

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ  
DEPARTAMENTO DE COMPUTAÇÃO  
CURSO DE ENGENHARIA DE COMPUTAÇÃO

ÁLVARO PEDROSO QUEIROZ – RA 1913255  
ISABELA NUNES CAETANO – RA 1914774  
LEONARDO DE SOUZA MATEUS – RA 1914782

IMPLEMENTAÇÃO DO ALGORITMO DE BUSCA A\* PARA O JOGO  
DOS 8

Relatório do Projeto Final para a disciplina de Estrutura de Dados

CORNÉLIO PROCÓPIO

2018

# SUMÁRIO

INTRODUÇÃO.....	1
1 N-PUZZLE .....	2
2 ESTRATÉGIAS DE BUSCA.....	3
2.1 Heurística .....	3
2.2 Algoritmo de busca A* .....	4
3 ESTRATÉGIAS DE IMPLEMENTAÇÃO .....	5
3.1 Estruturas de Dados utilizadas.....	5
3.2 Cálculo da heurística .....	6
3.3 Eliminação das redundâncias .....	6
3.4 Determinação de soluções possíveis .....	7
3.5 Algoritmo .....	7
4 RESULTADOS OBTIDOS.....	8
5 CONCLUSÃO .....	13
REFERÊNCIAS .....	14

## INTRODUÇÃO

O jogo dos N-Puzzle, bastante popular, é um jogo de tabuleiro quadrado, com peças dispostas de 1 a  $n$ , além de um campo vazio, do qual partindo de um estado inicial aleatório, deve-se chegar a um estado final, podendo mover somente as peças adjacentes ao espaço em branco.

Neste âmbito, é possível empregar soluções computacionais para o problema em questão. Por meio de análises lógicas, utilizando informações do domínio do problema, é possível selecionar o melhor caminho até encontrar sua solução.

Por conseguinte, quando tratamos da solução por meio de uma análise lógica empregamos o conceito de estratégias de busca, que são compostas por devidas estruturas de dados na implementação do algoritmo, o qual consiste o jogo.

A fim de encontrar o melhor caminho aplicou-se para a solução do problema em questão especificamente a estratégia de busca A\*, que promete alcançar o objetivo de forma mais eficiente, analisada sua completude, que diz respeito a complexidade de tempo e espaço.

## 1 N-PUZZLE

O jogo dos N-Puzzle é um jogo de tabuleiro quadrado que pertence à família dos blocos deslizáveis (Puzzles). Desta maneira, o tabuleiro pode ser de várias dimensões, entretanto, o mais comum é o de dimensão 3x3 (linhas x colunas), consistindo em oito peças e um espaço vazio, conhecido como jogo dos oito.

Por exemplo, no jogo dos oito (8 Puzzle), também conhecido como quebra-cabeça de oito ou jogo das oito fichas, as peças são números dispostos de 1 a 8, além de um espaço em branco, podendo ser movidas somente as adjacentes ao campo vazio, invertendo assim suas posições (espaço em branco x peça).

Seu objetivo é obter, a partir de uma posição inicial (estado inicial), uma posição “meta” (estado final), deslizando as peças no espaço vazio até que atinja o objetivo, como o exemplo da figura a seguir.

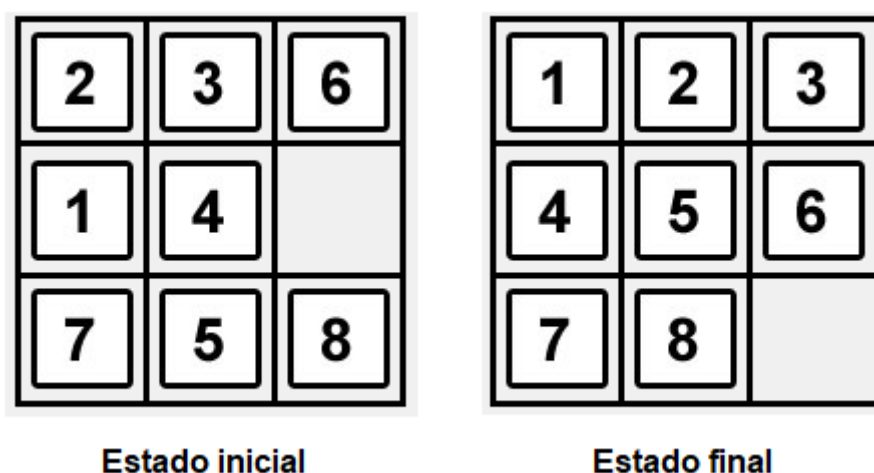


Figura 1 - Objetivo do jogo

Este estilo de jogo permite aplicar diversos métodos de solução, como os algoritmos de busca em profundidade, busca em largura, ou ainda técnicas de busca como a busca bidirecional, além do mais, permite a aplicação de testes e novos algoritmos em Inteligência Artificial (MONTESCO e TACO, 1999).

## **2 ESTRATÉGIAS DE BUSCA**

Dado um problema do mundo real é possível abstraí-lo a fim de empregar uma estratégia de busca que consiste no melhor caminho para alcançar um objetivo. Para tanto deve-se avaliar a situação atual, formular o objetivo, bem como o problema, conservar as informações relevantes e aplicar a busca que insere uma estratégia de controle.

Contextualizando, uma estratégia de busca pode ser definida pela escolha da ordem de expansão de nós visto que a partir da busca há a tomada de decisão que devem levar a solução pelo menor “custo”, conforme definiu (GAGNON, 2000).

Um algoritmo de busca é capaz de explorar um espaço de estados (conjunto de todos os estados acessíveis a partir de um estado inicial), que servirão para encontrar a sequência de ações que levam ao estado final.

### **2.1 Heurística**

Uma busca informada se refere a utilização de um conhecimento específico além da definição do problema, o que é promissor, visto que pode encontrar soluções mais eficientes do que uma busca não informada (busca cega) (LEE, [2000?]).

A heurística, neste caso, pode ser muito útil por quantificar a proximidade a um determinado objetivo, encontrando assim os caminhos mais promissões primeiro.

No jogo dos oito, a heurística pode ser definida de várias formas, assim como em outros problemas, entretanto, cada uma gera uma complexidade diferente ao algoritmo. Neste caso, pode-se utilizar como função de avaliação: o número de peças fora do lugar, soma das distâncias de cada número à sua posição final (por quadras).

A segunda opção é conhecida como distância “Manhattan” e de modo empírico, é possível determinar a qualidade da mesma, por meio do total de nós gerados pelo algoritmo de busca e pela profundidade da solução, o que definem o fator de ramificação.

Como a heurística da distância Manhattan possui um fator de ramificação menor, ela pode ser considerada melhor do que a outra opção e

ainda pode-se concluir que uma função heurística com valores mais altos é melhor desde que não leve muito tempo para ser computada (SAITO, 2018). Em suma, quanto menor o espaço de estados gerado, mais rápido o algoritmo encontra a solução.

## **2.2 Algoritmo de busca A\***

O algoritmo de busca A\* é uma das técnicas de buscas mais utilizadas. A\* trata de minimizar o custo total estimado da solução, por meio de uma função heurística que combina o custo real do caminho e o custo estimado para ir do nó até o objetivo.

Em suma, o fundamento é expandir o nó mais promissor (com a menor heurística) e evitar expandir caminhos que têm altas heurísticas. Para tanto, é preciso analisar não somente o nó mais promissor, como também a coleção de nós que foram gerados, mas ainda não foram expandidos, sendo a chamada borda.

### 3 ESTRATÉGIAS DE IMPLEMENTAÇÃO

Neste capítulo será apresentado a proposta de resolução do problema em questão, por meio de diferentes estruturas de dados e funções que aplicam a estratégia de busca A\*.

A ideia é expandir as possibilidades conforme suas respectivas heurísticas, para tanto, a maioria das funções foram aplicadas com recursividade visto que devem ser executadas enquanto o estado final não seja encontrado, ou seja, enquanto o tabuleiro de heurística igual a zero não seja encontrado.

#### 3.1 Estruturas de Dados utilizadas

A representação do tabuleiro foi implementada em uma matriz de valores inteiros, sendo o espaço em branco descrito pelo número nove. Dessa forma, a mesma foi gerada de forma aleatória, entretanto, ainda deve ser analisado se o tabuleiro gerado pode ser solucionado ou não.

Para percorrer o espaço de estados gerados (os possíveis movimentos do tabuleiro) optou-se por utilizar uma árvore de busca, conforme a imagem a seguir:

```
typedef struct arvore{
    int cod;
    int heuristica;
    int matriz[3][3];
    struct arvore* rsup;
    struct arvore* rinf;
    struct arvore* rdir;
    struct arvore* resq;
    char movimento[30];
    int qtdmov;
}Arvore;
```

Figura 2 - Árvore de busca

A fim de diminuir a complexidade do problema e do algoritmo, determinou-se para essa estrutura um código, que é diferente para cada novo estado gerado, o qual será utilizado para a busca nas listas, que serão aprofundadas a frente.

A heurística é essencial para a determinação do melhor caminho em cada novo nível da árvore. A matriz é o próprio tabuleiro, enquanto os ponteiros rsup, rinf, rdir e resq se referem às possibilidades de um estado, que são os movimentos para cima, para baixo, para a direita e para a esquerda, definidos em movimento.

Por fim, qtdmov se refere à própria profundidade da árvore na qual está o estado final, visto que a cada nível avançado pode-se computar um novo movimento.

Quanto ao gerenciamento dos possíveis caminhos foram criadas duas listas simples. Uma delas guarda as heurísticas das árvores que ainda não tiveram suas ramificações abertas e a outra guarda as árvores que já foram abertas.

### **3.2 Cálculo da heurística**

Como trabalhou-se com matrizes, para o cálculo da heurística, percorreu-se a mesma comparando com a matriz referente ao estado final e realizou-se a soma da quantidade de posições erradas de cada peça/número. Para tanto, encontrou-se a linha e coluna do valor e posteriormente a posição na qual o mesmo deveria estar, logo, comparando as posições foi possível determinar a quantidade de posições erradas, bem como a distância da posição ideal.

### **3.3 Eliminação das redundâncias**

Conforme a implementação tomou forma observou-se a necessidade do tratamento de alguns casos gerados pelo problema em questão. Um deles se refere a repetição de estados durante a escolha do melhor caminho o que acabava gerando um loop de um estado para o outro.

Para resolver o problema em questão implementou-se a comparação de uma matriz que poderia ser gerada (tendo a menor heurística) com as matrizes que já foram abertas, para que a mesma não pudesse ser gerada novamente e não houvesse a repetição de ciclos.

Em suma, a resolução do problema se deu de forma rápida e eficiente, visto que há havia sido criada uma lista com as matrizes abertas.



### 3.4 Determinação de soluções possíveis

Ao longo do desenvolvimento do algoritmo, percebeu-se que algumas situações iniciais, não eram possíveis de se resolver.

Para poder verificar se o estado inicial é solucionável, é preciso descrever todos os valores em sequência, em um vetor, e verificar suas inversões. Uma inversão é descrita quando um bloco precede outro bloco com um número menor, ou seja, da se a um par de blocos (a,b) tal que a aparece antes de b, mas  $a > b$ . Sendo assim, se o tabuleiro tiver a largura da grade com número ímpar, o número de inversões em uma situação solucionável será par (RYAN, 2004).

Para tanto, foi desenvolvido uma função que trata esse tipo de problema, ao começar o programa, cria-se uma matriz e verifica-se o número de inversões desta. Caso o número de inversões seja par, o algoritmo continua e busca a sua resolução, caso seja ímpar, o algoritmo avisa que esse estado inicial não possui uma solução.

### 3.5 Algoritmo

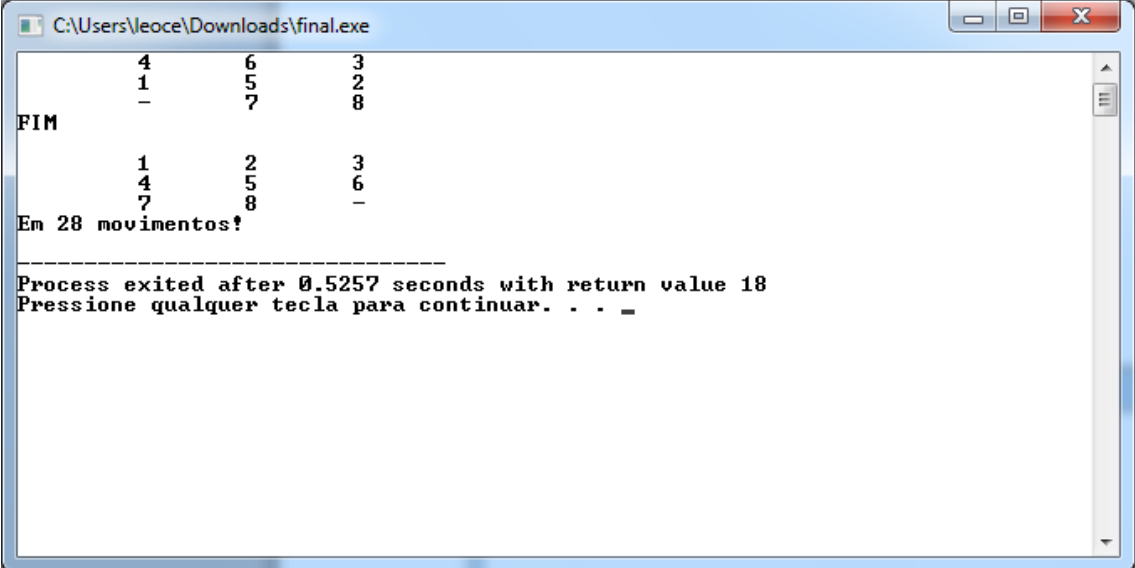
Neste âmbito, o algoritmo funciona da seguinte forma:

- Gera uma matriz aleatória que tenha uma solução possível
- Gera os possíveis movimentos
- Adiciona as heurísticas de cada nova possibilidade na lista 1, referente as matrizes fechadas
- Identifica a matriz com menor heurística, remove tal heurística da lista 1 e adiciona o código dessa matriz na lista 2, referente as matrizes já abertas

Sendo assim, o processo é repetido (de forma recursiva) até que seja encontrado a matriz com o estado final requerido e heurística igual a zero. Aplicando o algoritmo A\* pode ser que a primeira tentativa não seja elencada como o melhor caminho de forma definitiva (a priori) e então seja preciso retornar a outro estado para percorrer novos caminhos.

## 4 RESULTADOS OBTIDOS

De acordo com as estratégias já demonstradas o programa foi compilado para algumas situações e em todas as possibilidades o algoritmo chegou a uma resposta coerente. É notório que muitas vezes o estado final não foi obtido diretamente pelo menor caminho, porém, é possível afirmar que este consegue chegar ao estado final para todos os casos solúveis, conforme as figuras a seguir.



```
C:\Users\leoce\Downloads\final.exe

  4      6      3
  1      5      2
  -      7      8

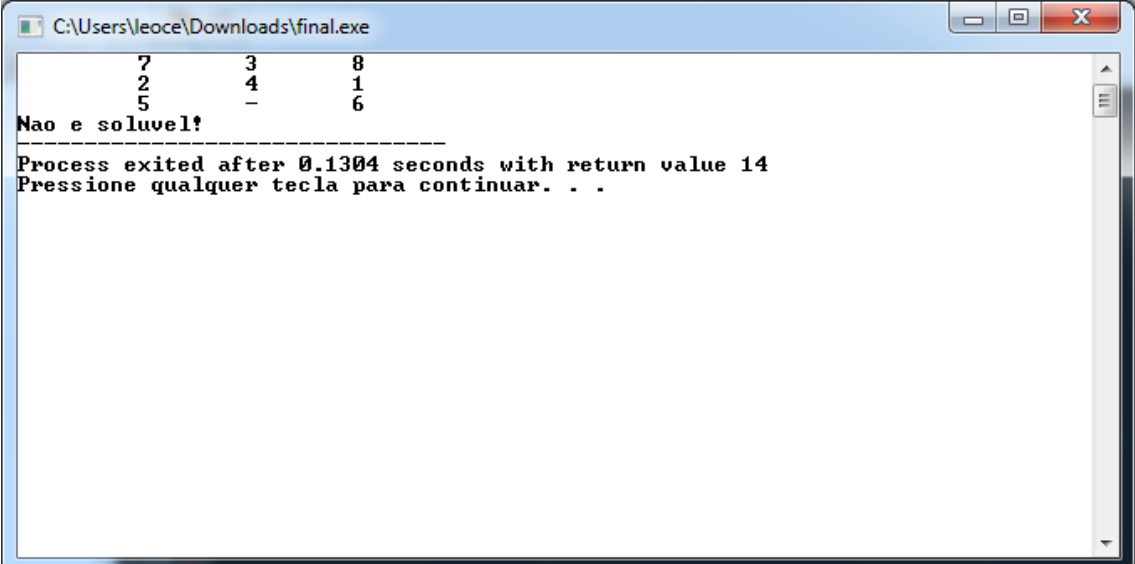
FIM

  1      2      3
  4      5      6
  7      8      -

Em 28 movimentos!

-----
Process exited after 0.5257 seconds with return value 18
Pressione qualquer tecla para continuar. . . _
```

Figura 3 - Exemplo de caso (01)



```
C:\Users\leoce\Downloads\final.exe

  7      3      8
  2      4      1
  5      -      6

Nao e soluvel!

-----
Process exited after 0.1304 seconds with return value 14
Pressione qualquer tecla para continuar. . .
```

Figura 4 - Exemplo de caso (02)

Além do mais, os métodos utilizados para prevenir redundâncias tiveram o efeito esperado. Por fim, pode-se notar também que o tempo de execução foi relativamente pequeno, mostrando assim a eficiência do algoritmo.

A seguir, é apresentando um exemplo de um estado inicial e seus respectivos passos para alcançar o estado final (Figura 5), bem como a resposta do algoritmo (Figuras 6, 7 8,).

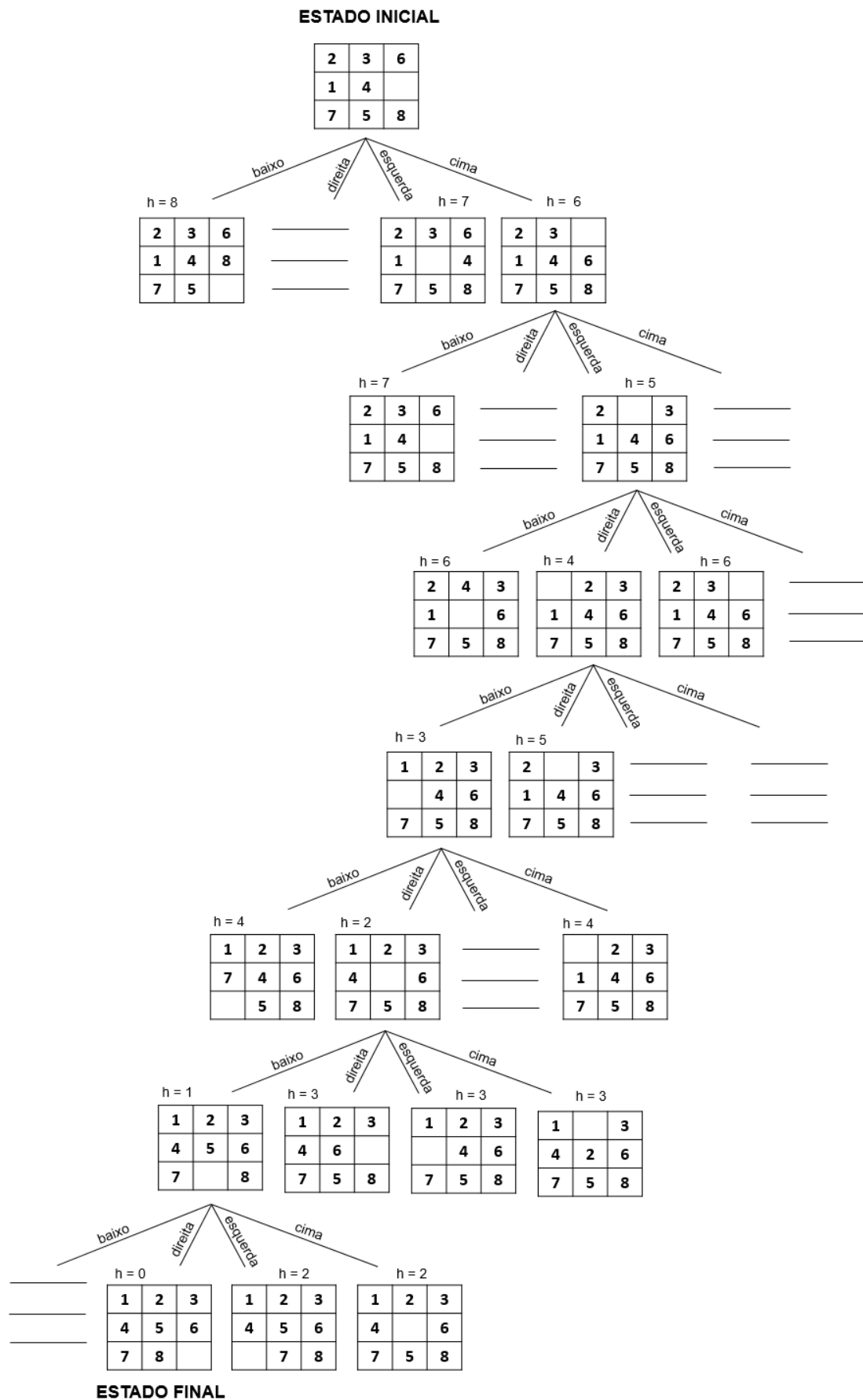


Figura 5 - Exemplo estratégia de busca

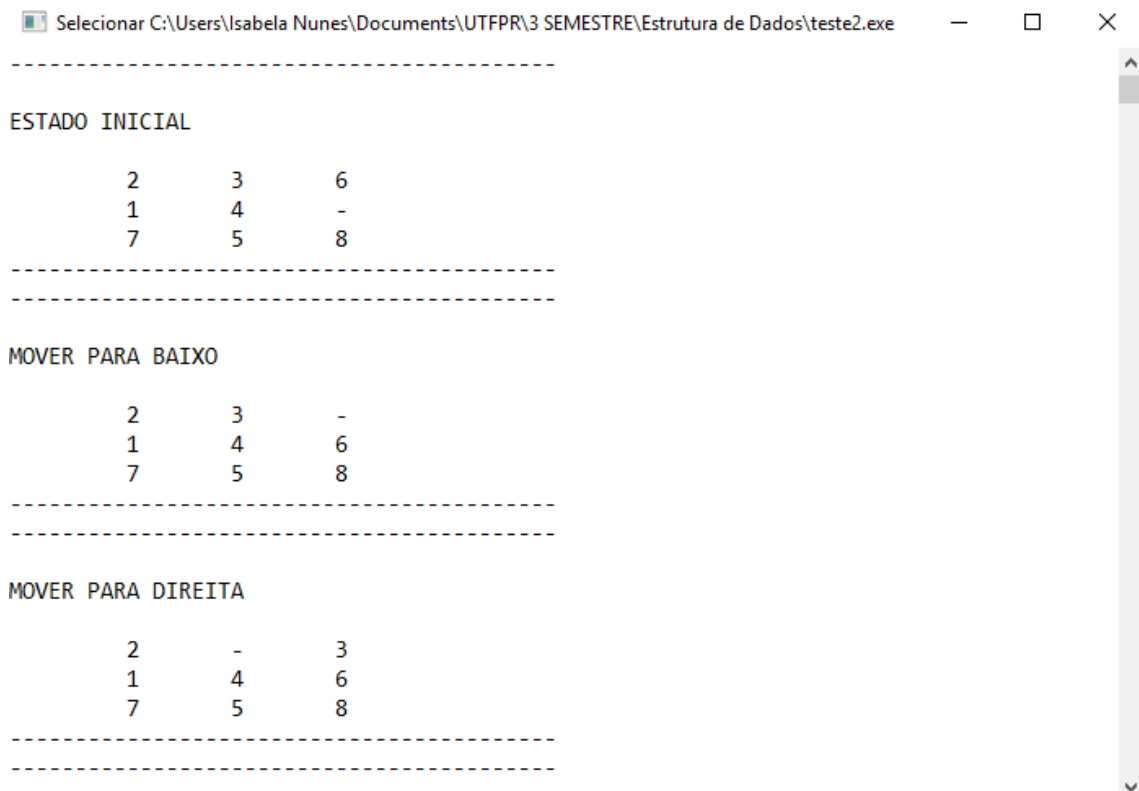


Figura 6 - Compilação do algoritmo (01)

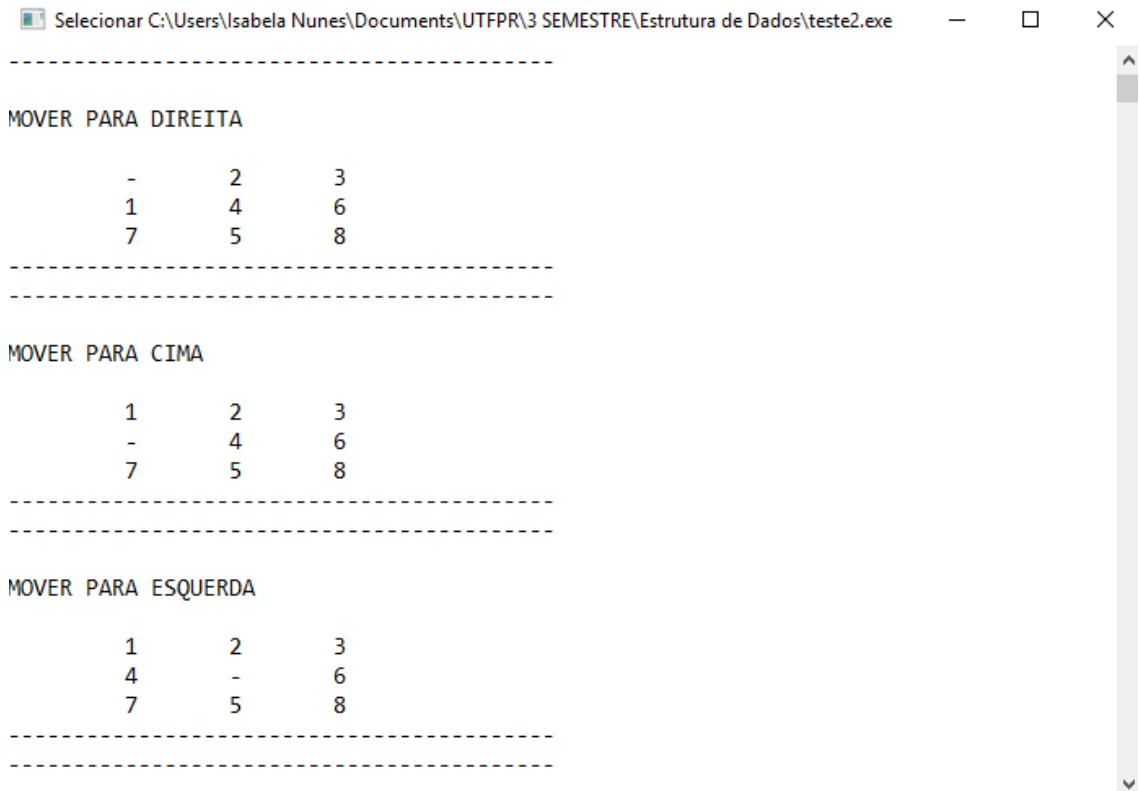


Figura 7 - Compilação do algoritmo (02)

```
Selecionar C:\Users\Isabela Nunes\Documents\UTFPR\3 SEMESTRE\Estrutura de Dados\teste2.exe

-----

MOVER PARA CIMA

    1      2      3
    4      5      6
    7      -      8
-----

MOVER PARA ESQUERDA

    1      2      3
    4      5      6
    7      8      -
-----

FIM

    1      2      3
    4      5      6
    7      8      -

Em 7 movimentos!

-----
Process exited after 0.1415 seconds with return value 18
Pressione qualquer tecla para continuar. . . █
```

Figura 8 - Compilação do algoritmo (03)

Como visto, o algoritmo seguiu a lógica esperada conforme a comparação das heurísticas entre as possibilidades de cada nível.

## 5 CONCLUSÃO

O N-Puzzle é um excelente jogo para utilização do algoritmo A\*, bem como aplicação de diferentes estruturas de dados. Outrossim, a complexidade do A\* depende diretamente da função heurística utilizada, além da efetividade quanto a tomada de decisões frente a diversas opções, levando em consideração a complexidade referente ao tempo.

Esta estratégia de busca pode ser comparada com a busca gulosa, que neste caso, poderia levar ao caminho mais longo (aumento na quantidade de passos). Neste âmbito, frequentemente o A\* é a única opção válida para um dado problema.

Dessa forma, para o melhor desempenho do algoritmo, foi utilizado uma lista simples para o armazenamento das heurísticas dos nós que ainda não foram abertos, sendo excluído esse valor da lista quando o mesmo fosse aberto e uma outra lista para guardar os nós que já foram abertos, sendo isto o mais viável para a lógica adotada.

Entretanto, ao longo do desenvolvimento do projeto, percebeu-se que poderíamos ter utilizado uma fila para tal armazenamento, sendo gravada, por questões de prioridade, a menor heurística sempre no início e sendo incrementada de forma ordenada, visando a economia de tempo ao invés de percorrer toda a lista toda vez que fosse procurar a menor heurística.

Para a escolha da utilização árvore foi levado em consideração o modo como o A\* funciona, o qual visita todos os nós antes da tomada de decisão, então, para um possível retorno, a árvore simplificaria o processo, pois com ela a partir da raiz pode-se saber a sequência das possibilidades visitando cada nível.

Em suma, a aplicação da estratégia de busca em questão se deu como satisfatória frente aos objetivos propostos, por meio das diversas estruturas de dados utilizadas e da heurística aplicada.

## REFERÊNCIAS

GAGNON, M. **Algoritmos básicos de busca**. École Polytechnique Montréal, 2000. Disponível em: <<http://www.professeurs.polymtl.ca/michel.gagnon/Disciplinas/Bac/IA/ResolProblema/resproblema.html>>. Acesso em: 06 jun. 2018.

LEE, H. D. **Estratégias de Busca**. Departamento Acadêmico de Informática - UTFPR/CT, [2000?]. Disponível em: <[http://www.dainf.ct.utfpr.edu.br/~fabro/IA\\_I/busca/IA\\_Estrategias\\_Busca\\_Inf.pdf](http://www.dainf.ct.utfpr.edu.br/~fabro/IA_I/busca/IA_Estrategias_Busca_Inf.pdf)>. Acesso em: 03 jun. 2018.

MONTESCO, C. A. E.; TACO, G. **Problema do Jogo das Oito Fichas (8 Puzzle) Fazendo uso da Busca Bidirecional**. Instituto de Ciências Matemáticas e de Computação, 1999. Disponível em: <[http://conteudo.icmc.usp.br/pessoas/sandra/G5\\_t2/8\\_Puzzle.htm](http://conteudo.icmc.usp.br/pessoas/sandra/G5_t2/8_Puzzle.htm)>. Acesso em: 01 jun. 2018.

RYAN, M. **Solvability of the Tiles Game**. The University of Birmingham, 2004. Disponível em: <<https://www.cs.bham.ac.uk/~mdr/teaching/modules04/java2/TilesSolvability.html>>. Acesso em: 11 jun. 2018.

SAITO, P. T. M. **Resolução de Problemas por Buscas - Buscas com informação**. Universidade Tecnológica Federal do Paraná. Cornélio Procópio, p. 44. 2018.