Documentație proiect Inteligență Artificială A*

Neagu Anastasia, grupa 234 – problema Șerpișori

1. Input

Programul se apelează din linia de comandă cu ajutorul a patru parametri: calea către folderul de input, calea către folderul de output, numărul de soluții căutate pentru toți algoritmii și timpul de timeout în secunde.

Un exemplu de apel ar fi: `python serpisori.py C:\Users\tiast\OneDrive\college\2\2.2\IA\astar_project C:\Users\tiast\OneDrive\college\2\2.2\IA\astar_project 1 300`

2. Parsarea fișierului de input

Fișierul de input este parsat respectând formatul dat ca exemplu; se efectuează optimizări detaliate mai jos.

3. Euristici

Euristica banală: Dacă starea este finală, se returnează 0; altfel, se returnează 1.

Prima euristică admisibilă: pentru fiecare șarpe care nu a ajuns la lungimea necesară pentru a forma latura pătratului cerut, se adaugă cea mai mică distanță Manhattan de la capul său la o stea – aceasta este distanța minimă care trebuie parcursă pentru a se apropia de o stare finală.

A doua euristică admisibilă: Se calculează numărul de bucăți din corpul fiecărui șarpe care nu sunt aliniate cu capul, atât pe verticală, cât și pe orizontală, dintre care se alege rezultatul minim, fiind numărul minim necesar acelui șarpe de a se îndrepta pentru a forma o latură a pătratului. Dintre acestea se alege maximul, presupunându-se că în acest număr de mișcări toți șerpii pot să ajungă la locul lor din poziția finală.

Euristica inadmisibilă: Înmulțirea rezultatului oricărei dintre cele două euristici admisibile cu un număr mare cum ar fi 100 sau 1000 rezultă într-o euristică inadmisibilă.

4. Exemple de fișiere de input

Un fișier de input care nu are soluții;

```
............
..aaA..B....
..cc...bb...
...C.D......
.....d......
.....d......
..*....*....
L=5
```

Un fișier de input care nu blochează pe niciun algoritm și să aibă ca soluții drumuri de lungime maxim 20;

Un fișier de input care să blocheze un algoritm la timeout, dar să fie minim un alt algoritm să dea soluție;

-

Un fișier de input care dă o stare inițială care este și finală.

aaAb.
D..b.
d..B.
dCcc.
L = 4

5. Validări și optimizări

Verificarea corectitudinii datelor de intrare – în timpul parsării fișierului de input se verifică corectitudinea lungimii șerpilor și absența altor caractere pe tablă.

Găsirea unui mod de a realiza din starea inițială că problema nu are soluții – se verifică dacă există în fișier un număr suficient de stele pentru a permite tuturor șerpilor să ajungă la lungimea necesară pentru a forma latura pătratului căutat; altfel, fișierul de input nu are soluție și căutarea este inutilă.

Găsirea unui mod de reprezentare a stării cât mai eficient – în loc să rețină toată matricea pentru fiecare stare, coordonatele tuturor părților unui șarpe sunt memorate, de la cap la coadă, într-un dicționar care are chei literele corespunzătoare fiecărui șarpe. Stelele de pe tablă sunt de asemenea memorate într-o listă de tupluri care reprezintă poziția fiecăreia.

Găsirea unor condiții din care să reiasă că o stare nu are cum sa conțină în subarborele de succesori o stare finală – în funcția de generare a succesorilor se verifică dacă vreun șarpe are lungimea mai mare sau egală cu latura pătratului dorit. Acel șarpe nu va putea niciodată să formeze o latură a pătratului, iar, nefiind posibil ca lungimea lui să scadă, subarborele format din succesorii lui nu va conține vreo stare finală. Din acest motiv, nodul nu va fi adăugat în lista de succesori.

Optimizare: implementarea eficientă a algoritmilor cu care se rulează programul, folosind eventual module care oferă structuri de date performante – algoritmul de BFS este implementat folosind modulul Queue, iar algoritmul A* este implementat cu o structură de tip Priority Queue.