vraća sve susjede neke konfiguracije
def susjedi(t):
    t = int(t)
    znamenke = []
    while t > 0:
        znamenke = [t % 10] + znamenke
        t //= 10
    while len(znamenke) < 4:
        znamenke = [0] + znamenke
    for i in range(4):
        for j in range(-1, 2, 2):
            tmp = znamenke[:]
            tmp[i] = (tmp[i] + j) % 10
            yield tmp[0] * 1000 + tmp[1] * 100 + tmp[2] * 10 + tmp[3]

##Funkcija vraća broj minimalnih koraka koje trebamo načiniti
##kako bismo došli iz kombinacije poc u kombinaciju zav
##pri čemu ne smijemo doći u kombinaciju ned.
def broj_koraka(poc, zav, ned):
    r = Queue()
    r.enqueue(poc)
    udaljenosti = dict()
    udaljenosti[poc] = 0
    ned |= {poc}
    while udaljenosti.get(zav, -1) < 0 and not r.isEmpty():
        t = r.dequeue()
        for k in susjedi(t):
            if not k in ned:
                udaljenosti[k] = udaljenosti[t] + 1
                r.enqueue(k)
                ned |= {k}
    return udaljenosti.get(zav, -1)

poc = int(input())
zav = int(input())
n = int(input())
ned = set()
for i in range(n):
    ned |= {int(input())}

print(broj_koraka(poc, zav, ned))
```

Zadatak 8.

File Edit Format Run Options Windows Help

```
# 06_08

from Stack import *
from matrica import *

##Klasu stack nadopunit ćemo metodama za ispis elemenata stoga
##te metodom koja će vraćati broj elemenata stoga.
class Stack2(Stack):
    def __init__(self):
        super().__init__()
        return

    def __repr__(self):
        s = ''
        for t in self:
            s += str(t)
        return s

    def velicina(self):
        return len(self)

##Program ćemo riješiti dubinskim pretraživanjem grafa.
##Krenut ćemo od čvora 1 i ići redom do čvorova do kojih je to moguće
##te se vraćati natrag kada više ne budemo mogli ići dalje.
##Svaki puta kad budemo obišli sve čvorove ispisat ćemo jedan obilazak.
##Postupak će biti gotov kada stog bude prazan.

##Funkcija vraća prvog susjeda čvora t koji je veći od ili jednak k.
def susjed(a, t, k):
    for i in range(k, a.n):
        if a[t, i] == 1:
            return i
    return -1

def obilazak(a):
    s = Stack2()
    s.push(0)
    ##Kako zbog vraćanja ne bismo ušli u beskonačnu petlju,
    ##u varijabli k čuvamo indeks najmanjeg dozvoljenog susjeda nekog čvora
    ##u koji možemo otići.
    k = 0
    while not s.isEmpty():
        ##Ako na stogu imamo 9 elemenata (kućica ima 8 crta)
        ##našli smo jedan obilazak.
        if s.velicina() == 9:
            print(s)
        t = s.pop()
        r = susjed(a, t, k)
        if r >= 0:
            a[t, r] = 0
            a[r, t] = 0
            s.push(t)
            s.push(r)
```

```
            br += 1
            k = 0
        elif not s.isEmpty():
            r = s.pop()
            s.push(r)
            a[r, t] = 1
            a[t, r] = 1
            k = t + 1
            br -= 1

a = Matrica(5, 5)
a[0, 1] = 1
a[1, 0] = 1
a[0, 2] = 1
a[2, 0] = 1
a[0, 4] = 1
a[4, 0] = 1
a[1, 2] = 1
a[2, 1] = 1
a[1, 4] = 1
a[4, 1] = 1
a[2, 3] = 1
a[3, 2] = 1
a[2, 4] = 1
a[4, 2] = 1
a[3, 4] = 1
a[4, 3] = 1
obilazak(a)
```

Zadatak 9.

File Edit Format Run Options Windows Help

```
# 06_09
from klasaTeGraf import *

##Potrebno je u k-tom retku težinske matrice pronaći najveći element

def najblizi(a, k):
    naj = []
    for i in range(a.n):
        if a[k, i] == float('inf'):
            naj.append(-1)
        else:
            naj.append(a[k, i])
    return naj.index(max(naj))

n = int(input())
m = int(input())
a = TeGraf(n)
for i in range(m):
    brid = input().split()
    r, s, t = int(brid[0]), int(brid[1]), int(brid[2])
    a[r, s] = t
    a[s, r] = t
k = int(input())
print(najblizi(a, k))
```

Zadatak 10.

File Edit Format Run Options Windows Help

```
# 06_10

from heapq import *

##Program se svodi na računanje minimalne udaljenosti od čvora s oznakom 0
##do čvora s zadanom oznakom k, za što ćemo koristiti Dijkstrin algoritam.
##Čvorovi će biti katovi, dok će grane biti udaljenost između dvaju katova
##direktno povezanih liftom.
##Za svaki kat u rječniku udaljenosti čuvat ćemo minimalno
##vrijeme koje nam je potrebno da bismo došli do tog kata i oznaku lifta
##s kojim je to minimalno vrijeme postignuto.
##Kada uzmemo neki čvor iz reda, pogledat ćemo koji sve liftovi voze
##do odgovarajućeg kata te ćemo ažurirati vremena za sve katove
##tih liftova. Kod ažuriranja vremena trebat ćemo voditi računa
##jesmo li mijenjali lift ili ne.

def udaljenost(brzine, liftovi):
    posjećeni = set()
    udaljenosti = dict()
    for i in range(100):
        udaljenosti[i] = (float('inf'), -1)
    udaljenosti[0] = (0, -1)
    red = []
    heappush(red, (udaljenosti[0], 0))
    while len(red) > 0:
        čvor = heappop(red)[1]
        if čvor not in posjećeni:
            posjećeni |= {čvor}
            for i in range(len(liftovi)):
                if čvor in liftovi[i]:
                    for r in liftovi[i]:
                        d = udaljenosti[čvor][0] + abs(čvor - r) * brzine[i]
                        if čvor != 0 and udaljenosti[čvor][1] != i:
                            d += 60
                        if d < udaljenosti[r][0]:
                            udaljenosti[r] = (d, i)
                            heappush(red, (udaljenosti[r], r))
    return udaljenosti



n = int(input())
k = int(input())
brzine = input().split()
for i in range(len(brzine)):
    brzine[i] = int(brzine[i])

liftovi = []
for z in range(n):
    lift = input().split()
    for i in range(len(lift)):
        lift[i] = int(lift[i])
    liftovi.append(lift)
```

```
d = udaljenost(brzine, liftovi)
if d[k][0] != float('inf'):
    print(d[k][0])
else:
    print(-1)
```