Resolução de Problemas por Meio de Busca
Inteligência Artificial

José Robertty de Freitas Costa[1], **Marianna de Pinho Severo**[1], **Marisa do Carmo Silva**[1]

[1]Campus Quixadá – Universidade Federal do Ceará (UFC)
Docente: Maria Viviane de Menezes

{robertty, mariannapinho, marisa296}@alu.ufc.br

1. Introdução

Neste trabalho são implementados três tipos de algoritmos de busca diferentes para solucionar o problema do mapa rodoviário da Romênia, que consiste em sair da cidade de Arad para chegar à cidade de Bucharest. Uma solução consiste em um caminho entre o estado inicial e o estado final, e o custo deste caminho.

Iremos, neste trabalho, fazer um teste computacional em relação ao tempo de execução dos algoritmos em 10 problemas (um problema consiste em duas cidades: origem e destino). Os algoritmos que iremos comparar são: busca em largura, busca em profundidade e busca de custo uniforme. Esses algoritmos são famosos e fáceis de implementar, e uma explicação sobre seu funcionamento e um pseudocódigo podem ser encontrados em [1].

Este relatório está organizado da seguinte forma: na Seção 2 apresentamos a formulação do problema de ir da cidade de Arad para a cidade de Bucharest; na Seção 3 descrevemos como implementamos os algoritmos de busca; na Seção 4 descrevemos os experimentos (10 problemas do mapa rodoviário da Romênia com diferentes origens e destinos), tais como as especificações de hardware e software; na Seção 5 mostramos o gráfico de tempo plotado para os 10 experimentos, realizando uma pequena discussão dos resultados; por fim, na Seção 6 concluímos qual algoritmo de busca é mais apropriado para resolver o problema com base em outras seções.

2. Formulação do problema

- **Estado Inicial**: Arad (cidade inicial).
- **Ações**: Ir(x,y) - Ir da cidade x para a cidade y.
- **O modelo de transição de estados**: o modelo de transição de estados está representado na Figura 1.
- **O teste de objetivo**: Bucharest (cidade objetivo).
- **O custo de caminho**: O custo será dado pela distância entre as cidades fornecidas no mapa.

3. Implementação dos algoritmos de busca

Antes de descrevermos os algoritmos, iremos descrever os atributos e a função de cada classe em nossa apresentação. Utilizamos neste trabalho as seguintes classes:

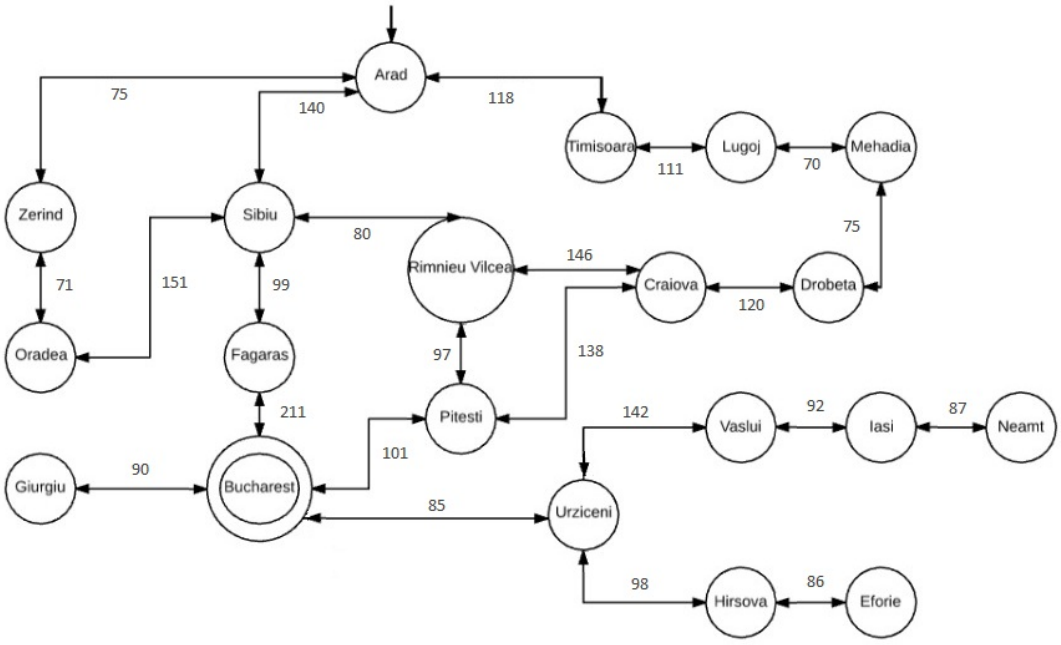

Figura 1. Modelo de transição de estados

- **State(int city)**: Irá indicar em qual estado estamos em nosso grafo. As cidades serão dadas por um inteiro do enum da classe City.

- **Action (State origin, State destination)**: Indica a ação de sair de um estado para outro estado.

- **City()**: Salvará o enum com a enumeração das cidades.

- **Inicialization()**: Irá inicializar nossa matriz de adjacência, do nosso grafo.

- **Node(State state, Node *father, Action action, int cost)**: Irá salvar cada nó da nossa árvore de busca, onde cada nó precisa de um estado, um nó que é seu pai na árvore, a ação que foi feita para chegar até esse nó e o seu custo.

O nosso estudo irá partir de um problema de encontrar um caminho que leve de uma cidade de origem até uma cidade objetivo no mapa rodoviário da Romênia. Iremos pensar em nossa implementação como se cada cidade representasse um nó em nossa árvore de busca e a, partir disso, iremos aplicar buscas muito conhecidas para resolver o mesmo problema. As buscas escolhidas neste trabalho foram:

- Busca em Largura
- Busca em Profundidade
- Busca de Custo Uniforme

A busca em largura consiste em, basicamente, expandir todos os nós de cada nível da árvore até conseguirmos encontrar o nosso nó objetivo (cidade objetivo). Para esse tipo de busca, iremos utilizar duas estruturas de dados: uma fila e um vetor. A fila determinará a ordem para a expansão e o vetor ficará responsável por salvar os nós já expandidos

na árvore, a fim de evitar ciclos. A utilização da fila se faz necessária pois precisamos expandir todos os nós de um nível para expandir outros de outro nível.

A busca em profundidade consiste na estratégia de sempre procurar expandir todos os nós de um lado da árvore de busca. Para isso, iremos utilizar novamente duas estruturas de dados: uma pilha e um vetor. A pilha determinará a ordem para a expansão e o vetor ficará responsável por salvar os nós já expandidos na árvore a fim de evitar ciclos. A utilização da pilha se faz necessário porque precisamos expandir sempre o último nó que foi obtido pela expansão do nó anterior.

A busca de custo uniforme consiste em procurar um caminho de menor custo que leve do nó origem até o nó objetivo. Para esse tipo de busca, precisamos de duas estruturas de dados: uma fila de prioridades e um vetor. A fila de prioridades determinará a ordem para a expansão e o vetor ficará responsável por salvar os nós já expandidos na árvore, a fim de evitar ciclos. A fila de prioridades é de extrema importância, pois a cada interação procuraremos expandir o nó de menor custo.

4. Experimentos

Utilizamos 10 casos de teste, que foram executados utilizando-se os três algoritmos diferentes, de forma a conseguirmos comparar o tempo de execução dos algoritmos em relação aos diversos casos de teste.

A Tabela 1 contém os casos de testes e tempo de execução de cada caso de teste (o tempo está em segundos).

Cidade Origem - Cidade Destino	Busca em Largura	Busca em Profundidade	Busca de Custo Uniforme
Arad - Bucharest	0.02	0.02	0.034
Oradea - Sibiu	0.016	0.016	0.016
Oradea - Zerind	0.014	0.018	0.009
Hirsova - Vaslui	0.015	0.017	0.017
Sibiu - Mehadia	0.027	0.021	0.035
Eforie - Oradea	0.026	0.032	0.045
Fagaras - Neamt	0.038	0.027	0.061
Drobeta - Urziceni	0.018	0.034	0.034
Timisoara - Fagaras	0.021	0.018	0.031
Zerind - Vaslui	0.039	0.029	0.091

Tabela 1. Tempos computados nos casos de teste

Em nosso problema, existe o custo de caminho e a Tabela 2 abaixo vem especificando o custo de caminho de cada um dos algoritmos nos diversos casos de teste.

Cidade Origem - Cidade Destino	Busca em Largura	Busca em Profundidade	Busca de Custo Uniforme
Arad - Bucharest	450	733	418
Oradea - Sibiu	151	151	151
Oradea - Zerind	71	71	71
Hirsova - Vaslui	240	240	240
Sibiu - Mehadia	439	421	421
Eforie - Oradea	326	326	326
Fagaras - Neamt	617	617	617
Drobeta - Urziceni	444	1064	444
Timisoara - Fagaras	357	826	357
Zerind - Vaslui	752	914	720

Tabela 2. Custo de caminho nos casos de teste

Especificações da máquina e do ambiente:

- Sistema operacional: Ubuntu 16.04 LTS
- Processador: Intel core i5-4210U 1.70GHz
- Memória: 7,7 GiB
- Tipo de sistema: 64-bit
- Ambiente de desenvolvimento: Qt - Creator
- Linguagem de desenvolvimento: C++

5. Gráfico de Tempo

No gráfico, as cores representam os tipos de busca. Assim:

- Busca em Largura: vermelho.
- Busca em Profundidade: verde.
- Busca de Custo Uniforme: preto.

As letras dos pares ordenados (Origem, Destino) representam as iniciais das cidades. Dessa forma: A = Arad, B = Bucharest, D = Drobeta, E = Eforie, F = Fagaras, H = Hirsova, M = Mehadia, N = Neamt, O = Oradea, S = Sibiu, T = Timisoara, U = Urziceni, V = Vaslui e Z = Zerind.

Observando o gráfico, é possível perceber que a busca de busto uniforme, na maioria das vezes, consome mais tempo de execução do que os outros dois algoritmos. Isso pode ser devido ao fato de, mesmo que o nó objetivo seja encontrado, ele só será considerado encontrado quando chegar sua vez de ser expandido, o que garante o menor custo.

Também é possível observar que a busca em largura ora consome um menor ora consome um maior tempo do que a busca em profundidade. Entretanto, a maioria das

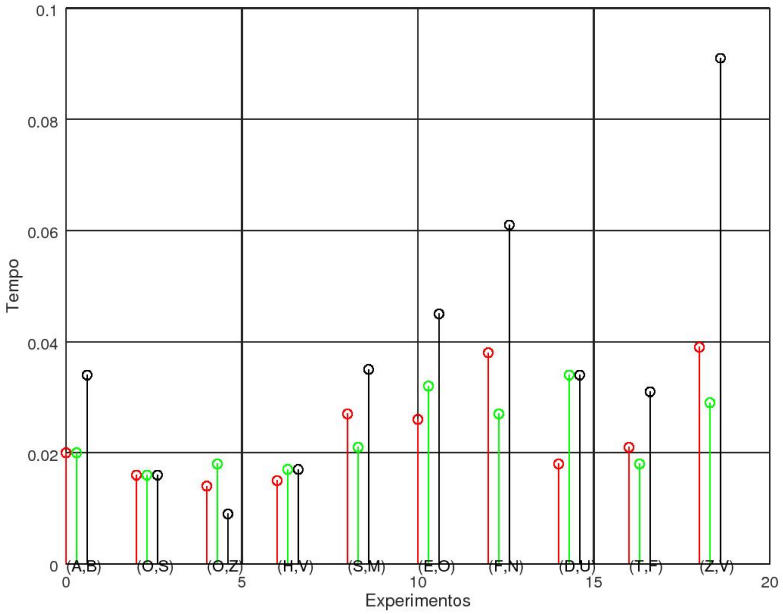

Figura 2. Gráfico de Tempo dos Experimentos

vezes a busca em profundidade precisou de menos tempo. Pode-se inferir que as buscas em largura e em profundidade seriam escolhas melhores para a solução do problema dado, se o critério mais importante adotado for o menor tempo.

6. Conclusão

A partir do gráfico e das tabelas podemos observar que é uma tarefa difícil de responder qual o melhor algoritmo, porém se observarmos a tabela 2 vemos que o algoritmo de custo uniforme encontra sempre solução melhor ou igual aos demais, mas em contrapartida em 60% dos casos ele possui o pior tempo de execução.

Em tempo de execução a busca em largura e a busca em profundidade se comportaram de forma similar de forma geral tendo a busca em largura 0.234 segundos de execução e a busca em profundidade 0.232 segundos de execução. Porém, ao nos depararmos com alguns casos a busca em profundidade geram caminhos com custo discrepantes em relação aos demais algoritmos (exemplo: caso Drobeta - Urziceni).

Podemos inferir a partir desta nossa análise simples dos dados em que o melhor algoritmo irá depender do que queremos. Se queremos a garantia de sempre ter o menor custo de caminho iremos optar por o algoritmo de custo uniforme mas se quisermos o de menor tempo de execução iremos optar ou por a busca em largura ou por a busca em profundidade (tendo em mente que a em profundidade poderá gerar custos de caminhos absurdos).

Referências

[1] Stuart Russell and Peter Norving. *Inteligência Artificial*. Elsevier, 3th edition, 2013.