Creita Andreea Georgiana

Grupa 232

La cules de flori

-documentatie-

Context

Considerăm că avem o grădină dreptunghiulară (reprezentată sub formă de matrice) în care cresc flori de diferite culori. Un omuleț dorește să culeagă flori pentru a face un buchet. Pentru aceasta se deplasează prin matrice în căutarea florilor. Regulile de deplasare și culegere a florilor sunt următoarele:

Omulețul se poate deplasa doar pe linie și coloană, nu și pe diagonală

Când ajunge într-o celulă cu floare, obligatoriu o culege.

Problema e că atunci când culege o floare de un anumit tip, toate florile de acel tip se supără pe el și îl atacă dacă ajunge pe o poziție vecină (pe linie/coloană sau diagonală) cu a lor. Florile sunt cu atât mai supărate cu cât omulețul are mai multe flori de felul lor în buchet. Astfel că distanța de la care îl atacă crește cu numărul de flori de acel fel din buchet. Dacă omulețul are k flori de acel fel nu se va putea apropia la o distanță Manhattan mai mică sau egală cu k față de florile de acel fel.

Totuși, spre norocul omulețului, florile mai și dorm. Fiecare tip de floare are k1 unități de timp în care e trează urmată de k2 unități de timp în care doarme. Fiecare pas al omulețului prin grădină indică trecerea unei unități de timp. Omulețul poate trece nestingherit pe lângă florile care dorm, dar, cum a fost scris mai sus, nu poate păși lângă o floare trează dacă are în buchet acea floare.

Omulețul nu are voie să culeagă mai multe flori sau flori de culori diferite față de cele cerute în buchet. Prin urmare, cum e stabilit că dacă ajunge într-o celulă cu floare, obligatoriu o culege, înseamnă că nu poate intra într-o celulă cu un anumit tip de floare, dacă aceasta nu e cerută în buchet sau deja a adunat toate florile de acel fel cerute în buchet.

Omulețul e grăbit și vrea să evite pașii inutili dus-întors de pe aceleași poziții. Astfel,omulețul nu poate reveni într-o celulă în care a mai fost, dacă atunci când a intrat ultima oară în celulă avea același buchet de flori. Deci pate reveni într-o celulă dacă între timp a cules alte flori.

După ce a cules toate florile cerute în buchet, omulețul poate ieși din grădină. Ieșirea se poate realiza de pe orice celulă aflată pe marginea matricei (grădinii)

Stări și tranziții

O stare e dată de poziția omulețului, starea florilor din gradină și conținutul buchetului. O tranziție e o deplasare a omulețului cu eventuala acțiune de culegere a unei flori.

Costul

Costul unei mutări este:

pentru mutarea pe linie, e egal cu 1+NF, unde NF este numărul de flori din buchet

Pentru mutarea pe coloană, e egal cu 1+NF/2, unde NF e numărul de flori din buchet

Excepție pentru cele scrise mai sus: când intră într-o celulă cu floare (pe care o culege), costul este întotdeauna 1.

1. (5%)Fișierele de input vor fi într-un folder a cărui cale va fi dată în linia de comanda.

Fisierele de input se afla intr-un folder denumit input_files, fisierele de output au fost generate intr-un folder denumit output_files. In terminal se apeleaza main.py input_files output_files nr_argumente timeout.

```
Total
PS C:\Users\Andreea\Desktop\an II\sem2\ai\lab2ML\cvtema> py main.py input_files output_files 2 300
```

```
PS C:\Users\Andreea\Desktop\an II\sem2\ai\lab2ML\cvtema> py main.py input_files output_files 1 5
```

2. (5%) Citirea din fisier + memorarea starii. Parsarea fișierului de input care respectă formatul cerut în enunț

```
def __init__(self, nume_fisier):
    f = open(nume_fisier, "r")
    sirFisier = f.read()
    listaLinii = sirFisier.strip().split("gradina")
    x = listaLinii[0].strip().split("\n")
    self.floriStatus = []
    cnt = 0
    for i in x:
        x[cnt] = x[cnt].split(" ")
        cnt += 1
    for i in x:
        self.floriStatus.append([i[0], int(i[1]), int(i[2])])
        #floriStatus retine tip_floare timp_treaza timp_adormita
    print(self.floriStatus)
    lung = len(self.floriStatus)
    self.rageFlori = []
    for i in range(lung):
        self.rageFlori.append(1)
    print(self.rageFlori)
    listaLinii.pop(0)
    listaLinii = listaLinii[0].split("buchet") #citirea de matrice
    a = listaLinii[0].strip().split("\n")
    self.start = []
    for i in a:
        self.start.append([x for x in i])
```

```
    print("START")
    print(self.start)  #matricea din input
    print("START")
    a = listaLinii[1].strip().split("\n")
    self.scopuri = []
    cnt = 0
    for i in a:
        a[cnt] = a[cnt].split(" ")
        cnt += 1
    for i in a:
        self.scopuri.append([i[0], int(i[1])])
    print("SCOPURI")
    print(self.scopuri) #nr de flori din buchet
    print("SCOPURI")
    self.culese = 0
    self.total = 0
    for i in range(len(self.scopuri)):
        self.total += self.scopuri[i][1]
    print("Total")
    print(self.total)  #nr total de flori din buchetul scop
    print("Total")
    self.noSol = 0
    for k in self.scopuri:
        add = 0
        for i in range(len(self.start)):
            for j in range(len(self.start[0])):
                if self.start[i][j] == k[0]:
                    add += 1
        if add < k[1]:
            self.noSol = 1  #validare
```

Pentru parsarea fisierului de input am folosit codul de mai sus, unde FloriStatus retine informatiile despre flori: tip_floare timp_treaza timp_adormita, start reprezinta matricea din fisierul de input, iar in scopuri retinem tipul de floare si cate flori trebuie culese pentru a finaliza buchetul.

3. (15%) Functia de generare a succesorilor

```
def genereazaSuccesori(self, nodCurent, tip_euristica="euristica banala"):
    listaSuccesori = []   #lista de succesori vida
    if nodCurent.lis == []:  #lista cu nodurile curente
        for i in range(len(self.scopuri)):  #parcurge lista cu bchetul scop
            nodCurent.lis.append([self.scopuri[i][0], 0]) # pune in nodCurent.lis literele florilor
    for lGol in range(len(nodCurent.info)): #(parcurge lista de liste -> matricea) luam pe rand fiecare
linie lGol din matrice
        try:
            cGol = nodCurent.info[lGol].index('@') #in lista respectiva caut elem cu @ -> omuletul
nostru(indicele pe linie e chiar coloana)
            break
        except:
            pass
```

```
    copieFlori = copy.deepcopy(nodCurent.floriStatus)
    copieMatrice = copy.deepcopy(nodCurent.info)
    for i in range(len(nodCurent.floriStatus)):     #verifica, pentru fiecare tip de floare, daca e treaza
sau nu
        if copieFlori[i][1] != 0:
            copieFlori[i][1] -= 1
        elif copieFlori[i][2] != 0:
            copieFlori[i][2] -= 1
        else:
            copieFlori[i] = copy.deepcopy(self.floriStatus[i])
            flor = copieFlori[i][0]                                 #dupa ce trece timpul de dormit,
schimba literele din mare in mic
            for k in range(len(copieMatrice)):
                for j in range(len(copieMatrice[0])):
                    if copieMatrice[k][j] == flor.upper():
                        copieMatrice[k][j] = flor.lower()
    # pus litere mari in loc de litere mici, daca copieFlori[i][1]==0 si copieFlori[i][2]!=0
    for i in copieFlori:   #daca floarea din floriStatus copie e egala cu floarea din matrice copie
        if i[1] == 0 and i[2] != 0:  #timp_treaza ajunge la 0 si timp_adormita e dif de zero
            for k in range(len(copieMatrice)):
                for j in range(len(copieMatrice[0])):
                    if copieMatrice[k][j] == i[0]:
                        copieMatrice[k][j] = copieMatrice[k][j].upper()  #schimba lit mica cu litera mare
=> floare adormita
    nodCurent.info = copieMatrice
    #parcurge lista cu floriStatus si vede daca floarea e adormita sau nu
    #daca e adormita, atunci se uita in matrice si daca
    #floarea de pe poz respectiva corespunde cu floarea din lista cu status
    #daca corespunde, atunci schimba lit mica cu lit mare ca sa ararte ca floarea a adormit


    # stanga, dreapta, sus, jos
    linieMax = len(nodCurent.info)
    colMax = len(nodCurent.info)
    #coordonatele vecinilor practic
    directii = [[lGol, cGol - 1], [lGol, cGol + 1], [lGol - 1, cGol], [lGol + 1, cGol]]
    costCnt = 0
    for lPlacuta, cPlacuta in directii:
        if 0 <= lPlacuta < linieMax and 0 <= cPlacuta < colMax and self.checkPos(lPlacuta, cPlacuta,
                                                                                 nodCurent.rageFlori,
                                                                                 nodCurent.info,
nodCurent.floriNr):
            copieMatrice = copy.deepcopy(nodCurent.info) #face o copie cu deep sa nu se modifice si in
liste daca o modificam
            lista = copy.deepcopy(nodCurent.lis)
            copieMatrice[lGol][cGol] = '.'  #pun pe locul omuletului vecinul care e '.' -> e loc liber
            anteriorCulese = nodCurent.culese
            copieRageFlori = copy.deepcopy(nodCurent.rageFlori)
            copieFloriNr = copy.deepcopy(nodCurent.floriNr)


            if costCnt == 0 or costCnt == 1:
                costArc = 1 + nodCurent.culese  #cost pt mutarea pe linie
            else:
                costArc = 1 + nodCurent.culese // 2  #cost pt mutarea pe coloana
            if copieMatrice[lPlacuta][cPlacuta] == '.':
                copieMatrice[lPlacuta][cPlacuta] = '@'  #si in locul unde era liber '.' pun omuletul daca
era liber
            else:
                tipFloare = copieMatrice[lPlacuta][cPlacuta].lower()  #flori treze in tipFloare
                for i in range(len(nodCurent.floriNr)):
                    if copieFloriNr[i][0] == tipFloare: #daca gaseste un tip de floare din buchet care sa
corespunda
                        copieFloriNr[i][1] -= 1         #cu floarea de pe poz vecina din matrice atunci
scade nr de flori ramase de cules
```

```
                    break
                copieMatrice[lPlacuta][cPlacuta] = '@' #dupa ce culege floarea, muta omuletul in locul ei
                anteriorCulese += 1
                costArc = 1  #cost intotdeauna 1 cand intra intr-o celula cu floare
                for i in range(len(lista)):
                    if lista[i][0] == tipFloare:
                        lista[i][1] += 1
                for i in range(len(nodCurent.floriStatus)):
                    if copieFlori[i][0] == tipFloare:
                        copieRageFlori[i] += 1

            if (not nodCurent.contineInDrum(
                    copieMatrice)) or anteriorCulese > nodCurent.culese:  # and not
self.nuAreSolutii(copieMatrice):
                # de adaugat costuri pe sus etc

                listaSuccesori.append(  #daca nu e in drum, atunci il adaug in lista de succesori
                    NodParcurgere(copieMatrice, nodCurent, copieFlori, copieRageFlori, copieFloriNr,
anteriorCulese,
                                  nodCurent.g + costArc,
                                  self.calculeaza_h(copieMatrice, tip_euristica, nodCurent.culese),
lista))


    return listaSuccesori
```

Pentru a genera succesorii, strategia va fi sa gasim coordonatele locului unde este pozitionat omuletul "@". In locul omuletului se pot pune doar elementele care se afla pe linii sau pe coloane, nu si pe diagonale, apoi se face interschimbarea intre omulet si locul vecin spre care se poate deplasa, care poate fi liber sau cu o floare. Daca e floare, atunci se creste si numarul de flori culese. Tot aici se verifica si daca floarea este treaza sau adormita, folosind functia upper() daca floarea nu mai are unitati de timp pentru timp_treaza sau se foloseste functie lower() daca timp_adormita ajunge la 0.

4. (5%) Calcularea costului pentru o mutare

```
if costCnt == 0 or costCnt == 1:
    costArc = 1 + nodCurent.culese  #cost pt mutarea pe linie
else:
    costArc = 1 + nodCurent.culese // 2  #cost pt mutarea pe coloana
if copieMatrice[lPlacuta][cPlacuta] == '.':
    copieMatrice[lPlacuta][cPlacuta] = '@'  #si in locul unde era liber '.' pun omuletul daca era liber
else:
    tipFloare = copieMatrice[lPlacuta][cPlacuta].lower()  #flori treze in tipFloare
    for i in range(len(nodCurent.floriNr)):
        if copieFloriNr[i][0] == tipFloare: #daca gaseste un tip de floare din buchet care sa corespunda
            copieFloriNr[i][1] -= 1         #cu floarea de pe poz vecina din matrice atunci scade nr de flori ramase de cules
            break
    copieMatrice[lPlacuta][cPlacuta] = '@' #dupa ce culege floarea, muta omuletul in locul ei
    anteriorCulese += 1
    costArc = 1  #cost intotdeauna 1 cand intra intr-o celula cu floare
```

```
listaSuccesori.append(  #daca nu e in drum, atunci il adaug in lista de succesori
    NodParcurgere(copieMatrice, nodCurent, copieFlori, copieRageFlori, copieFloriNr, anteriorCulese,
                  nodCurent.g + costArc,
                  self.calculeaza_h(copieMatrice, tip_euristica, nodCurent.culese), lista))
```

Pentru o mutare pe o pozitie din matrice unde nu se afla nicio floare, costul va fi 1 + numarul de flori din buchet pentru o mutare pe linie, 1+ numarul de flori din buchet / 2 pentru o mutare pe coloana si 1 daca pe pozitia respectiva exista vreo floare.

5. (5%) Testarea ajungerii în starea scop (indicat ar fi printr-o funcție de testare a scopului). Atenție, acolo unde nu se precizează clar în fișierul de intrare o stare finală înseamnă că funcția de testare a scopului doar verifică niște condiții precizate în enunț. Nu se va rezolva generând toate stările finale posibile fiindca e ineficient, ci se va verifica daca o stare curentă se potrivește descrierii unei stări scop.

```
#primeste ca parametru un nod si verifica daca se afla in starea finala sau nu
    #returneaza True sau False
    def testeaza_scop(self, nodCurent):
        for i in nodCurent.floriNr:
            if i[1] != 0:
                return 0
        return 1
```

6. (15% = 2+5+5+3) 4 euristici:

(2%) banala

(5%+5%) doua euristici admisibile posibile (se va justifica la prezentare si in documentație de ce sunt admisibile)

(3%) o euristică neadmisibilă (se va da un exemplu prin care se demonstrează că nu e admisibilă). Atenție, euristica neadmisibilă trebuie să depindă de stare (să se calculeze în funcție de valori care descriu starea pentru care e calculată euristica).

Euristica banala va return mereu 1, deci prioritizeaza costul default. Prima euristica returneaza cate flori mai avem de cautat, luand succesorii in ordinea crescatoare a cate flori mai sunt de cules. De exemplu, daca primim 6,7,2,3, euristica va lua 2, 3, 6, 7.

```
# euristica banala
def calculeaza_h(self, infoNod, tip_euristica="euristica banala", cul=0):
    if tip_euristica == "euristica banala":
        return 1    #returnam 1(costul minim) pentru ca avem macar o mutare
    else:
        return self.total - cul #eur 1
```

7.(10%) crearea a 4 fisiere de input cu urmatoarele proprietati:

a. un fisier de input care nu are solutii

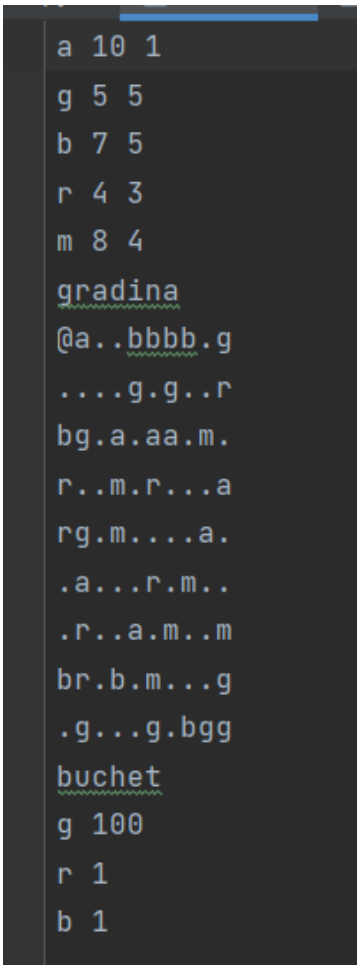
b. un fisier de input care da o stare initiala care este si finala (daca acest lucru nu e realizabil pentru problema, aleasa, veti mentiona acest lucru, explicand si motivul).

c. un fisier de input care nu blochează pe niciun algoritm și să aibă ca soluții drumuri lungime micuță (ca să fie ușor de urmărit), să zicem de lungime maxim 20.

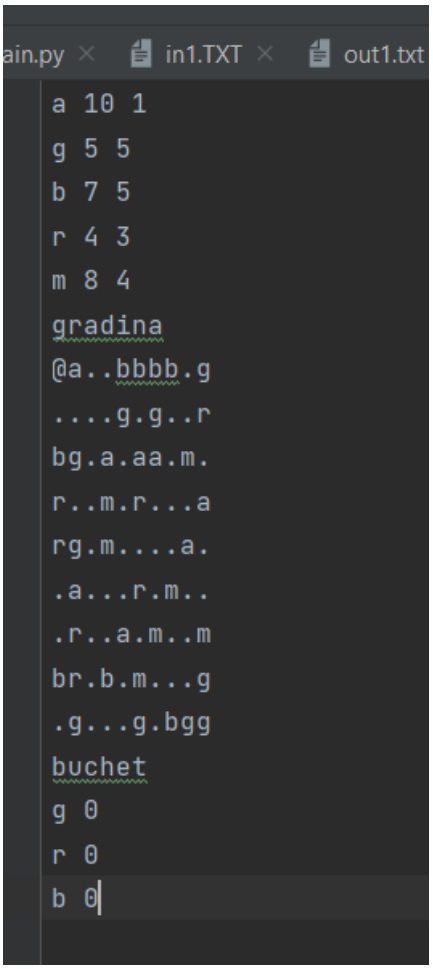
d. un fisier de input care să blocheze un algoritm la timeout, dar minim un alt algoritm să dea soluție (de exemplu se blochează DF-ul dacă soluțiile sunt cât mai "în dreapta" în arborele de parcurgere)

dintre ultimele doua fisiere, cel putin un fisier sa dea drumul de cost minim pentru euristicile admisibile si un drum care nu e de cost minim pentru cea euristica neadmisibila

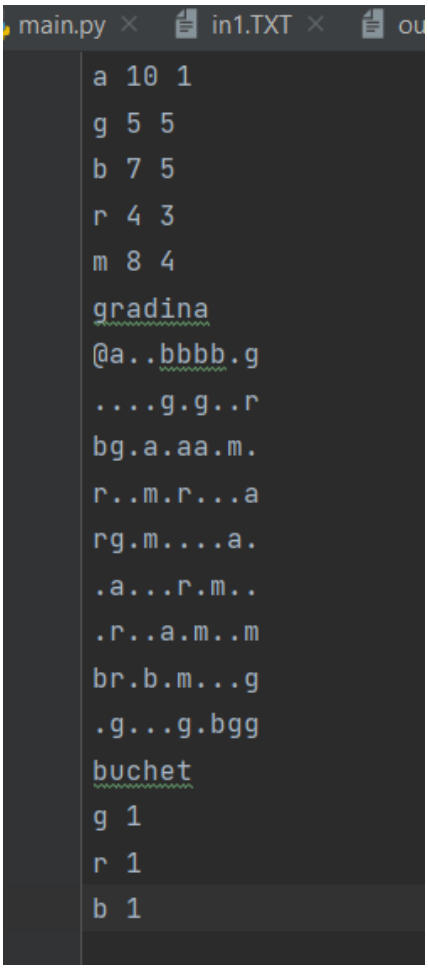
a.


a 10 1
g 5 5
b 7 5
r 4 3
m 8 4
gradina
@a..bbbb.g
....g.g..r
bg.a.aa.m.
r..m.r...a
rg.m....a.
.a...r.m..
.r..a.m..m
br.b.m...g
.g...g.bgg
buchet
g 100
r 1
b 1

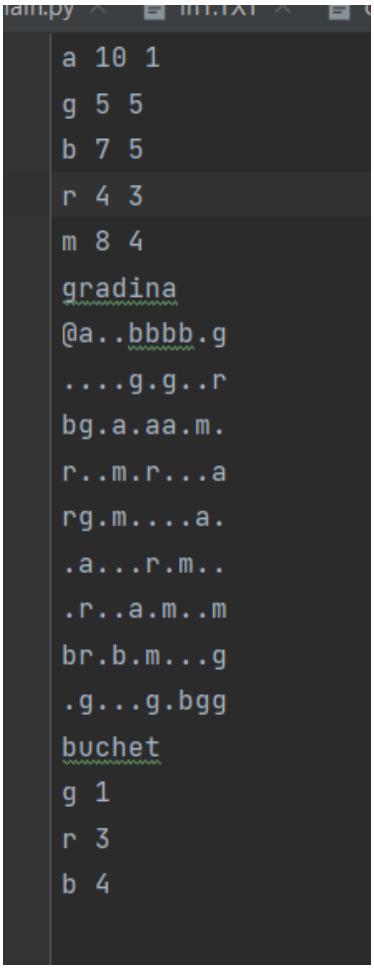
b.


ain.py
in1.TXT
out1.txt
a 10 1
g 5 5
b 7 5
r 4 3
m 8 4
gradina
@a..bbbb.g
....g.g..r
bg.a.aa.m.
r..m.r...a
rg.m....a.
.a...r.m..
.r..a.m..m
br.b.m...g
.g...g.bgg
buchet
g 0
r 0
b 0

c.


main.py
in1.TXT
ou
a 10 1
g 5 5
b 7 5
r 4 3
m 8 4
gradina
@a..bbbb.g
....g.g..r
bg.a.aa.m.
r..m.r...a
rg.m....a.
.a...r.m..
.r..a.m..m
br.b.m...g
.g...g.bgg
buchet
g 1
r 1
b 1

d.



8. (15%) Pentru cele NSOL drumuri(soluții) returnate de fiecare algoritm (unde NSOL e numarul de soluții dat în linia de comandă) se va afișa:

numărul de ordine al fiecărui nod din drum

lungimea drumului

costului drumului

timpul de găsire a unei soluții (atenție, pentru soluțiile de la a doua încolo timpul se consideră tot de la începutul execuției algoritmului și nu de la ultima soluție)

numărul maxim de noduri existente la un moment dat în memorie

numărul total de noduri calculate (totalul de succesori generati; atenție la DFI și IDA* se adună pentru fiecare iteratie chiar dacă se repetă generarea arborelui, nodurile se vor contoriza de fiecare dată afișându-se totalul pe toate iterațiile

între două soluții de va scrie un separator, sau soluțiile se vor scrie în fișiere diferite.

```
6)
cost:5
. a . . b b b b . G
. . . . G . G . . R
. . . a . a a . m .
@ . . m . R . . . a
R G . m . . . . a .
. a . . . R . m . .
. R . . a . m . . m
b R . b . m . . . G
. G . . . G . b G G
[['g', 1], ['r', 1], ['b', 1]]
Cost: 5
Lungime: 6
0.013962268829345703secunde
Maximul de noduri:4
Totalul de noduri:98

----------------
Solutii obtinute cu depth first:Solutie: 1)
```

9. (5%) Afisarea in fisierele de output in formatul cerut

Outputul este afisat in formatul cerut

```
Solutii obtinute cu breadth first:
```

```
cost:5
. a . . b b b b . G
. . . . G . G . . R
. . . a . a a . m .
@ . . m . R . . . a
R G . m . . . . a .
. a . . . R . m . .
. R . . a . m . . m
b R . b . m . . . G
. G . . . G . b G G
[['g', 1], ['r', 1], ['b', 1]]
Cost: 5
Lungime: 6
0.013962268829345703secunde
Maximul de noduri:4
Totalul de noduri:98

----------------
Solutii obtinute cu depth first:Solutie: 1)
cost:0
```

```
Solutii obtinute cu depth first:
```

```
46)
cost:103
. a . . b b b b . g
. . . . g . g . . R
. . . a . a a . m .
@ . . m . R . . . a
R g . m . . . . a .
. a . . . R . m . .
. R . . a . m . . m
b R . b . m . . . g
. g . . . g . b g g
[['g', 1], ['r', 1], ['b', 1]]
Cost: 103
Lungime: 46
0.019946575164794922secunde
Maximul de noduri:4
Totalul de noduri:107


----------------
Solutii obtinute cu depth first iterativ:**************
```

Solutii obtinute cu depth first iterativ

```
----------------
Solutii obtinute cu depth first iterativ:**************
Adancime maxima: 1**************
Adancime maxima: 2**************
Adancime maxima: 3**************
Adancime maxima: 4**************
Adancime maxima: 5**************
Adancime maxima: 6Solutie: 1)
cost:0
@ a . . b b b b . g
. . . . g . g . . r
b g . a . a a . m .
r . . m . r . . . a
r g . m . . . . a .
. a . . . r . m . .
. r . . a . m . . m
b r . b . m . . . g
. g . . . g . b g g
[['g', 0], ['r', 0], ['b', 0]]
2)
```

```
6)
cost:5
. a . . b b b b . G
. . . . G . G . . R
. . . a . a a . m .
@ . . m . R . . . a
R G . m . . . . a .
. a . . . R . m . .
. R . . a . m . . m
b R . b . m . . . G
. G . . . G . b G G
[['g', 1], ['r', 1], ['b', 1]]
Cost: 5
Lungime: 6
0.007980108261108398secunde
Maximul de noduri:4
Totalul de noduri:12


----------------



##################
```

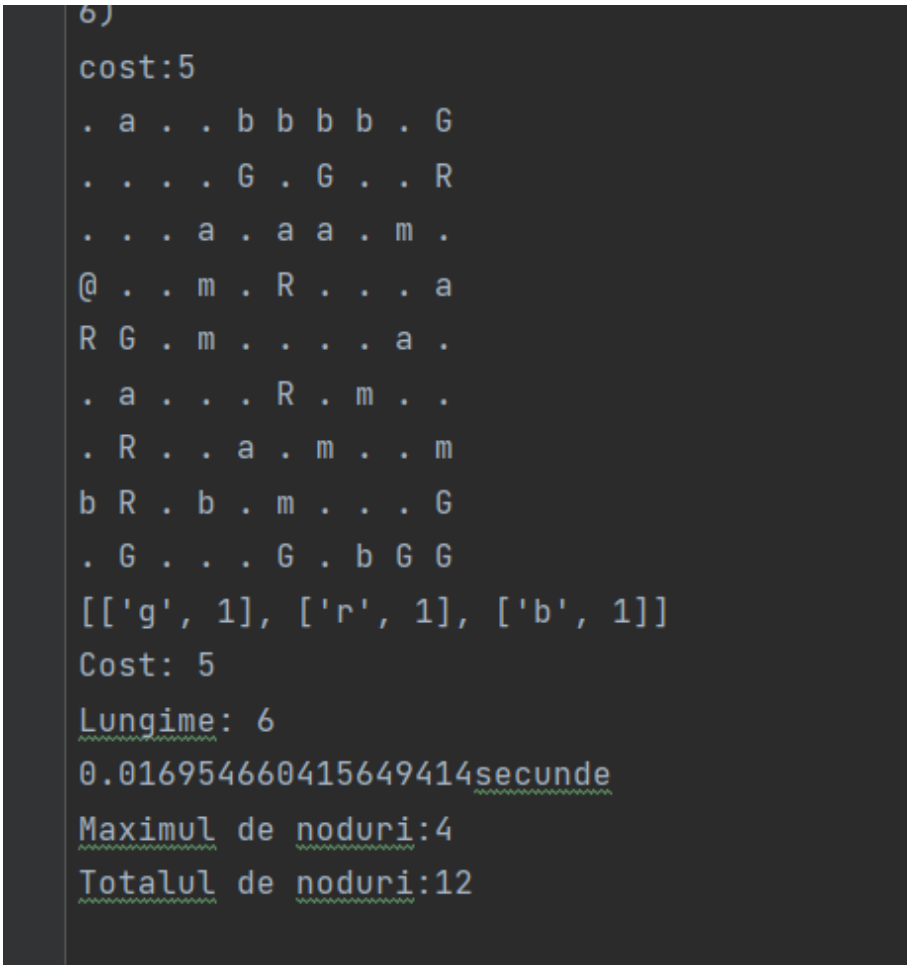
Solutii obtinute cu A*:

```
[[ g , 1], [ r , 0], [ b , 1]]
6)
cost:5
. a . . b b b b . G
. . . . G . G . . R
. . . a . a a . m .
@ . . m . R . . . a
R G . m . . . . a .
. a . . . R . m . .
. R . . a . m . . m
b R . b . m . . . G
. G . . . G . b G G
[['g', 1], ['r', 1], ['b', 1]]
Cost: 5
Lungime: 6
0.004984378814697266secunde
Maximul de noduri:23
Totalul de noduri:142


----------------
Solutie: 1)
cost:0
```

Solutii cu a_starOpt

```
6)
cost:5
. a . . b b b b . G
. . . . G . G . . R
. . . a . a a . m .
@ . . m . R . . . a
R G . m . . . . a .
. a . . . R . m . .
. R . . a . m . . m
b R . b . m . . . G
. G . . . G . b G G
[['g', 1], ['r', 1], ['b', 1]]
Cost: 5
Lungime: 6
0.004987716674804687 5secunde
Maximul de noduri:18
Totalul de noduri:119


----------------
```

Solutii cu ida_starSolutie:



Solutii cu ida_star

```
----------------
Solutii cu ida_starSolutie: 1)
cost:0
@ a . . b b b b . g
. . . . g . g . . r
b g . a . a a . m .
r . . m . r . . . a
r g . m . . . . a .
. a . . . r . m . .
. r . . a . m . . m
b r . b . m . . . g
. g . . . g . b g g
[]
Cost: 0
Lungime: 1
0.0secunde
Maximul de noduri:-1
Totalul de noduri:0
```

```
B R . B . M . . . g
. g . . . g . B g g
[['g', 1], ['r', 3], ['b', 3]]
24)
cost:37
. A . . . . . @ . g
. . . . . g . . R
B g . A . A A . M .
. . . M . . . . . A
R g . M . . . . A .
. A . . . . . M . .
. R . . A . M . . M
B R . B . M . . . g
. g . . . g . B g g
[['g', 1], ['r', 3], ['b', 4]]
Cost: 37
Lungime: 24
57.77106070518494secunde
Maximul de noduri:6341
Totalul de noduri:37572438

----------------
Solutii cu ida_starTimeout
Timeout
```

10.(5%+5%) Validări și optimizari.

Validare: Găsirea unui mod de a realiza din starea initială că problema nu are soluțiise afla in Graph __init__. Atunci cand se citeste inputul se verifica din prima daca nu are solutie.

```
        self.noSol = 0
        for k in self.scopuri:
            add = 0
            for i in range(len(self.start)):
                for j in range(len(self.start[0])):
                    if self.start[i][j] == k[0]:
                        add += 1
            if add < k[1]:
                self.noSol = 1  #validare
```

```
if gr.noSol == 1:
    fout.write("Nu exista solutii")
    exit()
```

Optimizare: găsirea unui mod de reprezentare a stării, cât mai eficient

Starile sunt reprezentate de cate o matrice ce contine mutarea facuta de omulet pe o pozitie vecina si de cate flori a cules omuletul, reprezentate printr-o lista de liste. Aceasta afisare este eficienta pentru ca permite vizalizarea fiecarei mutari facute si florile culese in mutarea respectiva.

11.(5%) Comentarii pentru clasele și funcțiile adăugate de voi în program

Codul este comentat acolo unde functiile nu sunt luate din laborator.

12.(5%) Documentație cuprinzând explicarea euristicilor folosite.

Input 1:

```
a 10 1
g 5 5
b 7 5
r 4 3
m 8 4
gradina
@a..bbbb.g
....g.g..r
bg.a.aa.m.
r..m.r...a
rg.m....a.
.a...r.m..
.r..a.m..m
br.b.m...g
.g...g.bgg
buchet
g 1
r 1
b 1
```

Algoritm	Euristica	Cost	Lungime	Noduri_generate	Maxim_noduri	Timp executie secunde
A*	Banala	5	6	142	23	0.004984378814697266
A*	Admisibila 1	5	6	34	11	0.0019948482513427734
A*_opt	Banala	5	6	119	18	0.0049877166748046875
A*_opt	Admisibila 1	5	6	44	12	0.0029916763305664062
IDA*	Banala	5	6	12	4	0.016954660415649414

IDA*	Admisibila 1	5	6	12	4	0.003989458084106445

Input 2:

```
a 10 1
g 5 5
b 7 5
r 4 3
m 8 4
gradina
@a..bbbb.g
....g.g..r
bg.a.aa.m.
r..m.r...a
rg.m....a.
.a...r.m..
.r..a.m..m
br.b.m...g
.g...g.bgg
buchet
g 1
r 2
b 0
```

Algoritm	Euristica	Cost	Lungime	Noduri_generate	Maxim_noduri	Timp executie
A*	Banala	7	7	6407	155	0.02094435691833496
A*	Admisibila1	7	7	122	20	0.001994609832763672
A*_opt	Banala	7	7	519	33	0.007977724075317383
A*_opt	Admisibila1	7	7	89	12	0.002993345260620117
IDA*	Banala	7	7	12	3	0.04089021682739258
IDA*	Admisibila1	7	7	12	3	0.009972572326660156

Se poate observa ca prima euristica este mult mai eficienta decat euristica banala pentru ambele inputuri.

Algoritmul A* s-a dovedit a fi mai rapid decat algoritmul A* optim.

In concluzie, fiecare algoritm are avantajele si dezavantajele lui. Algoritmii de tip A* si A* optim depind de euristica folosita, in schimb IDA* este eficienta pentru solutiile shallow.