**Evadarea Broscoceilor...**

Apelul programului:

python main.py *inputFiles outputFiles NSOL tOut*

inputFiles: director-ul unde se află input-urile

outputFiles: director-ul unde se vor afla output-urile

NSOL: numărul de soluții căutate

tOut: timeout-ul pentru fiecare algoritm în secunde

Ex apel: python main.py inputFiles outputFiles 10 30

În fișierul de input nu putem avea frunze cu id-ul ‘mal’

Euristica banală:

min = float('inf')  
for br in broaste:  
 frunzaCurenta = broaste[br][1]  
 if frunzaCurenta != 'mal':  
 x = frunze[frunzaCurenta][0]  
 y = frunze[frunzaCurenta][1]  
 distMal = raza - (x \*\* 2 + y \*\* 2) \*\* (1 / 2)  
 if distMal < min:  
 min = distMal  
if min != float('inf'):  
 return min  
return 0

Euristica returnează distanța minimă de la o broască la marginea lacului. Evident costul soluției minime nu poate fi mai mic decât acest minim.

Euristica admisibilă 1:

count = 0  
min = float('inf')  
for br in broaste:  
 frunzaCurenta = broaste[br][1]  
 if frunzaCurenta != 'mal':  
 count += 1  
 x = frunze[frunzaCurenta][0]  
 y = frunze[frunzaCurenta][1]  
 distMal = raza - (x \*\* 2 + y \*\* 2) \*\* (1 / 2)  
 if distMal < min:  
 min = distMal  
if min != float('inf'):  
 return count \* min  
return 0

Euristica calculează distanța minima de la o broască la marginea lacului și înmulțește acest minim cu numărul de broaște care încă nu se află la mal. Evident fiecare broască care nu se află pe mal trebuie să parcurgă cel puțin cât trebuie să parcurgă minim broasca cea mai apropiată de margine.

Euristica admisibilă 2:

h = 0  
for br in broaste:  
 frunzaCurenta = broaste[br][1]  
 if frunzaCurenta != 'mal':  
 x = frunze[frunzaCurenta][0]  
 y = frunze[frunzaCurenta][1]  
 distMal = raza - (x \*\* 2 + y \*\* 2) \*\* (1 / 2)  
 h += distMal  
return h

Euristica calculează suma distanțelor fiecărei broaște la mal. Evident costul minim nu poate fi mai mic decât această sumă, deoarece fiecare broască trebuie să parcurgă cel puțin distanța ei până la mal.

Euristica neadmisibilă:

h = 0  
for br in broaste:  
 frunzaCurenta = broaste[br][1]  
 istoricBroasca = parinte.istoricBroasca(br)  
 istoricBroasca.append(frunzaCurenta)  
 repetiti = len(istoricBroasca) - len(set(istoricBroasca))  
 if frunzaCurenta != 'mal':  
 x = frunze[frunzaCurenta][0]  
 y = frunze[frunzaCurenta][1]  
 distMal = (x \*\* 2 + y \*\* 2) \*\* (1 / 2) \* 100  
 h += distMal  
  
 if repetiti != 0:  
 h \*= repetiti + 1  
return h

Euristica calculează suma distanțelor de la o broasca la origine înmulțită cu 100 și după pentru fiecare repetiție de frunze vizitate înmulțește euristica cu (numărul repetițiilor + 1).

Fie inputul:

7  
Broscovina 6 id8  
id0 1 5 3 5  
id1 0 0 0 5  
id2 -1 1 3 8  
id3 0 2 0 7  
id4 2 2 3 10  
id5 3 0 1 5  
id6 -3 1 1 6  
id7 -4 1 3 7  
id8 -4 0 1 7  
id9 -5 0 2 8  
id10 -3 -3 4 12  
id11 1 -3 3 6  
id12 0 -2 0 5  
id13 -2 -1 3 9  
id14 -1 -1 7 15

H = dist(id8, (0,0)) \* 100 = 4 \* 100 = 400

H’ = id8 -> id9 -> mal = id8 -> id9 + id9 -> mal = 1 + 2 = 3

400 > 3 <=> H > H’ => euristica este neadmisiblă

Pentru inputul de mai sus:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Algoritm | Lungime | Cost | Nr. max de noduri existente la un moment dat | Nr. Total noduri | Timp |
| BF | 3 | 3.00 | 68 | 79 | 0.0019 |
| DF | 22 | 27.89 | 162 | 1757 | 0.0400 |
| DFI | 3 | 3.00 | 10 | 17 | 0.0009 |
| A\* b | 3 | 3.00 | 8 | 9 | 0.0010 |
| A\* a1 | 3 | 3.00 | 8 | 9 | 0.0009 |
| A\* a2 | 3 | 3.00 | 8 | 9 | 0.0000 |
| A\* n | 4 | 7.22 | 1847 | 2682 | 0.1420 |
| A\*optimizat b | 3 | 3.00 | 8 | 9 | 0.0006 |
| A\*optimizat a1 | 3 | 3.00 | 8 | 9 | 0.0009 |
| A\*optimizat a2 | 3 | 3.00 | 8 | 9 | 0.0009 |
| A\*optimizat n | 4 | 7.22 | 831 | 1664 | 0.1470 |
| IDA\* b | 3 | 3.00 | 10 | 9 | 0.0009 |
| IDA\* a1 | 3 | 3.00 | 10 | 9 | 0.0010 |
| IDA\* a2 | 3 | 3.00 | 10 | 9 | 0.0009 |
| IDA\* n | 4 | 7.22 | 62 | 66280 | 1.2060 |

Fie inputul:

7  
Broscovina 5 id1 Mormolocel 3 id12  
id0 1 5 3 5  
id1 0 0 0 5  
id2 -1 1 3 8  
id3 0 2 0 7  
id4 2 2 3 10  
id5 3 0 1 5  
id6 -3 1 1 6  
id7 -4 1 3 7  
id8 -4 0 1 7  
id9 -5 0 2 8  
id10 -3 -3 4 12  
id11 1 -3 3 6  
id12 0 -2 2 5  
id13 -2 -1 3 9  
id14 -1 -1 7 15

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Algoritm | Lungime | Cost | Nr. max de noduri existente la un moment dat | Nr. Total noduri | Timp |
| BF | 5 | 15.64 | 449423 | 478568 | 11.6761 |
| DF | - | - | - | - | timeout |
| DFI | 5 | 15.64 | 272 | 30404 | 0.3735 |
| A\* b | - | - | - | - | timeout |
| A\* a1 | 5 | 14.40 | 16790 | 17379 | 8.4143 |
| A\* a2 | 5 | 14.40 | 465 | 483 | 0.0109 |
| A\* n | 7 | 18.47 | 11644 | 13759 | 3.5321 |
| A\*optimizat b | - | - | - | - | timeout |
| A\*optimizat a1 | 5 | 14.40 | 7760 | 13365 | 5.5072 |
| A\*optimizat a2 | 5 | 14.40 | 465 | 483 | 0.0210 |
| A\*optimizat n | 7 | 18.47 | 7172 | 11754 | 8.3899 |
| IDA\* b | - | - | - | - | timeout |
| IDA\* a1 | 5 | 14.40 | 308 | 71825 | 0.9913 |
| IDA\* a2 | 5 | 14.40 | 127 | 2229 | 0.0369 |
| IDA\* n | 7 | 18.47 | 271 | 295815 | 5.1064 |

Validări:

def validateFrunze(self):  
 *'''  
 Validarea frunzelor primite. Atentie sirul 'mal' este rezervat pentru testarea scopului,  
 broastele se pot afla pe mal dar nu putem avea frunze cu id-ul 'mal'  
 '''* raza = self.start[0]  
 frunze = self.start[2]  
 for fr in frunze:  
 eq = frunze[fr][0]\*\*2 + frunze[fr][1]\*\*2  
 if fr == 'mal':  
 raise ValueError('Nu pot exista frunze cu id-ul "mal"')  
 if eq >= raza \*\* 2:  
 raise ValueError('Exista frunze in afara lacului')  
 if frunze[fr][2] < 0:  
 raise ValueError('Exista frunze cu numarul de insecte negativ')  
 if frunze[fr][3] < 0:  
 raise ValueError('Frunza are greutatea maxima acceptata negativa')  
 if frunze[fr][2] > frunze[fr][3]:  
 raise ValueError('Sunt prea multe insecte pe frunza')

În funcția validatFrunze fac următoarele validări:

* Verific dacă există frunze care să aibă id-ul ‘mal’, cuvântul fiind rezervat
* Verific dacă frunzele se află în interiorul lacului
* Verific dacă frunzele au număr negativ de insecte sau greutate maximă
* Verific dacă greutatea insectelor este mai mică sau egală cu capacitatea maximă de greutate a frunzei

def validateBroaste(self):  
 *'''  
 Validarea broastele primite. Atentie sirul 'mal' este rezervat pentru testarea scopului,  
 broastele se pot afla pe mal dar nu putem avea frunze cu id-ul 'mal'  
 '''* broaste = self.start[1]  
 frunze = self.start[2]  
 greutateFrunze = {fr : frunze[fr][3] - frunze[fr][2] for fr in frunze}  
 for br in broaste:  
 if broaste[br][0] <= 0:  
 raise ValueError('Exista broaste moarte (greutate <= 0)')  
 if broaste[br][1] != 'mal' and broaste[br][1] not in frunze:  
 raise ValueError('Exista broaste care stau pe frunze care nu exista')  
  
 frunzaCurenta = broaste[br][1]  
 greutateBroasca = broaste[br][0]  
 if frunzaCurenta != 'mal':  
 greutateFrunze[frunzaCurenta] -= greutateBroasca  
 if greutateFrunze[frunzaCurenta] < 0:  
 raise ValueError('Exita prea multe broaste pe o frunza')

În funcția validateBroaste fac următoarele validări:

* Verific dacă broaștele au greutate mai mare sau egală cu 1
* Verific dacă broaștele se află pe frunze care există
* Verific dacă broaștele au loc pe frunzele pe care stau