Universidade Federal de Santa Catarina
Departamento de Automação e Sistemas
Inteligência Artificial
Professores: Jomi F. Hübner

Manual de utilização dos algoritmos de Busca em Java

1 Introdução

Este manual explica como utilizar o pacote de algoritmos de busca implementados em Java. Com este pacote é possível resolver um problema de busca especificando-se apenas a classe que representa os estados, os algoritmos de busca

- largura,
- profundidade,
- profundidade iterativa,
- bi-direcional,
- heurística e
- subida da montanha

já estão implementados.

2 A interface Estado

A classe que representa o "estado"para cada problema específico deve implementar a interface `Estado` que tem os seguintes métodos:

- O método `getDescricao()` deve retornar uma string descrendo o problema. Esse método é eventualmente utilizado pela interface do programa.
- O método `ehMeta()` deve retornar *true* se o estado for o estado meta.
- O método `custo()` retorna o custo de gerar o estado (não o custo acumulado, este é calculado no Nodo da árvore).
- O método `sucessores()` deve retornar uma lista de estados sucessores.

3 Exemplo

Para demonstrar a utilização do pacote, será resolvido o problema do *O homem, o lobo, o carneiro e o cesto de alface*, que tem o seguinte enunciado:

> Uma pessoa, um lobo, um carneiro e um cesto de alface estão à beira de um rio. Dispondo de um barco no qual pode carregar apenas um dos outros três, a pessoa deve transportar tudo para a outra margem. Determine uma série de travessias que respeite a seguinte condição: em nenhum momento devem ser deixados juntos e sozinhos o lobo e o carneiro ou o carneiro e o cesto de alface.

Para resolver este problema, os estados devem ter as seguintes informações:

- Em que lado do rio estão cada um dos personagens (o homem, o lobo, o carneiro e o cesto de alface).
- Estrutura de dados: quatro variáveis caracter onde 'd' representa lado direito e 'e' lado esquerdo.

  ```
  char homem, lobo, carneiro, alface;
  ```

- Estado inicial:

  ```
  homem='e', lobo='e', carneiro='e', alface='e';
  ```

- Estado meta:

  ```
  homem='d', lobo='d', carneiro='d', alface='d';
  ```

A geração de sucessores pode ser resumida como

- "levar x da margem w para a margem z"
  - x é uma das variáveis { carneiro, alface, lobo } ou não levar nada,
  - w é 'd' ou 'e',
  - z é 'd' ou 'e'.
- Pré-condições
  - w é diferente de z,
  - o homem estar na margem w,
  - o homem estar na mesma margem de x,
  - não ficar na mesma margem ((o lobo e o carneiro) ou (o carneiro e o alface)) sem o homem.
- Efeitos
  - Se x é 'd', vira 'e'.
  - Se x é 'e', vira 'd'.
  - Se homem é 'd', vira 'e'.
  - Se homem é 'e', vira 'd'.

Segue a implementação desta classe (clique para obter o fonte .java):

3.1 Compilação e execução

As classes do pacote estão no arquivo `buscaJava.jar`, portanto este arquivo deve estar no classpath.

Para windows:

```
set CLASSPATH=C:\xxx\buscaJava.jar;.
cd src
javac exemplos/HLAC.java
java exemplos.HLAC
```

Para unix:

```
export CLASSPATH=/xxx/buscaJava.jar:.
cd src
javac exemplos/HLAC.java
java exemplos.HLAC
```

4 Utilização dos algoritmos de busca

4.1 Largura, Profundidade e Profundidade Iterativo

As classes que implementam a interface `Estado` podem utilizar os seguintes algoritmos de busca

- `new BuscaLargura().busca( Estado inicial )`: recebe o estado inicial e retorna o `Nodo` da árvore de busca que é a solução do problema. Com o método `montaCaminho()` da classe `Nodo`, pode-se imprimir a solução. Retorna `null` se não encontrar a solução.

  No método `main` está exemplificada a utilização destes algoritmos.

- `new BuscaProfundidade(int max).busca( Estado inicial )`: recebe o estado inicial e a profundidade máxima e retorna o `Nodo` solução.

- `new BuscaRecursiva(int max).busca( Estado inicial )`: implementação recursiva (sem lista de abertos) para a busca em profundidade.

- `new BuscaIterativo().busca( Estado inicial )`: implementação do busca em profundidade iterativo.

Mais informações sobre a classe Nodo e como ela pode ser utilizada podem ser obtidas na API do pacote e/ou nos programas exemplo (diretório `src/exemplos`).

4.2 Busca Bi-Direcional

Para a utilização do algoritmo de busca bi-direcional, invocado com `new BuscaBidirecional().busca( Estado inicial, Estado meta )`, a classe que representa o estado também deve implementar a interface `Antecessor`. Esta interface tem apenas um método:

- `antecessores()` que retorna uma lista de estados antecessores de um determinado estado.

No exemplo 8-Puzzle esta interface é implementada.

4.3 Busca A*

Para a utilização do algoritmo de busca A*, invocado com `new AEstrela().busca( Estado inicial )`, a classe que representa o estado também deve implementar a interface `Heuristica`. Esta interface tem o seguinte método:

- `h()` retorna a estimativa de custo para transformar um estado no estado meta;

Nos seguintes exemplos essa interface é implementada

- 8-Puzzle.
- N-Rainhas.
- Quadrado mágico.

4.4 Busca Subida da Montanha

Para a utilização do algoritmo de busca Subida da Montanha, invocado com `new SubidaMontanha().busca( Estado inicial )`, a classe que representa o estado também deve implementar a interface `Aleatorio`. Esta interface tem o seguinte método:

- `geraAleatorio()` que gera um estado aleatóriamente.

Nos seguintes exemplos essa interface é implementada

- N-Rainhas.
- Quadrado mágico.