Implementação e Análise de Algoritmos de Busca para o 8-Puzzle

Rony Soares de Freitas
Universidade do Estado de Minas Gerais
Pará de Minas, Brasil
ronyfreitas98@gmail.com

Vitor Soares Silva
Universidade do Estado de Minas Gerais
Divinópolis, Brasil
vitrola.vitor@gmail.com

***Resumo*—Este trabalho prático tem como objetivo o estudo, análise e comparação entre alguns tipos de algoritmos de busca implementado sobre o 8-Puzzle.**

***Index Terms*—Sistemas Inteligentes, Algoritmo de Busca, Algoritmos, Inteligência Artificial, 8-Puzzle.**

I. Introdução

Diante da enorme quantidade de problemas na sociedade contemporânea que podem ser resolvidos através de análise de dados também é grande a quantidade de estratégias que buscam facilitar tal tarefa. Com isso em mente, vamos abordar brevemente alguns algoritmos que tentam, cada um à sua maneira, resolver problemas com essa natureza de trabalhar em cima da analise de dados.

Algoritmos são nada mais nada menos do que a definição de um passo a passo, como uma receita de bolo ou uma lista de tarefas, com o intuito de resolver um problema. Neste caso, o problema abordado será o jogo 8-puzzle, e o tipo dos algoritmos usados para tal será Algoritmos de Busca.

II. Algoritmos de Busca

Em ciência da computação, um algoritmo de busca, em termos gerais é um algoritmo que toma um problema como entrada e retorna a solução para o problema, geralmente após resolver um número possível de soluções.

Segundo (ROCHA; DORINI, 2004), os algoritmos relacionados com otimização lidam com uma sequência de passos, sendo que em cada passo ha um conjunto de escolhas/opções.

A. Tipos de Buscas

1) Busca Cega: Busca sem Informação ou Busca Cega são algoritmos que não recebem nenhuma informação sobre o problema além de sua formulação. Estratégias de busca sem informação usam apenas a informação disponível na definição do problema, apenas geram sucessores e verificam se o estado objetivo foi atingido.

São exemplos de algoritmos de Busca Cega:

- Busca em largura
- Busca em profundidade

2) Busca com Heurística: Busca sem Informação ou Busca Heurística são algoritmos que utilizam conhecimento específico do problema na escolha do próximo nó a ser expandido. Aplica de uma função de avaliação a cada nó na fronteira do espaço de estados, essa função estima o custo de caminho do nó atual até o objetivo mais próximo utilizando uma função heurística.

São exemplos de algoritmos de Busca Heurística:

- Algoritmo Guloso
- Busca em A*

B. Busca em Largura

Diante da dificuldade para realizar uma busca em um grafo ou uma árvore o algoritmo de Busca em Largura vem para apresentar uma maneira de realizar tal tarefa. O primeiro ponto a ser trabalho é qual caminho seguir. Neste caso, usaremos a premissa de visitar cada um dos pontos, partindo do nó raiz, até o objetivo. Porém, esta estratégia se torna menos efetiva quando a distância do nó alvo se torna excessivamente grande.

Para visitar todos os pontos e não se perder e acabar revisitando pontos, o algoritmo usa um sistema de fila. Basicamente, antes de ir para um ponto da árvore todos os seus filhos são adicionados a uma fila. Após isso o ponto é verificado e caso não seja o objetivo o próximo da fila é submetido ao mesmo procedimento, então removido da fila.

O tempo para realização da busca está diretamente ligada ao tamanho da árvore e a distância entre o nó alvo e o início da busca normalmente partindo da raiz. Em um grafo que sua profundidade tende a ser muito grande, o tempo e o armazenamento necessário para realizar a busca crescem em conjunto.

Portanto, temos um algoritmo que nos garante que irá encontrar o objetivo, já que passa por todos os pontos sem exceção, porém dependendo do caso, pode demorar mais do que o esperado, então cabe ao usuário avaliar se esta técnica será a melhor a ideal.

C. Busca em Profundidade

Com a mesma proposta de realizar uma busca em grafos e árvores, a busca em profundidade utiliza de uma maneira diferente para decidir qual caminho trilhar até o objetivo. Parte em um nó até encontrar o objetivo ou até o a expansão dele não ser mais possível. Assim como na busca em largura, o

tempo está diretamente ligado ao tamanho do grafo. Porém, existe casos onde o pode-se ficar preso em um loop.

Um nó é escolhido e expandido até que não seja mais possível ou necessário. Em casos onde a profundidade é grande demais para ser armazenada na memória um limite de profundidade é estabelecido de modo a conseguir realizar a busca em parte da árvore sem atingir o estourar a memória, ou mesmo simplesmente para colocar um limite para o tempo de execução.

Alguns casos podem levar o algoritmo a ficar preso, então é necessário organizar o código de modo a não se prender em um loop. Existem uma série de variações e técnicas que abordam essa questão.

Portanto, assim como no caso da busca em largura, cabe um estudo de caso antes da execução da busca em si, levando em conta largura e profundidade da árvore a ser analisada de modo a ser aplicada a melhor técnica.

D. Algoritmo Guloso

O algoritmo Guloso é uma técnica que parte da ideia de que seguindo os melhores caminhos vai encontrar a melhor solução para o problema. Utilizando essa ideia, em alguns casos, ele consegue encontrar a solução ótima, que seria a melhor, ou uma das, dentre as soluções para o problema. Entretanto, ele é uma ótima escolha para problemas de NP-Completo.

Problemas de NP-Completo são problemas que não se consegue saber qual a melhor solução, pois caso seja investido mais tempo na busca é possível encontrar uma solução melhor. Basicamente, só poderia dizer que dada solução é a melhor, caso tenha testado todas as possibilidades, e estas tendem a crescer polinomialmente, assim como o tempo para realizar os cálculos.

Algumas vantagens deste algoritmo são: a fácil implementação; a velocidade de execução; podem encontrar a solução ideal. Porém, não deixa de ter desvantagens que incluem: nem sempre encontrar a solução ótima global; possibilidade de se prender em um loop; cálculos repetitivos.

Então, fica claro que entre vantagens e desvantagens, ele tem sim sua aplicação, se sobressaindo em alguns casos que os outros algoritmos não se dão muito bem, e sendo menos otimizado em outros casos. Cabe novamente o estudo quanto a sua aplicação.

*E. A**

O algoritmo A* é um algoritmo de busca em grafos, que se utiliza de uma heurística para auxiliar a ordem em que os nós são processados, com o intuito de diminuir o tempo de processamento. O A* pode ser usado quando um caminho mais curto não é rápido o suficiente, embora seja a única opção válida para um dado problema.

III. O Jogo 8-Puzzle

O jogo do 8-Puzzle é um jogo de tabuleiro de blocos deslizáveis , que consiste em um tabuleiro quadrado com nove divisões (matriz), contendo números (ou letras) e um espaço vazio. Esse espaço vazio permite que as peças vizinhas possam "deslizar". Ou seja, uma peça ao lado de um es paço vazio pode ser movida para essa lacuna, criando uma nova posição de lacuna. Em outras palavras, a lacuna pode ser trocada de posição com uma peça adjacente (horizontal e verticalmente). Dessa maneira pode-se mudar a configuração inicial da matriz, movimentando-se as peças. A meta do jogo é chegar a uma certa configuração final.

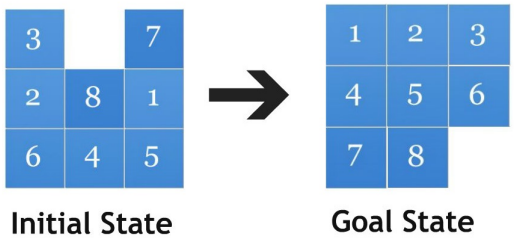

Figura 1. Jogo 8-Puzzle

O jogo consiste, portanto, em movimentar as peças partindo de uma configuração inicial, com as peças todas desordenadas, com o objetivo de chegar à configuração final.

IV. Implementação do Problema

O trabalho consiste em escrever dois algoritmos, um de busca cega e outro de busca com informação, que resolva o 8-puzzle. O algoritmos escolhidos são o Busca em Profundidade e o Algoritmo Guloso.

A. Ambiente e Recursos

A resolução para o problema foi desenvolvida utilizando a linguagem de programação C++ pelo IDE Visual Studio Code e foi executado para testes e obtenção de resultados em um notebook Dell, S.O. Ubuntu 18.04, processador Intel CORE i7, 8GB de memória RAM.

V. Resultados e Análise

Infelizmente nosso algoritmo não obteve os resultados desejados, sendo finalizado apensar em alguns instancias do problema proposto. Em alguns casos, o Algoritmo Guloso trava e o Algoritmo de Busca em Profundidade entra em loop.

Faremos a comparação sobre uma instância específica, onde os dois algoritmos foram executados e encontram a solução do problema com quantidade de interações e tempo diferentes.

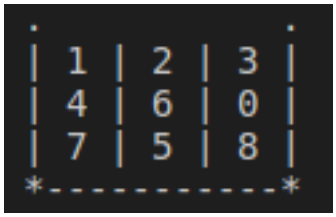

Figura 2. Instância analisada

A. Busca em Profundidade

A busca em profundidade, por trabalhar sem informações definidas está sujeita a não encontrar uma solução ótima, retornando a primeira solução encontrada. Por trabalhar com trabalhar com recursividade e criação de árvores de busca, o algoritmo tende a demandar mais recursos de hardware e também tempo de operação.

O algoritmo encontrou uma solução com 14 iterações e demandou um tempo total de 1.04587 ms (milissegundos).

B. Algoritmo Guloso

O algoritmo guloso, por ser uma busca com heurística, contém uma função heurística para auxiliar na tomada de decisão do próximo movimento da solução.

O algoritmo encontrou uma solução com 4 iterações e demandou um tempo total de 0.815087 ms (milissegundos).

VI. Considerações Finais

Esse trabalho abordou o 8-Puzzle, um jogo com implementações interessantes de vários algoritmos por sua estrutura e complexidade.

Nesse estudo, propusemos e comparamos o problema sobre um algoritmo de Busca Cega (Busca em Profundidade) e um algoritmo de Busca com Informação (Algoritmo Guloso), implementados em C++.

O trabalho alinhou nosso aprendizado teórico, visto em sala de aula, com desenvolvimento práticos de programação e análise.

O código implementado pode ser encontrado nas contas do GitHub de cada autor.

Referências

ROCHA, A.; DORINI, L. B. Algoritmos gulosos: definições e aplicações. *Campinas, SP*, v. 53, 2004.