Sfere căzătoare

Documentație

Punctul 1 (5%)

Cerinta: Fișierele de input vor fi într-un folder a cărui cale va fi dată în linia de comanda. În linia de comandă se va da și calea pentru un folder de output în care programul va crea pentru fiecare fișier de input, fișierul sau fișierele cu rezultatele. Tot în linia de comandă se va da ca parametru și numărul de soluții de calculat (de exemplu, vrem primele NSOL=4 soluții returnate de fiecare algoritm). Ultimul parametru va fi timpul de timeout. Se va descrie în documentație forma în care se apelează programul, plus 1-2 exemple de apel.

Implementare:

```
parser = ArgumentParser()
    parser.add_argument(
        "-dir_in", dest="dir_input", default="input", help="input folder"
    )
    parser.add_argument(
        "-dir_out",
        dest="dir_output",
        default="output",
        help="output folder",
    )
    parser.add_argument("-nsol", dest="nsol", default="1", help="numar solutii")
    parser.add_argument("-to", dest="timeout", default="5", help="Timeout")

    args = vars(parser.parse_args())
    dir_input, dir_output, nsol, timeout = (
        args["dir_input"],
        args["dir_output"],
        int(args["nsol"]),
        int(args["timeout"]),
    )
```

Model apelare: python {nume fișier} -dir_in {folder input} -dir_out {folder output} -nsol {număr soluții} -to {timeout}

Exemplu: python main.py -dir_in input -dir_out output -nsol 1 -to 5

Nota: Datorita valorilor default va merge si rularea fisierelor cu python main.py

Punctul 2 (5%)

Cerinta: Citirea din fisier + memorarea starii. Parsarea fișierului de input care respectă formatul cerut în enunț

Implementare:

```
if not os.path.isdir(dir_output):
        os.mkdir(dir_output)
    else:
        for file in os.listdir(dir_output):
            os.remove(dir_output + "/" + file)
    for fs in listaFisiereinput:
        for i in range(1, 7):
            nume_file_out = dir_output + "/output_" + fs + "__algo__"
            if i == 1:
                fout = open(nume_file_out + "DF", "w")
            elif i == 2:
                fout = open(nume_file_out + "BF", "w")
            elif i == 3:
                fout = open(nume_file_out + "DFI", "w")
            elif i == 4:
                fout = open(nume_file_out + "Astar", "w")
            elif i == 5:
                fout = open(nume_file_out + "Astar_opt", "w")
            else:
                fout = open(nume_file_out + "IDAstar", "w")
            fin = open(dir_input + "/" + fs, "r+")
            linie = fin.readlines()
            sfere = []
            iesiri = []
            matrice = []
            vizualizare = []
            mat_adiacenta = []
```

```
            mat_ponderi = []
            de_scos = []
            cost = 0
            input_type = 0
            for l in linie:
                lin = l.split()
                if lin == ["sfere"]:
                    input_type = 1
                    continue
                if lin == ["iesiri"]:
                    input_type = 2
                    continue
                if input_type == 0:
                    matrice.append(list(map(int, lin)))
                elif input_type == 1:
                    sfere.append(list(map(int, lin)))
                else:
                    iesiri.append(list(map(int, lin)))
            k = int(matrice[0][0])
            dist_max = int(matrice[0][1])
            matrice = matrice[1:]
            vizualizare = [
                ["." for i in range(len(matrice[0]))] for j in
range(len(matrice))
            ]
            for ies in iesiri:
                vizualizare[ies[0]][ies[1]] = "#"
            for sf in sfere:
                if vizualizare[sf[0]][sf[1]] == "#":
                    vizualizare[sf[0]][sf[1]] = "%"
                    de_scos.append(sf)
                else:
                    vizualizare[sf[0]][sf[1]] = "@"
```

Punctul 3 (15%)

Cerinta: Functia de generare a succesorilor

Implementare:

```
def generare_suc(self, matrice=matrice, sfere=sfere):
        """
        Verificam unde ar merge fiecare sfera dupa regula din enunt pentru a
genera succesorii
        :param matrice: matricea turnurilor
```

```
        :param sfere: vector cu pozitia sferelor
        :return: vectorul de succesori posibili ai configuratiei actuale
        """

        global sfere_temp
        succesori = []
        directii = [(-1, 0), (0, 1), (1, 0), (0, -1)]
        for sfr in sfere:
            cost_mutare = inf
            sfera_finala = -1
            contor_directie = 0
            for directie in directii:
                contor_directie += 1
                if (
                    0 <= sfr[0] + directie[0] < len(matrice)
                    and 0 <= sfr[1] + directie[1] < len(matrice)
                    and cost_mutare
                    > (
                        matrice[sfr[0]][sfr[1]]
                        - matrice[sfr[0] + directie[0]][sfr[1] + directie[1]]
                    )
                    * (
                        matrice[sfr[0]][sfr[1]]
                        - matrice[sfr[0] + directie[0]][sfr[1] + directie[1]]
                    )
                    and (
                        matrice[sfr[0]][sfr[1]]
                        - matrice[sfr[0] + directie[0]][sfr[1] + directie[1]]
                    )
                    > 0
                ):
                    cost_mutare = (
                        matrice[sfr[0]][sfr[1]]
                        - matrice[sfr[0] + directie[0]][sfr[1] + directie[1]]
                    ) * (
                        matrice[sfr[0]][sfr[1]]
                        - matrice[sfr[0] + directie[0]][sfr[1] + directie[1]]
                    )
                    sfera_finala = contor_directie
            sfere_temp.append(sfera_finala)
        for sfera_in in sfere:
            self.configuratie[sfera_in[0]][sfera_in[1]] = "."
        for sfera_noua in sfere_temp:
            self.configuratie[sfera_noua[0]][sfera_noua[1]] = "@"
        succesori.append(self.configuratie)
```

```
        return succesori
```

Punctul 4 (5%)

Cerinta: Calcularea costului pentru o mutare

Implementare:

```
def actualizare_cost_stare():
    """
    In sfere_temp avem pozitia actuala a sferelor si in sfere ultima pozitie,
adaugam la cost costul dintre mutarile consecutive si actualizam vectorul sfere
    """
    global cost, sfere, sfere_temp
    for i in range(len(sfere)):
        cost += (
            matrice[sfere[i][0]][sfere[i][1]]
            - matrice[sfere_temp[i][0]][sfere_temp[i][1]]
        ) * (
            matrice[sfere[i][0]][sfere[i][1]]
            - matrice[sfere_temp[i][0]][sfere_temp[i][1]]
        )
    sfere = copy.deepcopy(sfere_temp)
    sfere_temp.clear()
```

Punctul 5 (5%)

Cerinta: 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.

Implementare:

```
def eliminare_sfere_final(sfere_de_eliminat=de_scos):
    """
    Daca o sfera este pe o iesire, atunci eliminam sfera si iesirea din liste
    :param sfere_de_eliminat: pozitie ce se regaseste si sfera si iesirea
```

```
    """
    for afara in sfere_de_eliminat:
        iesiri.remove(afara)
        sfere.remove(afara)


def verif_stare_finala():
    """
    :return: avand in considerare ca elimin sferele din vectorul de sfere cand
ajung la iesire, atunci starea finala se indeplineste cand toate sferele au ajuns
la iesire, deci vectorul de sfere este gol
    """
    eliminare_sfere_final()
    return len(sfere) == 0


def verif_stare_finala_alternativa():
    """
    :return: daca toate sferele se afla pe iesiri
    """
    return all(item in iesiri for item in sfere)
```

Punctul 6 (15%)

Cerinta:

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).

Implementare:

```
def calculeaza_h(nod_actual, tip="banala"):
    '''
    Functie de calcularea euristicii (mai multe detalii despre fiecare in cadrul
documentatiei)
    :param nod_actual: nodul actual
    :param tip: tipul euristicii
    :return: valoarea euristicii
    '''
    if verif_stare_finala() == True:
        return 0
    elif tip == "banala":
        return 1
    elif tip == "ad_1":
        dist_min=inf
        for sf in nod_actual.sfere:
            for ies in nod_actual.iesiri:
                dist_min=min(dist_min,(abs(sf[0]-ies[0])+abs(sf[1]-ies[1])))
        return dist_min
    elif tip == "ad_2":
        return len(nod_actual.sfere)
    elif tip == "neadmisibila":
        dist_max = -inf
        for sf in nod_actual.sfere:
            for ies in nod_actual.iesiri:
                dist_max = max(dist_min, (abs(sf[0] - ies[0]) + abs(sf[1] -
ies[1])))
        return dist_max
```

Punctul 7 (10%)

Cerinta:

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)

e. 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

Implementare:

a)

input1.txt
```
4 5
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
sfere
0 0
iesiri
0 0
```

b)

input2.txt
```
4 5
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
sfere
2 2
1 2
iesiri
0 0
```

c)

input3.txt
```
5 10
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
sfere
1 1
iesiri
0 0
```

d) si e)

input4.txt

```
4 5
9 7 4 2 3 10 4 7 10 2
10 7 8 4 9 7 3 8 3 8
4 8 7 11 8 6 1 7 5 10
7 3 2 4 2 3 5 10 7 3
8 8 6 8 3 12 8 7 5 5
10 9 1 5 10 3 9 5 10 6
3 4 4 2 1 7 5 10 3 4
3 2 3 8 9 4 4 9 10 5
8 3 9 4 8 1 7 12 5 7
4 10 2 8 4 10 6 7 8 12
sfere
4 2
4 5
7 8
9 9
8 8
iesiri
4 0
4 9
0 2
9 6
8 0
```

Punctul 9 (5%)

Cerinta: Afisarea in fisierele de output in formatul cerut

Implementare:

```
def afisare(nume_fisier=fout, final=0):
    '''
    Functie de afisare a output-ului in fisier
    :param nume_fisier: numele fisierului de iesire
    :param final: parametru care arata daca programul este la final si trebuie
afisat costul final
```

```
    '''
    if final == 0:
        nume_fisier.write(str(step) + ")\n")
        nume_fisier.write("cost:" + str(cost) + "\n")
        nume_fisier.write("Matrice turnuri:\n")
        for line in matrice:
            nume_fisier.write("  ".join(map(str, line)) + "\n")

        nume_fisier.write("Matrice sfere:\n")
        for line in vizualizare:
            nume_fisier.write("  ".join(map(str, line)) + "\n")
        nume_fisier.write("\n")
    if final == 1:
        nume_fisier.write("cost final:" + str(cost) + "\n")
```

Punctul 10 (5%)

Cerinta:

Validare: verificarea corectitudinii datelor de intrare

Validare: găsirea unui mod de a realiza din starea inițială că problema nu are soluții. Validările și optimizările se vor descrie pe scurt în documentație.

Implementare:

```
def verif_input_valid():
    """
    Cazuri de input invalid:
    - daca sunt mai multe sfere decat iesiri
    - mai multe sfere in acelasi loc
    - mai multe iesiri in acelasi loc
    - daca o iesire nu este pe marginea grid-ului
    :return: True daca este valid si False daca nu este valid
    """
    if len(sfere) > len(iesiri):
        return False
    for sf in sfere:
        if sfere.count(sf) > 1:
            return False
```

```
    for ies in iesiri:
        if iesiri.count(ies) > 1:
            return False
        if (
            (ies[0] == 0 or ies[0] == len(matrice) - 1)
            or (ies[1] == 0 or ies[1] == len(matrice[0]) - 1)
        ) == False:
            return False
    return True
```

Punctul 11 (5%)

Cerinta: Comentarii pentru clasele și funcțiile adăugate de voi în program (dacă folosiți scheletul de cod dat la laborator, nu e nevoie sa comentați și clasele existente). Comentariile pentru funcții trebuie să respecte un stil consacrat prin care se precizează tipul și rolurile parametrilor, cât și valoarea returnată (de exemplu, reStructured text sau *Google python docstrings*).

Exemplu:

```
def __init__(self, matrice, configuratie, sfere, iesiri, distanta,
predecesor=None):
        '''
        :param matrice: matricea turnurilor
        :param configuratie: matricea de sfere
        :param sfere: vector cu pozitia sferelor
        :param iesiri: vector cu pozitia iesirilor
        :param distanta: distanta fata de prima configuratie
        :param predecesor: ultima configuratie inainte de cea actuala
        '''
```

Punctul 12 (5%)

Cerinta:

Documentație cuprinzând explicarea euristicilor folosite. În cazul euristicilor admisibile, se va dovedi că sunt admisibile. În cazul euristicii neadmisibile, se va

găsi un exemplu de stare dintr-un drum dat, pentru care h-ul estimat este mai mare decât h-ul real.

Implementare:

Euristica banala

Pentru ca matricea nu este matrice scop, exista cel putin o mutare, deci costul este minim 1, asa ca euristica banala va returna 1.

Euristica admisibila 1

Distanta minima va fi cea de la sfera cea mai apropiata de iesire pana la iesirea mentionata. In cazul in care mereu costul va fi 1 (caz ideal), atunci distanta va fi distanta Manhattan.

Euristica admisibila 2

Daca toate sferele ar avea un pas de cost 1 atunci costul ar fi len(sfere). Daca nu este aceasta configuratie atunci costul va fi mai mare sau egal.

Euristica neadmisibila

Se ia distanta cea mai mare dintre o sfera si o iesire. In cazul in care avem o sfera pe tabla care este la o distanta de 1 de o iesire si o distanta de 5 de o alta iesire, euristica va returna 5, dar costul va fi de fapt 1, inegalitatea 1>=5 fiind falsa.