Procura em Espaços de Estados

O tópico destas aulas são os problemas de procura em espaços de estados e os respectivos algoritmos para a sua resolução. É apresentada uma representação genérica destes problemas e uma implementação de vários algoritmos de procura cega (procura em profundidade, procura em largura e procura de custo uniforme) e de procura informada (procura sôfrega e procura A*). O código disponibilizado (em Java) visa o entendimento dos conceitos em detrimento da eficiência, possibilitando a fácil compreensão da matéria e distinção entre os algoritmos.

Dada a generalidade da modelação fornecida, será possível aplicar os algoritmos na resolução de diversos problemas de procura em espaços de estados. Complementarmente, fornece-se uma implementação para a representação de problemas de cálculo de rotas em mapas de estradas. Deste modo, será possível experimentar os diferentes algoritmos de procura e perceber as suas vantagens e inconvenientes para a resolução deste tipo de problemas.

1 - Instalação e utilização da implementação fornecida

Crie um novo projeto de JAVA (pode ser simples com uma classe Main) num IDE de sua escolha (por exemplo Eclipse), instale os pacotes Java fornecidos e leia atentamente o guião que explica o seu funcionamento detalhado. Experimente o seguinte código no qual se cria um grafo com o mapa da Roménia e se resolve o problema de encontrar o caminho mais curto entre Arad e Bucharest usando o algoritmo de procura em largura:

```
	public static void main(String[] args) {
		Graph graph = new Graph();
		graph.defineGraphRomenia();
		graph.showLinks();
		graph.showSets();
		Node n;
		n = graph.searchSolution("Arad", "Bucharest",
Algorithms.BreadthFirstSearch);
		graph.showSolution(n);
	}
```

Como resultado deverão aparecer:

todas as ligações e respectivos quilómetros

```
******************** LINKS ********************
Oradea: Sibiu (218.0); Zerind (56.0);
Zerind: Arad (51.0); Oradea (56.0);
Bucharest: Fagaras (180.0); Giurgiu (59.0); Pitesti (108.0); Urziceni (52.0);
Urziceni: Bucharest (52.0); Hirsova (103.0); Vaslui (229.0);
Arad: Sibiu (221.0); Timisoara (49.0); Zerind (51.0);
Mehadia: Dobreta (37.0); Lugoj (94.0);
Neamt: Iasi (94.0);
Iasi: Neamt (94.0); Vaslui (58.0);
R. Vilcea: Craiova (96.0); Pitesti (47.0); Sibiu (79.0);
Eforie: Hirsova (88.0);
Pitesti: Bucharest (108.0); Craiova (103.0); R. Vilcea (47.0);
Timisoara: Arad (49.0); Lugoj (54.0);
Craiova: Dobreta (96.0); Pitesti (103.0); R. Vilcea (96.0);
Hirsova: Eforie (88.0); Urziceni (103.0);
Vaslui: Iasi (58.0); Urziceni (229.0);
Giurgiu: Bucharest (59.0);
Sibiu: Arad (221.0); Fagaras (65.0); Oradea (218.0); R. Vilcea (79.0);
Dobreta: Craiova (96.0); Mehadia (37.0);
Fagaras: Bucharest (180.0); Sibiu (65.0);
Lugoj: Mehadia (94.0); Timisoara (54.0);
**********************************************
```

todas as regiões e respectivas cidades

```
******************** SETS ********************
Banat: Timisoara; Lugoj; Mehadia;
Crisana: Zerind; Oradea; Arad;
Dobrogea: Eforie; Hirsova;
Moldova: Neamt; Iasi; Vaslui;
Muntenia: Bucharest; Giurgiu; Urziceni; Pitesti;
Oltenia: R. Vilcea; Craiova; Dobreta;
Transilvania: Fagaras; Sibiu;
**********************************************
```

a solução obtida e os respectivos indicadores estatísticos do algoritmo

```
******************** SOLUTION ********************
Node Expansions: 8
Nodes Generated: 21
State Repetitions: 4
Runtime (ms):  1,125
| Arad      |   0 | [Arad, Sibiu] -> 221
| Sibiu     | 221 | [Sibiu, Fagaras] -> 65
| Fagaras   | 286 | [Fagaras, Bucharest] -> 180
| Bucharest | 466 |
**********************************************
```

Experimente com outras cidades de origem e destino e com diferentes algoritmos de procura.

2 - Comparação entre os diferentes algoritmos de procura

Compare o comportamento dos diferentes algoritmos de procura para calcular a melhor rota em cada um dos seguintes problemas. Para cada um deles, considere o número de nós expandidos, gerados e repetidos, assim como o tempo de execução e indique qual o algoritmo com a melhor performance para cada um dos casos. Verifique ainda se conseguiu obter a solução óptima:

	Cidade de origem	Cidade de destino
1	Arad	Bucharest
2	Bucharest	Oradea
3	Oradea	Bucharest
4	Timisoara	Neamt

3 - Problemas com passagem obrigatória numa província

Considere agora uma extensão ao problema anterior em que se pretende igualmente ir de uma cidade de origem para uma cidade de destino, mas onde se requer também a passagem por uma cidade que pertença a uma província pré-determinada. Assume-se que a cidade de origem é distinta da cidade de destino e que nem uma nem a outra pertencem à província em questão.

Por exemplo, considere que se pretende ir de Arad para Bucharest passando por uma cidade da província de Dobrogea. Como se pode ver acima, a província de Dobrogea contém duas cidades: Eforie e Hirsova. Por isso, neste caso uma solução seria iniciar o percurso em Arad escolher o melhor caminho para Eforie e finalmente escolher o melhor caminho de Eforie para Bucharest. A única solução alternativa a esta seria ir de Arad para Hirsova e posteriormente de Hirsova para Bucharest escolhendo sempre os caminhos mais curtos. Claro que se a província de passagem contivesse mais cidades, maior seria o número de possíveis soluções alternativas. Neste problema, tal como nos outros, estamos interessados na solução de menor custo, isto é, a que apresenta um percurso total mais curto.

Note que este problema pode ser resolvido usando um novo grafo que apenas contém os vértices correspondentes à cidade inicial, à cidade final e às cidades pertencentes à província de passagem. Para forçar a passagem pela província, o novo grafo teria que ter uma ligação da cidade inicial para todas as cidades dessa província e destas para a cidade destino. No exemplo acima teríamos que ter ligações entre: Arad e Eforie; Arad e Hirsova; Eforie e Bucharest; Hirsova e Bucharest. Para garantir o percurso total mais curto seria necessário associar a cada ligação deste novo grafo um custo correspondente ao do caminho mais curto entre as cidades envolvidas (que poderia ser calculado por um algoritmo de procura no grafo original).

No exemplo acima o novo grafo teria as seguintes ligações:

```
******************** LINKS ********************
Hirsova: Arad (610.0); Bucharest (155.0);
Bucharest: Eforie (243.0); Hirsova (155.0);
Arad: Eforie (698.0); Hirsova (610.0);
Eforie: Arad (698.0); Bucharest (243.0);
**********************************************
```

Usando neste novo grafo o método *searchSolution* para calcular o caminho mais curto entre a cidade de origem e a cidade de destino, neste caso Arad e Bucharest, obtém-se a solução pretendida. Esta solução pode ser visualizada pelo método *showSolution* evocado no grafo original.

No exemplo o resultado com o algoritmo de procura A* seria:

```
****************** SOLUTION ******************
Node Expansions: 28
Nodes Generated: 78
State Repetitions: 1
Runtime (ms):  3,999
| Arad      |   0 | [Arad, Sibiu, R. Vilcea, Pitesti, Bucharest, Urziceni,
Hirsova] -> 610
| Hirsova   | 610 | [Hirsova, Urziceni, Bucharest] -> 155
| Bucharest | 765 |
**********************************************
```

Implemente um novo método *searchSolution* em que além da identificação das cidades de origem e destino, e do algoritmo de procura, se pode especificar uma província de passagem (que não contenha nenhuma destas cidades). Este novo método deve criar automaticamente o novo grafo e usá-lo para resolver o problema como explicado acima.

4 - Problemas com passagem obrigatória numa sequência de províncias

Considere uma nova extensão ao problema em que além de se pretender ir de uma cidade de origem para uma cidade de destino, se requer a passagem por uma sequência pré-determinada de províncias (distintas entre si) pela ordem indicada. Assume-se que a cidade de origem é distinta da cidade de destino e que nem uma nem a outra pertencem a nenhuma das províncias em questão.

Implemente um novo método *searchSolution* em que além da identificação das cidades de origem e destino, e do algoritmo de procura, se pode especificar uma sequência de províncias de passagem (que não contenha nenhuma destas cidades).

No caso de uma viagem de Arad para Bucharest passando por uma cidade de Dobrogea e uma de Banat, o novo grafo teria as seguintes ligações:

```
******************** LINKS ********************
Hirsova: Arad (610.0); Lugoj (593.0); Mehadia (499.0); Timisoara (647.0);
Bucharest: Lugoj (438.0); Mehadia (344.0); Timisoara (492.0);
Arad: Eforie (698.0); Hirsova (610.0);
Mehadia: Eforie (587.0); Hirsova (499.0); Bucharest (344.0);
Lugoj: Eforie (681.0); Hirsova (593.0); Bucharest (438.0);
Eforie: Arad (698.0); Lugoj (681.0); Mehadia (587.0); Timisoara (735.0);
Timisoara: Eforie (735.0); Hirsova (647.0); Bucharest (492.0);
**********************************************
```

Neste exemplo, o resultado com o algoritmo de procura A* seria:

```
****************** SOLUTION ******************
Node Expansions: 105
Nodes Generated: 289
State Repetitions: 9
Runtime (ms): 11.512
| Arad      |   0 | [Arad, Sibiu, R. Vilcea, Pitesti, Bucharest, Urziceni,
Hirsova] -> 610
| Hirsova   | 610 | [Hirsova, Urziceni, Bucharest, Pitesti, Craiova, Dobreta,
Mehadia] -> 499
| Mehadia   | 1109 | [Mehadia, Dobreta, Craiova, Pitesti, Bucharest] -> 344
| Bucharest | 1453 |
**********************************************
```