Universidade De Évora
Departamento de Informática
Inteligência Artificial
Ano letivo 2019 - 2020

Inteligência Artificial: 1º Trabalho Prático

Alunos:
Luis Ressonha - 35003
Rúben Teimas - 39868

Docentes:
Paulo Quaresma

23 de Março de 2020

1 Introdução

Com este trabalho pretende-se que sejam utilizadas as capacidade de resolução de problemas como problemas de pesquisa no espaço de estados.

Para isto devemos entender a diferença entre pesquisa informada e não informada e os diferentes algoritmos abrangidos por cada uma das categorias. Devemos também compreender o conceito de heuristicas e ter a capacidade de propor heuristicas admissiveis para que possam posteriormente ser utilizadas na pesquisa informada.

2 Representação do Problema em Prolog

Para representar o estado inicial e final do nosso agente optámos por definir 2 predicados com um argumento, sendo este argumento um tuplo com as coordenadas das respetivas salas.

```
estado_inicial((1,1)).
estado_final((4,4)).
```

De seguida, de modo a "reconstruir"o labirinto, definimos um predicado *bloqueada/2* que recebe as coordenadas das 2 salas separadas pela porta.

```
% bloqueada(sala_1, sala_2).
bloqueada((1,1), (1,2)).
bloqueada((1,2), (1,1)).
bloqueada((2,1), (2,2)).
bloqueada((2,2), (2,1)).
bloqueada((3,1), (4,1)).
bloqueada((4,1), (3,1)).
bloqueada((3,2), (3,3)).
bloqueada((3,3), (3,2)).
bloqueada((4,2), (4,3)).
bloqueada((4,3), (4,2)).
```

Após represtar os estados e o espaço, optámos por definir um predicado *op/4* com as ações que o agente pode realizar, sendo elas mover-se para: a direita, baixo, cima e esquerda. Optámos por escolher esta ordem pois, estando o agente na sala (1,1) e querendo chegar à sala (4,4) as operações mais importantes serão mover-se para a direita e para baixo.

```
% op(estado_atual, operacao, estado_seguinte, custo).
op((X, Y), direita, (W, Y), 1):-
    X < 4,
    W is X+1,
    \+ bloqueada((X, Y), (W, Y)),
    nao_visitadas((W, Y)).

op((X, Y), baixo, (X, Z), 1):-
    Y < 4,
    Z is Y+1,
    \+ bloqueada((X, Y), (X,Z)),
    nao_visitadas((X,Z)).

op((X, Y), cima, (X, Z), 1):-
    Y > 1,
    Z is Y-1,
    \+ bloqueada((X, Y), (X,Z)),
    nao_visitadas((X,Z)).

```

```
20          op((X, Y), esquerda, (W, Y), 1):-
21              X > 1,
22              W is X-1,
23              \+ bloqueada((X, Y), (W, Y)),
24              nao_visitadas((W, Y)).
```

Por fim, de modo a evitar ciclos, definimos um predicado dinâmico *visitadas/1* onde são adicionadas as salas pelas quais o agente já passou. Para verificar se o agente já visitou uma sala definimos também o predicado *não_visitadas*.

```
1           nao_visitadas(X):-
2               \+ visitadas(X),
3               asserta(visitadas(X)).
```

3 Respostas

1.a) Tendo em conta este problema, o algortimo de pesquisa não informada que optámos por utilizar foi a **pesquisa em profundidade**, dado que utilizando esta pesquisa iremos visitar menos nós ainda que possam existir mais em memória simultâneamnete.

Para confirmar que este seria o algoritmo de pesquisa mais eficiente, decidimos resolver a pesquisa em profundidade em papel. Contudo, este método para além de aborrecido, não seria praticável caso o tamanho do labirinto aumentasse. Como tal, decidimos automatizar este processo utilizando os algortimos de pesquisa dados nas aulas e definindo 2 predicados dinâmicos *max_em_memoria/1* e *visitados/1*.

Em relação aos estados visitados limitámo-nos a incrementar o valor do predicado *visitados/1* cada vez que um estado era visitado.

Para descobrir o máximo de estados em memória em simultâneo definimos um predicado *max/3*. Este predicado recebe o valor do predicado *max_em_memoria/1* e o tamanho da estrutura(*Queue* caso seja a pesquisa em largura ou *Stack* caso seja em profundidade), devolvendo o maior dos 2 e atualizando de seguida o valor do predicado *max_em_memoria/1*.

```
1  :- dynamic(max_em_memoria/1).
2  :- dynamic(visitados/1).
3
4  % Pesquisa em Profundidade
5  pesquisa_profundidade([no(E,Pai,Op,C,P)|_],no(E,Pai,Op,C,P)) :-
6      retract(visitados(X)),
7      Y is X + 1,
8      asserta(visitados(Y)),
9      estado_final(E).
10 pesquisa_profundidade([E|R],Sol):-
11     expande(E,Lseg),
12     insere_inicio(Lseg,R,LFinal),
13     retract(max_em_memoria(X)),
14     length(LFinal, L),
15     max(X, L, Z),
16     asserta(max_em_memoria(Z)),
17     pesquisa_profundidade(LFinal,Sol).
18
19 expande(no(E,Pai,Op,C,P),L):-
20     findall(no(En,no(E,Pai,Op,C,P), Opn, Cnn, P1),
21                 (op(E,Opn,En,Cn), P1 is P+1, Cnn is Cn+C),
22                 L).
23
24 pesquisa_p:-
25     asserta(max_em_memoria(0)),
26     asserta(visitados(0)),
27     estado_inicial(S0),
28     pesquisa_profundidade([no(S0,[],[],0,0)], S),
29     write(S), nl,
30     write("Visitados: " ),
```

```
31          retract(visitados(Y)),
32          write(Y), nl,
33          write("Em Memoria: "),
34          retract(max_em_memoria(X)),
35          write(X).
```

1.b) Utilizando o algoritmo de pesquisa em profundidade modificado (acima apresentado) e a resolução da pesquisa em papel, podémos observar que, com a nossa representação do problema:

- **Foram Visitados:** 11 estados.
- **Passaram pela Stack:** 15 estados.
- **Máximo de Estados em Memória em Simultâneo:** 5 estados.

2)

2.a) A heuristica que propomos, dado o estado atual, calcula a diferença entre as coordenadas (X, Y) deste para o estado final e retorna a soma do módulo de cada uma das diferenças.

```
1           % Heuristica admissivel
2           heur((X, Y), R):-
3               estado_final(PF),
4               distancia((X, Y), PF, R).
5
6           %Calcula a distancia
7           distancia((X, Y), (W, Z), D):-
8               X1 is X-W,
9               abs(X1, AX),
10              Y1 is Y-Z,
11              abs(Y1, AY),
12              D is AX+AY.
13
14          abs(X, X):- X > 0.
15          abs(X, R):- R is -X.
```

2.b) O algoritmo de pesquisa informada que usámos foi o algortimo *A**, abordado nas aulas teóricas. Esta pesquisa, ao contrário de uma *pesquisa greedy*, tem em conta não só o custo da heuristica(n) (custo estimida de n até ao estado final) como o custo até chegar ao estado n.

Para a implementação do mesmo utilizámos o código de *pesquisa ordenada* facultado pelo docente. Apesar deste algortimo de pesquisa ser *não informado*, facilmente conseguimos adaptar para uma pesquisa informada simplesmente calculando o valor da heuristica. Assim os estados passam a ser ordenados pelo custo necessário para a eles ter chegado mais o valor da heuristica, chegando assim à *pesquisa A**.

```
%Pesquia A*
%representacao dos nos
%no(Estado,no_pai,Operador,Custo,CustoHeuristica,Profundidade)
:- dynamic(max_em_memoria/1).
:- dynamic(visitados/1).

pesquisa_aux([no(E,Pai,Op,C,CH,P)|_],no(E,Pai,Op,C,CH,P)) :-
    retract(visitados(X)),
    Y is X + 1,
    asserta(visitados(Y)),
    estado_final(E).
pesquisa_aux([E|R],Sol):-
    expandeInformada(E,Lseg),
    insere_ordenado(Lseg,R,LFinal),
    retract(max_em_memoria(X)),
    length(LFinal, L),
    max(X, L, Z),
    asserta(max_em_memoria(Z)),
    pesquisa_aux(LFinal,Sol).

expandeInformada(no(E,Pai,Op,C,CH,P),L):-
    findall(no(En,no(E,Pai,Op,C,CH,P), Opn, Cnn, CHn, P1),
                (op(E,Opn,En,Cn), P1 is P+1, Cnn is Cn+C, heur(
                    En, H), CHn is Cnn + H), L).

pesquisa_a:-
    asserta(max_em_memoria(0)),
    asserta(visitados(0)),
    estado_inicial(S0),
    pesquisa_aux([no(S0,[],[],0,0,0)], S),
    write(S), nl,
    write("Visitados: " ),
    retract(visitados(Y)),
    write(Y), nl,
    write("Em Memoria: " ),
    retract(max_em_memoria(X)),
    write(X).

ins_ord(E, [], [E]).
ins_ord(no(E,Pai,Op,C,CH,P), [no(E1,Pai1,Op1,C1,CH1,P1)|T],[no(E
    ,Pai,Op,C,CH,P),no(E1,Pai1,Op1,C1,CH1,P1)|T]) :- CH =< CH1.
ins_ord(no(E,Pai,Op,C,CH,P), [no(E1,Pai1,Op1,C1,CH1,P1)|T],[no(
    E1,Pai1,Op1,C1,CH1,P1)|T1]) :-
    ins_ord(no(E,Pai,Op,C,CH,P), T, T1).

insere_ordenado([],L,L).
insere_ordenado([A|T], L, LF):-
```

```
45          ins_ord(A,L,L1),
46          insere_ordenado(T, L1, LF).
```

2.3) Voltando a repetir o processo aplicado na pergunta *1.b)* para o algoritmo de *pesquisa A** chegámos aos seguintes resultados:

- **Foram Visitados:** 11 estados.
- **Máximo de Estados em Memória em Simultâneo:** 5 estados.

4 Respostas

Ainda que não tenhamos obtido qualquer diferença de eficiência entre a pesquisa informada e não informada, estamos cientes das vantagens da pesquisa informada.

Esta "invisibildade"dos resultados esperados deve-se principalmente ao reduzido tamanho do espaço sobre o qual estamos a trabalhar e à ordem pela qual definimos as operações realizadas pelo agente.

Nota: Após chamar um dos predicados de pesquisa é necessário reconsultar o ficheiro caso se queira chamar novamente um predicado de pesquisa.