Manual Técnico

Logotipo da Instituto Politécnico de Setúbal - Escola Superior de Tecnologia de Setúbal

Puzzle dos Pontos e das Caixas

**Docentes da disciplina**

Joaquim Filipe

Cédric Grueau

**Um trabalho realizado por**

Artur Esteves 140221076 Engenharia Informática

Daniel Costa 120221058 Engenharia Informática

Conteúdo

[Introdução 3](#_Toc469791097)

[Ambiente de desenvolvimento 3](#_Toc469791098)

[Software 3](#_Toc469791099)

[Requisitos 3](#_Toc469791100)

[Algoritmo geral 4](#_Toc469791101)

[Conteúdo da aplicação 4](#_Toc469791102)

[Algoritmos 4](#_Toc469791103)

[Breadth-First 4](#_Toc469791104)

[Depth-First 5](#_Toc469791105)

[A\* 5](#_Toc469791106)

[IDA\* 5](#_Toc469791107)

[Heurística 5](#_Toc469791108)

[Funções 6](#_Toc469791109)

[Diagrama - Interação com Utilizador 9](#_Toc469791110)

[Análise dos resultados 9](#_Toc469791111)

[Resultados 9](#_Toc469791112)

[Quadro Comparativo 12](#_Toc469791113)

[Limitações da aplicação 12](#_Toc469791114)

[Conclusão 12](#_Toc469791115)

[Bibliografia 13](#_Toc469791116)

[Fontes Literárias 13](#_Toc469791117)

[Fontes de endereço 13](#_Toc469791118)

[Anexos 14](#_Toc469791119)

[Puzzle.Lisp 14](#_Toc469791120)

[Procura.Lisp 22](#_Toc469791121)

[Projeto.Lisp 31](#_Toc469791122)

# Introdução

No âmbito da disciplina Inteligência Artificial, do curso de Engenharia informática do Instituto Politécnico de Setúbal – Escola Superior de Tecnologia de Setúbal, foi proposto pelos docentes da disciplina o desenvolvimento de um projeto, em LISP, denominado por “Puzzle dos pontos e caixas”, de modo que os alunos possam aplicar os seus conhecimentos adquiridos nas aulas práticas e laboratórios da disciplina.

Este jogo consiste em fechar um determinado número de caixas a partir de uma configuração inicial do tabuleiro. Para alcançar o objetivo será necessário desenhar um arco entre dois pontos adjacentes, tanto na horizontal como vertical, e quando o tiver 4 arcos desenhados, entre pontos adjacentes, temos uma caixa fechada.

Para mais informações sobre as características do jogo é favor consultar o enunciado do projeto de forma a integrar-se no jogo e o seu modo de procedimento.

# Ambiente de desenvolvimento

## Software

A Tabela 1 Softwares utilizados, contém os softwares utilizados para o desenvolvimento deste projeto:

Tabela 1 Softwares utilizados

|  |  |
| --- | --- |
| LispWorks 7.0 (x64) | Ferramenta de desenvolvimento |
| Notepad++ | Editor de Texto |
| GIT | Repositório |

## Requisitos

A Tabela 2 Requisitos mínimos software, contém os requisitos mínimos e recomendados das máquinas de modo a suportar o LispWorks 7.0 (64bits):

Tabela 2 Requisitos mínimos software

|  |  |
| --- | --- |
| Requisitos mínimos | |
| Windows (x64) | **Windows (x32)** |
| Processador AMD64|Intel 64  Memória de disco: 170MB  Sistema Operativo: Windows Vista SP2 (x64) | Processador Pentium 4  Memória de disco: 170MB  Sistema Operativo: Windows Vista SP2 (x64) |
| Macintosh (x64) | **Macintosh (x32)** |
| Processador Intel-based Macintosh  Memória de disco: 285MB  Sistema Operativo: OS X v.10.5 | Processador Intel-based Macintosh  Memória de disco: 265MB  Sistema Operativo: OS X v.10.5 |

# Algoritmo geral

## Conteúdo da aplicação

Este projeto foi proposto com a seguinte estrutura:

* Problemas.dat

Ficheiro que contêm os tabuleiros, isto é, os problemas que serão para resolver pelo programa. Cada problema é uma lista que contém uma lista de arcos verticais e arcos horizontais.

* Projeto.lisp

Ficheiro responsável pela interação com o utilizador, imprimindo menus no ecrã da máquina com as opções.

* Puzzle.lisp

Ficheiro responsável pela gestão das regras do domínio da aplicação. Neste ficheiro são encontrados os operadores, validações e outras funções úteis para o seu desenvolvimento.

* Procura.lisp

Neste ficheiro encontra-se a procura genérica e os algoritmos, assim como os sucessores, funções de cálculo (penetrância e fator de ramificação) e funções auxiliares. Responsável pela execução da procura de um problema até encontrar a solução objetiva.

**NOTA:**

Gostaríamos de alertar para a utilização dos ficheiros nos diversos Sistemas Operativos (Mac OS X; LINUX; Elementary entre outros), na qual não tínhamos qualquer meio de utilizar nesses mesmos ambientes.

## Algoritmos

Neste projeto foi proposto 4 algoritmos, na qual ao longo das próximas secções vamos descrever algumas das suas características.

### *Breadth-First*

A procura em largura (em Inglês Breadth-first) é um método de procura não-informada que expande todos os nós de uma árvore de uma solução. Este método realiza uma procura exaustiva numa árvore inteira, sem considerar o seu alvo de procura, até que ele o encontre. Este algoritmo não utiliza uma heurística.

Do ponto de vista do algoritmo, todos os nós filhos obtidos pela expansão de um nó são adicionados a uma fila (FIFO). Normalmente, nós que ainda não foram estudados pelos seus vizinhos são colocados num *container* (uma lista) designada por “lista de aberto". Uma vez examinados, são colocados num outro *container, a “lista de fechados”.*

### *Depth-First*

A procura em profundidade (*Depth-first* em Inglês) é um algoritmo de procura em profundidade realiza uma procura, não-informada, que progride através da expansão do primeiro nó filho da árvore de procura, e se aprofunda cada vez mais, até que o alvo da procura seja encontrado ou até que ele se depare com um nó que não possui filhos (nó folha). Então a procura retrocede (*backtrack*) e começa no próximo nó.

### A\*

A procura A\* é um tipo de procura informada (método heurística), ao contrário dos algoritmos já apresentados, que utiliza heurísticas de forma a direcionar a procura de uma solução. Recorrendo a uma heurística e estimando o custo de cada nó até a solução (composto por **g** (custo até ao nó em questão) e **h** (custo restante até à solução) é possível ordenar os nós a serem expandidos e assim explorar os que são mais prováveis de levar a uma solução. Desta forma reduz-se o número de nós a expandir tornando a pesquisa mais rápida.

### IDA\*

O IDA\* utiliza o mesmo conceito que o A\*, mas só que tenta reduzir a quantidade de memória utilizada através do uso de limiares de custo, **L.**

Isto é só mantém em memória, e expandem-se, nós que tenham um valor inferior ao limite estimado no nó inicial. Caso se chegue a um caso em que não se encontram nós que satisfaçam esta condição, volta a fazer executar a procura e escolhe um novo limiar, que assumo como limite o valor mais baixo por entre os nós descartados. Repete-se este procedimento até encontrar a solução.

**NOTA:**

Tanto o algoritmo A\* como IDA\* algoritmos encontram a solução ótima caso a heurística utilizada seja admissível.

# Heurística

As heurísticas acrescentam conhecimento sobre o domínio da aplicação aos algoritmos de procura informados (A\* e IDA\*). São regras empíricas que podem oferecer aceleração no processo de procura. As heurísticas são admissíveis quando, para qualquer grafo, descobre sempre o caminho ótimo até achar o objetivo. Neste projeto foi desenvolvido duas heurísticas, sendo uma proposta pelos docentes e outra ao nosso critério.

* Heuristica1

Fornecida pelo enunciado do projeto, consiste numa subtração do número de caixas fechadas de um tabuleiro pelo número de caixas ainda a fechar, escritas pelo utilizador e ainda subtrai 1. Esta heurística é admissível, visto que o valor heurístico da solução dá 0 (h=0)(Ver Resultados). A forma apresentada será a seguinte:

* + - **h(x) = o(x) + c(x) – 1**, onde o(x) é o numero de caixas a fechar e c(x) o número de caixas fechadas de um determinado tabuleiro.
* Heuristica2

Heurística é proposta pelos alunos que realizaram o projeto onde consiste numa subtração do número de caixas fechadas de um tabuleiro pelo número de caixas ainda a fechar. Trata-se de uma heurística simples e eficaz e para alguns casos é admissível. A forma apresentada será a seguinte:

* + - **h(x) = o(x) + c(x),** onde o(x) é o numero de caixas a fechar e c(x) o número de caixas fechadas de um determinado tabuleiro.

# Funções

Segue uma lista das funções principais, existentes em cada ficheiro, assim como a sua descrição:

* Projeto.lisp

|  |  |
| --- | --- |
| Função | Descrição |
| (iniciar) | Função que inicializa o programa, chama a função que pede ao utilizador colocar a diretoria e esse será passada como argumento para carregar os ficheiros. |
| (inserir-diretoria) | Função que pede ao utilizador para colocar a raiz da pasta onde se encontram os ficheiros do projeto. |
| (load-files) | Função que carrega os ficheiros de puzzle e procura para o programa e executa o menu" |
| (menu-inicial) | Apresenta o menu principal do programa. |
| (iniciar-procura) | Função que pede ao utilizador toda a informação necessária para começar uma procura no espaço de estados. |
| (ler-tabuleiro) | Lista todos os problemas e faz a leitura numa diretoria que chama o ficheiro que contém todos os problemas. |
| (ler-numero-objectivo-caixas) | Lê o numero de caixas que o utilizador pretende fechar. |
| (ler-algoritmo) | Função que recebe do utilizador o algoritmo que pretende utilizar do problema anteriormente escolhido. |
| (ler-heuristica) | No caso do utilizador escolher o algoritmo a-asterisco ou ida-asterisco, será mostrado o menu que mostra as duas heurísticas que pretende usar na procura |
| (ler-profundidade) | Função que lê a profundidade máxima do utilizador caso este escolha o algoritmo depth-first (procura em profundidade). |
| (resultados) | Imprime os resultados do jogo. |
| (sem-resultados) | Função que imprime num ficheiro do tipo .DAT que não existe solução de determinado nó. |

* Procura.lisp

|  |  |
| --- | --- |
| Função | Descrição |
| (procura-generica) | Permite procurar a solução de um problema usando a procura no espaço de estados. A partir de um estado inicial,  de uma função que gera os sucessores e de um dado algoritmo |
| (procura-ida-asterisco) | Permite procurar a solução de um problema usando procura no espaço de estados. A partir de um estado inicial,  gera os sucessores e do algoritmo IDA\* a partir de um estado inicial e um limiar. |
| (novo-limiar) | Função auxiliar da procura do algoritmo IDA\*, que serve para implementar um novo limiar caso seja maior que o custo. |
| (bfs) | Função que executa o algoritmo de procura em largura. Faz uma junção da lista de abertos com os sucessores. |
| (dfs) | Função que executa o algoritmo de procura em profundidade. Faz uma junção da lista de sucessores com os abertos. |
| (a-asterisco) | Função que executa o algoritmo a\*. Faz uma junção da lista de sucessores com os abertos, tendo em conta a ordenação do valor do custo. |
| (ida-asterisco) | Função que executa o algoritmo ida\*. Faz uma validação do custo do nó com o limiar que é passado pelo argumento, caso contrário passará ao próximo nó sucessor. |
| (sucessores) | Função que retornada uma lista com todos os sucessores de um nó. |
| (existep) | Retorna valor verdadeiro, se o nó existir na lista. Para o algoritmo dfs, o conceito de nó repetido é particular. |
| (existe-solucao) | Verifica se existe uma solução ao problema numa lista de sucessores para o algoritmo dfs |
| (penetrancia) | Retorna o valor da penetrância dos nos gerados até o nó objetivo sobre dos nos totais gerados. |
| (fator-ramificacao) | Retorna o valor do fator de ramificação, compreendido entre uma margem. |

* Puzzle.lisp

|  |  |
| --- | --- |
| Função | Descrição |
| (criar-no) | Cria uma lista que representa um nó. Um nó é composto pelo estado que é o tabuleiro, este é um parâmetro obrigatório, é composto também por outros parâmetros, como a profundidade a que se encontra, pela heurística deste mesmo nó e pelo nó pai, ou seja, o nó que o gerou.  A profundidade e a heurística por omissão têm valor nil, enquanto que o nó pai por defeito é NIL. |
| (get-no-estado) | Retorna o estado do nó, que é representado pelo tabuleiro. |
| (get-no-profundidade) | Retorna a profundidade em que o nó se encontra. |
| (get-no-heuristica) | Retorna a heurística do nó. |
| (get-no-pai) | Retorna o nó pai deste nó, ou seja, o nó que gerou este nó. |
| (custo) | Retorna o valor do custo do nó (f). Soma do valor da profundidade com o valor heurístico. |
| (get-arcos-horizontais) | Retorna a lista dos arcos horizontais de um tabuleiro. |
| (get-arcos-verticais) | Retorna a lista dos arcos verticais de um tabuleiro. |
| (Inserir-arco-na-posicao) | Insere um arco (representado pelo valor [T]) numa lista que representa o conjunto de arcos dum tabuleiro. A posição representada pela linha e a coluna de destino são valores inteiros passados como argumentos. |
| (inserir-arco-vertical) | Insere um arco vertical (representado pelo valor [T]) num tabuleiro passado como argumento. |
| (inserir-arco-horizontal) | Insere um arco horizontal (representado pelo valor [T]) num tabuleiro passado como argumento. |
| (possível-adicionar-arco) | Recebe índices de linha e coluna e uma lista de arcos horizontais ou de arcos verticais e verifica se naquela posição o valor é [T], se for devolve [NIL], se for [NIL] devolve [T]. |
| (operadores) | Cria uma lista com todos os operadores do problema dos Pontos e das Caixas. |
| (numero-linhas-tabuleiro) | Retorna o número de linhas verticais de um tabuleiro. |
| (numero-colunas-tabuleiro) | Retorna o número de linhas verticais de um tabuleiro. |
| (caixas-fechadas) | Retorna o numero de caixas fechadas de um tabuleiro. |
| (contar-objetivo) | Função que irá contar se a caixa está fechada ou não, isto é, se a função auxiliar conta-caixa-fechada |
| (contar-nils-lista) | Função que irá contar o numero de NILS's existentes numa lista.Se não existir dará valor 0, ou seja, teriamos uma lista so com T,o que resulta uma caixa fechada. |
| (heuristica1) | Usada uma heurística que privilegia os tabuleiros com o maior número de caixas fechadas. |
| (heuristica2) | Heurística efetuada pelos alunos que se baseia na heuristica1. |
| (solucaop) | Devolve [T] se o número de caixas a fechar for igual ao número de caixas fechadas do nó, e devolve [NIL] se não for |
| (caminho-solucao) | Retorna o caminho até à solução. Ou seja, todos os estados desde a inicial à solução. |

# Diagrama - Interação com Utilizador

O diagrama de sequência que se segue, é uma representação de baixo nível, da interação que o utilizador terá quando inicia o programa na consola.



Ilustração 1 DIagrama sequências da interação inicial

# Análise dos resultados

## Resultados

Os resultados seguintes, serão mostrados por ordem dos problemas:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Problema A para 4 caixas objetivo | | | | | | |
|  | BFS | DFS | A\* - Heuristica1 | A\* - Heuristica2 | IDA\* - Heuristica1 | IDA\* - Heuristica2 |
| Nós Gerados | 21824 | 1890 | 3860 | 3860 | SN | SN |
| Nós Expandidos | 2127 | 1855 | 310 | 310 |
| Profundidade | 4 | 4 | 4 | 4 |
| Penetrância | 1,8328446^-4 | 0.0021164022 | 0.0010362695 | 0.0010362695 |
| Fator de ramificação | - | 6.229594 | 3.9558578 | 3.9558578 |
| Caixas Fechadas | 4 | 4 | 4 | 4 |
| Valor Heurístico | NIL | NIL | 0 | 1 |
| Tempo decorrido(s) | 1 | 0 | 0 | 0 |

\***SN – Sem Solução**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Problema B com 7 caixas objetivo | | | | | | |
|  | BFS | DFS | A\* - Heuristica1 | A\* - Heuristica2 | IDA\* - Heuristica1 | IDA\* - Heuristica2 |
| Nós Gerados | 198 | 16 | 198 | 198 | 85 | 85 |
| Nós Expandidos | 15 | 1 | 15 | 15 | 8 | 8 |
| Profundidade | 1 | 1 | 1 | 1 | 7 | 7 |
| Penetrância | 0.005050505 | 0.0625 | 0.005050505 | 0.005050505 | 0.08235294 | 0.08235294 |
| Fator de ramificação | 15.324541 | 1.682121 | 15.324541 | 15.324541 | 1.2273736 | 1.2273736 |
| Caixas Fechadas | 7 | 7 | 7 | 7 | 7 | 7 |
| Valor Heurístico | NIL | NIL | 1 | 2 | 0 | 1 |
| Tempo decorrido(s) | 0 | 0 | 0 | 0 | 0 | 0 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Problema C com 6 caxas objetivo | | | | | | |
|  | BFS | DFS | A\* - Heuristica1 | A\* - Heuristica2 | IDA\* - Heuristica1 | IDA\* - Heuristica2 |
| Nós Gerados | 2588 | 170 | 565 | 565 | 117 | 117 |
| Nós Expandidos | 168 | 90 | 35 | 35 | 9 | 9 |
| Profundidade | 2 | 6 | 2 | 2 | 8 | 8 |
| Penetrância | 7.727975E-4 | 0.03529412 | 0.003539823 | 0.003539823 | 0.06837607 | 0.06837607 |
| Fator de ramificação | 12.596058 | 1.682121 | 5.3200994 | 5.3200994 | 1.2273736 | 1.2273736 |
| Caixas Fechadas | 6 | 6 | 6 | 6 | 6 | 6 |
| Valor Heurístico | Nil | Nil | 0 | 1 | 0 | 1 |
| Tempo decorrido(s) | 0 | 0 | 0 | 0 | 0 | 0 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Problema D com 3 caixas objetivo | | | | | | |
|  | BFS | DFS | A\* - Heuristica1 | A\* - Heuristica2 | IDA\* - Heuristica1 | IDA\* - Heuristica2 |
| Nós Gerados | - | - | - | - | 1073 | 1073 |
| Nós Expandidos | - | - | - | - | 33 | 33 |
| Profundidade | - | - | - | - | 32 | 32 |
| Penetrância | - | - | - | - | 0.029822927 | 0.029822927 |
| Fator de ramificação | - | - | - | - | 1.2273736 | 1.2273736 |
| Caixas Fechadas | - | - | - | - | 3 | 3 |
| Valor Heurístico | - | - | - | - | 0 | 1 |
| Tempo decorrido(s) | - | - | - | - | 0 | 0 |

**NOTA:**

O problema D não está preenchido em alguns algoritmos, uma vez que apresenta erros de memória (*You are approaching the heap size limit for the Personal Edition of LispWorks.If you choose to continue now you are advised to save your work at regular intervals*.).

Ou seja, o programa LispWorks está intencionalmente limitado de modo a evitar os chamados “crashes” com outros programas. Assim, este problema acaba por ser uma limitação que apresentamos (Limitações da aplicação).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Problema E com 4 caixas objetivo | | | | | | |
|  | BFS | DFS | A\* - Heuristica1 | A\* - Heuristica2 | IDA\* - Heuristica1 | IDA\* - Heuristica2 |
| Nós Gerados | 1402 | 42 | 1402 | 1402 | 427 | 427 |
| Nós Expandidos | 36 | 1 | 36 | 36 | 13 | 13 |
| Profundidade | 1 | 1 | 1 | 1 | 12 | 12 |
| Penetrância | 7.132668E-4 | 0.023809525 | 7.132668E-4 | 7.132668E-4 | 0.028103045 | 0.028103045 |
| Fator de ramificação | 36.697667 | 1.682121 | 36.697667 | 36.697667 | 1.2273736 | 1.2273736 |
| Caixas Fechadas | 4 | 4 | 4 | 4 | 4 | 4 |
| Valor Heurístico | NIL | NIL | 1 | 2 | 0 | 1 |
| Tempo decorrido(s) | 0 | 0 | 0 | 0 | 0 | 0 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Problema F com 7 caixas objetivo | | | | | | |
|  | BFS | DFS | A\* - Heuristica1 | A\* - Heuristica2 | IDA\* - Heuristica1 | IDA\* - Heuristica2 |
| Nós Gerados | - | - | - | - | 5093 | 5093 |
| Nós Expandidos | - | - | - | - | 68 | 68 |
| Profundidade | - | - | - | - | 67 | 67 |
| Penetrância | - | - | - | - | 0.013155311 | 0.013155311 |
| Fator de ramificação | - | - | - | - | 1.2273736 | 1.2273736 |
| Caixas Fechadas | - | - | - | - | 7 | 7 |
| Valor Heurístico | - | - | - | - | 0 | 1 |
| Tempo decorrido(s) | - | - | - | - | 0 | 0 |

**NOTA:**

O problema F não está preenchido em alguns algoritmos, uma vez que apresenta erros de memória (*You are approaching the heap size limit for the Personal Edition of LispWorks.If you choose to continue now you are advised to save your work at regular intervals*.).

Ou seja, o programa LispWorks está intencionalmente limitado de modo a evitar os chamados “crashes” com outros programas. Assim, este problema acaba por ser uma limitação que apresentamos (Limitações da aplicação).

## Quadro Comparativo

No quadro seguinte (Tabela 3 Quadro comparativo dos resultados), podemos ter uma ideia do tempo de execução para cada problema, utilizando algoritmos diferentes:

Tabela 3 Quadro comparativo dos resultados – Tempo de execução

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Problemas | Breadh-First | Depth-First | A\* (heuristica1) | A\* (heuristica2) | IDA\* (heuristica1) | IDA\* (heuristica2) |
| A | 1 | 0 | 0 | 0 | 0 | 0 |
| B | 0 | 0 | 0 | 0 | 0 | 0 |
| C | 0 | 0 | 0 | 0 | 0 | 0 |
| D | - | - | - | - | 0 | 0 |
| E | 0 | 0 | 0 | 0 | 0 | 0 |
| F | - | 0 | - | - | 0 | 0 |

Podemos concluir que, de um modo geral, esta procura acaba por ser eficaz pois consegue resolver os problemas num curto tempo (< 1 segundo) como mostra a Tabela 3 Quadro comparativo dos resultados – Tempo de execução. Também temos que ter em atenção que o tempo que está a ser apresentado tem como base os Resultados, poderá existir variações do valor temporal ao utilizar outros dados.

# Limitações da aplicação

Neste projeto, existe uma limitação em relação a memória do software. Em certos problemas, quando tentamos soluciona-los, é apresentado uma mensagem de erro e fecha o programa. Estivemos a estudar o assunto e chegamos a conclusão que, como já referido, o programa LispWorks está intencionalmente limitado de modo a evitar os chamados “crashes” com outros programas.

# Conclusão

Com o desenvolvimento deste primeiro projeto, ao longo das últimas semanas foi bastante enriquecedor no que toca aos assuntos abordados de LISP, e são sem dúvida uma mais valia na futura vida académica e profissional, no entanto, e de notar que é uma linguagem que apresenta um grau de dificuldade exigente. Inicialmente, encontramos algumas dificuldades de codificação, mas assim que essa foi ultrapassada deparamo-nos com erros, de limitação de memória, ao testar a solução dos problemas, que acabou por ser uma limitação deste projeto. Em relação aos algoritmos foram todos bem-sucedidos, e de modo eficaz, isto é, o programa está capaz de apresentar a solução do problema num curto espaço de tempo, no entanto, podemos concluir que na procura dos algoritmos não informados, o Breadth-First é o que gera maior número de nós, comparativamente ao Depth-First, o que leva a ocupar mais memória, sendo menos eficiente. Relativamente aos algoritmos informados, devido ao uso da heurística, podemos verificar que geram menor número de nós comparativamente ao Depth-First e Breadth-First, o que leva a ocupar menor memória do programa.

# Bibliografia

## Fontes Literárias

* Filipe, Joaquim (2015) - Algoritmos de Procura em Espaço de Estados, Setúbal.

## Fontes de endereço

* LispWorks. Disponível em <http://www.lispworks.com/support/system-requirements.html> . Acesso em 7 de Dezembro de 2016.
* Design and Analysis of Algorithms. Disponível em <https://www.ics.uci.edu/~eppstein/161/960215.html>. Acesso em 2 de Dezembro de 2016.
* University of Washington, Computer Science & Engineering. Disponível em <http://faculty.washington.edu/dbp/SAPACLISP-1.x/basic-math.lisp>. Acesso em 23 de Novembro de 2016.
* University of Washington, Computer Science & Engineering. Disponível em <https://courses.cs.washington.edu/courses/cse415/06wi/notes/IDA.pdf>. Acesso em 29 Novembro de 2016.
* Heap Size, LispWorks User Guide - 11 Mar 2008*.* Disponível em <http://www.lispworks.com/documentation/lw51/LWUG/html/lwuser-456.htm> . Acesso em 17 Dezembro de 2016.
* Prof. Michel Gagnon, Resolução de Problemas. Disponível em <http://www.professeurs.polymtl.ca/michel.gagnon/Disciplinas/Bac/IA/ResolProb/resproblema.html>. Acesso em 18 dezembro de 2016.

# Anexos

## Puzzle.Lisp

;;;; puzzle.lisp

;;;; Disciplina de IA - 2016 / 2017

;;;; Programador: Artur Esteves - 140221076

;;;; Programador: Daniel Costa - 120221058

;;;; Funções do domínio do problema

;;; Construtor

;; cria-no

(defun cria-no (estado &optional (profundidade 0) (heuristica nil) (no-pai nil))

"Cria uma lista que representa um nó; Um nó é composto pelo estado que é o tabuleiro, este é um parâmetro obrigatório, é composto também por outros parâmetros, como

a profundidade a que se encontra, pela heurística deste mesmo nó e pelo nó pai, ou seja, o nó que o gerou. A profundidade e a heurística por omissão têm valor nil, enquanto que o nó pai por defeito é NIL"

(list estado profundidade heuristica no-pai)

)

;;; Metodos de Consulta

;;get-no-estado

(defun get-no-estado (no) "Retorna o estado do nó, que é representado pelo tabuleiro"

(car no)

)

;;get-no-profundidade

(defun get-no-profundidade (no) "Retorna a profundidade em que o nó se encontra"

(cadr no) ; Igual a: (car (cdr no))

)

;;get-no-heuristica

(defun get-no-heuristica (no) "Retorna a heurística do nó"

(caddr no) ; Igual a: (car (cdr (cdr no)))

)

;;get-no-pai

(defun get-no-pai (no) "Retorna o nó pai deste nó, ou seja, o nó que gerou este nó"

(cadddr no) ; Igual a: (car (cdr (cdr (cdr no))))

)

;;custo

(defun custo (no)"retorna o valor do custo do nó (f). Soma do valor da profundidade com o valor heuristico."

(+ (get-no-profundidade no)(get-no-heuristica no))

)

;;get-arcos-horizontais

(defun get-arcos-horizontais (tabuleiro) "Retorna a lista dos arcos horizontais de um tabuleiro"

(car tabuleiro)

)

;;get-arcos-verticais

(defun get-arcos-verticais (tabuleiro) "Retorna a lista dos arcos verticiais de um tabuleiro"

(car (cdr tabuleiro))

)

;;; Funcões auxiliares dos operadores

;;inserir-arco-na-posicao

(defun inserir-arco-na-posicao (indice lista) "Insere um arco (representado pelo valor [T]) no índice da lista recebida"

(cond

((null lista) nil)

((= indice 1) (cons T (cdr lista)))

(T (cons (car lista) (inserir-arco-na-posicao (- indice 1) (cdr lista))))

)

)

;;inserir-arco-na-posicao-aux

(defun inserir-arco-na-posicao-aux (linha coluna lista) "Insere um arco (representado pelo valor [T]) numa lista que representa o conjunto de arcos dum tabuleiro. A posição representada pela linha e a coluna de destino são valores inteiros passados como argumentos"

(cond

((null lista) nil)

((= linha 1) (cons (inserir-arco-na-posicao coluna (car lista)) (cdr lista)))

(T (cons (car lista) (inserir-arco-na-posicao-aux (- linha 1) coluna (cdr lista))))

)

)

;;; Operadores

;;arco-vertical

(defun inserir-arco-vertical (linha coluna tabuleiro) "Insere um arco vertical (representado pelo valor [T]) num tabuleiro passado como argumento"

(let ((arcos-horizontais (get-arcos-horizontais tabuleiro))

(arcos-verticais (get-arcos-verticais tabuleiro)))

(cond

((or (null tabuleiro) (not (possivel-adicionar-arco linha coluna arcos-verticais))) nil)

(T (cons arcos-horizontais (list (inserir-arco-na-posicao-aux linha coluna arcos-verticais))))

)

)

)

;;arco-horizontal

(defun inserir-arco-horizontal (linha coluna tabuleiro) "Insere um arco horizontal (representado pelo valor [T]) num tabuleiro passado como argumento"

(let ((arcos-horizontais (get-arcos-horizontais tabuleiro))

(arcos-verticais (get-arcos-verticais tabuleiro)))

(cond

((or (null tabuleiro) (not (possivel-adicionar-arco linha coluna arcos-horizontais))) nil)

(T (cons (inserir-arco-na-posicao-aux linha coluna arcos-horizontais) (list arcos-verticais)))

)

)

)

;;possivel-adicionar-arco

(defun possivel-adicionar-arco (linha coluna lista) "Recebe indices de linha e coluna e uma lista de arcos horizontais ou de arcos verticais e verifica se naquela posição o valor é [T], se for devolve [NIL], se for [NIL] devolve [T]"

(cond

((null lista) nil)

((= linha 1) (possivel-adicionar-arco-aux coluna (car lista)))

(T (possivel-adicionar-arco (- linha 1) coluna (cdr lista)))

)

)

;;possivel-adicionar-arco-aux

(defun possivel-adicionar-arco-aux (indice lista)

(cond

((null lista) nil)

((and (= indice 1) (eql (car lista) nil)) T)

(T (possivel-adicionar-arco-aux (- indice 1) (cdr lista)))

)

)

;;operadores

(defun operadores () "Cria uma lista com todos os operadores do problema dos Pontos e das Caixas"

(list 'inserir-arco-horizontal 'inserir-arco-vertical)

)

;;numero-linhas-tabuleiro

;; ATENÇÃO: Esta função devolve o número de linhas entre os pontos. Para devolver o número exacto de linhas é preciso somar 1 ao resultado da função."

(defun numero-linhas-tabuleiro (tabuleiro) "Retorna o número de linhas horizontais de um tabuleiro"

(get-dimensao-aux (car (get-arcos-verticais tabuleiro)))

)

;;numero-colunas-tabuleiro

;; ATENÇÃO: Esta função devolve o número de colunas entre os pontos. Para devolver o número exacto de colunas é preciso somar 1 ao resultado da função."

(defun numero-colunas-tabuleiro (tabuleiro) "Retorna o número de linhas verticais de um tabuleiro"

(get-dimensao-aux (car (get-arcos-horizontais tabuleiro)))

)

;;get-dimensao-aux

;; Teste: (get-dimensao-aux (get-arcos-horizontais (tabuleiro-teste1))) -> Resultado: 4

(defun get-dimensao-aux (lista) "Dada uma lista de arcos devolve o número de listas existentes"

(cond

((null lista) 0)

(T (+ 1 (get-dimensao-aux (cdr lista))))

)

)

;;; Função Validação de Caixas

;;caixas-fechadas

(defun caixas-fechadas (tabuleiro) "retorna o numero de caixas fechadas de um tabuleiro."

(cond

((null tabuleiro) nil)

(t

(contar-objetivo (caixas-fechadas-aux (get-arcos-horizontais tabuleiro)(get-cabecas-por-coluna tabuleiro)))

)

)

)

;;caixas-fechadas-aux

;; Esta função vai buscar os arcos horizontais e verticais e aplica uma função auxiliar de modo a juntar todas as caixas numa só lista.

(defun caixas-fechadas-aux (horizontais verticais) "retorna a lista completa das caixas verticais e horizontais de um tabuleiro"

(cond

(

(or (null horizontais)

(null verticais)

)

nil

)

(t

(append (caixas-fechadas-aux2 (car horizontais) (cadr horizontais) (car verticais))

(caixas-fechadas-aux (cdr horizontais) (cdr verticais))

)

)

)

)

;;caixas-fechadas-aux2

;; Esta função cria uma lista com a cabeça da 1ºlinha e da 2ºlinha e ainda a cabeça do resto da coluna.

(defun caixas-fechadas-aux2(linha1 linha2 coluna)"retorna uma lista com as caixas das duas linhas e da duas colunas."

(cond

(

(or

(null linha1)

(null linha2)

(null coluna)

)

nil)

(t

(cons

(list (car linha1)(car linha2)(car coluna)(cadr coluna))

(caixas-fechadas-aux2 (cdr linha1)(cdr linha2)(cdr coluna))

)

)

)

)

;;get-cabecas-lista-aux

(defun get-cabecas-lista-aux(lista n)"Função que vai buscar as cabeças de uma linha vertical ou horizontal,escolhida pelo utilizador, e cria numa nova lista."

(cond

((null lista)nil)

(t

(cons

(nth n (car lista))

(get-cabecas-lista-aux (cdr lista)n)

)

)

)

)

;;get-cabecas-por-coluna

(defun get-cabecas-por-coluna(tabuleiro)"retorna as listas de cada cabeca de cada coluna"

(contador 0 (numero-colunas-tabuleiro tabuleiro) 'get-cabecas-lista-aux (get-arcos-verticais tabuleiro))

)

;; contador

(defun contador (i tamanhoMax funcao &rest argumentos) "contador que executa determinada ação ao longo de uma lista. Para quando indice for = tamanho da lista."

(cond

((= i tamanhoMax) nil)

(T (ecase funcao

(get-cabecas-lista-aux (cons

(get-cabecas-lista-aux (car argumentos) i)

(contador (+ i 1) tamanhoMax funcao (car argumentos))))

)

)

)

)

;; contar-objetivo

(defun contar-objetivo(lista) "Função que irá contar se a caixa está fechada ou não, isto é, se a função auxiliar conta-caixa-fechada. "

(cond

((null lista)0)

((= (contar-nils-lista (car lista)) 0) (+ 1 (contar-objetivo (cdr lista))))

(t (contar-objetivo(cdr lista)))

)

)

;; contar-nils-lista

(defun contar-nils-lista (lista) "Função que irá contar o numero de NILS's existentes numa lista.Se não existir dará valor 0, ou seja, teriamos uma lista so com T,o que resulta uma caixa fechada."

(cond

((null lista) 0)

((equal 'NIL (car lista)) (+ 1 (contar-nils-lista (cdr lista))))

(T (contar-nils-lista (cdr lista)))

)

)

;;; Heuristicas

;; heuristica1

(defun heuristica1 (tabuleiro numero-caixas-a-fechar) "Usada uma heurística que priveligia os tabuleiros com o maior número de caixas fechadas"

(- numero-caixas-a-fechar (caixas-fechadas tabuleiro) 1)

)

;; heuristica2

(defun heuristica2 (tabuleiro numero-caixas-a-fechar) "Usada uma heurística que priveligia os tabuleiros com o maior número de caixas fechadas"

(/ (+ numero-caixas-a-fechar (caixas-fechadas tabuleiro)) 2)

)

;; Solução

;; solucaop

(defun solucaop (no numero-caixas-a-fechar) "Devolve [T] se o número de caixas a fechar for igual ao número de caixas fechadas do nó, e devolve [NIL] se não for"

(= (caixas-fechadas (get-no-estado no)) numero-caixas-a-fechar)

)

;; caminho-solucao

(defun caminho-solucao (no &optional (solucao nil)) "Retorna o caminho até à solução. Ou seja todos os estados desde o inicial à solução."

(cond

((null (get-no-estado no)) solucao)

(T (caminho-solucao (get-no-pai no) (cons (get-no-estado no) solucao)))

)

)

## Procura.Lisp

;;;; procura.lisp

;;;; Disciplina de IA - 2016 / 2017

;;;; Programador: Artur Esteves - 140221076

;;;; Programador: Daniel Costa - 120221058

;;;; Implementação dos algoritmos de procura

;;; Implementação das Procuras

;; procura-generica

(defun procura-generica (no-inicial prof-max f-solucao f-sucessores f-algoritmo lista-operadores f-heuristica numero-objectivo-caixas &aux (tempo-inicial (get-universal-time)))

"Permite procurar a solucao de um problema usando a procura no espaÃ§o de estados. A partir de um estado inicial,

de uma funcao que gera os sucessores e de um dado algoritmo. De acordo com o algoritmo pode ser usada um limite

de profundidade, uma heuristica e um algoritmo de ordenacao

"

(let ( (abertos (list no-inicial)) (fechados nil) )

(loop

(let ((no-atual (first abertos)))

(cond

((null abertos) (return nil))

((funcall f-solucao no-atual numero-objectivo-caixas) (return (list no-atual (length abertos) (length fechados) (- (get-universal-time) tempo-inicial) )))

((existep no-atual fechados f-algoritmo) (setf abertos (rest abertos))); se o no ja existe nos fechados e ignorado

(T

(let\* ((lista-sucessores (funcall f-sucessores no-atual lista-operadores f-algoritmo prof-max f-heuristica numero-objectivo-caixas))

(solucao (existe-solucao lista-sucessores f-solucao f-algoritmo numero-objectivo-caixas)));verifica se existe uma solucao nos sucessores para o dfs

(cond

(solucao (return (append (list solucao) (list (+ (length abertos) (length lista-sucessores)) (length fechados) (- (get-universal-time) tempo-inicial))))); devolve a solucao, com o tempo de execucao

(T (progn

(setf abertos (funcall f-algoritmo (rest abertos) lista-sucessores))

(setf fechados (cons no-atual fechados))

)

)

)

)

)

)

)

)

)

)

;;procura-ida-asterisco

(defun procura-ida-asterisco (no-inicial prof-max f-solucao f-sucessores f-algoritmo lista-operadores f-heuristica numero-objectivo-caixas &aux (tempo-inicial (get-universal-time)))

"Permite procurar a solucao de um problema usando procura no espaço de estados. A partir de um estado inicial,

gera os sucessores e do algoritmo IDA\* a partir de um estado inicial e um limiar.

"

(let ( (abertos (list no-inicial))

(fechados nil)

(limiar (funcall f-heuristica (get-no-estado no-inicial) numero-objectivo-caixas)))

(loop

(let ((no-atual (first abertos)))

(cond

((null abertos) (return nil))

((and (funcall f-solucao no-atual numero-objectivo-caixas) (>= limiar (custo no-atual))) (return (list no-atual (length abertos) (length fechados))))

((existep no-atual fechados f-algoritmo) (setf abertos (rest abertos))); se o no ja existe nos fechados e ignorado

(T

(let\* ((lista-sucessores (funcall f-sucessores no-atual lista-operadores f-algoritmo prof-max f-heuristica numero-objectivo-caixas))

(solucao (existe-solucao lista-sucessores f-solucao f-algoritmo numero-objectivo-caixas)));verifica se existe uma solucao nos sucessores para o dfs

(cond

((null lista-sucessores) (return nil)) ;; retorna nil quando esta a null a lista de sucessores

((and solucao (>= limiar (custo solucao))) (return (append (list solucao) (list (+ (length abertos) (length lista-sucessores)) (+ 1 (length fechados)) (- (get-universal-time) tempo-inicial)))))

(T

(progn

(setf limiar (novo-limiar limiar lista-sucessores))

(setf abertos (funcall f-algoritmo (rest abertos) lista-sucessores limiar))

(setf fechados (cons no-atual fechados))

)

)

)

)

)

)

)

)

)

)

;;; Funções auxiliares dos Sucessores

;;get-novo-limiar

(defun novo-limiar (limiar sucessores) "Função auxiliar da procura do algoritmo IDA\*, que serve para implementar um novo limiar caso seja maior que o custo."

(let ((min-custo (custo (first (sort sucessores #'< :key #'custo)))))

(cond

((null sucessores) limiar)

((> min-custo limiar) min-custo)

(T limiar)

)

)

)

;;; Algoritmos

;; Breadht-First (Procura em largura)

(defun bfs (abertos sucessores)

(append abertos sucessores)

)

;; Depth-First (Procura em profundidade)

(defun dfs (abertos sucessores)

(append sucessores abertos)

)

;; A\*

(defun a-asterisco (abertos sucessores)

(sort (append abertos sucessores) #'< :key #'custo)

)

;; IDA\*

(defun ida-asterisco (abertos sucessores limiar)

(cond

((null sucessores) abertos)

(

(> (custo (car sucessores))limiar) ;; se o custo do sucessor for <= que o limiar ele volta a fazer

(ida-asterisco abertos (cdr sucessores) limiar)

)

(T

(cons

(car sucessores)

(ida-asterisco abertos (cdr sucessores) limiar)

)

)

)

)

;;; Sucessores

;;sucessores

(defun sucessores (no operadores algoritmo-procura profundidade funcao-heuristica numero-objectivo-caixas) "Dado um nó é retornada uma lista com todos os sucessores desse mesmo nó"

(let\* ((operador (car operadores))

(numero-linhas (numero-linhas-tabuleiro (get-no-estado no)))

(numero-colunas (numero-colunas-tabuleiro (get-no-estado no)))

(lista-linhas-colunas-possiveis (cond

((eql operador 'inserir-arco-horizontal) (reverse (lista-combinacoes (+ numero-linhas 1) numero-colunas)))

((eql operador 'inserir-arco-vertical) (reverse (lista-combinacoes (+ numero-colunas 1) numero-linhas))))))

(cond

((null operadores) nil)

((and (equal 'dfs algoritmo-procura) (= (get-no-profundidade no) profundidade)) nil)

(T (append (sucessores-todas-possibilidades no operador lista-linhas-colunas-possiveis funcao-heuristica numero-objectivo-caixas) (sucessores no (cdr operadores) algoritmo-procura profundidade funcao-heuristica numero-objectivo-caixas)))

)

)

)

;;; Funções auxiliares dos Sucessores

;;sucessores-todas-possibilidades

(defun sucessores-todas-possibilidades (no operador possibilidades funcao-heuristica numero-objectivo-caixas)

(let\* ((primeira-possibilidade (car possibilidades))

(possibilidades-validas (not (null possibilidades)))

(resultado (cond (possibilidades-validas (sucessores-aux no (list operador primeira-possibilidade) funcao-heuristica numero-objectivo-caixas)) (T nil)))

(resultado-avaliado (cond ((null resultado) nil) (T (list resultado)))))

(cond

((null possibilidades) nil)

(T (append resultado-avaliado (sucessores-todas-possibilidades no operador (cdr possibilidades) funcao-heuristica numero-objectivo-caixas)))

)

)

)

;;sucessores-aux

(defun sucessores-aux (no lista-operador-parametros funcao-heuristica numero-objectivo-caixas)

(let\* ((operador (car lista-operador-parametros))

(tabuleiro (get-no-estado no))

(parametros (append (cadr lista-operador-parametros) (list tabuleiro)))

(resultado-operacao (apply operador parametros))

(resultado (cria-no resultado-operacao (+ 1 (get-no-profundidade no)) (cond ((not (null funcao-heuristica)) (funcall funcao-heuristica tabuleiro numero-objectivo-caixas)) (T nil)) no)))

(cond

((null resultado-operacao) nil)

(T resultado)

)

)

)

;;lista-combinacoes

(defun lista-combinacoes (maximo-linhas maximo-colunas) "Recebe uma lista e retorna um conjunto de listas que representa todas as combinacoes possiveis"

(cond

((zerop maximo-linhas) nil)

(T (append (combinacoes-numero-lista maximo-linhas (criar-lista-numeros maximo-colunas)) (lista-combinacoes (- maximo-linhas 1) maximo-colunas)))

)

)

;; combinacoes-numero-lista

(defun combinacoes-numero-lista (numero lista) "Devolve uma lista com várias listas compostas pelo elemento recebido e um elemento da lista recebida"

(let ((ultimo-elemento (car (last lista))))

(cond

((null lista) nil)

(T (cons (list numero ultimo-elemento) (combinacoes-numero-lista numero (reverse (cdr (reverse lista))))))

)

)

)

;; criar-lista-numeros

(defun criar-lista-numeros (tamanho &optional (valor-por-omissao 1)) "Devolve uma lista com o tamanho recebido como argumento e o valor dos elementos da lista é recebido se não por omissão têm todos o valor 1"

(cond

((zerop tamanho) nil)

(T (cons valor-por-omissao (criar-lista-numeros (- tamanho 1) (+ valor-por-omissao 1))))

)

)

;;; Funções de Verificações

;;existep

(defun existep (no lista-nos algoritmo)"Retorna verdadeiro se o nó existir na lista.Para o algoritmo dfs,o conceito de nó repetido é particular."

(let\* ((no-comparado (existep-aux no lista-nos))

(valido (not (null no-comparado))))

(cond

((and (eql algoritmo 'dfs) valido (= (get-no-profundidade no) (get-no-profundidade no-comparado))) T) ; algoritmo dfs

((and (eql algoritmo 'bfs) valido) T)

((and (eql algoritmo 'a-asterisco) valido) T)

((and (eql algoritmo 'ida-asterisco) valido) T)

(T nil)

)

)

)

;existep-aux

(defun existep-aux (no lista-nos)"Verifica se um nó existe numa lista de nos"

(cond

((null lista-nos) nil)

((equal (get-no-estado no) (get-no-estado (car lista-nos)))(car lista-nos))

(T (existep-aux no (cdr lista-nos)))

)

)

;; existe-solucao

(defun existe-solucao (lista f-solucao f-algoritmo numero-objectivo-caixas) "Verifica se existe uma solucao ao problema numa lista de sucessores para o algoritmo dfs"

(cond

((not (eql f-algoritmo 'dfs)) nil)

((null lista) nil)

((funcall f-solucao (car lista) numero-objectivo-caixas) (car lista))

(T (existe-solucao (cdr lista) f-solucao f-algoritmo numero-objectivo-caixas))

)

)

;;; Funções de Cálculo

;;penetrancia

(defun penetrancia (no nos-gerados) "Retorna o valor da penetrância dos nos gerados até o nó objetivo sobre dos nos totais gerados"

(cond

((not (equal nos-gerados 0)) (float (/ (get-no-profundidade no) nos-gerados)))

)

)

;;fator-ramificacao

; B+B^2+B^3+...+B^L=T [L comprimento do caminho até ao objetivo, T numero total de nós gerados]

(defun fator-ramificacao (L valor-t &optional (margem-erro 0.5) (bmin 1) (bmax 10e11)) "retorna o valor do fator de ramificaçao do no"

(let\* ((bmedio (/ (+ bmin bmax) 2)))

(cond

((< (- bmax bmin) margem-erro) (/ (+ bmax bmin) 2))

((> (- (f-polinomial L bmedio) valor-t) margem-erro) (fator-ramificacao L valor-t margem-erro bmin bmedio))

(T

(float(fator-ramificacao L valor-t margem-erro bmedio bmax))

)

)

)

)

;;; Funções auxiliares

(defun f-polinomial (polinomio x)

(cond

((= polinomio 1) x)

(t (+ (expt x polinomio) (f-polinomial (- polinomio 1) x)))

)

)

## Projeto.Lisp

;;;; projecto.lisp

;;;; Disciplina de IA - 2016 / 2017

;;;; Programador: Artur Esteves - 140221076

;;;; Programador: Daniel Costa - 120221058

;;;; Funções de interacção com o utilizador, de carregamento dos outros ficheiros do projecto e leitura e escrita em ficheiros

;;; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;;; Inicialização do Programa

;;iniciar

(defun iniciar ()"Função que inicializa o programa, chamando a função que apresenta o menu inicial."

(progn

(let ((caminho (inserir-diretoria)))

(load-files caminho)

)

)

)

;;inserir-diretoria

(defun inserir-diretoria() "Função que pede ao utilizador para colocar a raiz da pasta onde se encontram os ficheiros do projeto."

(progn

(format t "~%Introduza o caminho do ficheiro ~%")

(format nil (read-line))

)

)

;;load-files ()

(defun load-files (caminho) "Funcao que carrega os ficheiros de puzzle e procura para o programa e executa o menu"

(progn

(compile-file (concatenate 'string caminho "\\puzzle.lsp"))

(compile-file (concatenate 'string caminho "\\procura.lsp"))

(load (concatenate 'string caminho "\\puzzle.ofasl"))

(load (concatenate 'string caminho "\\procura.ofasl"))

(menu-inicial caminho)

)

)

;; menu-inicial

(defun menu-inicial (caminho) "Apresenta o menu principal do programa na consola. Sendo possível iniciar uma procura ou sair do programa"

(loop

(progn

(format t "~%> ------------------------------------------------------")

(format t "~%>| Puzzle dos Pontos e das Caixas |")

(format t "~%>| |")

(format t "~%>| 1. Iniciar Procura |")

(format t "~%>| 2. Regras do Jogo |")

(format t "~%>| 3. Exemplo Puzzle |")

(format t "~%>| 4. Sair |")

(format t "~%>| |")

(format t "~%> ------------------------------------------------------")

(format t "~%> Opcao")

(format t "~%> ")

(let ((opcao (ler-teclado)))

(cond

((not (numberp opcao)) (menu-inicial caminho))

((and (<= opcao 5) (>= opcao 1)) (cond

((= opcao 1) (iniciar-procura caminho))

((= opcao 2) (regras-jogo))

((= opcao 3) (imprime-tabuleiro))

((= opcao 4) (progn (format t "PROGRAMA TERMINADO")) (return))

)

)

(T (progn

(format t "~%> Opcao Invalida!")

(format t "~%> Opcoes Validas: [1, 2]")

(terpri)

(format t "~% ")

)

)

)

)

)

)

)

;; iniciar-procura

(defun iniciar-procura (caminho)

"Pede ao utilizador toda a informação necessário para começar uma procura no espaço de estados.

Sendo necessário fornecer o estado inicial, o algoritmo de procura e consoante o algoritmo escolhido é indicada a profundidade máxima e a heurística"

(let\* ( (no (cria-no (ler-tabuleiro caminho)))

(numero-objectivo-caixas (ler-numero-objectivo-caixas))

(algoritmo (ler-algoritmo))

(profundidade (cond ((eql algoritmo 'dfs) (ler-profundidade)) (T 9999)))

(heuristica (cond ((not (or (eql algoritmo 'dfs) (eql algoritmo 'bfs))) (ler-heuristica)) (T nil)))

(tempo-inicial (get-universal-time))

(solucao (cond (

(eql algoritmo 'ida-asterisco) (procura-ida-asterisco no profundidade 'solucaop 'sucessores algoritmo (operadores) heuristica numero-objectivo-caixas)

)

(T (procura-generica no profundidade 'solucaop 'sucessores algoritmo (operadores) heuristica numero-objectivo-caixas))

)

)

)

(cond

((null solucao) (sem-resultados no caminho))

(T

(resultados no profundidade algoritmo heuristica solucao tempo-inicial caminho)

)

)

)

)

;

;; ler-tabuleiro

(defun ler-tabuleiro (caminho) "Lista todos os estados inicias possíveis, recebe a escolha do utilizador e retorna a sua escolha caso esta seja válida"

(progn

(format t "~%>")

(format t "~%> Escolha o estado/tabuleiro inicial do problema ")

(format t "~%> Possibilidades: ")

(format t "~%> Tabuleiro A -> Caixas fechadas: 1 | Dimensao: 3 x 3")

(format t "~%> Tabuleiro B -> Caixas fechadas: 5 | Dimensao: 4 x 4")

(format t "~%> Tabuleiro C -> Caixas fechadas: 4 | Dimensao: 4 x 4")

(format t "~%> Tabuleiro D -> Caixas fechadas: 0 | Dimensao: 5 x 4")

(format t "~%> Tabuleiro E -> Caixas fechadas: 2 | Dimensao: 6 x 6")

(format t "~%> Tabuleiro F -> Caixas fechadas: 0 | Dimensao: 7 x 7")

(format t "~%> Tabuleiro G -> Discussão")

(format t "~%> Estado inicial: ")

(format t "~%> ")

(let\* ((opcao (ler-teclado))

(opcao-valida (existe-lista opcao '(a b c d e f))))

(with-open-file (ficheiro (concatenate 'string caminho "\\problemas.dat") :direction :input :if-does-not-exist :error)

;(with-open-file (ficheiro (concatenate 'string (diretoria-atual)"problemas.dat") :direction :input :if-does-not-exist :error)

(cond

((not opcao-valida) (progn

(format t "~%> Opcao Invalida!")

(format t "~% ")

(terpri)

(ler-tabuleiro caminho)))

((equal opcao 'a) (nth 0 (read ficheiro)))

((equal opcao 'b) (nth 1 (read ficheiro)))

((equal opcao 'c) (nth 2 (read ficheiro)))

((equal opcao 'd) (nth 3 (read ficheiro)))

((equal opcao 'e) (nth 4 (read ficheiro)))

((equal opcao 'f) (nth 5 (read ficheiro)))

;((equal opcao 'g) (nth 6 (read ficheiro))) ; se for adicionado ao nosso ficheiro é o problema 6, se for adicionado num ficheiro novo é o problema 1

)

)

)

)

)

;; ler-numero-objectivo-caixas

;; TODO: receber o tabuleiro e verificar o numero máximo de caixas a fechar!, para nao serem introduzidas por ex: 100000 caixas a fechar

(defun ler-numero-objectivo-caixas () "Le do utilizador o número objectivo de caixas a fechar"

(progn

(format t "~%> Qual o numero de caixas a fechar, ou seja, o objectivo ?")

(format t "~%> ")

(let ((resposta (ler-teclado)))

(cond

((not (numberp resposta)) (ler-numero-objectivo-caixas))

((>= resposta 1) resposta)

(T (ler-numero-objectivo-caixas))))

)

)

;; ler-algoritmo

(defun ler-algoritmo () "Recebe do utilizador o nome do algoritmo a usar para efectuar a procura"

(progn

(format t "~%> Qual o algoritmo que pretende usar para efectuar a procura?")

(format t "~%> Serao apresentadoss todos os algoritmos com e o nome a introduzir no sistema: ")

(format t "~%> Breadth-first Search -> bfs")

(format t "~%> Depth-first Search -> dfs")

(format t "~%> A\* Search -> a-asterisco")

(format t "~%> IDA\* Search -> ida-asterisco")

(format t "~%> Algoritmo a usar: ")

(format t "~%> ")

(let\* ((resposta (ler-teclado))

(opcao-valida (existe-lista resposta '(bfs dfs a-asterisco ida-asterisco))))

(cond

(opcao-valida resposta)

(T (progn

(format t "~%> Opcao Invalida!")

(format t "~% ")

(terpri)

(ler-algoritmo)

)

)

)

)

)

)

;; ler-heurística

(defun ler-heuristica () "Recebe do utilizador a decisão de qual heurística usar"

(progn

(format t "~%> Qual a heuristica que pretende aplicar na procura?")

(format t "~%> Possibilidades: ")

(format t "~%> 1. Proposta pelos professores")

(format t "~%> 2. Proposta pelos alunos")

(format t "~%> Heuristica a usar: ")

(format t "~%> ")

(let\* ((resposta (ler-teclado)))

(cond

((or (not (numberp resposta)) (or (> resposta 2) (< resposta 1)))

(progn

(format t "~%> Opcao Invalida!")

(format t "~% ")

(terpri)

(ler-heuristica)

))

(T (cond

((= resposta 1) 'heuristica1)

((= resposta 2) 'heuristica2)

)

)

)

)

)

)

;; ler profundidade do algoritmo dfs

(defun ler-profundidade () "Le do utilizador a profundidade para o algoritmo dfs"

(progn

(format t "~%> Qual a profundidade que pretende ?")

(format t "~%> ")

(let ((resposta (ler-teclado)))

(cond

((or (not (numberp resposta)) (or (> resposta 99999) (<= resposta 0)))

(progn

(format t "~%> Opcao Invalida! Valores compreendidos entre [0,9999]")

(format t "~% ")

(terpri)

(ler-profundidade)

))

(T resposta)

)

)

)

)

;; regras-jogo

(defun regras-jogo() "Apresenta as regras do jogo dos pontos e das caixas"

(format t "~%>

---------------- Regras do Puzzle dos Pontos e das Caixas -----------------

| O objetivo do puzzle consiste em fechar um determinado número de caixas |

| a partir de uma configuração inicial do tabuleiro. |

| Quando o número de caixas por fechar é atingido, o puzzle está resolvido. |

| A resolução do puzzle consiste portanto em executar a sucessão de traços |

| que permite chegar a um estado onde o número de caixas por fechar é |

| alcançado. |

---------------------------------------------------------------------------"

)

)

(defun imprime-tabuleiro ()

(format t "~%> Tabuleiro Exemplo com 1 caixa fechada!")

(format t "~%>

● ● ●--●

| |

● ● ●--●

|

● ● ● ●

|

● ● ● ●

")

)

;;; Estatisticas

;;sem-resultados

(defun sem-resultados (no-inicial diretoria) "Função que imprime num ficheiro do tipo .DAT que não existe solução de determinado nó"

(with-open-file (ficheiro (concatenate 'string diretoria "\\estatisticas.dat")

:direction :output

:if-exists :append

:if-does-not-exist :create)

(format ficheiro "~%Estado inicial: ~s ~%" no-inicial)

(format ficheiro "Sem Solução")

(format t "~%Estado inicial: ~s ~%" no-inicial)

(format t "Sem Solução")

)

)

;; resultados

(defun resultados (no-inicial profundidade-maxima algoritmo heuristica solucao tempo-inicial diretoria) "Função que imprime num ficheiro do tipo .DAT as estatisticas do jogo."

(let\*

(

(tamanho-lista-abertos (car (cdr solucao)))

(no-solucao (car solucao))

(estado-solucao (get-no-estado no-solucao))

(tamanho-lista-fechados (+ (car (cdr (cdr solucao))) 1))

(nos-gerados (- (+ tamanho-lista-abertos tamanho-lista-fechados) 1))

(profundidade (get-no-profundidade (get-no-estado solucao)))

(no-final (get-no-estado solucao))

(tempo (- (get-universal-time) tempo-inicial))

(caminho (caminho-solucao no-solucao))

(valor-heuristico (get-no-heuristica no-solucao))

)

(with-open-file (ficheiro (concatenate 'string diretoria "\\estatisticas.dat")

:direction :output

:if-exists :append

:if-does-not-exist :create)

;; Esta parte será escrita no ficheiro do tipo .DAT

(format ficheiro "Gerado em ~s~%" (current-date-string))

(format ficheiro "~%Estado inicial: ~s ~%" no-inicial)

(format ficheiro "~%Estado final: ~s ~%" estado-solucao)

(format ficheiro "~%Profundidade maxima: ~s ~%" profundidade-maxima)

(format ficheiro "~%Algoritmo: ~s ~%" algoritmo)

(format ficheiro " ~s ~%" heuristica)

(format ficheiro "~%Profundidade: ~s ~%" profundidade)

(format ficheiro "~%Nos Gerados: ~s ~%" nos-gerados)

(format ficheiro "~%Nos expandidos: ~s ~%" tamanho-lista-fechados)

(format ficheiro "~%Penetrancia: ~s ~%"(penetrancia no-final nos-gerados));

(format ficheiro "~%Fator de Ramificacao: ~s ~%" (fator-ramificacao profundidade tamanho-lista-fechados))

(format ficheiro "~%Valor Heuristico: ~s ~%" valor-heuristico)

(format ficheiro "~%Caminho ate a solucao: ~s ~%" caminho)

(format ficheiro "~%Caixas Fechadas: ~s ~%" (caixas-fechadas (get-no-estado no-final)))

(format ficheiro "~%Tempo decorrido: ~s segundos ~%" tempo)

(format ficheiro "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_~%")

)

;;Esta parte será mostrada na consola

(format t "Gerado em ~s~%" (current-date-string))

(format t "~%Estado inicial: ~s ~%" no-inicial)

(format t "~%Estado final: ~s ~%" estado-solucao)

(format t "~%Profundidade maxima: ~s ~%" profundidade-maxima)

(format t "~%Algoritmo: ~s ~%" algoritmo)

(format t "~%Heuristica: ~s ~%" heuristica)

(format t "~%Profundidade: ~s ~%" profundidade)

(format t "~%Nos Gerados: ~s ~%" nos-gerados)

(format t "~%Nos expandidos: ~s ~%" tamanho-lista-fechados)

(format t "~%Penetrancia: ~s ~%" (penetrancia no-final nos-gerados))

(format t "~%Fator de Ramificacao: ~s ~%" (fator-ramificacao profundidade tamanho-lista-fechados))

(format t "~%Valor Heuristico: ~s ~%" valor-heuristico)

(format t "~%Caminho ate a solucao: ~s ~%" caminho)

(format t "~%Caixas Fechadas: ~s ~%" (caixas-fechadas (get-no-estado no-final)))

(format t "~%Tempo decorrido: ~s segundos ~%" tempo)

(format t "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_~%")

)

)

;;; Funções Auxiliares

;; existe-lista

(defun existe-lista (elemento lista) "Retorna [T] se o elemento recebido existe na lista recebida e devolve [NIL] se o elemento não existe"

(cond

((null lista) nil)

((eql elemento (car lista)) T)

(T (existe-lista elemento (cdr lista)))

)

)

;

;; ler-teclado

(defun ler-teclado () "Ler do teclado algo do utilizador"

(read)

)

;

;; current-date-string [Data actual]

(defun current-date-string () "Retorna a data no formato de string"

(multiple-value-bind (sec min hr day mon yr dow dst-p tz)

(get-decoded-time)

(declare (ignore dow dst-p tz))

(format nil "~A-~A-~A : ~A:~A:~A" yr mon day hr min sec)

)

)