Implementações e Análises com Busca Cega e Busca com Informação

Lucas Fontes Buzuti
Departamento de Engenharia Elétrica
Centro Universitário FEI
São Bernardo do Campo-SP, Brasil
lucas.buzuti@outlook.com

***Resumo*—Esse artigo tem uma finalidade acadêmica nas compreensões e implementações de algoritmos de busca cega e busca com informação. Os algoritmos: busca em largura, busca em profundidade, subida da encosta e A* foram analisados e comparados a partir de uma condição inicial e um objetivo, mediante a resolver o Jogo-dos-Oito. Nas implementações foi utilizada a programação orientada a objeto na linguagem *C++*, tendo em foco a otimização e a velocidade na execução dos algoritmos.**

***Index Terms*—busca cega, busca com informação, programação orientada a objeto, C++**

I. Introdução

Esse artigo tem em seu objetivo as compreensões e implementações dos algoritmos: busca em largura (Breadth First Search), busca em profundidade (Deep First Search), subida da encosta (Hill-Climbing) e A* (A Star) para resolver o Jogo-dos-Oito, sendo que os dois primeiros algoritmos são categorizados como busca cega e os restantes como busca informada (ou heurística). Além das implementações, uma análise comparativa e crítica dos algoritmos são propostas. A implementação tem como alvo a utilização da linguagem *C++*, pois é uma linguagem focada na otimização e na velocidade da execução de algoritmos.

Uma busca visa encontrar uma solução para um problema através de um mecanismo algorítmico que procura no espaço de soluções o objetivo do problema que foi proposto.

Mesmo sendo algoritmos da década de 1950 e 1960, mostram ser eficientes nas década seguintes e variações de seu mecanismo são propostos para resolver problemas mais complexos [1] [2].

II. Teoria

A. Busca em Largura

Em 1959, Moore utilizou busca em largura para encontrar o caminho mais curto para sair de um labirinto [7]. Dois anos depois, Lee publicou um artigo sobre algoritmo de roteamento de fio (wire routing) [4].

A busca em largura expande todos os nós (estados) de cada profundidade do grafo até encontrar o objetivo, mostrado na Figura 1. A expansão ocorre no nó anexado no inicio da lista, removendo-o e inserindo seus nós filhos ao final da lista, podendo apontar a lista como uma fila, portanto, a implementação da busca em largura usufrui da estrutura de dados fila (FIFO).

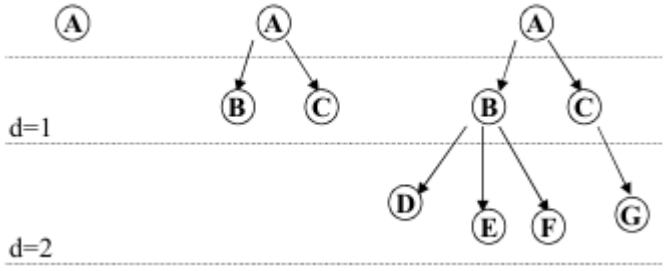

Figura 1. Exemplo da busca em largura. $FILA \Rightarrow \{A\} \rightarrow \{B, C\} \rightarrow \{C, D, E, F\} \rightarrow \{D, E, F, G\}$

Em sua caracterização a busca em largura é completa (sempre encontra uma solução se existir), é ótima (encontra o menor caminho, no qual sempre expande o primeiro nó mais raso) e apresenta um custo de tempo b^d, em que b é todos os nós gerados a partir de um nó e d é a profundidade (nível) ao decorrer das expansões dos nós.

B. Busca em Profundidade

Busca em profundidade teve sua primeira citação no século XIX pelo matemático francês Charles Trémaux com o objetivo de resolver labirintos [5] [6].

Na busca em profundidade expande o primeiro nó (estados) da primeira profundidade do grafo em segui expande o segundo nó da segunda profundidade e assim por diante até encontrar o objetivo, mas, quando um nó final não é o objetivo, o algoritmo retorna para expandir os nós que ainda se encontram na fronteira do espaço de estados, esse retorno é conhecido como *backtracking*, mostrado na Figura 2. A expansão ocorre no nó anexado no inicio da lista, removendo-o e inserindo seus nós filhos no inicio da lista, podendo apontar a lista como uma pilha, portanto, a implementação da busca em profundidade usufrui da estrutura de dados pilha (LIFO).

Em sua caracterização a busca em profundidade não é completa (pois precisa-se definir um limite de profundidade senão o algoritmo pode ficar paralisado ao descer um caminho muito longo), não há garantia de ser ótima e apresenta um custo de tempo b^m, em que b é todos os nós gerados a partir de

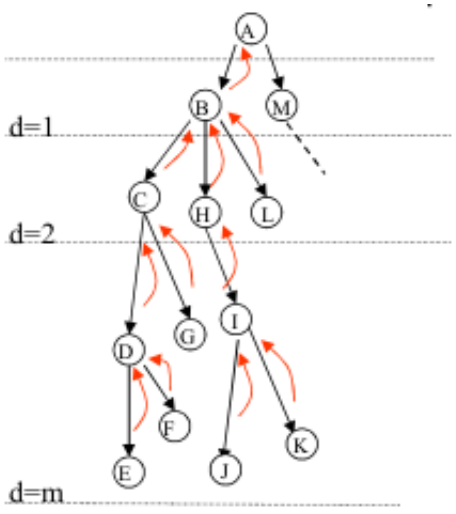

Figura 2. Exemplo da busca em profundidade. $PILHA \Rightarrow \{A\} \rightarrow \{B, M\} \rightarrow \{C, H, L, M\} \rightarrow \{D, G, H, L, M\} \rightarrow \{E, F, G, H, L, M\} \rightarrow \{F, G, H, L, M\} \rightarrow \{G, H, L, M\} \rightarrow \{H, L, M\} \rightarrow \{I, L, M\} \rightarrow \{J, K, L, M\} \rightarrow \{K, L, M\} \rightarrow \{L, M\} \rightarrow \{M\}...$

um nó e m é a profundidade (nível) ao decorrer das expansões dos nós.

C. Subida da Encosta

Subida da encosta é um algoritmo guloso (greedy algorithm), no qual segue uma heurística de resolução do problemas proposto para fazer a escolha ótima em cada estágio com a intenção de encontrar um ótimo global [8]. Entretanto, utilizar algoritmos gulosos podem não produzir uma solução ótima, mas uma função heurística, quando muito bem dimensionada permite a reprodução da solução ótima e o processos de busca tornam-se mais velozes.

O algoritmo expande o primeiro nó encontrado cuja a função heurística é melhor que a do nó pai que o gerou e ignora os demais nós, mostrado na Figura 3. Entretanto, por ignorar os demais nós, a caracterização da subida da encosta não é completa e não é ótima (podendo não gerar uma solução ótima), mas, em contrapartida o custo de tempo é baixo sendo para o melhor caso d e no pior caso b^d, em que b é todos os nós gerados a partir de um nó e d é a profundidade (nível) ao decorrer das expansões dos nós.

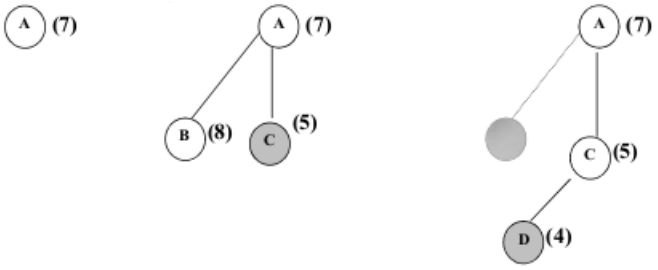

Figura 3. Exemplo da subida da encosta.

D. A*

Em 1968, Hart, Nilsson e Raphael do Stanford Research publicaram um artigo com o tema "Uma base formal para a determinação heurística de caminhos de custo mínimos", no qual referiram-se ao algoritmo A* (A-Estrela) [3]. O algoritmo A* surgiu como uma etapa do projeto Shakey, que tinha o objetivo de projetar um robô móvel que pudesse tomar suas próprias ações.

A busca A* é a busca Melhor-Escolha guiada pela função de avaliação $f(n)$, ou seja, alcança um melhor desempenho usando $f(n)$ para guiar sua busca. A* expande o nó de menor valor de $f(n)$ na fronteira do espaço de estados até encontrar o objetivo, ilustrado na Figura 4. A função de avaliação é composta pela heurística e além da heurística, pode-se ter o custo de cada ramo da árvore de busca, sendo assim define-se a função de avaliação como:

$$f(n) = g(n) + h(n), \tag{1}$$

onde $g(n)$ representa a função de custo de n ao nó inicial e $h(n)$ representa a função heurística. Se $f(n) = h(n)$, então a busca, guiada pela função de avaliação, depende única e exclusivamente da função heurística. No caso de h ser admissível, $f(n)$ nunca irá superestimar o custo real da melhor solução através de n, logo, o A* encontrará a solução ótima em termos de custo.

Em sua caracterização a busca A* é completa, é ótima com relação ao custo (minimiza o custo quando a heurística é admissível) e apresenta um custo de tempo b^d, em que b é todos os nós gerados a partir de um nó e d é a profundidade (nível) ao decorrer das expansões dos nós. Entretanto, o A* é um algoritmo caro computacionalmente, mas é o melhor algoritmo para encontrar soluções ótimas com relação ao custo.

III. Proposta e Implementação

Esse artigo propõe utilizar a linguagem *C++* para as implementações[1] dos algoritmos busca em largura, busca em profundidade, subida da encosta e A* para resolver o Jogo-dos-Oito. Além das implementações, uma análise comparativa e crítica dos algoritmos são propostas. Estas implementações foram feitas em um computador com sistema operacional *Linux* e com o compilador *GCC*, a versão da linguagem utilizada foi a *C++11*. Para desenvolver os algoritmos, foi utilizada a programação orientada a objeto (POO), onde visa a construção de classes e métodos.

Para resolver o Jogo-dos-Oito foi desenvolvida uma classe *puzzle8*, no qual toda a logica do jogo está modelada em forma de métodos e atributos de classe. No Jogo-dos-Oito têm-se apenas quatro operadores reversíveis da expansão de um estado, sendo esses: cima, baixo, direita e esquerda. Todavia, a definição dos operadores são estabelecidas mediante o posicionamento peça " 0 ", i.e., na Equação 2 onde está sendo representada um estado inicial do jogo os operadores

[1] https://github.com/buzutilucas/scientific-programming/tree/master/Ex03

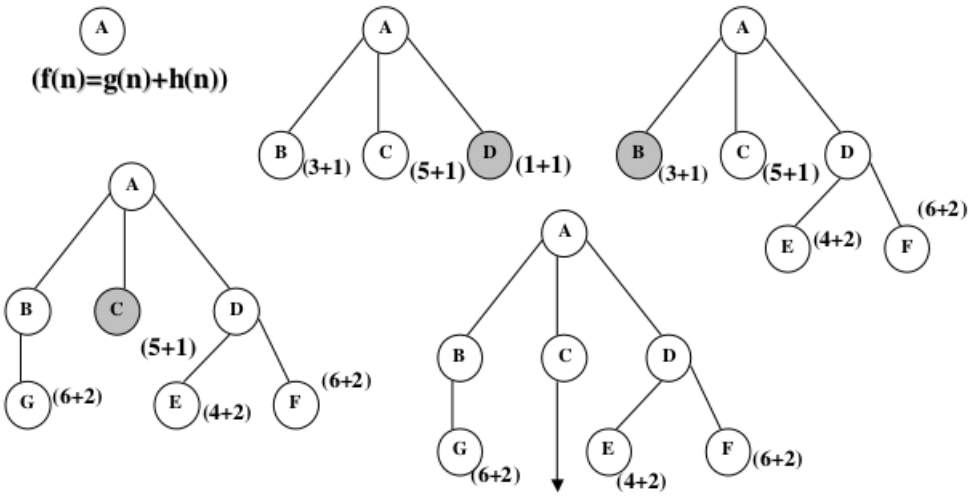

Figura 4. Exemplo da busca A*.

reversíveis são: cima, direita e esquerda. Na Figura 5, ilustra-se o diagrama UML da classe *puzzle8*.

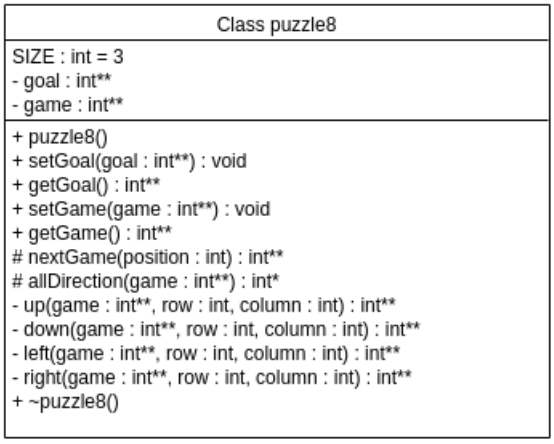

Figura 5. Diagrama UML da classe *puzzle8*.

A partir do Jogo-dos-Oito modelado foram devolvidas as classes dos algoritmos busca em largura, busca em profundidade até o nível 13, subida da encosta e A*. Na Figura 6, ilustra-se os diagramas UML dos algoritmo.

IV. Experimentos e Resultado

Para analisar a eficiência dos algoritmos no problema do Jogo-dos-Oito foi desenvolvido um arquivo de teste em $C++$ denominado *main* e nesse arquivo definiu-se o estado inicial sendo:

$$S(0) = \begin{bmatrix} 4 & 1 & 6 \\ 3 & 2 & 8 \\ 7 & 0 & 5 \end{bmatrix}, \tag{2}$$

e o estado objetivo sendo:

$$S(n) = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}. \tag{3}$$

Após a definição do estado inicial e objetivo, gerou-se um timer para computar o tempo em que cada algoritmo leva para encontrar o estado objetivo. Para os algoritmos que utilizam-se de heurística tais como: subida da encosta e A*, a distância de Manhattan foi adotada. Na Tabela I, têm-se os tempos de processamento de cada algoritmo.

Tabela I
Tempos de processamento dos algoritmos.

Algoritmo	**Tempo (s)**
Busca em Largura	24.1827
Busca em Profundidade (nível 13)	15.2506
Subida da Encosta	*Objetivo não encontrado!*
Busca A*	0.0324

Analisando quantitativamente os resultados da Tabela I, observa-se que o A* tem o menor tempo de processamento até o estado objetivo mas, a busca em largura e profundidade também conseguem encontrar o estado objetivo, entretanto, com um tempo de processamento muito superior do A*. Para computar a busca em profundidade foi definido um limite de nível para ocorrer a busca do objetivo, senão a busca entraria em um loop e o tempo de processamento seria infinito, mas se o limite do nível for inferior ao do objetivo o algoritmo não consegue encontra-lo, em outras palavras, busca em profundidade deve ser evitada quando m é muito maior que d ou mesmo quando não é conhecido, mas, para problemas com várias soluções usar essa busca pode ser bem mais rápida do que busca em largura. A subida da encosta

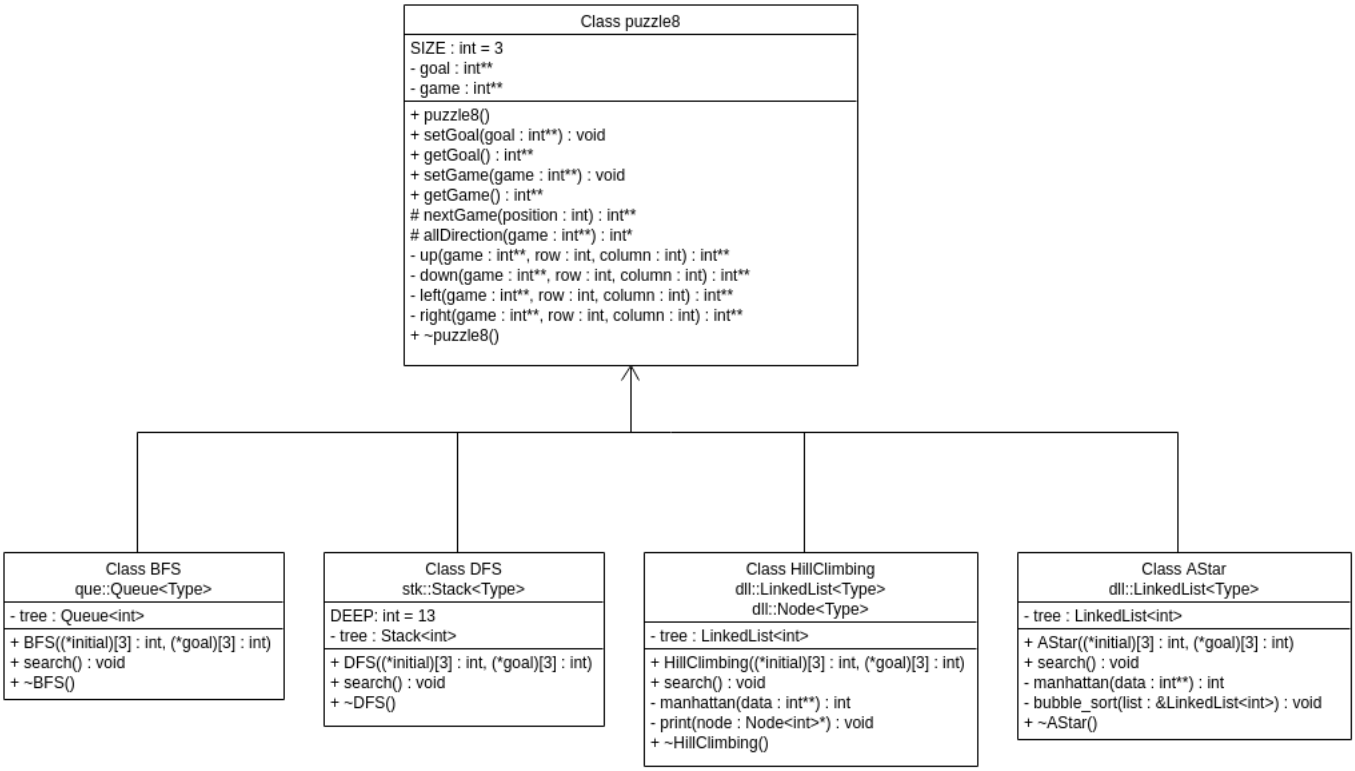

Figura 6. Classe *BFS* (busca em largura), classe *DFS* (busca em profundidade), classe *HillClimbing* (subida da encosta) e classe *AStar* (A*).

não consegue encontrar o objetivo, porque necessita-se de uma função heurística muito bem dimensionada, então a utilização da distância de Manhattan para esse estado inicial leva o algoritmo a ficar estagnado em um ótimo local.

V. Trabalhos Correlatos

Trabalhos sobre buscas tem uma alta relevantes nos dias atuais, pois há uma necessidade de encontrar o objetivo com o caminho mais rápido de menor custo. Algoritmos aprendizado de maquinas utilizam-se das teorias ou dos próprios algoritmos em sua modelação matemática e em seus aperfeiçoamentos [1] [2].

VI. Conclusão

Nesse artigo pode-se compreender e analisar dos algoritmos: busca em largura (Breadth First Search), busca em profundidade (Deep First Search), subida da encosta (Hill-Climbing) e A* (A Star) para resolver o Jogo-dos-Oito. A busca A* teve o menor tempo de processamento em relação aos outros algoritmos, portanto sendo o melhor algoritmo dos quatro, mas computacionalmente é muito caro. Entretanto, pode-se optar pela busca em largura ou pela busca em profundidade para problemas mais complexo que o Jogo-dos-Oito. No caso da subida da encosta pode-se utilizar uma outra heurística para o algoritmo conseguir se desenvolver e encontrar o objetivo.

Referências

[1] Al-Betar, Mohammed Azmi et al. "β-Hill Climbing Algorithm for Sudoku Game." 2017 Palestinian International Conference on Information and Communication Technology (PICICT) (2017): 84-88.

[2] Kagan, Eugene and Irad Ben-Gal. "A group testing algorithm with online informational learning." (2014).

[3] Hart, Peter E., Nils J. Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths."IEEE transactions on Systems Science and Cybernetics 4.2 (1968): 100-107.

[4] Lee, Chin Yang. "An algorithm for path connections and its applications."IRE transactions on electronic computers 3 (1961): 346-365.

[5] Even, Shimon. Graph algorithms. Cambridge University Press, 2011.

[6] Sedgewick, Robert. "Algorithms in C++, Part 5: Graph algorithms. Addition-wesley."(2002).

[7] Moore, Edward F. "The shortest path through a maze."Proc. Int. Symp. Switching Theory, 1959.

[8] Russell, Stuart J.; Norvig, Peter (2003), Artificial Intelligence: A Modern Approach (2nd ed.), Upper Saddle River, New Jersey: Prentice Hall, pp. 111–114,