Inteligência Artificial

Resolução de problemas como problemas de pesquisa no espaço de estados

42470: João Cavaco

42501: Gerson Abreu

43014: Nuno Sousa

2020/2021

Problema 1

a)

Para a construção do tabuleiro e os devidos obstáculos, foi definido um predicado *lim/2* para verificar se uma determinada posição se encontra dentro do tabuleiro, uma série de factos "bloqueadas" que guarda as posições que o agente não pode aceder.

```
lim(X,Y):-
    X < 7,
    X >=0,
    Y < 7,
    Y>=0.

estado_inicial((1,0)).
estado_final((4,6)).

bloqueadas((0,5)).
bloqueadas((6,2)).
bloqueadas((5,2)).
bloqueadas((3,1)).
bloqueadas((3,2)).
bloqueadas((3,3)).
bloqueadas((6,4)).
```

Neste caso decidimos optar por definir um predicado op/4 indicando as ações que é possível o agente realizar. Estas ações são: dir (direita), sobe(subir), esq(esquerda) e desc(descer).

```
%representacao dos operadores
%op(Estado_atual, operador, estado_seguinte, custo)
op((X, Y), dir, (Z, Y), 1):-
    lim(X,Y),
    Z is X+1,
    \+ bloqueadas((Z,Y)).

op((X, Y),sobe, (X, Z), 1):-
    lim(X,Y),
    Z is Y+1,
    \+ bloqueadas((X, Z)).

op((X, Y), esq, (Z, Y), 1):-
    lim(X,Y),
    Z is X-1,
    \+ bloqueadas((Z,Y)).

op((X, Y), desce, (X, Z), 1):-
    lim(X,Y),
    Z is Y-1,
    \+ bloqueadas((X, Z)).
```

b)

O algoritmo de pesquisa não informada mais eficiente para resolver este problema foi o algoritmo de pesquisa em profundidade cujo código está na figura Pesquisa não informada.

c)

Pesquisa em Largura:

- Custo / Profundidade: 9
- Estados visitados: 169
- Estados em memória: 81

Pesquisa em Profundidade:

- Custo / Profundidade: 17
- Estados visitados: 25
- Estados em memória: 18

Deste modo concluímos que a pesquisa em profundidade mostra-se mais eficiente para este problema em específico, tendo menor número de estados visitados e menor número de estados guardados em memória.

d)

Para a realização da pesquisa informada foram utilizadas duas heurísticas, a heurística de Manhattan e uma alternativa. A de Manhattan calcula a distância de um estado ao estado final, sendo que os nós são expandidos de acordo com este valor, o nó de menor valor é expandido primeiro. Quanto à heurística alternativa, prevê sempre o valor 0 tornando assim o algoritmo de pesquisa A* na variante best-first search do algoritmo de pesquisa em largura.

```
% Heuristica de Manhattan
h((X,Y),Val):- estado_final((Xf,Yf)),
               mod(Vi, Xf, X), mod(Vj, Yf, Y),
               Val is (Vi+Vj).

% Heuristica alternativa
h((X,Y),0).
```

e)

O algoritmo de pesquisa informada mais eficiente utilizando ambas as heurísticas é o algoritmo de pesquisa greedy pois este visita e armazena menos estados em memória ao mesmo tempo que o algoritmo A*. O código do algoritmo encontra-se no anexo Pesquisa informada.

f)

Heuristica de Manhattan

Algoritmo A*:

- Estados visitados 98
- Estados em memória 97

Algoritmo Greedy:

- Estados visitados 9
- Estados em memória 15

Heuristica de valor 0

Algoritmo A*:

- Estados visitados 170
- Estados em memória 81

Algoritmo Greedy:

- Estados visitados 169
- Estados em memória 81

Problema 2

a)

Para a construção do tabuleiro e os devidos obstáculos, foi definido um predicado *lim/2* para verificar se uma determinada posição se encontra dentro do tabuleiro, uma série de factos "bloqueadas" que recebe as coordenadas das posições que o agente e a caixa não podem aceder, os estados são agora representados pela combinação das coordenadas do agente e da caixa.

```
lim(A,B) :-
    A < 7,
    A >= 0,
    B < 7,
    B >= 0.


estado_inicial(((1,0),(1,1))).
estado_final(((4,5),(4,6))).


bloqueadas((0,5)).
bloqueadas((6,2)).
bloqueadas((5,2)).
bloqueadas((3,1)).
bloqueadas((3,2)).
bloqueadas((3,3)).
bloqueadas((6,4)).
```

Neste caso decidimos optar por definir um predicado "*op*"indicando as ações que o agente e a caixa podem realizar. Para se moverem é adicionado ao eixo em que irá deslocar +1 ou -1, sendo assim possível movimentarem-se nas quatro direções possíveis. É necessário verificar as dimensões do tabuleiro e verificar se a casa não é bloqueada. Para a movimentação da caixa, o agente tem que estar do lado oposto ao movimento, movendo-se também, implementando assim o ato de empurrar.

```
%op(Estado_act,operador,Estado_seg,Custo)
op(((X,Y),(Xc,Yc)), (N,M), ((X1,Y1),(X1c,Y1c)), 1):-
    member(N,[1, -1]),
    member(M,[0]),
    X is Xc-N, Y is Yc - M,
    X1 = Xc, Y1 = Yc,
    X1c is Xc+N, Y1c is Yc + M,
    \+ bloqueadas((X1c,Y1c)),
    lim(X1c,Y1c).

op(((X,Y),(Xc,Yc)), (N,M), ((X1,Y1),(X1c,Y1c)), 1):-
    member(N,[0]),
    member(M,[1, -1]),
    X is Xc-N, Y is Yc - M,
    X1 = Xc, Y1 = Yc,
    X1c is Xc+N, Y1c is Yc + M,
    \+ bloqueadas((X1c, Y1c)),
    lim(X1c,Y1c).

op(((X,Y),(Xc,Yc)), (N,M), ((X1,Y1),(Xc,Yc)), 1):-
    member(M,[1,-1]),
    member(N,[0]),
    X1 is X+N, Y1 is Y+M, lim(X1,Y1),
(    X1,Y1) \= (Xc,Yc),
    \+ bloqueadas((X1, Y1)).
```

```
op(((X,Y),(Xc,Yc)), (N,M), ((X1,Y1),(Xc,Yc)), 1):-
    member(M,[0]),
    member(N,[1,-1]),
    X1 is X+N, Y1 is Y+M, lim(X1,Y1),
(    X1,Y1) \= (Xc,Yc),
    \+ bloqueadas((X1, Y1)).
```

b)

O algoritmo de pesquisa não informada mais eficiente a resolver este problema foi o algoritmo de pesquisa em profundidade cujo código está na figura Pesquisa não informada.

c)

Largura:

- Custo / Profundidade: 16
- Estados visitados: 5538
- Estados em memória: 978

Profundidade:

- Custo / Profundidade: 82
- Estados visitados: 1150
- Estados em memória: 109

d)

Para a realização da pesquisa informada foram utilizadas duas heurísticas, a heurística de Manhattan que tem em consideração a posição do agente e uma heurística de Manhattan que tem em consideração a posição da caixa.

```
%heuristicas
% Alterar o nome do predicado para h antes de utilizar
hAgente( ((Xa,Ya),_) ,Val):-
    estado_final(((Xfa,Yfa),_)),
    mod(Vi, Xfa, Xa),
    mod(Vj, Yfa, Ya),
    Val is (Vi+Vj).

hCaixa(( _,(Xc,Yc) ),Val):-
    estado_final((_,(Xfc,Yfc))),
    mod(Vi, Xfc, Xc),
    mod(Vj, Yfc, Yc),
    Val is (Vi+Vj).

mod(Vj,X,Y) :-X<Y,!, Vj is Y-X.
mod(Vj,X,Y) :-Vj is X-Y.
```

e)

O algoritmo de pesquisa informada mais eficiente utilizando ambas as heurísticas é o algoritmo de pesquisa greedy pois este visita e armazena menos estados em memória ao mesmo tempo que o algoritmo A*. O código do algoritmo encontra-se no anexo Pesquisa informada.

f)

Heuristica da Posição do Agente

Algoritmo A*:

- Estados visitados: 2920
- Estados em memória: 820

Algoritmo Greedy:

- Estados visitados: 664
- Estados em memória: 315

Heuristica da Posição da Caixa

Algoritmo A*:

- Estados visitados: 1631
- Estados em memória: 637

Algoritmo Greedy:

- Estados visitados: 654
- Estados em memóri: 75

Instruções para correr o código

Utilizar o swi-prolog visto que o gprolog ”rebenta”no ex2 com a pesquisa não informada devido ao tamanho da stack.

Carregar o ficheiro pni.pl para obter as respostas às questões do enunciado relativas à pesquisa não informada através dos seguintes comandos.

- pesquisa(ex1,largura).
- pesquisa(ex1,profundidade).
- pesquisa(ex2,largura).
- pesquisa(ex2,profundidade).

Carregar o ficheiro pi.pl para obter as respostas às questões do enunciado relativas à pesquisa informada através dos seguintes comandos.

- pesquisa(ex1,largura).
- pesquisa(ex1,profundidade).
- pesquisa(ex2,largura).
- pesquisa(ex2,profundidade).

Código

```
:-dynamic(estados_visitados/1).
:-dynamic(fechado/1).

pesquisa(Problema,Alg):-
  consult(Problema),
  estado_inicial(S0),
  reset_estados_visitados, % reset e inicializa
  pesquisa(Alg,[no(S0,[],[],0,0)],Solucao),
  escreve_seq_solucao(Solucao),
  retractall(fechado(_)).

guarda(L):-
  retract(estados_em_memoria(T)),
   maior(T,L,M),
    assertz(estados_em_memoria(M)).

pesquisa(largura,Ln,Sol):-
  pesquisa_largura(Ln,Sol).

pesquisa(profundidade,Ln,Sol):-
  pesquisa_profundidade(Ln,Sol).

pesquisa_largura([no(E,Pai,Op,C,P)|_],no(E,Pai,Op,C,P)):-
    estado_final(E).

pesquisa_largura([E|R],Sol):-
  expande(E,Lseg), esc(E),
  E=no(Ei,_Pai,_Op,_C,_P),
  assertz(fechado(Ei)),
  insere_fim(Lseg,R,Resto),
```

```
  inc_estados_visitados,
  length(Resto, L), guarda(L),
  pesquisa_largura(Resto,Sol).

pesquisa_profundidade([no(E,Pai,Op,C,P)|_],no(E,Pai,Op,C,P)):-
    estado_final(E).

pesquisa_profundidade([E|R],Sol):-
  expande(E,Lseg), esc(E),
  E=no(Ei,_Pai,_Op,_C,_P),
  assertz(fechado(Ei)),
  insere_fim(R,Lseg,Resto),
  inc_estados_visitados,
  length(Resto, L), guarda(L),
  pesquisa_profundidade(Resto,Sol).

expande(no(E,Pai,Op,C,P),L):-
  findall(no(En,no(E,Pai,Op,C,P),Opn,Cnn,P1),
(    op(E,Opn,En,Cn),
    P1 is P+1,
    Cnn is Cn+C,
    \+fechado(En)),
    L).

insere_fim([],L,L).
insere_fim(L,[],L).
insere_fim(R,[A|S],[A|L]):-
  insere_fim(R,S,L).

escreve_seq_solucao(no(E,Pai,Op,Custo,Prof)):- nl,nl,
  escreve_seq_accoes(no(E,Pai,Op,_,_)),
  write(custo(Custo)),nl,
```

```
  write(profundidade(Prof)),nl,
  estados_visitados(A),
  estados_em_memoria(B),
  write(estados_visitados(A)),nl,
  write(estados_em_memoria(B)),nl.

escreve_seq_accoes([]).
escreve_seq_accoes(no(E,Pai,Op,_,_)):-
  escreve_seq_accoes(Pai),
  write(e(Op,E)),nl.

esc(A):- write(A), nl.

inc_estados_visitados:-
  retract(estados_visitados(Y)),
  Z is Y + 1,
  assertz(estados_visitados(Z)).

reset_estados_visitados:-
  retractall(estados_visitados(_)),
  retractall(estados_em_memoria(_)),
  asserta(estados_visitados(0)),
  asserta(estados_em_memoria(0)).

maior(X,Y,Z):-
  X > Y -> Z = X;
  X < Y -> Z = Y;
  Z = X.
```

Pesquisa não informada

```
:-dynamic(fechado/1).
:-dynamic(maxNL/1).
```

```
:-dynamic(nos/1).

maxNL(0).
nos(0).

inc:-
  retract(nos(N)),
  N1 is N+1,
  asserta(nos(N1)).

actmax(N):-
  maxNL(N1),
  N1 >= N,!.

actmax(N):-
  retract(maxNL(_N1)),
  asserta(maxNL(N)).

pesquisa(Problema,Alg):-
  consult(Problema),
  estado_inicial(S0),
  pesquisa(Alg,[no(S0,[],[],0,1,0)],Solucao),
  escreve_seq_solucao(Solucao),
  retract(nos(Ns)),retract(maxNL(NL)),
  asserta(nos(0)),asserta(maxNL(0)),
  write(nos(visitados(Ns),lista(NL))).

pesquisa(a,E,S):- pesquisa_a(E,S).

pesquisa(g,E,S):- pesquisa_g(E,S).

pesquisa_a([no(E,Pai,Op,C,HC,P)|_],no(E,Pai,Op,C,HC,P)):-
```

```
  estado_final(E),
  inc.

pesquisa_a([E|R],Sol):-
  inc,
  asserta(fechado(E)),
  expande(E,Lseg),
  esc(E),
  insere_ord(Lseg,R,Resto),
  length(Resto,N),
  actmax(N),
  pesquisa_a(Resto,Sol).

pesquisa_g([no(E,Pai,Op,C,HC,P)|_],no(E,Pai,Op,C,HC,P)):-
  estado_final(E).

pesquisa_g([E|R],Sol):-
  inc,
  asserta(fechado(E)),
  expande_g(E,Lseg),
  %esc(E),
  insere_ord(Lseg,R,Resto),length(Resto,N), actmax(N),
  pesquisa_g(Resto,Sol).

expande(no(E,Pai,Op,C,HC,P),L):-
  findall(no(En,no(E,Pai,Op,C,HC,P),Opn,Cnn,HCnn,P1),
(    op(E,Opn,En,Cn),
    \+ fechado(no(En,_,_,_,_,_)),
  P1 is P+1,
  Cnn is Cn+C, h(En,H),
  HCnn is Cnn+H), L).
```

```
expande_g(no(E,Pai,Op,C,HC,P),L):-
  findall(no(En,no(E,Pai,Op,C,HC,P),
    Opn,Cnn,H,P1),
(    op(E,Opn,En,Cn),
    \+ fechado(no(En,_,_,_,_,_)),
  P1 is P+1, Cnn is Cn+C, h(En,H)), L).

insere_ord([],L,L).
insere_ord([A|L],L1,L2):-
  insereE_ord(A,L1,L3),
   insere_ord(L,L3,L2).

insereE_ord(A,[],[A]).
insereE_ord(A,[A1|L],[A,A1|L]):-
  menor_no(A,A1),!.
insereE_ord(A,[A1|L], [A1|R]):-
  insereE_ord(A,L,R).

menor_no(no(_,_,_,_,N,_), no(_,_,_,_,N1,_)):-
  N < N1.

escreve_seq_solucao(no(E,Pai,Op,Custo,_HC,Prof)):-
  write(custo(Custo)),nl,
  write(profundidade(Prof)),nl,
  escreve_seq_accoes(no(E,Pai,Op,_,_,_)),
  maxNL(Tata),
  write(Tata),nl.

escreve_seq_accoes([]).
escreve_seq_accoes(no(E,Pai,Op,_,_,_)):-
  escreve_seq_accoes(Pai),
  write(e(Op,E)),nl.
```

```
esc(A):- write(A), nl.
```

Pesquisa informada