Métodos de Busca

Estudo de caso utilizando métodos de busca de IA

Iago Costa das Flores   
Universidade do Sul e Sudeste do ParáMarabá, Pará  
univercomput@unifesspa.edu.br

Warley Rabelo Galvão  
Universidade do Sul e Sudeste do ParáMarabá, Pará  
wrgalvao@unifesspa.edu.br

Gustavo Oliveira Lacerda  
Universidade do Sul e Sudeste do ParáMarabá, Pará  
gust4oliveira@unifesspa.edu.brKayro Santos Costa  
Universidade do Sul e Sudeste do Pará  
Marabá, Pará kayrosantos@unifesspa.edu.br

***Abstract*—This document aims to present programmable solutions created for a traveler trajectory problem, where we can represent this problem through a graph, which facilitates the creation of our solutions developed in python programming language. Solutions were demonstrated through deep search, in breadth, A\*, greedy and uniform.**

***keywords — search, path, heurística, method (****key words****)***

# Introdução

O seguinte artigo visa mostrar os resultados obtidos com a implementação dos métodos de busca de Inteligência Artificial, como busca em profundidade, busca em largura, A\*, gulosa e uniforme que foram implementadas, com o intuito de encontrar uma determinada cidade em um mapa da escolha do docente da disciplina de Inteligência Artificial do curso de Engenharia da Computação da Universidade Federal do Sul e Sudeste do Pará.

# Representação do Problema

O objetivo é encontrar a cidade neamt utilizando os métodos de busca em profundidade, largura, A\*, gulosa e uniforme em um mapa contendo 20 cidades e analisar a performance de cada método em particular em termos de complexidade, espaço, tempo de execução e completude e a partir dessas informações definir qual o método mais aconselhável para a solução do problema.

A representação do problema em grafo contendo as cidades em forma de árvore e a representação real do mapa em grafo encontra-se em anexo no envio do artigo juntamente com os códigos.

III. Busca em Profundidade

O método busca de profundidade, consiste em uma busca que intuitivamente começa do nó raiz, ou seja, ele seleciona algum nó como sendo raiz, quando estamos tratando de grafos, e explora cada um dos seus ramos, antes de retroceder.

Mais especificamente um algoritmo de busca em profundidade progride através do primeiro nó filho da árvore de busca, e vai se aprofundando cada vez mais até que o alvo da busca seja encontrado ou até que ele se depare com um nó que não possui filhos, o que chamamos de nó folha, quando chegamos no nó folha a busca retrocede para o próximo nó anterior e começa novamente a busca em outro ramo.

A busca em profundidade é o método com complexidade mais inferior aos demais citados neste artigo. Mas devido esse método percorrer um ramo até o último nó para verificar se chegou ao destino, é um dos menos utilizados em resolução de problemas, pois depende da complexidade e do tamanho do grafo problema apresentado, podendo ser muito útil e de fácil implementação em grafos pequenos e menos complexos, visando o ganho de tempo, não de performance.

Para o nosso exemplo de grafo, o método de busca em profundidade conseguiu achar o destino, ou nó final, a partir do nó inicial, mas por se tratar de um grafo com complexidade baixa não levou um tempo consideravelmente grande para achar o destino, o que normalmente é o contrário.

Ainda levando em consideração o exemplo em questão, podemos pontuar que apesar do método ter encontrado o destino, ele não foi suficientemente eficiente, tendo ido a um nó folha que não era o destino final e retrocedeu novamente para por fim achar o ramo certo para o destino procurado, por isso ele pode ser usado para solucionar o problema, mas não seria o melhor e mais eficiente método para isso.

Para a implementação desse método utilizamos a linguagem de programação PYTHON, por ser uma linguagem de fácil manipulação, além disso utilizamos uma estrutura de dados do tipo pilha, implementada por meio de uma lista, onde foram colocados os nomes de cada cidade, e criado um grafo do tipo vetor para localizar e ratificar quais cidades eram adjacentes às outras.

O tempo de execução desse método foi de 0,00498 s, sendo calculado a partir do uso da função time() presente na biblioteca *time* da linguagem python.

IV. Busca em Largura.

O método de busca em largura, no qual consiste em percorrer a árvore de dados em largura invés da profundidade, de tal forma que a árvore é percorrida em camadas utilizando como estrutura de dados uma fila FIFO, ou seja, o primeiro a entrar é o primeiro a sair o método de busca em largura foi desenvolvido em linguagem python e aplicado para situação problema apresentada anteriormente.

A busca em Largura não é um método de alta complexidade, visto que usa uma estrutura de dados do tipo FILA FIFO de mediana complexidade de implementação. Além disso, utilizando dicionário de cidades com a linguagem python facilita muito a implementação do método, visto que o mapa pode ser definido com facilidade na linguagem.

O método de busca em largura mostrou-se capaz de encontrar a cidade desejada (neamt), porém vale ressaltar que o método precisou percorrer o mapa inteiro até encontrar a cidade desejada, portanto, em termo de espaço o método de busca em largura não é o mais aconselhável para solução da situação problema em particular.

O tempo de execução para a busca em largura foi calculado utilizando a função time da biblioteca time da linguagem python e o tempo decorrido para encontrar Neamt foi de 0,00708 **s.**

Portanto, diante das informações mostradas verificou-se que o método de Busca em Largura apesar de ter se mostrado capaz de alcançar o objetivo não é o método mais eficiente para a solução do problema em particular, visto que foi necessário percorrer o mapa inteiro até encontrar a cidade desejada (neamt).

V. Busca Gulosa.

Um algoritmo guloso se baseia na técnica de escolher a melhor opção local do desafio ofertado na esperança de encontrar uma ótima solução global. Aplicando-o na problemática de obter o melhor caminho entre dois destinos, este algoritmo usa a heurística de distância em linha reta entre as possibilidades de caminhos e o destino, escolhendo a opção com a menor distância para o destino.

Com o uso do Google Maps para obter as distâncias em linha reta entre as cidades do mapa e o destino, foi criado um arquivo de texto com o nome de cada cidade e sua respectiva distância. Após a leitura do arquivo de texto pelo algoritmo e a definição de cada cidade e seus vizinhos. É configurado o ponto de partida e o destino, passando estes dois atributos para o método de busca gulosa implementado, o método faz a varredura das distâncias da cidade atual com o destino, a cidade com menor distância é escolhida para ser a próxima no caminho, e assim, recursivamente é encontrado o destino.

O método foi capaz de encontrar o destino, Neamt, no segundo menor caminho possível, percorrendo a distância de 856 Km, passando somente pelas cidades que o algoritmo julgou necessárias e descartando o gasto de recursos com processamento em cidades que não ofereciam um bom caminho do ponto de vista do algoritmo.

Capturando o tempo com auxílio da biblioteca time e sua função time do Python, que captura tempo atual em segundos no início da execução do algoritmo guloso, e no final da execução. Desta forma foi possível calcular o tempo de execução do algoritmo guloso para buscar o destino. O tempo gasto foi de 0,01288 s.

VI. Busca A\*

O método de Busca A\* é considerado um método heurístico informado. Possui Completude, não é ótimo, não é irrevogável e não é admissível.

Um algoritmo A\* efetua um cálculo de custo para percorrer os nós do grafo. No caso da aplicação para encontrar o melhor caminho para chegar de uma cidade a outra foram utilizados dois dados para fazer o cálculo de custo. A distância em linha reta entre as cidades em relação a cidade alvo e a distância entre o nó atual e seu adjacente. Com isso a cada nó vai sendo calculado a função de avaliação e determinando o próximo passo a seguir.

Em resumo, foi necessário percorrer 9 nós até achar a cidade objetivo, partindo de Arad até chegar em Neamt. Com um tempo total de execução de 0,0842 s.

VII. Busca Uniforme

Um algoritmo Uniforme efetua um cálculo de custo entre as arestas do nó atual escolhendo sempre a menor distância entre os nós até encontrar o alvo. Ao chegar no alvo o algoritmo prossegue processando as arestas destes alvos até chegar no seu destino. Sempre mantendo a prioridade no processamento das arestas com a menor distância entre o nó de origem e o sendo processado.

Devido a essa característica de processar o que está mais próximo do ponto de partida, o algoritmo de busca uniforme tem sua demanda de processamento diretamente conectada com a distância e a quantidade de possibilidades de chegada ao destino. Somente após calcular todas as possibilidades é possível determinar a melhor rota. Fazendo-o assim um algoritmo de alta demanda de processamento.

Com auxílio do grafo gerado foi atribuído as arestas possíveis com os vizinhos de cada cidade, em seguida, definindo o ponto de partida e o de chegada, o algoritmo inicia sua busca por caminhos viáveis para chegar ao destino, armazenando cada possibilidade, o caminho percorrido e a distância necessária para alcançar o objetivo por este caminho. Ao fim, é feita uma comparação entre as possibilidades. O caminho com melhor distância entre o ponto de saída até o ponto de chegada é escolhido e mostrado ao usuário.

Em conclusão, o método obteve êxito ao encontrar o menor caminho possível. No total, foram obtidas 6 rotas, com distâncias variando de 824 Km até 1244 Km. A menor rota, de 824 Km de distância percorre as cidades de Arad, Sibiu, Rimnicu Valcea, Pitești, Bucharest, Urziceni , Vaslui e Iasi para então chegar em Neamt com sucesso.

A fim de obter este resultado, novamente com a ajuda da biblioteca time do Python foi possível calcular o tempo de execução de todo o algoritmo de busca, que no total, foram necessários 0,1099 s para realizar a busca de todas as rotas e mostrar ao usuário a melhor opção.

# VIII. Conclusão

Após executar todos os métodos em apenas uma máquina verifica-se que todos os métodos de busca foi capaz de encontrar neamt, ou seja, todos os métodos possuem completude para o caso em particular, por outro lado, é possível perceber que nem todos os métodos são admissíveis, como no caso de busca em largura que percorreu 19 cidades até encontrar neamt. Em termos de admissibilidade, o método de busca que destacou-se foi a busca uniforme, porém, a busca uniforme também foi o método que consumiu mais recursos do computador visto que todas as possíveis rotas são testadas e armazenadas para finalmente mostrar ao usuário a rota de melhor caminho. Em termos de complexidade, o método que destacou-se foi a busca em profundidade encontrando neamt em apenas 0,00498 s. Portanto, do ponto de vista do usuário o método mais adequado de uso seria a busca uniforme, pois o mesmo mostra a rota de menor caminho para o usuário evitando assim custos adicionais, como uso de combustível entre outros, por outro lado, em termos de complexidade o método de busca mais adequado que consome poucos recursos do computador foi a busca em profundidade, necessitando de pouco espaço de memória e encontrando neamt em apenas 0,00498 s, menor tempo de execução em comparação com os outros métodos.

##### References

1. Phaneendhar, Reddy, and Vanam, “Shortest path using A\* Algorithm” , Indiana State University , Terre Haute, IN, USA, December 2011.