Ficha de laboratório Nº 6: O Problema das Vasilhas de Água

Inteligência Artificial - Escola Superior de Tecnologia de Setúbal

Prof. Joaquim Filipe
Eng. Filipe Mariano

Objetivos da ficha

Com este laboratório pretende-se que os alunos pratiquem a implementação de métodos de procura em espaço de estados, sendo os principais objetivos:

- Reforçar a aprendizagem da sintaxe da linguagem Lisp
- Utilizar tipos abstratos de dados para representar os estados de um problema em Lisp
- Implementar a procura de soluções de um problema em Lisp

1. O Problema das Vasilhas de Água

Pretende-se medir 1 litro de água com o recurso a duas vasilhas (A e B), não graduadas, que cheias contêm exatamente 3 e 5 litros de água (respetivamente) e um tanque de água. Para isso, pode optar por uma das seguintes ações:

1. Encher uma vasilha utilizando a água do tanque;
2. Verter o conteúdo de uma das vasilhas na outra;
3. Deitar o conteúdo de uma vasilha para o tanque.

A solução do problema pode ser obtida através de uma sucessão das três operações apresentadas, até que alguma das vasilhas fique com 1 litro de água.

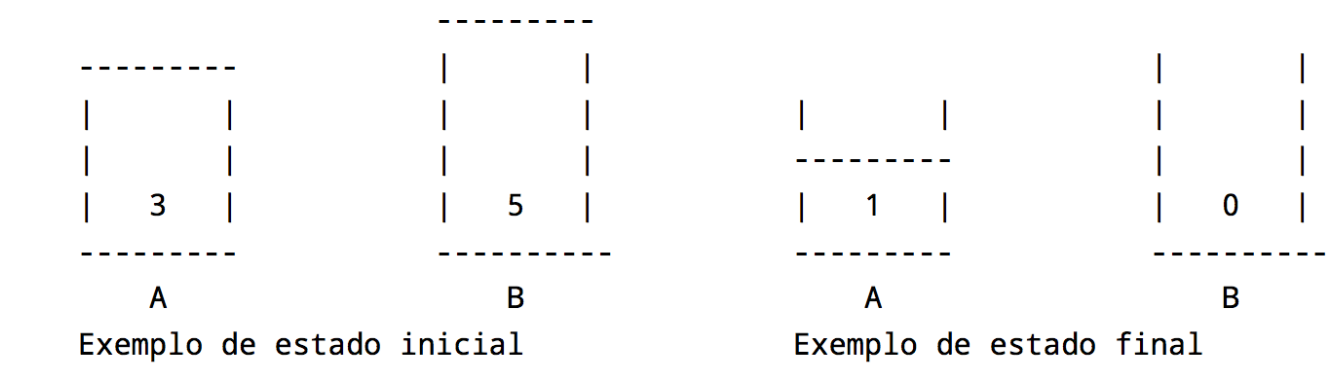

Figura 1: Exemplos de estado inicial e final do problema.

2. Programa a Desenvolver

Construa um programa para ler do teclado o valor das duas vasilhas, e procurar a sequência de operações necessária para medir um litro de água numa das duas vasilhas. Encontrará uma proposta de esqueleto para a implementação no ficheiro `laboratorio6.lisp` disponível no Moodle. A procura deve ser feita

implementando um algoritmo que permite procurar a solução na largura, denominado ***Breadth-First Search*** (BFS) e um outro algoritmo para procurar na profundidade, chamado ***Depth-First Search*** (DFS).

3. Representação do Estado para o Problema

No intuito de representar o estado do problema das vasilhas de água, decidiu-se representar as vasilhas por uma lista com dois átomos, em que o primeiro átomo é o número de litros na vasilha A e o segundo átomo o número de litros na vasilha B.

4. Criação de Nós para Auxílio na Árvore de Procura

Uma vez identificado como se irá representar o estado deste problema, é importante proceder à criação de um tipo abstrato de dados que será utilizado na expansão da árvore de procura mediante o algoritmo escolhido. Esse tipo abstrato de dados, doravante denominado de **Nó**, irá ser uma lista com três elementos principais:

- O estado do problema, ou seja, o conteúdo das vasilhas;
- A profundidade do nó na árvore;
- O nó "pai", ou seja, o nó imediatamente acima e do qual o atual nó foi gerado.

5. Exercícios

Em primeiro lugar, deverá analisar o ficheiro `laboratorio6.lisp` que contém algumas funções que disponibilizamos para auxiliar na realização deste laboratório.

Tipo abstrato de dados

1. **Seletores:** Defina as funções `vasilha-a-conteudo`, `vasilha-b-conteudo`, `no-estado`, `no-profundidade` e `no-pai` que recebem um nó e que retornam respetivamente o conteúdo da vasilha A, o conteúdo da vasilha B, a lista com os conteúdos das vasilhas, o valor da profundidade do nó e a lista que representa o nó pai.

2. **Funções auxiliares:** Criar um predicado `no-solucaop` que verifica se um nó é solução, ou seja, devolve verdadeiro se o conteúdo de alguma das vasilhas for de 1 litro de água.

Operadores

3. **Operadores:** Defina as funções `vazar-a`, `vazar-b`, `encher-a`, `encher-b`, `transferir-a-b`, `transferir-b-a`, que recebem um estado e que retornam um novo estado, no qual o valor das vasilhas foi alterado de acordo com o operador. Se o operador não se aplica (e.g. `encher-a` enquanto a vasilha A do estado já está no seu valor máximo) a função deve retornar `NIL`.

Geração dos nós sucessores

4. **Funções auxiliares:** Defina a função `novo-sucessor`, que permite gerar um sucessor de um determinado nó e recebe dois argumentos de entrada: um nó e o nome do operador a aplicar. A função retorna um nó sucessor no qual o estado foi atualizado pela aplicação do operador, a profundidade foi incrementada de um valor e o nó pai atualizado com a inserção do nó recebido por parâmetro.

5. **Sucessores:** Defina a função `sucessores`, que permite gerar os sucessores de um determinado nó. A função recebe como argumentos de entrada: 1) Um nó; 2) A lista de operadores do problema (nomes das 6 funções que implementam os operadores); 3) O nome do algoritmo usado para a procura; 4) A profundidade máxima de expansão da árvore de procura (opcional).
   A função retorna a lista de nós sucessores do nó recebido por parâmetro. Se o algoritmo de procura for o DFS e o nó recebido por parâmetro tiver uma profundidade igual à profundidade máxima de procura, a função não irá gerar nenhum sucessor.

Algoritmos: Funções auxiliares e de ordenação de nós

6. **Ordenação para Breadth-First Search:** Defina a função `abertos-bfs` que recebe duas listas. A primeira lista representa um conjunto de nós da lista dos abertos e a segunda lista representa um conjunto de nós sucessores. A função retorna a junção destas listas de acordo com o algoritmo de procura em largura.

7. **Ordenação para Depth-First Search:** Defina a função `abertos-dfs` que recebe duas listas. A primeira lista representa um conjunto de nós da lista dos abertos e a segunda lista representa um conjunto de nós sucessores. A função retorna a junção destas listas de acordo com o algoritmo de procura em profundidade.

8. **Verificar Existência de um Nó:** Defina o predicado `no-existep` que permite verificar se um nó existe numa lista. O predicado recebe três argumentos de entrada: um nó, uma lista de nós e o nome do algoritmo. Retorna verdadeiro se existir algum nó na lista com o mesmo estado.

9. **Breadth-first Search:** Defina a função `bfs` que irá efetuar a procura em largura-primeiro e recebe como argumento o nó inicial, a função objetivo a ser testada, a função sucessores e a lista de operadores e deverá retornar o nó solução encontrado.

    - Utilize como argumento de entrada opcional duas listas, denominadas de `ABERTOS` e `FECHADOS`, que permitam ir guardando na primeira os novos nós sucessores gerados ainda por expandir e na segunda os nós já expandidos.
    - Deve recorrer à função `no-existep` no intuito de verificar se existem nós sucessores com o mesmo estado de outros já gerados, sendo para tal necessário reconhecer se um estado sucessor já está em fechados e nesse caso não colocar o nó correspondente na lista de abertos.

10. **Depth-first Search:** Defina a função `dfs` que irá efetuar a procura em profundidade-primeiro e recebe como argumento o nó inicial, a função objetivo a ser testada, a função sucessores, a lista de operadores e a profundidade máxima e deverá retornar o nó solução encontrado.

    - Utilize como argumento de entrada opcional duas listas, denominadas de `ABERTOS` e `FECHADOS`, que permitam ir guardando na primeira os novos nós sucessores gerados ainda por expandir e na segunda os nós já expandidos.
    - Deve recorrer à função `no-existep` no intuito de verificar se existem nós sucessores com o mesmo estado de outros já gerados, sendo que para o DFS o conceito de nó repetido é particular, uma vez que quando são gerados sucessores que já estão em abertos ou fechados pode ser necessário recalcular a profundidade dos nós correspondentes.