

# **TRABALHO DE INTELIGÊNCIA ARTIFICIAL**

Arthur Fernandes Ribeiro Costa – 11911BCC059

Gabriel Zeitoum dos Santos – 11911BCC021

Henrique Macarini de Paula - 11911BCC003

Vinícius Goulart de Farias Meres - 11811BCC009



**Todo os arquivos deste trabalho está no [GitHub](#)**

**Professora Dra. Gina M. B. de Oliveira**

## Sumário

Introdução .....	3
Experimentos .....	4
Busca Simples.....	4
Busca em Largura .....	4
Busca em Profundidade Iterativa .....	6
Busca de Menor Custo .....	8
Busca Informada .....	8
Busca A* .....	8
Busca Local .....	11
Subida de Encosta (Hill Climbing).....	11
Recristalização Simulada (Simulated Annealing) .....	14
Comparações dos algoritmos.....	15
Labirinto .....	15
Quebra-cabeça .....	18
Conclusão .....	18

## Introdução

O trabalho consiste na implementação de diversas buscas aprendidas em sala de aula para resolver dois problemas, que são eles: um labirinto e um quebra-cabeça. Ambos de tamanho  $N \times N$ . Para cada busca, um algoritmo apenas deve ser capaz de resolver os dois jogos.

O jogo do labirinto consiste em uma matriz formada pelos elementos 0, 1, 2 e 3, que representam lugares onde não podemos passar, lugares que podemos passar, posição inicial e destino de chegada, respectivamente. Devemos achar o melhor caminho possível da posição inicial ( $\text{matriz}[i][j] = 2$ ) até o destino ( $\text{matriz}[i][j] = 3$ ).

Já o jogo do quebra-cabeça consiste em uma matriz formada pelos elementos de 0 até  $(N \times N) - 1$ , onde  $N$  é a dimensão da matriz, ou seja, com  $N = 3$  teríamos uma matriz  $3 \times 3$  e seus elementos seriam de 0 até 8 ( $(3 \times 3) - 1$ ). Temos um estado inicial com os números fora de ordem e o estado final desse jogo é deixar os números em ordem, sendo que o elemento 0 deve estar na posição central da matriz, ou seja,  $\text{matriz}[N/2][N/2] = 0$ . Para melhor entendimento, veja as imagens abaixo:

2	0	0	0	1	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1
0	0	0	1	0	1	0	1	0	0
0	0	0	1	0	1	0	1	0	0
0	0	1	1	1	1	0	1	0	0
0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	1	0	0
0	0	1	3	1	1	1	1	0	0

Estado inicial jogo do labirinto. O objetivo é partir da célula que contém o 2 e chegar até a célula que contém o 3 passando apenas pelas células que contém 1's.

5	4	
6	1	8
7	3	2

Estado Inicial

1	2	3
8		4
7	6	5

Estado Meta

Jogo do quebra-cabeça com um tabuleiro onde  $N = 3$ , ou seja, uma matriz  $3 \times 3$ .

A linguagem escolhida para implementação desses algoritmos foi **C++** por ser uma extremamente rápida e também com vários recursos.

## Experimentos

### Observações:

- Todos os algoritmos foram testados **com os mesmos estados iniciais**
- As **saídas** para cada problema que serão mostradas nesse tópico são das entradas mostradas no exemplo na seção anterior
- Os algoritmos foram implementados no **Visual Studio Code**, porém foram executados no terminal devido alguns problemas para compilar na própria IDE
- Os tempos de execução foram medidos usando o comando **time** no terminal conforme a figura a seguir.

```
artfrc@DESKTOP-H59V9DE: /r x + v
artfrc@DESKTOP-H59V9DE:/mnt/c/Users/arthur/OneDrive/Área de Trabalho/UFU/IA/FinalWork/BuscaSimples/BuscaLargura$
time g++ buscaLargura.cpp -o a && ./a < ../Labirinto/inputs/tamanho10/01 > outputsLabirinto/tamanho10/01

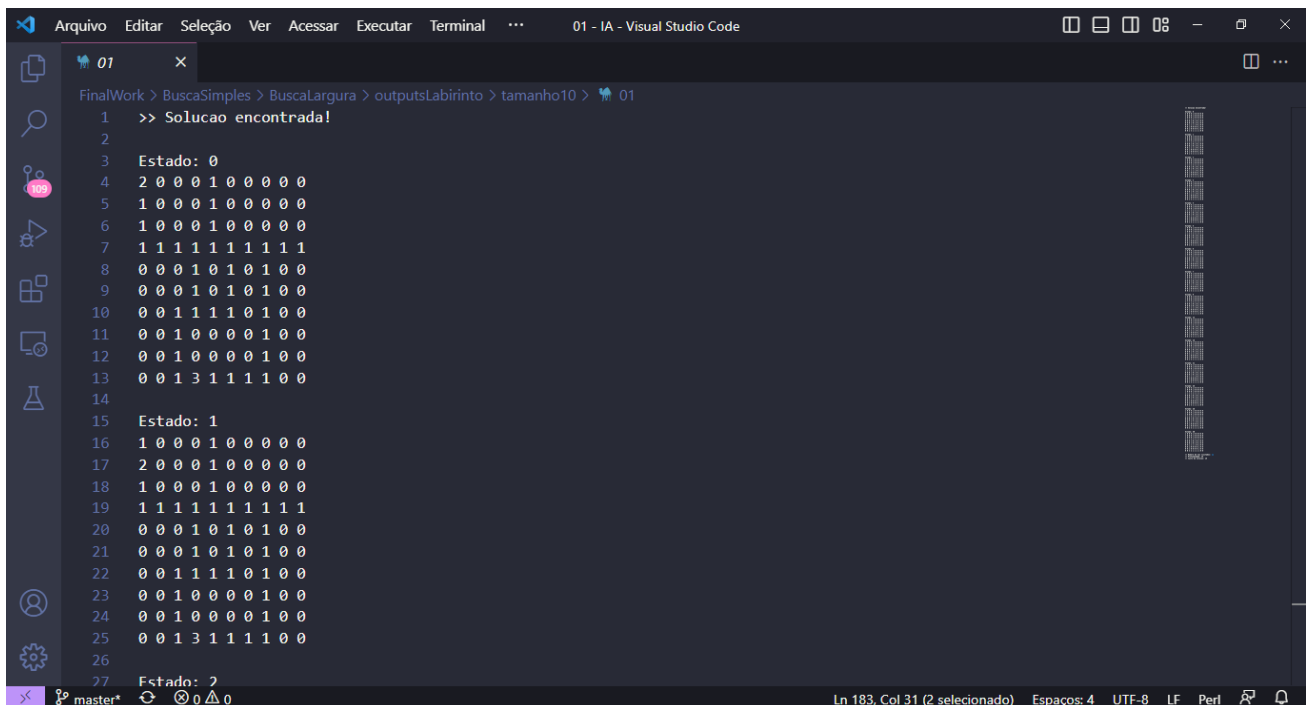
real    0m1.033s
user    0m0.688s
sys     0m0.328s
artfrc@DESKTOP-H59V9DE:/mnt/c/Users/arthur/OneDrive/Área de Trabalho/UFU/IA/FinalWork/BuscaSimples/BuscaLargura$
```

O comando acima executa o arquivo (buscaLargura.cpp), pega a entrada de dados que está no arquivo "01" (../Labirinto/inputs/tamanho10/01), escreve a saída no arquivo "01" (outputsLabirinto/tamanho10/01) e por fim, calcula o tempo de execução. O tempo usado foi **real**.

## Busca Simples

### Busca em Largura

Este com certeza foi o algoritmo mais fácil de implementar devido ser um algoritmo mais famoso e bastante usado para resolução de diversos problemas em maratonas de programação, por exemplo.



```
01
FinalWork > BuscaSimples > BuscaLargura > outputsLabirinto > tamanho10 > 01
1  >> Solucao encontrada!
2
3  Estado: 0
4  2 0 0 0 1 0 0 0 0 0
5  1 0 0 0 1 0 0 0 0 0
6  1 0 0 0 1 0 0 0 0 0
7  1 1 1 1 1 1 1 1 1 1
8  0 0 0 1 0 1 0 1 0 0
9  0 0 0 1 0 1 0 1 0 0
10 0 0 1 1 1 1 0 1 0 0
11 0 0 1 0 0 0 0 1 0 0
12 0 0 1 0 0 0 0 1 0 0
13 0 0 1 3 1 1 1 1 0 0
14
15 Estado: 1
16 1 0 0 0 1 0 0 0 0 0
17 2 0 0 0 1 0 0 0 0 0
18 1 0 0 0 1 0 0 0 0 0
19 1 1 1 1 1 1 1 1 1 1
20 0 0 0 1 0 1 0 1 0 0
21 0 0 0 1 0 1 0 1 0 0
22 0 0 1 1 1 1 0 1 0 0
23 0 0 1 0 0 0 0 1 0 0
24 0 0 1 0 0 0 0 1 0 0
25 0 0 1 3 1 1 1 1 0 0
26
27 Estado: 2
```

```
Arquivo  Editar  Seleção  Ver  Acessar  Executar  Terminal  Ajuda  01 - IA - Visual Studio Code

FinalWork > BuscaSimples > BuscaLargura > outputsLabirinto > tamanho10 > 01
160 1 0 0 0 1 0 0 0 0 0
161 1 0 0 0 1 0 0 0 0 0
162 1 0 0 0 1 0 0 0 0 0
163 1 1 1 1 1 1 1 1 1 1
164 0 0 0 1 0 1 0 1 0 0
165 0 0 0 1 0 1 0 1 0 0
166 0 0 1 1 1 1 0 1 0 0
167 0 0 1 0 0 0 0 1 0 0
168 0 0 1 0 0 0 0 1 0 0
169 0 0 2 3 1 1 1 1 0 0
170
171 Estado: 30
172 1 0 0 0 1 0 0 0 0 0
173 1 0 0 0 1 0 0 0 0 0
174 1 0 0 0 1 0 0 0 0 0
175 1 1 1 1 1 1 1 1 1 1
176 0 0 0 1 0 1 0 1 0 0
177 0 0 0 1 0 1 0 1 0 0
178 0 0 1 1 1 1 0 1 0 0
179 0 0 1 0 0 0 0 1 0 0
180 0 0 1 0 0 0 0 1 0 0
181 0 0 3 2 1 1 1 1 0 0
182
183 >> Numero de nos visitados: 32
184 >> Profundidade: 14
185 >> Custo da solucao: 15
186

Ln 186, Col 1  Espaços: 4  UTF-8  LF  Perl
```

Saída para labirinto 10x10

```
Arquivo  Editar  Seleção  Ver  Acessar  Executar  Terminal  ...  tamanho3 - IA - Visual Studio Code

tamanho3
FinalWork > BuscaSimples > BuscaLargura > outputsQuebraCabeca > tamanho3
1 >> Solucao encontrada!
2
3 Estado: 0
4 5 4 0
5 6 1 8
6 7 3 2
7
8 Estado: 2
9 5 0 4
10 6 1 8
11 7 3 2
12
13 Estado: 5
14 5 1 4
15 6 0 8
16 7 3 2
17
18 Estado: 11
19 5 1 4
20 6 3 8
21 7 0 2
22
23 Estado: 24
24 5 1 4
25 6 3 8
26 7 2 0
27

Ln 1, Col 1  Espaços: 4  UTF-8  LF  Texto sem Formatação
```

```
FinalWork > BuscaSimples > BuscaLargura > outputsQuebraCabeca > tamanho3
97
98 Estado: 33976
99 2 0 3
100 1 4 5
101 6 7 8
102
103 Estado: 49058
104 0 2 3
105 1 4 5
106 6 7 8
107
108 Estado: 66277
109 1 2 3
110 0 4 5
111 6 7 8
112
113 Estado: 88542
114 1 2 3
115 4 0 5
116 6 7 8
117
118 >> Numero de nos visitados: 110368
119 >> Profundidade: 22
120 >> Custo da solucao: 23
121
```

Saída para quebra-cabeça 3x3

### Busca em Profundidade Iterativa

Este também foi fácil de implementar pelo fato de a busca em profundidade também ser um algoritmo muito famoso, bastante usado também na resolução de muitos problemas, então precisou fazer apenas alguns ajustes para tornar uma busca iterativa, ou seja, limitar a profundidade máxima e ir aumentando gradativamente conforme o algoritmo vai fazendo a busca pela solução.

```
Arquivo  Editar  Seleção  Ver  Acessar  Executar  Terminal  Ajuda  01 - IA - Visual Studio Code
01
1 >> Solucao encontrada!
2
3 Estado: 0
4 2 0 0 0 1 0 0 0 0 0
5 1 0 0 0 1 0 0 0 0 0
6 1 0 0 0 1 0 0 0 0 0
7 1 1 1 1 1 1 1 1 1 1
8 0 0 0 1 0 1 0 1 0 0
9 0 0 0 1 0 1 0 1 0 0
10 0 0 1 1 1 1 0 1 0 0
11 0 0 1 0 0 0 0 1 0 0
12 0 0 1 0 0 0 0 1 0 0
13 0 0 1 3 1 1 1 1 0 0
14
15 Estado: 1
16 1 0 0 0 1 0 0 0 0 0
17 2 0 0 0 1 0 0 0 0 0
18 1 0 0 0 1 0 0 0 0 0
19 1 1 1 1 1 1 1 1 1 1
20 0 0 0 1 0 1 0 1 0 0
21 0 0 0 1 0 1 0 1 0 0
22 0 0 1 1 1 1 0 1 0 0
23 0 0 1 0 0 0 0 1 0 0
24 0 0 1 0 0 0 0 1 0 0
25 0 0 1 3 1 1 1 1 0 0
26
27 Estado: 2
```

```
Arquivo  Editar  Seleção  Ver  Acessar  Executar  Terminal  Ajuda  01 - IA - Visual Studio Code

FinalWork > BuscaSimples > BuscaProfiterativa > outputsLabirinto > tamanho10 > 01
160 1 0 0 0 1 0 0 0 0 0
161 1 0 0 0 1 0 0 0 0 0
162 1 0 0 0 1 0 0 0 0 0
163 1 1 1 1 1 1 1 1 1 1
164 0 0 0 1 0 1 0 1 0 0
165 0 0 0 1 0 1 0 1 0 0
166 0 0 1 1 1 1 0 1 0 0
167 0 0 1 0 0 0 0 1 0 0
168 0 0 1 0 0 0 0 1 0 0
169 0 0 2 3 1 1 1 1 0 0
170
171 Estado: 14
172 1 0 0 0 1 0 0 0 0 0
173 1 0 0 0 1 0 0 0 0 0
174 1 0 0 0 1 0 0 0 0 0
175 1 1 1 1 1 1 1 1 1 1
176 0 0 0 1 0 1 0 1 0 0
177 0 0 0 1 0 1 0 1 0 0
178 0 0 1 1 1 1 0 1 0 0
179 0 0 1 0 0 0 0 1 0 0
180 0 0 1 0 0 0 0 1 0 0
181 0 0 3 2 1 1 1 1 0 0
182
183 >> Numero de nos visitados: 24
184 >> Profundidade: 14
185 >> Custo da solucao: 15
186

master*
```

Saída para labirinto 10x10

```
Arquivo  Editar  Seleção  Ver  Acessar  Executar  Terminal  Ajuda  tamanho3 - IA - Visual Studio Code

tamanho3
FinalWork > BuscaSimples > BuscaProfiterativa > outputsQuebraCabeca > tamanho3
1 >> Solucao encontrada!
2
3 Estado: 0
4 5 4 0
5 6 1 8
6 7 3 2
7
8 Estado: 38356
9 5 0 4
10 6 1 8
11 7 3 2
12
13 Estado: 38357
14 5 1 4
15 6 0 8
16 7 3 2
17
18 Estado: 53282
19 5 1 4
20 6 8 0
21 7 3 2
22
23 Estado: 58071
24 5 1 4
25 6 8 2
26 7 3 0
27

master*
```

```
FinalWork > BuscaSimples > BuscaProfiterativa > outputsQuebraCabeca > tamanho3
136 6 8 5
137
138 Estado: 60852
139 1 2 3
140 4 7 0
141 6 8 5
142
143 Estado: 60853
144 1 2 3
145 4 7 5
146 6 8 0
147
148 Estado: 60854
149 1 2 3
150 4 7 5
151 6 0 8
152
153 Estado: 60855
154 1 2 3
155 4 0 5
156 6 7 8
157
158 >> Numero de nos visitados: 61394
159 >> Profundidade: 30
160 >> Custo da solucao: 31
161
```

Saída para quebra-cabeça 3x3

## Busca de Menor Custo

Não foi necessário fazer um algoritmo específico para essa busca, pois devido os custos para todos os vizinhos de cada estado sempre custar 1 (lado direito, esquerdo, cima e baixo na matriz), então acaba caindo em uma busca em largura.

## Busca Informada

### Busca A\*

Esse já foi um algoritmo que teve mais dificuldade para entender como ele funcionava, mas após compreendê-lo, então foi fácil sua implementação.

```
FinalWork > BuscaInformada > outputsLabirinto > tamanho10 > 01
1 >> Solucao encontrada!
2
3 Estado: 0
4 2 0 0 0 1 0 0 0 0
5 1 0 0 0 1 0 0 0 0
6 1 0 0 0 1 0 0 0 0
7 1 1 1 1 1 1 1 1 1
8 0 0 0 1 0 1 0 1 0
9 0 0 0 1 0 1 0 1 0
10 0 0 1 1 1 1 0 1 0
11 0 0 1 0 0 0 0 1 0
12 0 0 1 0 0 0 0 1 0
13 0 0 1 3 1 1 1 1 0
14
15 Estado: 1
16 1 0 0 0 1 0 0 0 0
17 2 0 0 0 1 0 0 0 0
18 1 0 0 0 1 0 0 0 0
19 1 1 1 1 1 1 1 1 1
20 0 0 0 1 0 1 0 1 0
21 0 0 0 1 0 1 0 1 0
22 0 0 1 1 1 1 0 1 0
23 0 0 1 0 0 0 0 1 0
24 0 0 1 0 0 0 0 1 0
25 0 0 1 3 1 1 1 1 0
26
27 Estado: 2
```



```
Arquivo Editar Seleção Ver Acessar Executar Terminal Ajuda 01 - IA - Visual Studio Code
01 x
FinalWork > BuscaInformada > outputsLabirinto > tamanho10 > 01
159 Estado: 24
160 1 0 0 0 1 0 0 0 0 0
161 1 0 0 0 1 0 0 0 0 0
162 1 0 0 0 1 0 0 0 0 0
163 1 1 1 1 1 1 1 1 1 1
164 0 0 0 1 0 1 0 1 0 0
165 0 0 0 1 0 1 0 1 0 0
166 0 0 1 1 1 1 0 1 0 0
167 0 0 1 0 0 0 0 1 0 0
168 0 0 1 0 0 0 0 1 0 0
169 0 0 2 3 1 1 1 1 0 0
170
171 Estado: 25
172 1 0 0 0 1 0 0 0 0 0
173 1 0 0 0 1 0 0 0 0 0
174 1 0 0 0 1 0 0 0 0 0
175 1 1 1 1 1 1 1 1 1 1
176 0 0 0 1 0 1 0 1 0 0
177 0 0 0 1 0 1 0 1 0 0
178 0 0 1 1 1 1 0 1 0 0
179 0 0 1 0 0 0 0 1 0 0
180 0 0 1 0 0 0 0 1 0 0
181 0 0 3 2 1 1 1 1 0 0
182
183 >> Numero de nos visitados: 38
184 >> Profundidade: 14
185 >> Custo da solucao: 15
Ln 1, Col 1 Espaços: 4 UTF-8 LF Perl
```

Saída para labirinto 10x10

```
Arquivo Editar Seleção Ver Acessar Executar Terminal Ajuda tamanho31 - IA - Visual Studio Code
tamanho31 x
FinalWork > BuscaInformada > outputsQuebraCabeca > tamanho31
1 >> Solucao encontrada!
2
3 Estado: 0
4 5 4 0
5 6 1 8
6 7 3 2
7
8 Estado: 2
9 5 0 4
10 6 1 8
11 7 3 2
12
13 Estado: 5
14 5 1 4
15 6 0 8
16 7 3 2
17
18 Estado: 10
19 5 1 4
20 6 3 8
21 7 0 2
22
23 Estado: 20
24 5 1 4
25 6 3 8
26 7 2 0
27
Ln 1, Col 1 Espaços: 4 UTF-8 LF Texto sem Formatação
```

```
Arquivo Editar Seleção Ver Acessar Executar ... tamanho31 - IA - Visual Studio Code
tamanho31 x
FinalWork > BuscaInformada > outputsQuebraCabeca > tamanho31
96 6 7 8
97
98 Estado: 7864
99 2 0 3
100 1 4 5
101 6 7 8
102
103 Estado: 9956
104 0 2 3
105 1 4 5
106 6 7 8
107
108 Estado: 12118
109 1 2 3
110 0 4 5
111 6 7 8
112
113 Estado: 14529
114 1 2 3
115 4 0 5
116 6 7 8
117
118 >> Numero de nos visitados: 181440
119 >> Profundidade: 22
120 >> Custo da solucao: 23
121
```

Saída para quebra-cabeça 3x3 usando heurística de número de peças na posição errada

```
Arquivo Editar Seleção Ver Acessar Executar ... tamanho32 - IA - Visual Studio Code
tamanho32 x
FinalWork > BuscaInformada > outputsQuebraCabeca > tamanho32
1 >> Solucao encontrada!
2
3 Estado: 0
4 5 4 0
5 6 1 8
6 7 3 2
7
8 Estado: 2
9 5 0 4
10 6 1 8
11 7 3 2
12
13 Estado: 5
14 5 1 4
15 6 0 8
16 7 3 2
17
18 Estado: 7
19 5 1 4
20 6 3 8
21 7 0 2
22
23 Estado: 16
24 5 1 4
25 6 3 8
26 7 2 0
27
```

```
Arquivo  Editar  Seleção  Ver  Acessar  Executar  ...  tamanho32 - IA - Visual Studio Code

FinalWork > BuscaInformada > outputsQuebraCabeca > tamanho32
98 Estado: 2705
99 2 0 3
100 1 4 5
101 6 7 8
102
103 Estado: 3391
104 0 2 3
105 1 4 5
106 6 7 8
107
108 Estado: 3936
109 1 2 3
110 0 4 5
111 6 7 8
112
113 Estado: 4283
114 1 2 3
115 4 0 5
116 6 7 8
117
118 >> Numero de nos visitados: 181440
119 >> Profundidade: 22
120 >> Custo da solucao: 23
121
```

Saída para quebra-cabeça 3x3 usando heurística da soma das distâncias até a posição correta

## Busca Local

### Subida de Encosta (Hill Climbing)

Não compreendemos exatamente como implementar essa busca, mas seguimos toda a ideia descrita no slide. Começamos com um estado inicial, procuramos todos os vizinhos possíveis desse estado e caso exista algum estado que esteja mais próximo da solução, então fazemos a busca novamente com esse novo estado. Se não existir um vizinho que esteja mais próximo da solução, então geramos um estado aleatório e verificamos a partir dele refazemos todo o processo até chegar ou não na solução.

Foi estabelecido um limite para essa busca, pois chega em um determinado momento que fica muito difícil o algoritmo achar um estado aleatoriamente que não tenha sido visitado. Então caso o algoritmo gere X estados aleatórios e todos eles já foram visitados, então o algoritmo encerra e devolve o último estado encontrado antes de tentar buscar um novo estado aleatório.

```
Arquivo  Editar  Seleção  Ver  Acessar  Executar  ...  01 - IA - Visual Studio Code

FinalWork > BuscaLocal > SubidaEncosta > outputsLabirinto > tamanho10 > 01

1  >> Solucao encontrada:
2
3  Estado: 0
4  2 0 0 0 1 0 0 0 0 0
5  1 0 0 0 1 0 0 0 0 0
6  1 0 0 0 1 0 0 0 0 0
7  1 1 1 1 1 1 1 1 1 1
8  0 0 0 1 0 1 0 1 0 0
9  0 0 0 1 0 1 0 1 0 0
10 0 0 1 1 1 1 0 1 0 0
11 0 0 1 0 0 0 0 1 0 0
12 0 0 1 0 0 0 0 1 0 0
13 0 0 1 3 1 1 1 1 0 0
14
15 Estado: 1
16 1 0 0 0 1 0 0 0 0 0
17 2 0 0 0 1 0 0 0 0 0
18 1 0 0 0 1 0 0 0 0 0
19 1 1 1 1 1 1 1 1 1 1
20 0 0 0 1 0 1 0 1 0 0
21 0 0 0 1 0 1 0 1 0 0
22 0 0 1 1 1 1 0 1 0 0
23 0 0 1 0 0 0 0 1 0 0
24 0 0 1 0 0 0 0 1 0 0
25 0 0 1 3 1 1 1 1 0 0
26
27 Estado: 2
28
```

```
Arquivo  Editar  Seleção  Ver  Acessar  Executar  Terminal  Ajuda  01 - IA - Visual Studio Code

FinalWork > BuscaLocal > SubidaEncosta > outputsLabirinto > tamanho10 > 01

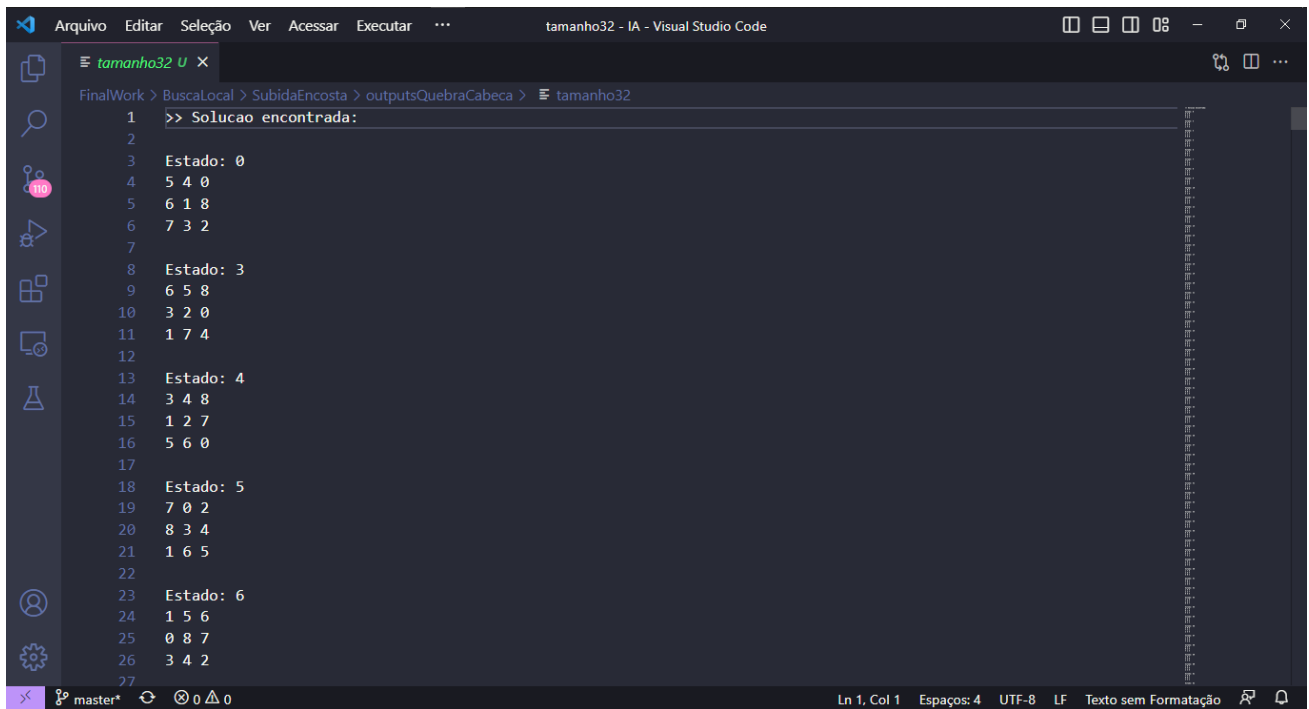
208 1 0 0 0 1 0 0 0 0 0
209 1 0 0 0 1 0 0 0 0 0
210 1 0 0 0 1 0 0 0 0 0
211 1 1 1 2 1 1 1 1 1 1
212 0 0 0 1 0 1 0 1 0 0
213 0 0 0 1 0 1 0 1 0 0
214 0 0 1 1 1 1 0 1 0 0
215 0 0 1 0 0 0 0 1 0 0
216 0 0 1 0 0 0 0 1 0 0
217 0 0 1 3 2 1 1 1 0 0
218
219 Estado: 22
220 1 0 0 0 1 0 0 0 0 0
221 1 0 0 0 1 0 0 0 0 0
222 1 0 0 0 1 0 0 0 0 0
223 1 1 1 2 1 1 1 1 1 1
224 0 0 0 1 0 1 0 1 0 0
225 0 0 0 1 0 1 0 1 0 0
226 0 0 1 1 1 1 0 1 0 0
227 0 0 1 0 0 0 0 1 0 0
228 0 0 1 0 0 0 0 1 0 0
229 0 0 1 2 3 1 1 1 0 0
230
231 >> Numero de nos visitados: 24
232 >> Custo da solucao: 19
233
```

Saída para labirinto 10x10

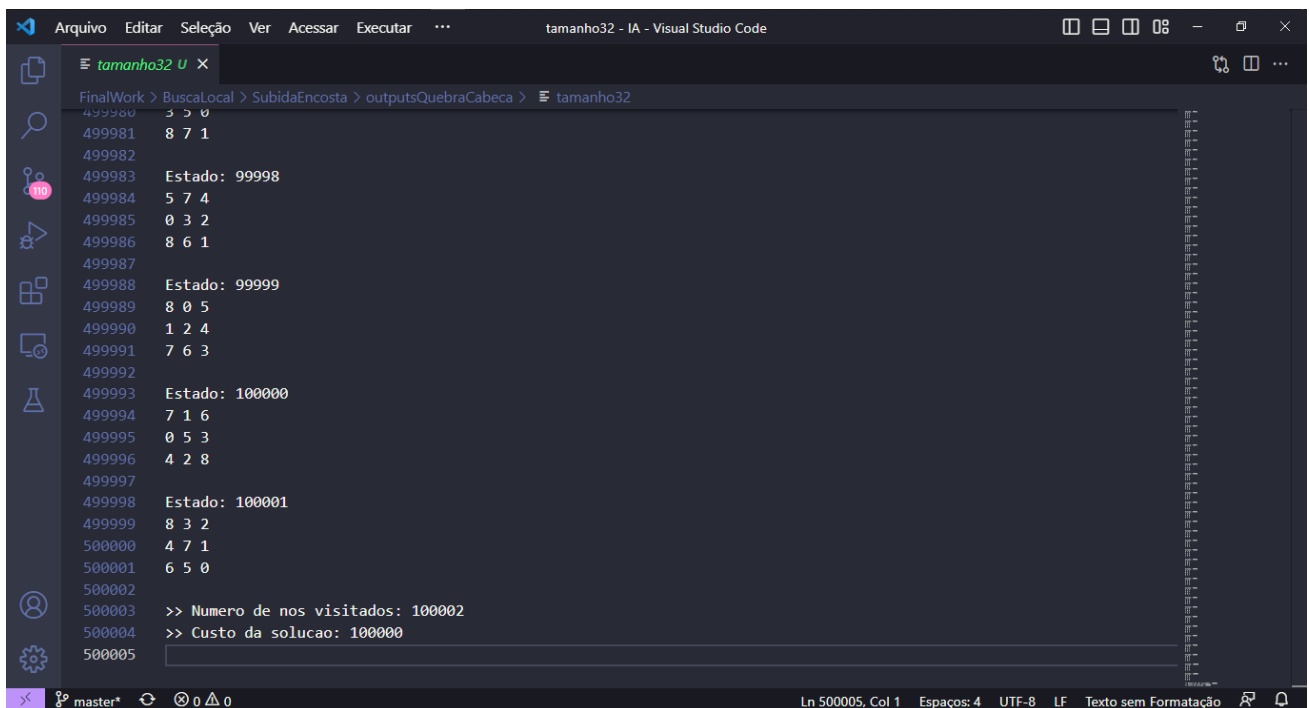
```
tamanho31 U x
FinalWork > BuscaLocal > SubidaEncosta > outputsQuebraCabeca > tamanho31
1  >> Solucao encontrada:
2
3  Estado: 0
4  5 4 0
5  6 1 8
6  7 3 2
7
8  Estado: 3
9  2 7 1
10 4 5 0
11 8 6 3
12
13 Estado: 4
14 3 0 4
15 2 5 8
16 6 7 1
17
18 Estado: 5
19 7 2 8
20 1 5 3
21 0 4 6
22
23 Estado: 6
24 4 8 3
25 1 0 2
26 6 7 5
27
```

```
Arquivo Editar Seleção Ver Acessar Executar ... tamanho31 - IA - Visual Studio Code
tamanho31 U x
FinalWork > BuscaLocal > SubidaEncosta > outputsQuebraCabeca > tamanho31
499979 5 7 3
499980 4 2 8
499981 1 0 6
499982
499983 Estado: 99998
499984 2 6 3
499985 1 0 8
499986 7 4 5
499987
499988 Estado: 99999
499989 0 6 2
499990 7 8 3
499991 5 4 1
499992
499993 Estado: 100000
499994 2 7 0
499995 1 6 8
499996 5 4 3
499997
499998 Estado: 100001
499999 2 8 5
500000 6 3 4
500001 0 1 7
500002
500003 >> Numero de nos visitados: 100002
500004 >> Custo da solucao: 100000
500005
```

Saída para quebra-cabeça 3x3 usando heurística de número de peças na posição correta



```
FinalWork > BuscaLocal > SubidaEncosta > outputsQuebraCabeca > tamanho32
1  >> Solucao encontrada:
2
3  Estado: 0
4  5 4 0
5  6 1 8
6  7 3 2
7
8  Estado: 3
9  6 5 8
10 3 2 0
11 1 7 4
12
13 Estado: 4
14 3 4 8
15 1 2 7
16 5 6 0
17
18 Estado: 5
19 7 0 2
20 8 3 4
21 1 6 5
22
23 Estado: 6
24 1 5 6
25 0 8 7
26 3 4 2
27
```



```
FinalWork > BuscaLocal > SubidaEncosta > outputsQuebraCabeca > tamanho32
499999 3 5 0
499981 8 7 1
499982
499983 Estado: 99998
499984 5 7 4
499985 0 3 2
499986 8 6 1
499987
499988 Estado: 99999
499989 8 0 5
499990 1 2 4
499991 7 6 3
499992
499993 Estado: 100000
499994 7 1 6
499995 0 5 3
499996 4 2 8
499997
499998 Estado: 100001
499999 8 3 2
500000 4 7 1
500001 6 5 0
500002
500003 >> Numero de nos visitados: 100002
500004 >> Custo da solucao: 100000
500005
```

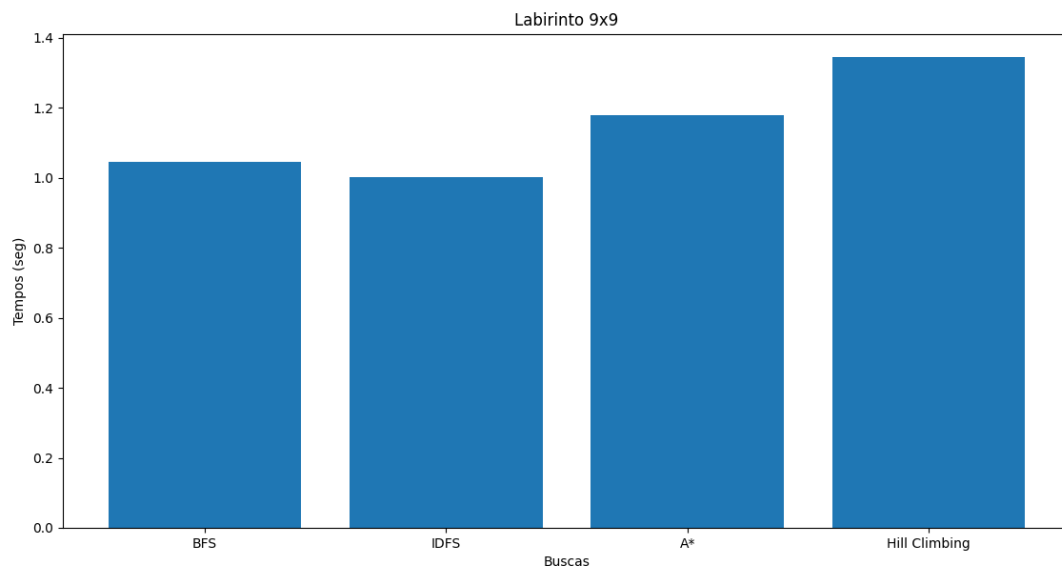
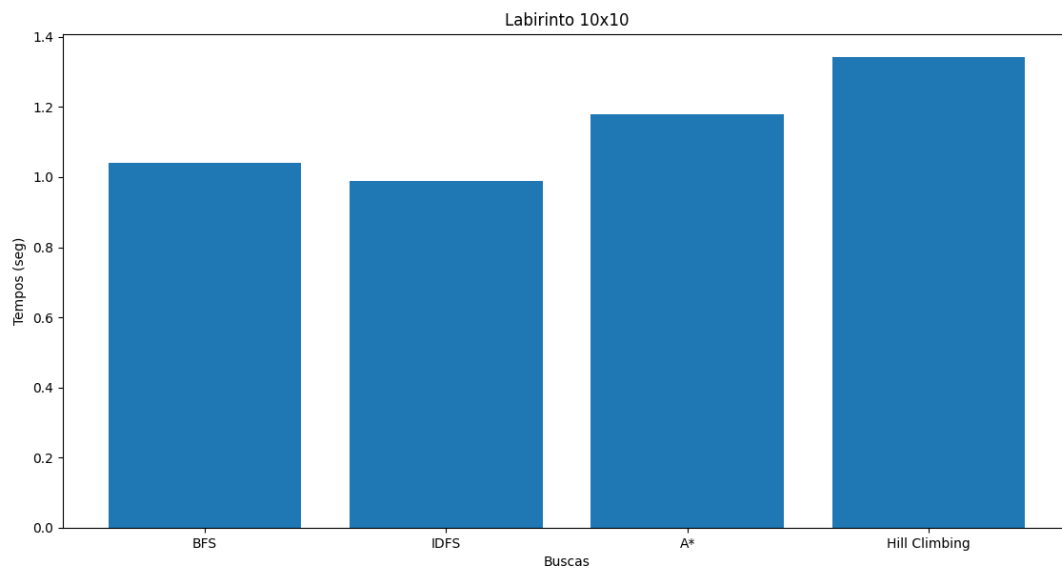
Saída para quebra-cabeça 3x3 usando heurística 50 - soma das distâncias até a posição correta (100 - ... para tabuleiro com 15 peças)

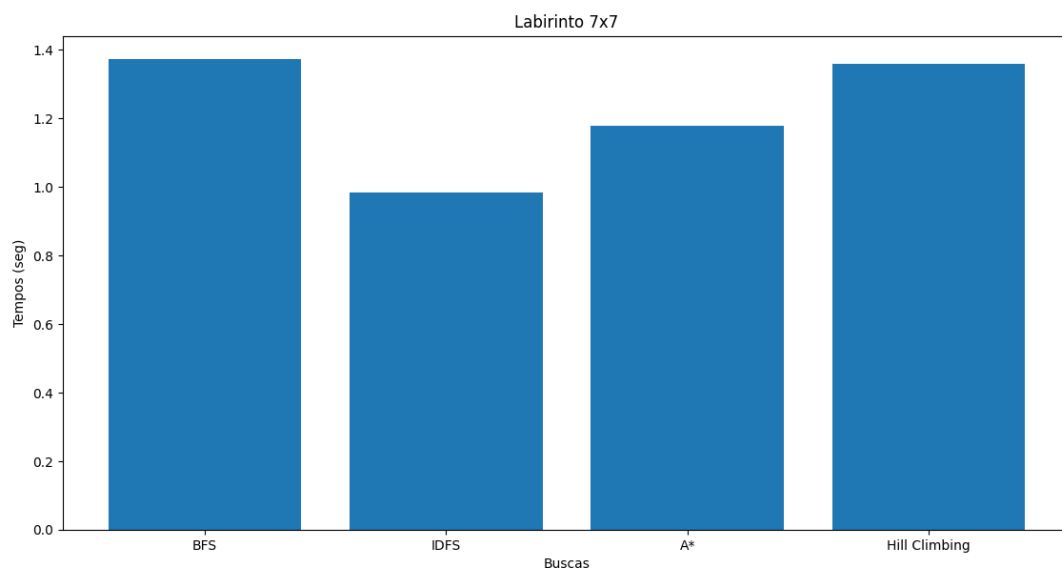
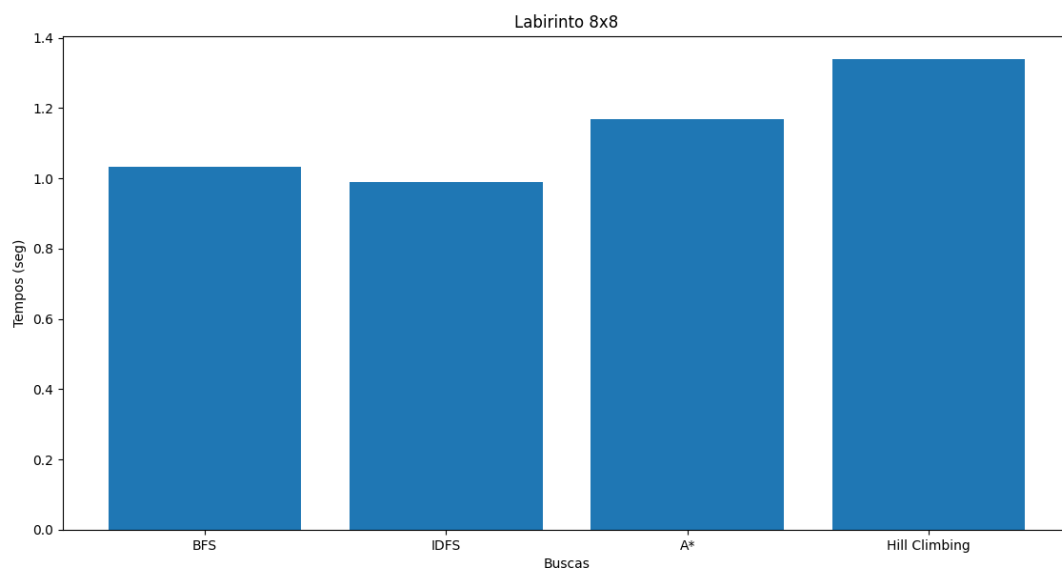
### Recristalização Simulada (Simulated Annealing)

Esse foi feito um algoritmo similar ao de Hill Climbing, mas já aceitamos que não está correto e devido isso não fizemos testes para ele.

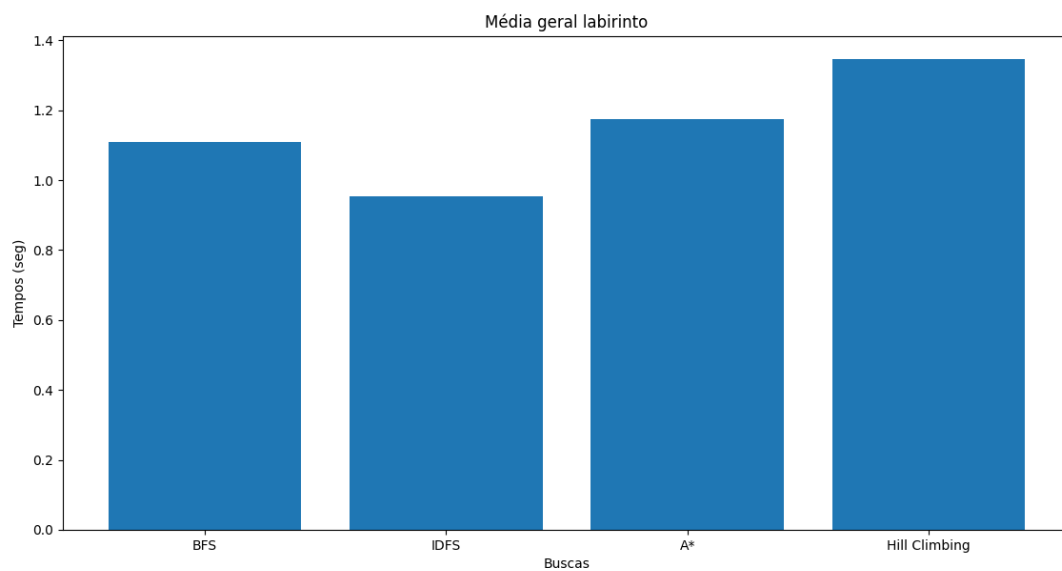
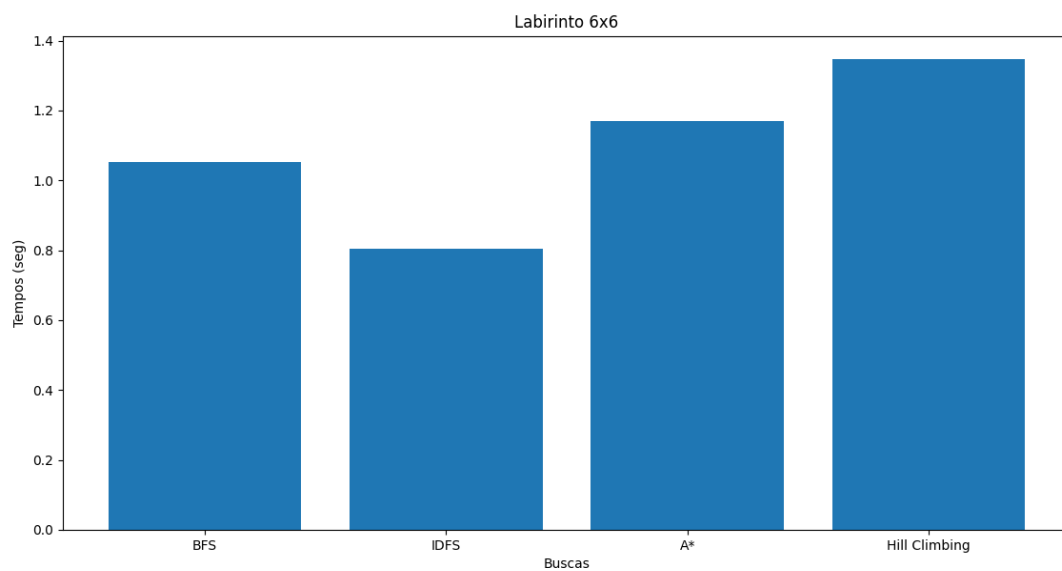
## Comparações dos algoritmos

### Labirinto

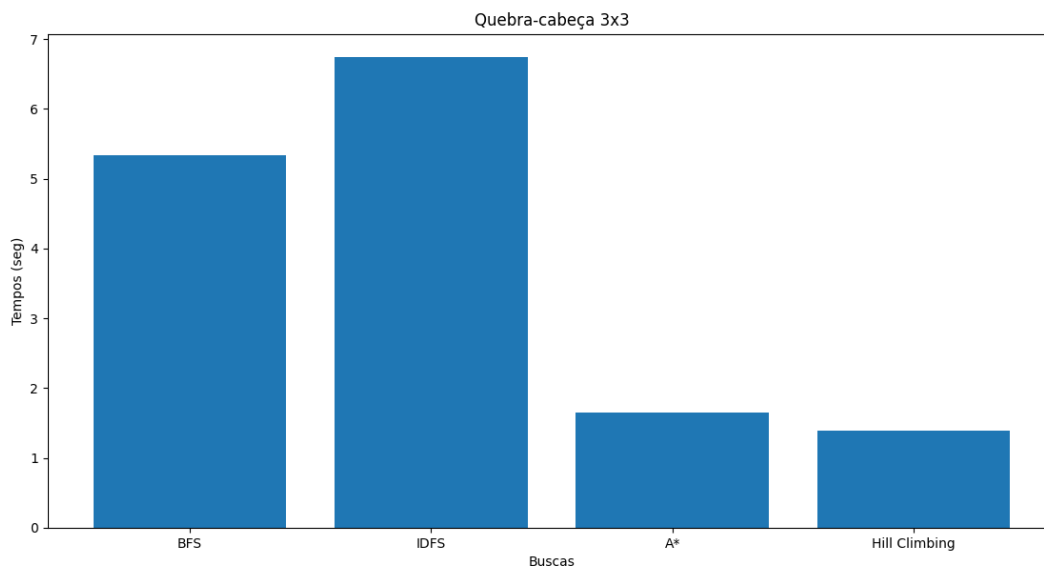








## Quebra-cabeça



## Conclusão

Analisando os gráficos da seção anterior, podemos notar que no jogo do **labirinto** o algoritmo que foi **mais** eficiente foi **busca em profundidade iterativa (IDFS)** e o **menos** eficiente foi **Hill Climbing**.

Já no jogo do **quebra-cabeça** por mais incrível que pareça, foi totalmente o contrário. O **mais** eficiente foi **Hill Climbing** e o **menos** eficiente foi a **busca em profundidade iterativa**. Temos que levar em consideração que Hill Climbing **não achou a solução correta**. Logo, o algoritmo mais eficiente com **solução correta** foi **busca A\***.

Como mencionado anteriormente, é bastante provável que contenha erros nos algoritmos, principalmente no de busca local.

As maiores dificuldades desse trabalho foram: pensar em um modo de fazer apenas um algoritmo resolver os dois problemas e também compreender como funcionava os algoritmos de busca local.