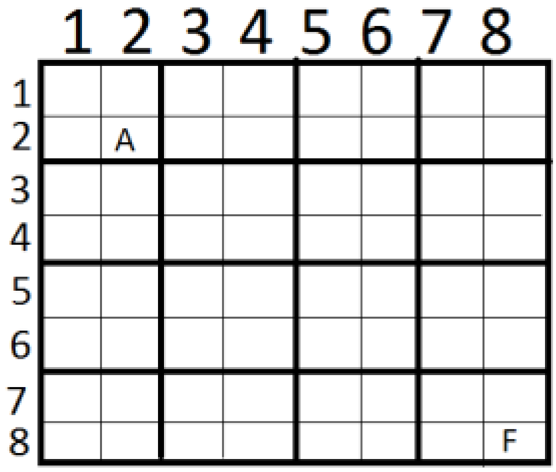
Inteligência Artificial

Trabalho 2015/2016

16/03/16

UNIVERSIDADE DE ÉVORA



Professora:
Irene Pimenta Rodrigues

Realizado por:
João Calhau - 31621
José Pimenta - 31677

Índice

Introdução

Este trabalho enquadra-se na disciplina de Inteligência Artificial e vamos abordar pesquisas não informadas e pesquisas informadas para resolução de um exercicio definido pela professora, em que temos de pesquisar um caminho numa matriz NxN de um dado ponto até chegar a outro ponto.
Iremos assim tentar dar o nosso melhor, e iremos tentar utilizar os métodos que achemos mais correctos ou propícios à boa evolução do trabalho e que no final se concretize o que nos é pedido.

Exercicio 1:

Para representar o espaço de estados decidimos fazê-lo utilizado 1 tuplo para o estado inicial que contém a letra do agente e a sala de partida(que é 1 tuplo com a posição X e Y), e o mesmo se aplica ao estado final:

```
%estado_inicial((agente, sala inicial))
estado_inicial((a, (2,2))).

%estado_final((agente, sala final)).
estado_final((a, (8,8))).
```

Para ajudar nas operações e definir o tamanho da matriz usamos ainda umas variáveis estáticas que definem a largura e profundidade:

```
%largura(Y)
largura(8).

%profundidade(X)
profundidade(8).
```

Para realizar o problema temos ainda as transições que não se podem realizar, ou seja, as portas bloqueadas (transição em ambos os sentidos):

```
%not_possible(casa_inicial, casa_final)
not_possible((1,2), (1,3)).
not_possible((1,3), (1,2)).
not_possible((2,3), (2,2)).
not_possible((2,2), (2,3)).
not_possible((3,4), (4,4)).
not_possible((4,4), (3,4)).
not_possible((4,5), (3,5)).
not_possible((3,5), (4,5)).
```

Para a resolução do problema temos ainda uma definição dynamic Visited que regista os nós que já foram passados de modo a evitar loops:

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

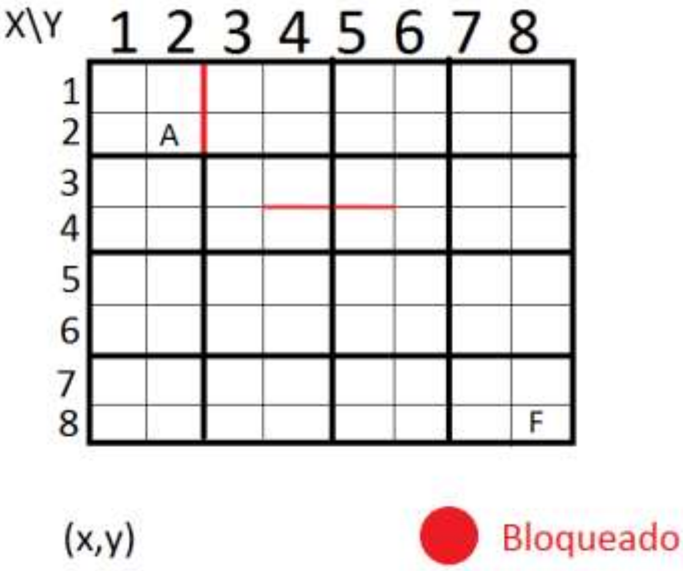
Para as operações, temos 4 que são andar para baixo, direita, cima e esquerda. Com base no nó em que se encontra e conforme a operação verifica-se se o nó a que se chega com essa operação é um nó que ainda não foi visitado ou se é um nó que não tenha porta bloqueante.
As operações estão definidas então da seguinte maneira:

```
%op(Estado_atual, operador, estado_seguinte, custo)
op((S, (X, Y)), desce, (S, (Z, Y)), 1) :-
        profundidade(Prof),
        X < Prof,
        K is X+1,
        (visited((K,Y))
            -> fail
            ;(not_possible((X, Y), (K, Y))
                -> fail
                ; asserta(visited((K,Y))),
                Z is X+1
            )
        ).


op((S, (X, Y)), dir, (S, (X, Z)), 1) :-
        largura(Larg),
        K is Y+1,
        Y < Larg,
        (visited((X,K))
            -> fail
            ;(not_possible((X, Y), (X, K))
                -> fail
                ;asserta(visited((X,K))),
                Z is Y+1
            )
        ).
```

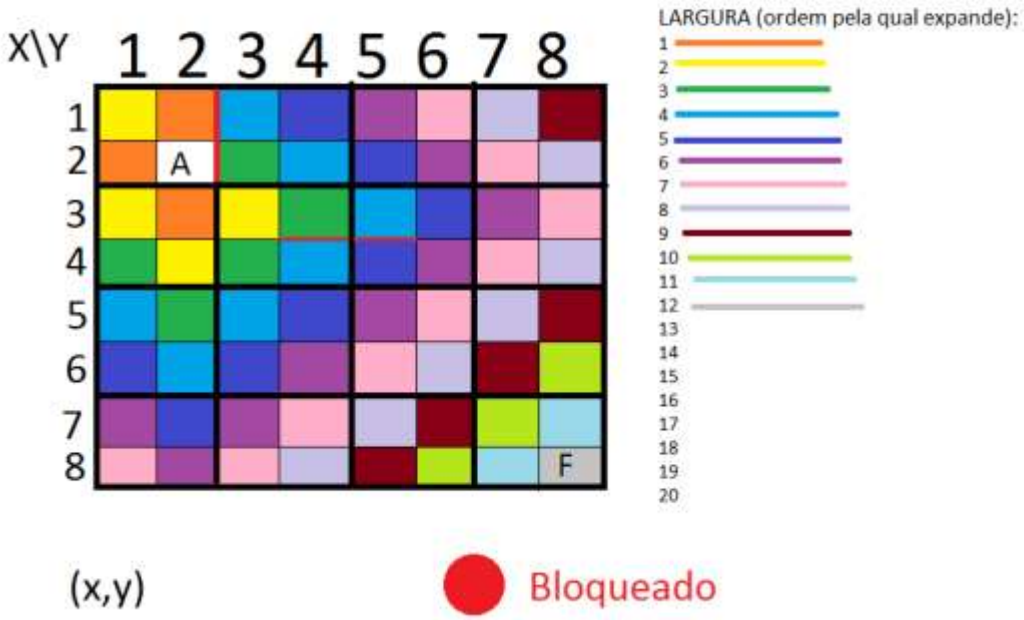
```
op((S, (X, Y)), sobe, (S, (Z, Y)), 1) :-
        X > 1,
        K is X-1,
        (visited((K,Y))
            -> fail
            ;(not_possible((X, Y), (K, Y))
                ->fail
                ;asserta(visited((K,Y))),
                Z is X-1
            )
        ).

op((S, (X, Y)), esq, (S, (X, Z)), 1) :-
        Y > 1,
        K is Y-1,
        (visited((X,K))
            -> fail
             ;(not_possible((X, Y), (X, K))
                -> fail
                ;asserta(visited((X,K))),
                Z is Y-1
            )
         ).
```

Pergunta 1 a)

Usando como exercicio o definido em cima de uma sala 8x8 e começando o agente na sala (2,2) e tentando ir para a sala (8,8) tendo as portas bloqueadas entre (1,2) e (1,3), (2,3) e (2,2), (3,4) e (4,4), e (4,5) e (3,5), um desenho representativo seria o seguinte:



Tendo em conta que guardamos os locais por onde passamos, em termos de largura o algoritmo vai expandindo como um “ripple”, propagando-se pelos vizinhos em redor. Logo em termos de largura o algoritmo ia funcionar algo como:



Assim e sabendo que são guardados nós por onde já se passou, no pior dos casos corre-se a matriz toda que será N*N, neste caso 8*8 = 64 nós + 1 que é o nó inicial para o qual tenta ir (pois este inicialmente não está registado como já visitado).
Sendo assim se o estado final se encontrar a uma distância tanto no X como no Y maior que metade de N (ou seja, neste caso se a diferença entre o estado inicial fosse >= a 4, em principio todos os nós seriam preenchidos através do algoritmo).

Numa pesquisa de profundidade, e tendo em a ordem das nossas operações em conta, sendo a ordem descer, direita, cima, esquerda, o algoritmo vai descendo nas posiçoes até não dar mais, e depois vai para a direita e atinge facilmente a posição final. Mas caso a ordem fosse por exemplo esquerda, cima, direita, descer, o algoritmo demora mais tempo a resolver.

Assim sendo, quando se verifica que estado_inicial(a,b) e estado_final (c,d), e que c-a >= N/2 e que d-b >= N/2, deve-se utilizar a pesquisa em profundidade.
Em caso contrário, o estado final está relativamente perto do estado inicial e assim uma pesquisa em largura é mais rápida.

A pesquisa iterativa funciona, mas devido a guardar-se os nós possiveis da matriz, e sendo este valo maximo = N*N, assim a pesquisa profundidade iterativa mostrou-se como o pior algoritmo.

Pergunta 1 b)

LARGURA:
Número de nós visitados: 64
Número máximo de nós em memória: 10

PROFUNDIDADE:
Número de nós visitados: 13
Número máximo de nós em memória: 18
(Ordem de operações definidas no nosso trabalho: descer, direita, cima, esquerda)

PROFUNDIDADE:
Número de nós visitados: 43
Número máximo de nós em memória: 22
(Ordem de operações: esquerda, cima, direita, descer)

PROFUNDIDADE ITERATIVA:
Número de nós visitados: 541
Número máximo de nós em memória: 19
(Ordem de operações definidas no nosso trabalho: descer, direita, cima, esquerda)

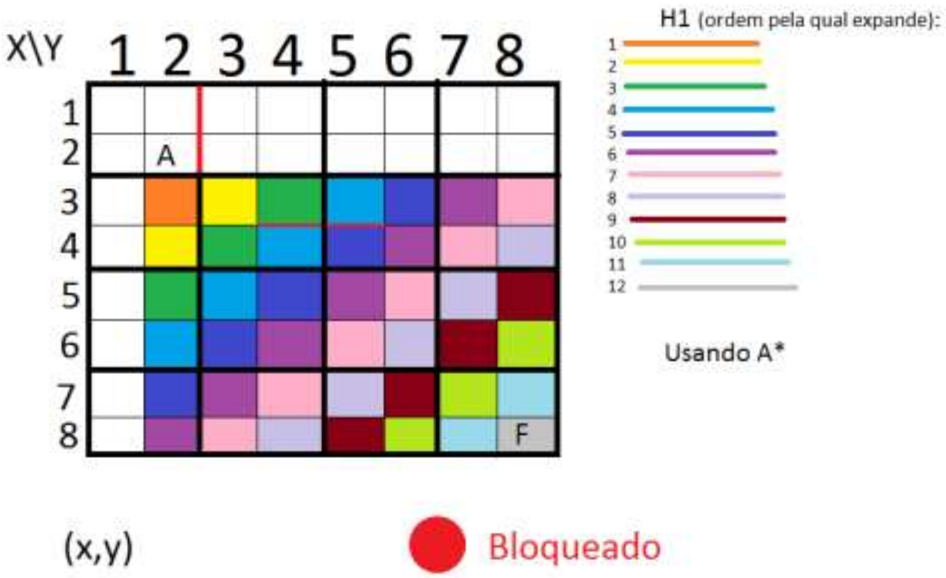
É o pior algoritmo de todos para este problema devido a ser um problema limitado pelo tamanho N*N e assim visitar nós repetidas vezes desnecessáriamente.

Pergunta 2 a) e b)

Heuristicas propostas:

h1 em que compara-se a diferença entre X atual e o X final, e a diferença entre o Y atual e o Y final. Soma-se o valor que é a distância do fim, e assim optando por um caminho mais curto. Este caminho segue a ordem das operações definidas no caso de igualdade de custo até ao final. Seguindo assim no caso do exercicio realizado nos sentidos para baixo e direita.

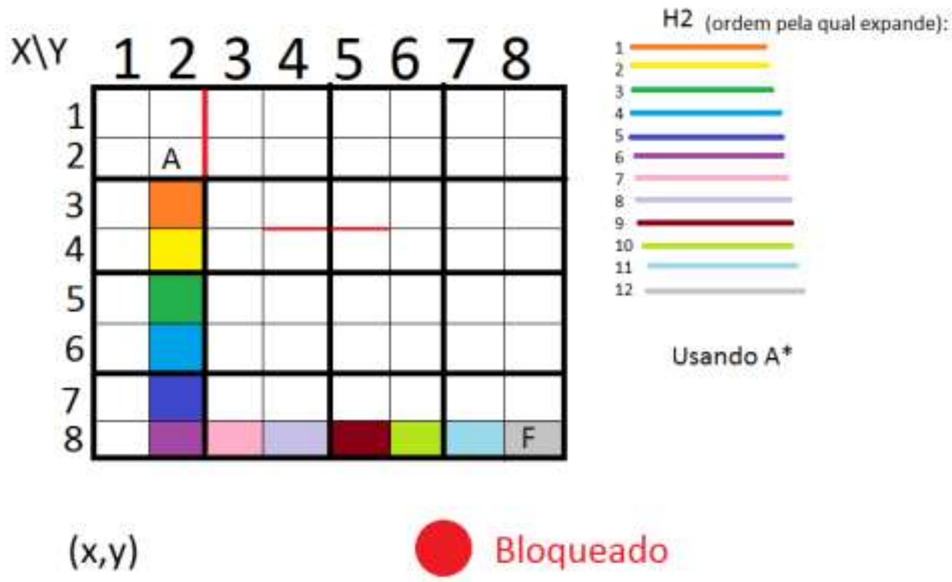
```
h1((Cx,Cy),C):-
    estado_final((Fx,Fy)),
    (Cx>=Fx

        -> K1 is Cx-Fx,
        (Cy>=Fy
            -> K2 is Cy-Fy,
               C is K1 + K2
            ;  K2 is Fy-Cy,
               C is K1 + K2
            )


        ; K1 is Fx-Cx,
         (Cy>=Fy
            -> K2 is Cy-Fy,
               C is K1 + K2
            ;  K2 is Fy-Cy,
               C is K1 + K2
            )
        ).
```



h2 em que compara-se a diferença entre X atual e o X final, e a diferença entre o Y atual e o Y final. Soma-se o valor que é a distância do fim, e assim optando por um caminho mais curto. Este caminho segue a ordem das operações definidas no caso de igualdade de custo até ao final.

É uma heuristica parecida à h1, mas difere no sentido que os valores de X são duplicados (após valor restante até ao final) e sendo assim, movimentar-se na horizontal (valor Y) apenas tem 1 de Custo C, enquanto que movimentar-se para cima ou para baixo tem custo 2, e assim prefere movimentar-se 1º verticalmente e só depois horizontalmente, sendo assim uma heuristica ainda melhor que h1, pelo menos neste exercicio.

```
h((Cx,Cy),C):-
    estado_final((Fx,Fy)),
    (Cx>=Fx
        -> K1 is Cx-Fx,
           K3 is K1*2,
        (Cy>=Fy
            -> K2 is Cy-Fy,
               C is K3 + K2
            ;  K2 is Fy-Cy,
               C is K3 + K2
            )
        ; K1 is Fx-Cx,
           K3 is K1*2,
         (Cy>=Fy
            -> K2 is Cy-Fy,
               C is K3 + K2
            ;  K2 is Fy-Cy,
               C is K3 + K2
            )
        ).
```



Pergunta 2 c)

H1:
Número de nós visitados: 43
Número máximo de nós em memória: 19

H2:
Número de nós visitados: 13
Número máximo de nós em memória: 18

As heuristicas seguem o algoritmo A*, considerado bom para este exercicio.
A melhor heuristica comprovada para este algoritmo foi assim o H2.

Conclusão

Após a realização deste trabalho ficámos a conhecer melhor como funcionam algoritmos de pesquisa não informada e pesquisa informada. Vimos que conforme o problema e conforme o problema seja disposto assim podem ser aplicados um tipo de algoritmos ou outro, e que nem sempre há um algoritmo perfeito para o nosso problema.

Tivémos de fazer coisas que inicialmente não pensámos, tais como guardar nós por onde já passámos e assim evitar criar loops, voltando para sitios por onde já passámos e assim passar num máximo de N*N nós (excepto no caso de profundidade iterativa que passa por muitos mais nós devido à sua repetição).

Em temos da 2ª parte usando heuristicas, foi um pouco complicado ao ínicio definirmos as heuristicas, mas após termos percebido como funcionam e como são utilizadas, mais facilmente as conseguimos definir e conseguir pôr a funcionar.