Uninformed Search

Problema aleasa pentru utilizarea algoritmilor de uninformed search este problema Robotului in Labirint. Aceasta problema este reprezentata printr-un labirint 2D. In acest labirit exista un robot care, in starea initiala se afla in centrul labirintului. In cadrul labirintului exista o iesire. Scopul acestei probleme este ca robotul sa gaseasca iesirea, tinand cont ca acesta nu poate decat sa mearga in 4 directii: Nord, Est, Sud, Vest. Tot in labirint pot exista si ziduri peste care robotul nu poate trece, oprindu-se in acestea.

In rezolvarea acestei probleme am folosit pentru identificarea unei stari un tuplu (x, y) care reprezinta pozitia in matrice, care contine valorile 0, 1 si 2, unde valoarea 0 reprezinta zonele din labirint libere, unde robotul poate merge, valoarea 1 reprezinta zidurile peste care robotul nu poate trece, iar valoarea 2 reprezinta iesirea din labirint. Labirintul este citit dintr-un fisier de intrare, iar solutia este scrisa intr-un fisier de iesire, stare cu stare, unde pozitia actuala a robotului este notata cu “R”, iar pasii pe care i-a urmat robotul pana la acel moment au fost notati cu “Pn” unde n=0,x, x fiind numarul de pasi ai robotului.

In rezolvarea problemei am create o clasa numita RobotProblem care contine labirintul initial sub forma unei matrici, starea initiala sub forma unui tuplu (x, y) reprezentand pozitia in labirint si actiunile posibile. Aceasta clasa mai contine functii get care intorc diferite valori din labirint. Totodata, mai exista si o functie numita “get\_actions” primeste ca parametru o stare (un tuplu) si returneaza toate actiunile posibile pe care le poate lua robotul din aceasta stare si o functie numnita “get\_child\_node” care primiste 2 parametrii, starea si actiunea si returneaza noua stare plecand din starea primita ca parametru si luand actiunea primita ca parametru.

O a doua clasa utilitara este o clasa numita Solution care contine un dictionar unde cheia este starea copil, iar valoarea este starea parinte din care am ajuns la starea copil. Aceasta clasa exista pentru printarea solutiei pas cu pas sub un format inteligibil.

In cadrul acestei probleme s-au folosit 2 algoritmi:

1. Parcurgerea in latime
   1. Preia starea initiala a problemei
   2. Verifica daca starea initiala este chiar iesirea din labirint, daca da termina cu solutia avand doar starea initiala
   3. Altfel, initializeaza frontiera (care este o coada) cu starea initiala, si initializeaza lista de vizitati cu o lista goala
   4. Cat timp mai am elemente in frontiera atunci scoate prima stare din frontiera, adauga acest element in lista de vizitati (pentru a nu-l mai luat in considerare din nou).
   5. Pentru starea scoasa din coada, ia toate actiunile posibile din acea stare si pentru fiecare actiune ia copilul acestei stari. Daca acesta a fost deja explorat (adica daca este in lista de explorati sau in frontiera) nu-l mai luam in considerare.
   6. Adauga elemental in solutie (pentru a stii din ce nod am ajuns in acest copil)
   7. Daca am ajuns la solutie atunci printeaza solutia
   8. Altfel adauga copilul in frontiera
   9. Daca nu am ajuns la iesirea din labirint si coada frontiera este goala atunci algoritmul nu poate ajunge la o solutie (nu exista solutie pentru labirintul dat).
2. Iterative Deepening Search: Cautare in latime cu limita, unde limita pleaca de la 0 pana la infinit. In cazul de fata, am ales ca limita superioara n \* m unde n si m sunt lungimea, respective latimea labirintului, pentru ca solutia nu poate avea o lungime mai mare decat numarul maxim de pasi posibili in labirint. Exista 3 raspunsuri posibile pentru acest algoritm: Success, Failure si Cutoff unde Cutoff este returnat atunci cand se atinge limita de pasi.
   1. Algoritmul pleaca de la stearea initiala si limita la acest moment
   2. Daca am ajuns la solutie atunci returneaza Success
   3. Altfel, daca limita este 0 atunci returneaza Cutoff
   4. Altfel, marcheaza starea ca fiind deja vizitata. Fac acest lucru pentru a nu ma intoarce iarasi la starea aceasta
   5. Ia toate actiunile posibile din starea actuala
   6. Pentru fiecare stare posibila, ia copilul in care ajung din starea actuala facand actiunea data
   7. Daca a fost deja vizitat atunci nu-l luam in considerare
   8. Altfel, rezultatul algoritmului va fii un call recursive al algoritmului unde nodul va fii copilul, limita va fii limita – 1 si unde se da lista de vizitati actualizata pana la acest moment
   9. Daca rezultatul este success atunci adauga starea si copilul in solutie si returneaza din recursivitate Success

Parcurgerea in latime este mai eficienta deoarece, in cazul Iterative Deepening Search luam fiecare valoare a limite, sis pre exemplu, daca rezultatul se afla la 10 pasi cand avem limita 9 ajunge la un pas aproape de rezultat dar nu continuam catre acesta ci incepem din nou rularea algoritmului cu limita fiind 10 unde vom gasii solutia. Astfel, ajungem de mai multe ori aproape de solutie dar incercam diferite valori ale lui limit.