Manual Técnico

Inteligência Artificial - Escola Superior de Tecnologia de Setúbal

2020/2021

Problema do Quatro

Realizado por:

André Nascimento nº160221075

Eduardo Ferreira nº110221031

Arquitetura do sistema

Para melhor organização do código correspondente ao problema, foram criados 4 ficheiros:

Procura.LISP - Implementa os algoritmos relevantes para a resolução do problema.

Project.LISP - Liga todos os ficheiros LISP e executa-os como se fossem um só, neste ficheiro, também estão as funções de input/output.

Puzzle.LISP - Implementa as funções todas para se poder movimentar e manusear o tabuleiro.

problems.dat - Contem os diversos nós iniciais que vão ser avaliados.

Entidades e a sua implementação

Como entidades e tipos abstratos de dados, os alogoritmos possuem uma entidade chamada "initialNode" onde terá sempre o estado inicial do tabuleiro e das peças como forma de poder iniciar o algoritmo facilmente.

Como entidades opcionais, existe o "moves" onde basicamente vai simbolizar a jogada que está a ser verificada no momento. Esta entidade também tem a a capacidade de ter registado as jogadas que foram feitas anteriormente por aquele nó.

Temos também a entidade "abertos" onde será guardado as jogadas de todos os nós separadamente dependendo do algoritmo em questão será chamado o primeiro dos abertos (BFS e DFS) ou a jogada que tenha melhor valor (A*).

Temos a entidade "fechados" onde vai ser armazenado todas as jogadas/nós que já foram verificados.

Temos a entidade "counter" que servirá como forma de verificar quantas vezes a função vai iterar sobre ela mesma.

Temos a entidade "depth" que pertence exclusivamente ao algoritmo "DFS" que serve para saber a profundidade maxima que pode explorar.

Algoritmos e a sua implementação

(é preciso apresentar os dados referentes para cada problema e qual foi a sua solução, tambem é necessário indicar o numero de nós gerados, expandido, penetrancia, o factor de ramificação médio e o tempo de execução)

Os algoritmos que foram implementados neste projeto foram:

- BFS (Breath First Search)
- DFS (Depth First Search)
- A*

BFS:

```
CL-USER 2 > (iniciar)
Qual o algoritmo que pretende testar?BFS
Qual o problema que pretende testar?1
Timing the evaluation of (BFS BOARDNPIECES)

User time    =        0.000
System time  =        0.000
Elapsed time =        0.003
Allocation   = 1492296 bytes
0 Page faults
Calls to %EVAL    30567
(((PRETA QUADRADA ALTA CHEIA) (1 3)) "Numero de iteracoes" 1 "Penetrancia:" 1/16 "Numero de nos gerados:" 16)
```

Figura 1: Problema 1 com BFS.

```
CL-USER 3 > (iniciar)
Qual o algoritmo que pretende testar?BFS
Qual o problema que pretende testar?2
Timing the evaluation of (BFS BOARDNPIECES)

User time    =        0.125
System time  =        0.000
Elapsed time =        0.111
Allocation   = 60503944 bytes
0 Page faults
Calls to %EVAL    1272420
(((PRETA REDONDA BAIXA CHEIA) (1 1) (BRANCA REDONDA BAIXA OCA) (1 3)) "Numero de iteracoes" 25 "Penetrancia:" 1/204 "Numero de nos gerados:" 408)
```

Figura 2: Problema 2 com BFS.

```
CL-USER 4 > (iniciar)
Qual o algoritmo que pretende testar?BFS
Qual o problema que pretende testar?3
Timing the evaluation of (BFS BOARDNPIECES)

User time    =        0.859
System time  =        0.000
Elapsed time =        0.878
Allocation   = 490405208 bytes
0 Page faults
Calls to %EVAL    10090514
(((BRANCA QUADRADA ALTA OCA) (3 0) (BRANCA QUADRADA ALTA CHEIA) (1 0)) "Numero de iteracoes" 86 "Penetrancia:" 2/5445 "Numero de nos gerados:" 5445)
```

Figura 3: Problema 3 com BFS.

O problema 4, 5 e 6 chega ao limite da memória Heap do lispworks, isto devido ao facto que o BFS percorre os nós "horizontalmente", ou seja, ele verifica primeiro um nivel antes de descer para o proximo, devido a isso, ele vai fazer várias iterações e criar vários nós na lista de abertos, criando problemas na memória.

DFS:

```
CL-USER 7 > (iniciar)
Qual o algoritmo que pretende testar?DFS
Qual o problema que pretende testar?1
Timing the evaluation of (DFS BOARDNPIECES)

User time    =        0.015
System time  =        0.000
Elapsed time =        0.005
Allocation   = 1495456 bytes
0 Page faults
Calls to %EVAL    30493
(((PRETA QUADRADA ALTA CHEIA) (1 3)) "Numero de iteracoes" 1 "Penetrancia:" 1/16 "Numero de nos gerados:" 16)
```

Figura 4: Problema 1 com DFS.

```
CL-USER 8 > (iniciar)
Qual o algoritmo que pretende testar?DFS
Qual o problema que pretende testar?2
Timing the evaluation of (DFS BOARDNPIECES)

User time    =        0.000
System time  =        0.000
Elapsed time =        0.007
Allocation   = 3478416 bytes
0 Page faults
Calls to %EVAL    71959
(((PRETA REDONDA BAIXA CHEIA) (1 1) (BRANCA REDONDA BAIXA OCA) (1 3)) "Numero de iteracoes" 2 "Penetrancia:" 2/41 "Numero de nos gerados:" 41)
```

Figura 5: Problema 2 com DFS.

```
CL-USER 9 > (iniciar)
Qual o algoritmo que pretende testar?DFS
Qual o problema que pretende testar?3
Timing the evaluation of (DFS BOARDNPIECES)

User time    =        0.015
System time  =        0.000
Elapsed time =        0.026
Allocation   = 13890912 bytes
0 Page faults
Calls to %EVAL    286531
(((PRETA QUADRADA ALTA OCA) (2 1) (PRETA QUADRADA ALTA CHEIA) (1 2) (BRANCA QUADRADA BAIXA OCA) (1 1) (BRANCA QUADRADA ALTA OCA) (0 1) (BRANCA QUADRADA ALTA CH
" 1/51 "Numero de nos gerados:" 255)
```

Figura 6: Problema 3 com DFS.

```
CL-USER 10 > (iniciar)
Qual o algoritmo que pretende testar?DFS
Qual o problema que pretende testar?4
Timing the evaluation of (DFS BOARDNPIECES)

User time    =        0.015
System time  =        0.000
Elapsed time =        0.021
Allocation   = 11370496 bytes
0 Page faults
Calls to %EVAL    234204
(((PRETA QUADRADA ALTA OCA) (1 3) (PRETA QUADRADA ALTA CHEIA) (1 2) (BRANCA QUADRADA BAIXA CHEIA) (1 0) (BRANCA QUADRADA ALTA CHEIA) (1 1)) "Numero de iteraco
" 362)
```

Figura 7: Problema 4 com DFS.

```
CL-USER 11 > (iniciar)
Qual o algoritmo que pretende testar?DFS
Qual o problema que pretende testar?5
Timing the evaluation of (DFS BOARDNPIECES)

User time    =        0.015
System time  =        0.000
Elapsed time =        0.031
Allocation   = 15579440 bytes
0 Page faults
Calls to %EVAL    320252
(((BRANCA QUADRADA BAIXA CHEIA) (0 3) (BRANCA QUADRADA BAIXA OCA) (0 2) (BRANCA QUADRADA ALTA OCA) (0 0) (BRANCA QUADRADA ALTA CHEIA) (0 1)) "Numero de iterac
:" 734)
```

Figura 8: Problema 5 com DFS.

```
CL-USER 12 > (iniciar)
Qual o algoritmo que pretende testar?DFS
Qual o problema que pretende testar?6
Timing the evaluation of (DFS BOARDNPIECES)

User time    =        0.031
System time  =        0.000
Elapsed time =        0.030
Allocation   = 16116656 bytes
0 Page faults
Calls to %EVAL    332179
(((BRANCA QUADRADA BAIXA CHEIA) (0 3) (BRANCA QUADRADA BAIXA OCA) (0 2) (BRANCA QUADRADA ALTA OCA) (0 0) (BRANCA QUADRADA ALTA CHEIA) (0 1)) "Numero de iterac
:" 846)
```

Figura 9: Problema 6 com DFS.

A*:

```
CL-USER 13 > (iniciar)
Qual o algoritmo que pretende testar?A*
Qual o problema que pretende testar?1
Timing the evaluation of (A-STAR BOARDNPIECES)

User time    =        0.031
System time  =        0.000
Elapsed time =        0.005
Allocation   = 2010792 bytes
0 Page faults
Calls to %EVAL    40132
(((PRETA QUADRADA ALTA CHEIA) (0 2)) "Numero de iteracoes:" 1 " Penetrancia:" 1/16 " Nos gerados:" 16)
```

Figura 10: Problema 1 com A*.

```
CL-USER 15 > (iniciar)
Qual o algoritmo que pretende testar?A*
Qual o problema que pretende testar?3
Timing the evaluation of (A-STAR BOARDNPIECES)

User time    =        0.046
System time  =        0.000
Elapsed time =        0.057
Allocation   = 31059992 bytes
0 Page faults
Calls to %EVAL    638584
(((BRANCA QUADRADA ALTA OCA) (3 0) (BRANCA QUADRADA ALTA CHEIA) (1 0)) "Numero de iteracoes:" 3 " Penetrancia:" 2/209 " Nos gerados:" 209)
```

Figura 12: Problema 3 com A*.

```
CL-USER 4 : 1 > (iniciar)
Qual o algoritmo que pretende testar?A*
Qual o problema que pretende testar?4
Timing the evaluation of (A-STAR BOARDNPIECES)

User time    =        0.125
System time  =        0.000
Elapsed time =        0.124
Allocation   = 67814696 bytes
0 Page faults
Calls to %EVAL    1569753
(((PRETA QUADRADA ALTA CHEIA) (3 0) (BRANCA QUADRADA BAIXA CHEIA) (2 0) (BRANCA QUADRADA ALTA CHEIA) (1 0)) "Numero de iteracoes:" 3 " Penetrancia:" 3/299 " Nc
```

Figura 13: Problema 4 com A*.

```
CL-USER 17 > (iniciar)
Qual o algoritmo que pretende testar?A*
Qual o problema que pretende testar?5
Timing the evaluation of (A-STAR BOARDNPIECES)

User time    =        0.171
System time  =        0.000
Elapsed time =        0.172
Allocation   = 96995912 bytes
0 Page faults
Calls to %EVAL    2055642
(((PRETA QUADRADA ALTA CHEIA) (3 2) (BRANCA QUADRADA ALTA OCA) (1 2) (BRANCA QUADRADA ALTA CHEIA) (0 2)) "Numero de iteracoes:" 3 " Penetrancia:" 3/590 " Nos ;
```

Figura 14: Problema 5 com A*.

```
CL-USER 18 > (iniciar)
Qual o algoritmo que pretende testar?A*
Qual o problema que pretende testar?6
Timing the evaluation of (A-STAR BOARDNPIECES)

User time    =        0.328
System time  =        0.000
Elapsed time =        0.323
Allocation   = 182272264 bytes
0 Page faults
Calls to %EVAL    3871204
(((BRANCA QUADRADA BAIXA CHEIA) (0 3) (BRANCA QUADRADA BAIXA OCA) (0 2) (BRANCA QUADRADA ALTA OCA) (0 1) (BRANCA QUADRADA ALTA CHEIA) (0 0)) "Numero de iterac
```

Figura 15: Problema 6 com A*.

Descrição das opções tomadas

(Aspectos que não tinhamos a certeza e usamos uma coisa e o porque de a usarmos)

No algoritmo BFS, o grupo teve como ideia definir uma variavel let "abertos_let" cujo o que essa variavel vai fazer é:

- Caso seja a primeira iteração, ele irá mostrar todas as jogadas possiveis do tabuleiro inicial
- Caso não seja a primeira iteração, ele irá fazer um "apend" dos nós que já estavam abertos com os nós que abriram agora no nó estado que esta a ser verificado de momento, esta ordem foi escolhida assim com o objetivo de simular o BFS, pois o BFS percorre primeiro todos os nós de um nivel antes de descer para o proximo.

Depois temos uma variavel let denominada "fechados_let" que o que vai fazer é:

- Caso seja a primeira iteração, ele irá colocar o primeiro nó dos "abertos" nos "fechados".
- Caso não seja a primeira iteração, ele irá fazer um "append" onde irá guardar a jogada (nó) atual juntamente com os nós que tinham sido previamente guardados nos "fechados", registando assim, todos os nós que foram percorridos na árvore.

De seguida temos a variavel let chamada "cleanList" que consiste basicamente em remover dos "abertos_let" os nós repetidos que estejam mais à esquerda, embora esta variavel não seja necessária, o grupo achou correto utiliza-la por uma questão de performance, pois assim poderia-mos impedir de gerar nós que já tenham sido previamente gerados por um outro nó que possua as mesma jogadas, mas trocadas.

Por fim, temos as verificações, onde caso ele encontre uma solução, este termina e dá o resultado desejado. Caso os "abertos_let" seja null(esteja vazio), então ele irá dar uma mensagem que percorreu a árvore toda e não encontrou o resultado.

Por fim a verificação "T" chama o BFS outra vez para simular/recriar a recursividade, enviando como parametros o nó inicial, proxima jogada a ser analisada, os "abertos" sem a jogada a ser analisada, os "fechados" e um contador para saber quantas iterações foram necessárias.

No algoritmo DFS, o grupo mante-ve uma estrutura similar ao algoritmo BFS, mas com as alterações necessárias para simular corretamente o DFS, ou seja:

- O DFS possui os mesmos argumentos que o BFS mais um novo argumento chamado "depth" (profundidade), pois no DFS, é necessário possuir uma profundidade limite. O grupo optou por atribuir a essa variavel a quantidade de peças que o tabuleiro começa na reserva, pois técnicamente, é impossivel efetuar mais jogadas sequenciais do que existe peças.
- Iremos tambem ter uma variavel let chamada "abertos_let" que funciona da mesma maneira que a do BFS, com a unica diferenca de ser o "append" onde ele irá guardar os sucessores de uma jogada sempre à esquerda no lugar de estar à direita, de forma a simular-mos o descer de uma árvore, temos tambem uma nova verificação caso o número de jogadas atinja a profundidade, ele irá subir na árvore.
- Temos tambem a variavel let "fechados_let" que funciona exatamente da mesma maneira que a do BFS.
- Temos tambem uma variavel let "cleanList" que tem o mesmo objetivo que o do BFS e é utilizada pelo mesmo motivo.
- Por fim, temos as nossas condições, onde a primeira verificação é caso tenha obtido uma solução, o DFS devolverá a informação pedida.
- Temos tambem a verificação caso o "abertos_let" seja nulo, então não existe resultado.
- E por fim, no nosso "T", ele irá invocar o DFS novamente, para funcionar de uma forma puramente recursiva.

No algoritmo A*, irá ter uma estrutura similar aos anteriores, embora funcione de uma forma muito diferente. Em termos de argumentos, ele possui o nó inicial (tabuleiro + peças que começou o jogo), a variavel "moves", onde terá as jogadas efetuadas, ou seja, os nós que vamos trabalhar.

Temos tambem o argumento dos "abertos" e dos "fechados".

O código começa com uma variavel let chamada "abertos_let", onde caso os "abertos" seja nulo, devolverá todas as jogadas possiveis que se expandem do nó inicial e caso não seja nulo, irá guarda nos "abertos_let", todas as jogadas geradas do nó verificado mais a que ainda não foram verificadas e já se encontravam previamente nos abertos.

Depois iremos ter a varivel let chamada "best_move" onde através da heuristica a ser utilizada, ele irá, procurar nos abertos pela jogada que tenha o melhor (menor) valor.

Temos agora tambem a variavel let "fechados_let" que caso esteja na primeira iteração, irá registar o "best_move" e caso não seja a primeira iteração ele irá fazer um "append" da "best_move" atual com as que foram previamente utilizadas.

Por fim temos a nossa condição com verificações, onde a primeira é, caso tenho encontrado uma solução, ele irá devolver a informação pedida. A nossa proxima verificação é cao os "abertos" sejam nulo, então ele percorreu todas as jogadass possiveis e não encontrou uma solução.

por fim temos a nossa ultima verificação que chama o A* para criar uma recursividade enviando como argumentos o nó inicial, a "best_move" encontrada para que possa gerar os seus sucessores, os "abertos_let" sem a "best_move" pois ela vai ser verificada, os "fechados_let" e um contador para saber o número de iterações.

Limitações técnicas

Em relação ás limitações do projeto, este mostra um erro na memória Heap em certos problemas, isto acontece devido à versão grátis do lispworks é limitada.

Tambem é necessário extender a Stack para que possa correr certos problemas em certos algoritmos, por isso aconselha-se a utilizar o comando "(extend-current-stack 1000)", embora 1000 seja um bocado exagerado e desnecessário, garante que consegue correr

todos.