Universidade Federal de Alagoas / UFAL
Programa Multidisciplinar de Pós-graduação
em Modelagem Computacional de Conhecimento

Disciplina: Inteligência Computacional
Professor: Aydano Pamponet Machado
Alunos: Fernando Antonio Dantas Gomes Pinto
Jalves Mendonça Nicácio
Sunny Kelma Oliveira Miranda

Lista de Exercícios Busca

1. O Modelo de Resolução de Problemas

Antes de analisarmos os problemas propostos nas questões da lista, explanaremos resumidamente de que maneira implementamos as soluções de cada problema, bem como os algoritmos de busca discutidos nos capítulos 3 e 4 do livro de Russell e Norvig (2004).

Basicamente, distribuímos cada artefato programado entre três grandes grupos ou pacotes: **busca**, **modelo** e **problema**. A ilustração 01 demonstra como foi realizada a distribuição das classes e interfaces criadas para resolução desta lista:

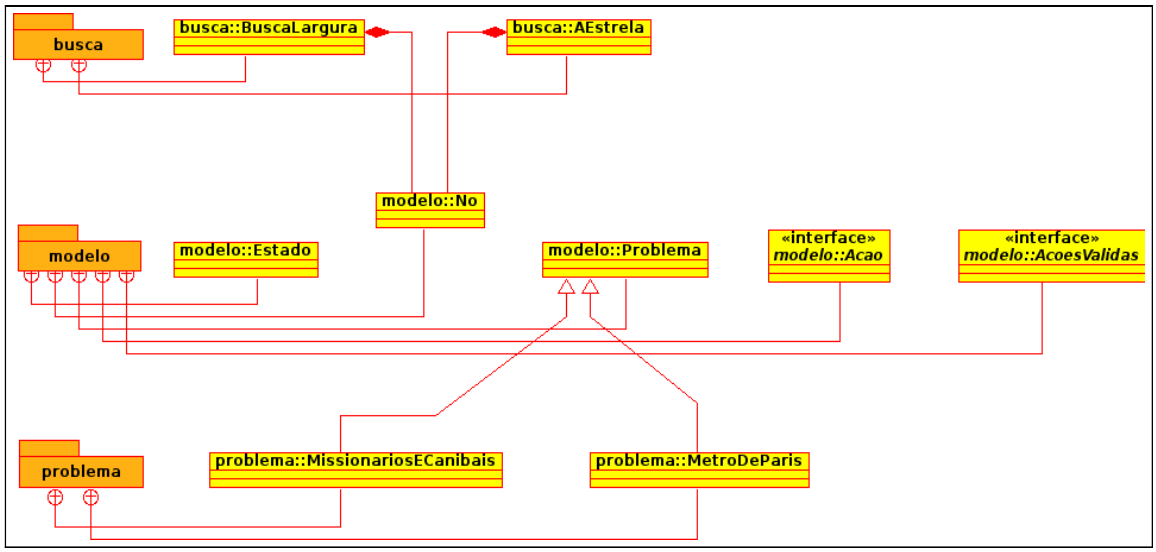

Ilustração 1: distribuição de todas as classes e interfaces implementadas

Na ilustração 1 podemos observar as principais classes construídas. Nem todas as classes, nem tampouco todas as associação entre as classes, estão explicitadas nesta figura. Contudo, a ilustração 1 deixa transparecer como os elementos de um pacote se comunicam com elementos de outros pacotes, permitindo desta forma o funcionamento do modelo construído.

No pacote de *busca*, implementamos as classes que efetivamente realizam a busca em um determinado problema, como por exemplo, a classe *AEstrela*. Para que as classes codificadas no pacote *busca* funcionassem independente do contexto, ou seja, de um problema específico, decidimos utilizar a abordagem de programação para interface. Desta forma, um algoritmo de busca realiza a busca através da chamada do método `buscar(Problema problema)`, onde *problema* é um objeto da classe genérica *Problema* (integrante do pacote modelo). Somente na execução do método principal é que especificamos quem é de fato o problema que está sendo investigado pelo algoritmo de busca.

O pacote modelo possui as definições genéricas de *problema*, *estado* e *ações*. Tais definições foram baseadas na formalização dos componentes de um problema (Russel; Norvig, 2004), a saber:

- Estado inicial – o estado em que o agente começa a percorrer o problema;
- Função sucessor – a função que gera os estados sucessores de um determinado estado,

baseado na descrição das ações disponíveis para este estado;

- Teste de objetivo – função que teste se um determinado estado é estado-objetivo;
- Custo de caminho – função que atribui um custo a cada caminho encontrado pelo agente;

Segundo Russel e Norvig (2004), os elementos listados acima definem um problema. Neste contexto, a solução para um determinado problema se encontra na determinação de um caminho desde o estado inicial até o estado objetivo. É natural que alguns algoritmos de busca se empenhem em encontrar o caminho ótimo, isto é, o caminho que possui o menor custo.

Da mesma forma como foi feito no pacote busca, o pacote modelo também não possui nenhuma implementação de um estado ou problema reais (problemas tais propostos nas questões da lista). No entanto, o pacote modelo possui todos os elementos necessários para que uma classe do pacote *busca* possa manipular qualquer problema real que seja definido no pacote de problema, desde que tal problema implemente e estenda as classes do pacote modelo. Em outras palavras, o pacote modelo é um ***framework*** para a implementação de qualquer problema que siga o modelo proposto neste trabalho.

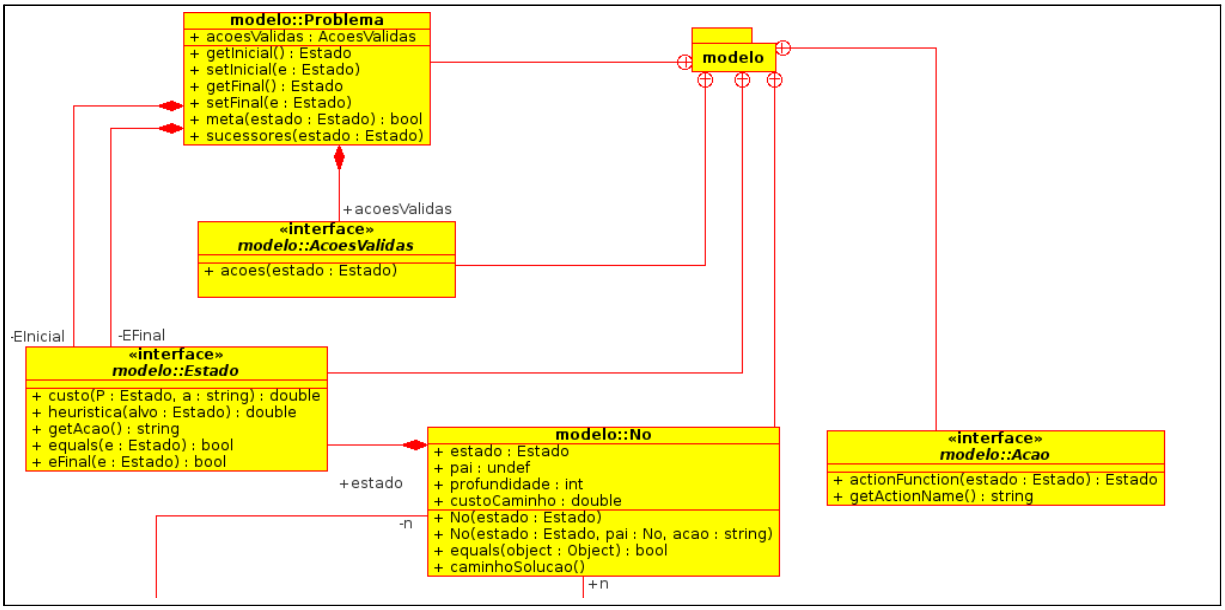

Ilustração 2: pacote modelo – Diagrama de classe: modela um problema de forma geral, através de seus estados e ações do agente.

A figura 2 traz o diagrama de classes do pacote modelo. É importante notar que *Estado* é apenas uma interface. Da mesma forma, *Acao* e *AcoesValidas*. As únicas classes realmente implementadas são *Problema* e *No*. Veremos mais adiante que a classe *Problema* é estendida no pacote problema, pelas classes que realmente implementam os detalhes dos problemas apresentados em cada questão.

A classe Nó é necessária para que as classes que implementam os algoritmos de busca possam se mover por entre os estados gerados a partir da função sucessor. A função sucessor, como um elemento do problema, está representado através do método `sucessores(Estado estado)`, que pertence, naturalmente, à classe Problema.

Na ilustração 3, expandimos o diagrama de classe apresentado na ilustração 2 para incluir as classes do pacote busca. Observe que todas os algoritmos de busca utilizam a classe No, que é imprescindível para o controle do caminho percorrido e seu respectivo custo. Observe ainda que a classe *Aestrela* implementa uma classe abstrata chamada *BuscaMelhorEscolha*. Isto é importante porque permite que implementemos sem grandes dificuldades outros algoritmos da mesma família, como por exemplo a Busca de Subida da Encosta.

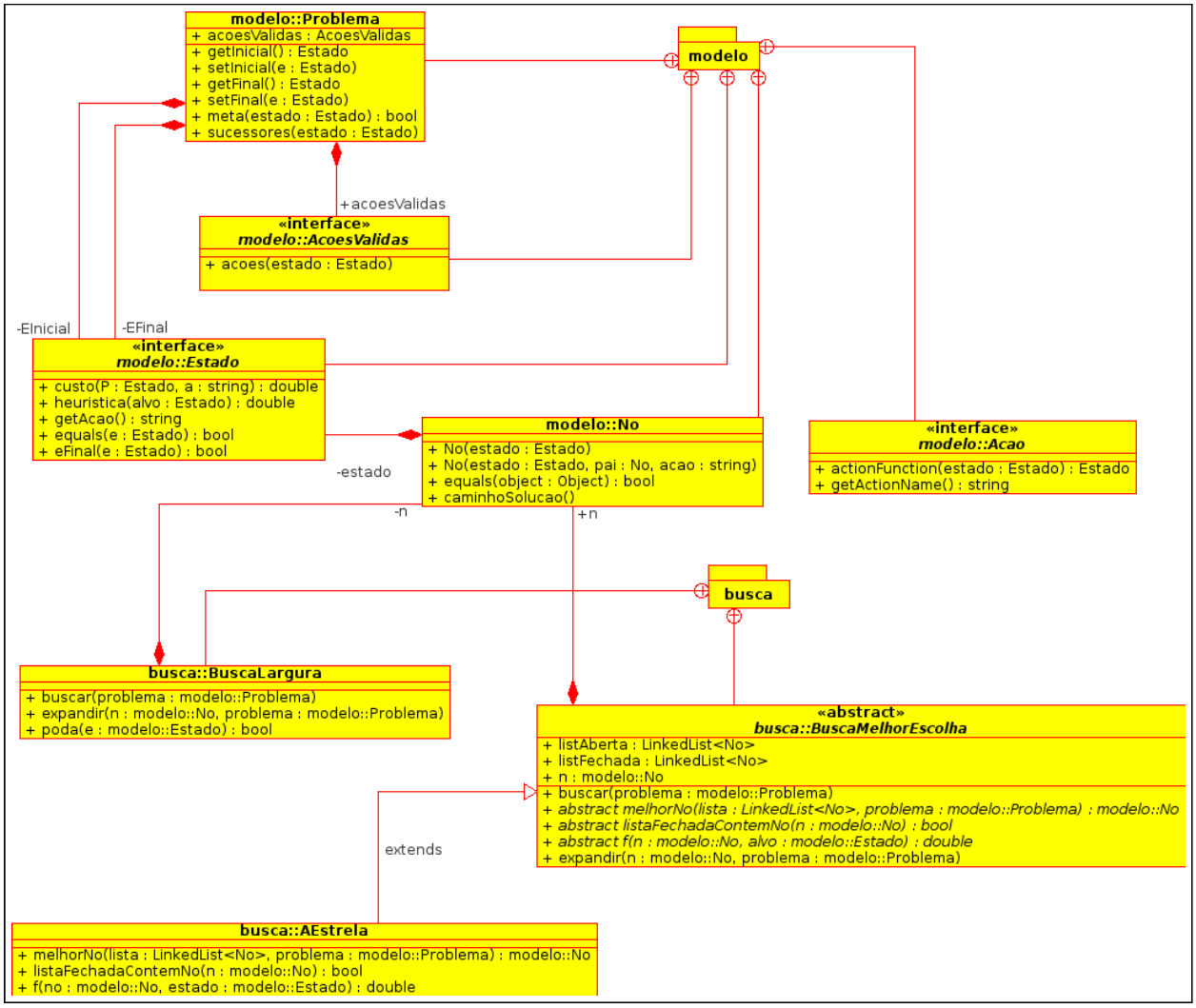

Ilustração 3: Diagrama de classe estendido, incluindo as classes de buscas

2. As questões da lista de Exercícios

2.1 – O problema dos Missionários e Canibais

Três missionários e três canibais estão em um lado do rio, juntamente com um barco que pode conter uma ou duas pessoas. Descubra um meio de fazer todos atravessarem o rio, sem deixar que um grupo de missionários de um lado fique em número menor que o número de canibais nesse lado do rio. Esse problema é famoso em IA, porque foi assunto do primeiro artigo que abordou a formulação de problemas a partir de um ponto de vista analítico (Amarel, 1968).

Implemente e resolva o problema de forma ótima, utilizando um algoritmo de busca apropriado. É boa ideia verificar a existência de estados repetidos?

Representação dos Estados

Para resolver este problema, consideraremos como representação dos estados o número de canibais e de missionários em cada margem do rio. É fácil perceber que os números de canibais e missionários em cada margem são complementares de 3, ou seja, se em uma margem existem 3 missionários e 2 canibais, na margem oposta haverá zero missionários e 1 canibal.

Desta forma podemos representar cada estado pela tripla de valores (c,m,b), onde c é o número de canibais, m é o número de missionários e b a margem onde o barco se encontra. Por motivos de simplificação do problema, podemos fazer nossa representação do estado sempre observar apenas a margem

esquerda do rio, já que é trivial inferir a situação na margem oposta.

Assim, considerando a representação de estado, E = (2,2,direita) indica que na margem esquerda existem dois canibais e dois missionários, e ainda que o barco se encontra na margem direita. Obviamente, considerando a mesma representação, podemos facilmente inferir que na margem direita (onde se encontra o barco) existem um canibal e um missionário.

Ações do agente

Como o barco comporta no máximo duas pessoas, as ações possíveis são:

- levar 1 missionário
- levar 1 canibal
- levar 2 missionários
- levar 2 canibais
- levar 1 missionário e 1 canibal.

Para cada estado expandido, devemos verificar quais ações são válidas. Um exemplo prático desta verificação é impedir que o estado gerado crie uma situação em que o número total de missionários ou canibais ultrapasse a três.

Outra restrição importante é em relação ao número de missionários de cada margem do rio, que deve ser sempre maior ou igual ao número de canibais.

Resultado da busca

Utilizamos o algoritmo de busca em largura para percorrer o espaço de estados do problema dos missionários e canibais. A solução encontrada é escrita pelo programa conforme demonstramos abaixo. Para que a busca seja ótima, utilizamos custo uniforme em todas as ações.

```
********** O PROBLEMA DOS MISSIONÁRIOS E CANIBAIS ****************
Resultado da Busca:
EstadoMC [barco=e, canibais=3, missionarios=3, acão=]
EstadoMC [barco=d, canibais=1, missionarios=3, acão=Levar 2 Canibais]
EstadoMC [barco=e, canibais=2, missionarios=3, acão=Levar 1 Canibal]
EstadoMC [barco=d, canibais=0, missionarios=3, acão=Levar 2 Canibais]
EstadoMC [barco=e, canibais=1, missionarios=3, acão=Levar 1 Canibal]
EstadoMC [barco=d, canibais=1, missionarios=1, acão=Levar 2 Missionários]
EstadoMC [barco=e, canibais=2, missionarios=2, acão=Levar 1 Missionário e 1 Canibal]
EstadoMC [barco=d, canibais=2, missionarios=0, acão=Levar 2 Missionários]
EstadoMC [barco=e, canibais=3, missionarios=0, acão=Levar 1 Canibal]
EstadoMC [barco=d, canibais=1, missionarios=0, acão=Levar 2 Canibais]
EstadoMC [barco=e, canibais=1, missionarios=1, acão=Levar 1 Missionário]
EstadoMC [barco=d, canibais=0, missionarios=0, acão=Levar 1 Missionário e 1 Canibal]
Custo do caminho: 11.0
Profundidade do alvo: 11
************************************************************
```

2.2 - O problema do Metrô de Paris

Suponha que queremos construir um sistema para auxiliar um usuário do metrô de Paris a saber o trajeto mais rápido entre a estação onde ele se encontra e a estação de destino. O usuário tem um painel com o mapa, podendo selecionar a sua estação de destino. O sistema então acende as luzes sobre o mapa mostrando o melhor trajeto a seguir (em termos de quais estações ele vai atravessar., e quais as conexões mais rápidas a fazer – se for o caso).

Considere que:

- *A distância em linha reta entre duas estações quaisquer é dada em uma tabela. Para facilitar a vida, considere apenas 4 linhas do metrô.*

- *A Velocidade média de um trem é de 30km/h;*
- *Tempo gasto para trocar de linha dentro de mesma estação (fazer baldeação) é de 4 minutos.*

Formule e implemente este problema em termos de estado inicial, estado final, operadores e função de avaliação para Busca heurística com A.*

No problema do metrô de Paris, adotamos como representação do Estado o seguinte conjunto de informações:

- O número da estação de metrô
- A linha a qual esta estação pertente
- informação sobre baldeação – informações sobre as linhas de metrô que passam pela estação, quando esta estação dá acesso a mais de uma linha de metrô.

A informação sobre baldeação é importante porque informa ao agente de busca quando é possível trocar de linha de metrô, assim como também ajuda na formação da função heurística, visto que a realização de baldeação de linha pode ser mais custoso em um determinado ponto do caminho, mas nada impede que o caminho de menor custo seja justamente o caminho que utilize baldeações.

Alguns exemplos da representação de um estado seriam

- E = (1, azul, null) – representando a estação UM que pertence à linha azul e não possui opção de baldeação.
- E = (4, azul, azul-verde) – representando a estação QUATRO, que pertence à linha Azul e possui baldeação com a linha verde.
- E = (4, verde, azul-verde) – representando a estação QUATRO, que pertence à linha Verde e possui baldeação com a linha azul.

Observe que cada estação que possui opção de baldeação pode ser representada de duas formas diferentes, dependendo do caminho percorrido pelo agente. Pela ótica do agente de busca, se ele está percorrendo a linha azul em busca da estação 8 (linha verde ou amarela) e alcança a estação 4, naturalmente ele entenderá que essa estação pertence à linha que ele vem percorrendo (linha azul). Uma vez estando na estação *4-azul*, o agente pode modificar seu trajeto para a estação *4-verde*. Toda vez que o agente realizar uma ação onde ele não muda de estação, mas muda de linha, dizemos que o agente realizou uma ação de baldeação.

Ações do agente

Neste contexto, o agente pode realizar as seguintes ações:

- Ir(estacao)
- Baldear(Estação)

Resultado da busca

Utilizamos o algoritmo A* (A Estrela) para a busca do estado alvo. Para demonstração, escolhemos a estação 1 como estado inicial e a estação 14 como estado final.

Como o algoritmo A* exige a utilização de funções heurísticas, precisávamos representar, em alguma estrutura de dados, as informações fornecidas pela lista através da tabela de distâncias e do mapa do metrô. No programa, portanto, todas essas informações necessárias estão à disposição em uma classe chamada ModeloMetro.java, utilizando o padrão de projeto *singleton.*

A saída do programa é como se observa abaixo:

```
*************** O PROBLEMA DO METRÔ DE PARIS ****************
Saída: estação 1 - Chegada: Estação 14
Resultado da Busca:
EstadoMP [estacao=1, linha=azul, nomeAcao=]
EstadoMP [estacao=2, linha=azul, nomeAcao=ir]
EstadoMP [estacao=3, linha=azul, nomeAcao=ir]
EstadoMP [estacao=3, linha=vermelha, nomeAcao=baldear]
EstadoMP [estacao=13, linha=vermelha, nomeAcao=ir]
EstadoMP [estacao=13, linha=verde, nomeAcao=baldear]
EstadoMP [estacao=14, linha=verde, nomeAcao=ir]
Custo do caminho: 84.0
Profundidade do alvo: 6
*****************************************************************
```

3. Instruções para utilização do programa IA.jar (anexado)

1) **Abra um terminal e navegue até a pasta onde se encontra o arquivo IA.jar**
2) **execute o comando** java -jar IA.jar

4. Referências

RUSSEL, T.; NORVIG P.. Inteligência Artificial. Rio de Janeiro: Elsevier, 2004.

Amarel, S. (1968). On representations of problems of reasoning about actions, Machine Intelligence, (3), 131—171