**UNIVERSIDADE ESTADUAL DE MARINGÁ**

**DEPARTAMENTO DE INFORMÁTICA**

**CIÊNCIA DA COMPUTAÇÃO**

**MODELAGEM E OTIMIZAÇÃO ALGORITMICA**

**PROFº.: ADEMIR APARECIDO CONSTANTINO**

**ANÁLISE DO COMPORTAMENTO DAS HEURÍSTICAS DO ALGORITMO A\* APLICADAS NA RESOLUÇÃO DO JOGO DE TABULEIRO DE 15 PEÇAS**

JOÃO OTÁVIO BIONDO RA: 81746

MARCO AURÉLIO DEOLDOTO PAULINO RA: 82473

**MARINGÁ**

**11 DE JANEIRO DE 2016**

1. **Introdução**

O puzzle 15-puzzle também conhecido, em inglês, como, Boss Puzzle, Game of fifteen e Mystic Square, em português, o jogo é mais conhecido como jogo do 15.

O jogo dos 15 é um puzzle de deslocamento com o objetivo de ordenar em forma crescente os números de 1 a 15 em tabuleiro com largura e comprimento igual a quatro, ou seja, uma matriz quadrática de ordem quatro, a ordenação pode ser de forma horizontal ou vertical ou em formas mais exóticas, como em caracol. Também existe o 8-puzzle, ou se preferir, jogo dos 8, que nada mais é uma versão para ordenar os números de 1 a 8 em uma matriz quadrática de ordem três.

Esse modelo de puzzle pode ser considerado com um problema clássico de modelagem algorítmica envolvendo heurística. Algumas heurísticas mais populares neste contexto são os números de peças fora de seu lugar final e a soma distância de cada peça de seu lugar final.

O objetivo deste trabalho é implementar o algoritmo A-estrela para obter o menor custo para chegar no estado final do puzzle a partir de um estado inicial dado.

O algoritmo A foi descrito pela primeira vez em 1968 por Peter Hart, Nils Nilsson e Bertram Raphael. Utilizando o algoritmo A com uma heurística apropriada obtém-se um comportamento ótimo que é o algoritmo A-estrela.

Para ter-se uma heurística apropriada e assim um comportamento ótimo é preciso obedecer a seguinte regra:

h’(n) ≤ h(n)

Onde h’(n) é a estimativa do custo do menor caminho entre o estado n e o estado final, chamada de *função heurística* e h(n) é o custo do menor do caminho entre o estado n e o estado final.

Para o presente trabalho foi empregado as seguintes 5 heurísticas:

1. Número de peças fora de seu lugar na configuração final;
2. Número de peças fora de ordem na sequência numérica das 15 pelas, seguindo a ordem das posições no tabuleiro;
3. Para cada peça fora de seu lugar é soma a sua distância retangular para alcançar o seu lugar final.
4. A soma com pesos das três primeiras heurísticas, em que os pesos deve ser igual a 1.
5. Escolher a heurística de maior resultado entre as heurísticas 1, 2 e 3.

As linguagens de programação utilizadas foram C++ e Java, assim sendo os capítulos estão organizados da seguinte forma. No capítulo 1, é explicado o algoritmo A-estrela implementado em Java e seus respectivos resultados. No capítulo 2, é equivalente ao capítulo 1 para o algoritmo A-estrela em C++. Na conclusão é apresentada a comparação de resultados entre os dois algoritmos.

1. **Algoritmo em Java**
2. **package** main;
4. **import** java.io.BufferedReader;
5. **import** java.io.FileInputStream;
6. **import** java.io.FileNotFoundException;
7. **import** java.io.IOException;
8. **import** java.io.InputStreamReader;
9. **import** java.util.ArrayList;
10. **import** java.util.HashMap;
11. **import** java.util.TreeMap;
13. **class** Main {
15. **public** **static** Puzzle buscaMenor(ArrayList<Puzzle> lista){
16. **int** compara = Integer.MAX\_VALUE;
17. Puzzle puzzleRetorno = **null**;
18. **for**(Puzzle puzzle : lista){
19. **if**(puzzle.custoTotal < compara){
20. compara = puzzle.custoTotal;
21. puzzleRetorno = puzzle;
22. }
23. }
24. **return** puzzleRetorno;
25. }
27. **public** **static** **class** Puzzle{
28. **int**[][] tabuleiro = **new** **int**[4][4];
29. **int** altura;             //G(x)
30. **int** custoHeuristica;    //H'(X)
31. **int** custoTotal;         //F(X)
32. **int**[] posicaoZero;
33. String hash;
34. String pai;
36. **public** Puzzle(**int**[][] tab){
37. **this**.tabuleiro = tab;
38. **this**.altura = 0;
39. **this**.posicaoZero = **this**.procuraZero();
40. **this**.custoHeuristica = **this**.calculaHeuristica3();
42. **this**.custoTotal = custoHeuristica + altura;
43. **this**.hash = **this**.custoTotal + **this**.toHash();
44. **this**.pai = "Primeiro";
45. **if** (**this**.custoTotal >= 10){
46. **this**.hash = "h"+**this**.hash;
47. }
48. }
50. **public** Puzzle(**int**[][] tab, **int** altura, String pai){
51. **this**.tabuleiro = tab;
52. **this**.altura = altura;
53. **this**.posicaoZero = **this**.procuraZero();
54. **this**.custoHeuristica = **this**.calculaHeuristica5();
56. **this**.custoTotal = custoHeuristica + altura;
58. **this**.hash = **this**.custoTotal + **this**.toHash();
59. **this**.pai = pai;
60. **if** (**this**.custoTotal >= 10){
61. **this**.hash = "h"+**this**.hash;
62. }
63. }
65. **private** **int** calculaHeuristica1(){
66. **int**[][] estadoFinal = **new** **int**[][] {{1, 2, 3, 4},{12, 13, 14, 5},{11, 0, 15, 6}, {10, 9, 8, 7}};
67. **int** soma = 0;
68. **for**(**int** i = 0; i < 4; i++){
69. **for**(**int** j = 0; j < 4; j++){
70. **if**(**this**.tabuleiro[i][j] != estadoFinal[i][j]){
71. soma += 1;
72. }
73. }
74. }
75. **return** soma;
76. }
78. **private** **int** calculaHeuristica2(){
79. **int** retorno = 0;
80. **int**[] vetor = **new** **int**[16];
81. vetor[0] = **this**.tabuleiro[0][0];
82. vetor[1] = **this**.tabuleiro[0][1];
83. vetor[2] = **this**.tabuleiro[0][2];
84. vetor[3] = **this**.tabuleiro[0][3];
85. vetor[4] = **this**.tabuleiro[1][3];
86. vetor[5] = **this**.tabuleiro[2][3];
87. vetor[6] = **this**.tabuleiro[3][3];
88. vetor[7] = **this**.tabuleiro[3][2];
89. vetor[8] = **this**.tabuleiro[3][1];
90. vetor[9] = **this**.tabuleiro[3][0];
91. vetor[10] = **this**.tabuleiro[2][0];
92. vetor[11] = **this**.tabuleiro[1][0];
93. vetor[12] = **this**.tabuleiro[1][1];
94. vetor[13] = **this**.tabuleiro[1][2];
95. vetor[14] = **this**.tabuleiro[2][2];
96. vetor[15] = **this**.tabuleiro[2][1];
98. **for**(**int** l = 1; l < 16; l++){
99. **if**(vetor[l] != vetor[l-1] + 1){
100. retorno++;
101. }
102. }
103. **return** retorno - 1;
104. }
106. **private** **int** calculaHeuristica3(){
107. **int** retorno = 0;
108. **int**[][] matriz = **new** **int**[16][2];
110. matriz[0][0] = 2;
111. matriz[0][1] = 1;
113. matriz[1][0] = 0;
114. matriz[1][1] = 0;
116. matriz[2][0] = 0;
117. matriz[2][1] = 1;
119. matriz[3][0] = 0;
120. matriz[3][1] = 2;
122. matriz[4][0] = 0;
123. matriz[4][1] = 3;
125. matriz[5][0] = 1;
126. matriz[5][1] = 3;
128. matriz[6][0] = 2;
129. matriz[6][1] = 3;
131. matriz[7][0] = 3;
132. matriz[7][1] = 3;
134. matriz[8][0] = 3;
135. matriz[8][1] = 2;
137. matriz[9][0] = 3;
138. matriz[9][1] = 1;
140. matriz[10][0] = 3;
141. matriz[10][1] = 0;
143. matriz[11][0] = 2;
144. matriz[11][1] = 0;
146. matriz[12][0] = 1;
147. matriz[12][1] = 0;
149. matriz[13][0] = 1;
150. matriz[13][1] = 1;
152. matriz[14][0] = 1;
153. matriz[14][1] = 2;
155. matriz[15][0] = 2;
156. matriz[15][1] = 2;

159. **for**(**int** i = 0; i < 4; i++){
160. **for**(**int** j = 0; j < 4; j++){
161. **int** numero = **this**.tabuleiro[i][j];
162. **int** linha = matriz[numero][0];
163. **int** coluna = matriz[numero][1];
164. **int** dist = Math.abs(linha - i) + Math.abs(coluna - j);
165. retorno += dist;
167. }
168. }
170. **return** retorno;
171. }
173. **private** **int** calculaHeuristica4(){
174. **int** h1 = **this**.calculaHeuristica1();
175. **int** h2 = **this**.calculaHeuristica2();
176. **int** h3 = **this**.calculaHeuristica3();
178. **double** aux = (0.1\*h1 + 0.1\*h2 + 0.8\*h3);
179. **int** retorno;
180. aux = Math.ceil(aux);
181. retorno = (**int**) aux;
183. **return** retorno;
185. }
187. **private** **int** calculaHeuristica5(){
188. **int** h1 = **this**.calculaHeuristica1();
189. **int** h2 = **this**.calculaHeuristica2();
190. **int** h3 = **this**.calculaHeuristica3();
192. **int** aux = Math.max(h1, h2);
193. **int** retorno = Math.max(aux, h3);
195. **return** retorno;
197. }
199. **private** **int**[] procuraZero(){
200. **int**[] retorno = **new** **int**[2];
201. **for**(**int** i = 0; i < 4; i++){
202. **for**(**int** j = 0; j < 4; j++){
203. **if**(**this**.tabuleiro[i][j] == 0){
204. retorno[0] = i;
205. retorno[1] = j;
206. }
207. }
208. }
209. **return** retorno;
210. }
212. **public** **void** mostraFormato(){
213. System.out.println(**this**.hash);
214. System.out.println("Custo = " + **this**.altura + " -- Heuristica = " + **this**.custoHeuristica);
215. System.out.println("CUSTO TOTAL = " + **this**.custoTotal);
216. **for**(**int** i = 0; i < 4; i++){
217. **for**(**int** j = 0; j < 4; j++){
218. System.out.print(**this**.tabuleiro[i][j] + " ");
219. }
220. System.out.println();
221. }
222. System.out.println();
223. }
225. **public** **boolean** verifica(){
226. **int**[][] resposta = {{1, 2, 3, 4},{12, 13, 14, 5},{11, 0, 15, 6}, {10, 9, 8, 7}};
227. **for**(**int** i = 0; i < 4; i++){
228. **for**(**int** j = 0; j < 4; j++){
229. **if**(**this**.tabuleiro[i][j] != resposta[i][j]){
230. **return** **false**;
231. }
232. }
233. }
234. **return** **true**;
235. }
237. **private** String toHash(){
238. String retorno = "";
239. **for**(**int** i = 0; i < 4; i++){
240. **for**(**int** j = 0; j < 4; j++){
241. retorno = retorno + "-";
242. retorno = retorno + **this**.tabuleiro[i][j];
244. }
245. }
246. **return** retorno;
247. }
249. **public** ArrayList<Puzzle> geraSucessor(){
250. ArrayList<Puzzle> listaSucessor = **new** ArrayList<>();
251. **int** auxTroca;
252. **int** linha = **this**.posicaoZero[0];
253. **int** coluna = **this**.posicaoZero[1];
254. **int**[][] formato1 = **new** **int**[4][4];
255. **for**(**int** i = 0; i < 4; i++){
256. System.arraycopy(**this**.tabuleiro[i], 0, formato1[i], 0, 4);
257. }
258. **if**(coluna < 3){
259. **int**[][] formatoNovo = **new** **int**[4][4];
260. **for**(**int** i = 0; i < 4; i++){
261. System.arraycopy(formato1[i], 0, formatoNovo[i], 0, 4);
262. }
263. auxTroca = formatoNovo[linha][coluna];
264. formatoNovo[linha][coluna] = formatoNovo[linha][coluna+1];
265. formatoNovo[linha][coluna+1] = auxTroca;
266. Puzzle novo1 = **new** Puzzle(formatoNovo, **this**.altura+1, **this**.hash);
267. listaSucessor.add(novo1);
269. }
270. **if**(coluna > 0){
271. **int**[][] formatoNovo = **new** **int**[4][4];
272. **for**(**int** i = 0; i < 4; i++){
273. System.arraycopy(formato1[i], 0, formatoNovo[i], 0, 4);
274. }
275. auxTroca = formatoNovo[linha][coluna];
276. formatoNovo[linha][coluna] = formatoNovo[linha][coluna-1];
277. formatoNovo[linha][coluna-1] = auxTroca;
278. Puzzle novo2 = **new** Puzzle(formatoNovo, **this**.altura+1, **this**.hash);
279. listaSucessor.add(novo2);
280. }
281. **if**(linha > 0){
282. **int**[][] formatoNovo = **new** **int**[4][4];
283. **for**(**int** i = 0; i < 4; i++){
284. System.arraycopy(formato1[i], 0, formatoNovo[i], 0, 4);
285. }
286. auxTroca = formatoNovo[linha][coluna];
287. formatoNovo[linha][coluna] = formatoNovo[linha-1][coluna];
288. formatoNovo[linha-1][coluna] = auxTroca;
289. Puzzle novo3 = **new** Puzzle(formatoNovo, **this**.altura+1, **this**.hash);
290. listaSucessor.add(novo3);
291. }
292. **if**(linha < 3){
293. **int**[][] formatoNovo = **new** **int**[4][4];
294. **for**(**int** i = 0; i < 4; i++){
295. System.arraycopy(formato1[i], 0, formatoNovo[i], 0, 4);
296. }
297. auxTroca = formatoNovo[linha][coluna];
298. formatoNovo[linha][coluna] = formatoNovo[linha+1][coluna];
299. formatoNovo[linha+1][coluna] = auxTroca;
300. Puzzle novo4 = **new** Puzzle(formatoNovo, **this**.altura+1, **this**.hash);
301. listaSucessor.add(novo4);
302. }
303. **return** listaSucessor;
304. }
306. }

309. **public** **static** **int** algoritmoAEstrela(Puzzle estadoInicial){
310. **int**[][] estadoFinal = **new** **int**[][] {{1, 2, 3, 4},{12, 13, 14, 5},{11, 0, 15, 6}, {10, 9, 8, 7}};
312. TreeMap<String, Puzzle> listaAberto = **new** TreeMap<>();
313. HashMap<String, Puzzle> listaFechado = **new** HashMap<>();
315. listaAberto.put(estadoInicial.hash, estadoInicial);
316. **int** it = 0;
317. **while**(!listaAberto.isEmpty()){
318. String str = listaAberto.firstKey();
319. Puzzle v = listaAberto.get(str);
320. listaAberto.remove(str);
322. listaFechado.put(v.hash, v);
323. ArrayList<Puzzle> vSucessor = **new** ArrayList<>();
324. **if**(v.verifica()){
325. **return** v.custoTotal;
326. } **else** {
327. vSucessor = v.geraSucessor();
328. **if**(vSucessor.isEmpty()){
329. **continue**;
330. }
332. **for**(Puzzle m : vSucessor){
333. **if**((!listaAberto.containsKey(m.hash) && !listaFechado.containsKey(m.hash)) || ((listaAberto.containsKey(m.hash) || listaFechado.containsKey(m.hash)) && m.altura <= v.altura)){
334. listaAberto.put(m.hash, m);
335. **if**(listaFechado.containsKey(m.hash)){
336. listaFechado.remove(m.hash);
337. }
338. }
339. }
340. }
342. }
344. **return** -1;
345. }
347. **public** **static** **void** main(String[] args) **throws** FileNotFoundException, IOException {
349. //Tempo inicial
350. **long** tempoIn = System.currentTimeMillis();
352. //==============================================================//
353. //===========================LEITURA============================//
354. String nomeArquivo = "teste.txt";
355. FileInputStream stream = **new** FileInputStream(nomeArquivo);
356. InputStreamReader reader = **new** InputStreamReader(stream);
357. //InputStreamReader reader = new InputStreamReader(System.in);
358. BufferedReader br = **new** BufferedReader(reader);
360. **int**[][] entrada = **new** **int**[4][4];
361. String linha  = br.readLine();
362. **int**[] vetor;
363. String sublinha[];
364. sublinha = linha.split(" ");
365. **int** k = 0;
366. **for**(**int** i = 0; i < 4; i++){
367. **for**(**int** j = 0; j < 4; j++){
368. entrada[i][j] = Integer.parseInt(sublinha[k]);
369. k++;
370. }
371. }

374. //=============================================================//
375. Puzzle estadoInicial = **new** Puzzle(entrada);
376. System.out.println(algoritmoAEstrela(estadoInicial));
378. **long** tempoTotal = System.currentTimeMillis() - tempoIn;

381. System.out.println("Tempo gasto: " + tempoTotal);
383. }


387. }

Para escrever o algoritmo A-estrela, a decisão mais importante a ser tomada é como será a implementada os conjuntos dos estados abertos e fechados, visto que, em casos mais complexo, é certo que o gargalo ocorrerá nas operações de busca, inserção e remoção nestes conjuntos.

Para o caso do 15-puzzle, os estados iniciais e finais são apenas um, portanto, não houve grande preocupação com o custo das operações envolvendo o mesmo.

A coleção de objetos utilizada para implementar o conjunto de estados abertos foi o *TreeMap* e para o conjunto de estados fechado foi escolhido o *HashMap*. Ambas coleções serão explicadas abaixo:

* ***HashMap*:** Utiliza de uma tabela Hash para armazenar seus elementos. O tempo de acesso aos elementos é constante devido a função hash utilizada, explicada logo adiante, não permite chave repetida.
* ***TreeMap:*** Utiliza o algoritmo Red-Black para ordenação da árvore de elementos, garantindo a ordenação ascendente, com isso temos a busca pelo menor elemento constante. Também utiliza tabela Hash tornando o acesso constante.

A função hash construiu uma *string* a partir do custo total de cada tabuleiro para chegar ao estado final e a posição de suas peças, isso garante chaves únicas, devido que cada tabuleiro uma formação única. Para garantir a ordenação correta foi utilizada uma artimanha, que foi colocar um caractere de código ascii maior que os números visto que seria inserido um tabuleiro com custo 10 antes do tabuleiro com custo 9, já que o código do caractere 1 é menor que o código do caractere 9, abaixo será exemplificado, o caso para custo maior e menor que 10.

* **Custo Total: 9.**
* **Formato:**

1 2 3 4

12 13 14 5

11 0 15 6

10 9 8 7

* **Hash:** 9-1-2-3-4-12-13-14-5-11-0-15-6-10-9-8-7
* **Custo Total: 13.**
* **Formato:**

1 2 3 4

0 13 14 5

12 11 15 6

10 9 8 7

**Hash:** h13-1-2-3-4-0-13-14-5-12-11-15-6-10-9-8-7

A seguir, será apresentada os resultados de interações para os seguintes estados iniciais e para cada heurística

Os estados iniciais:

2 3 4 5 1 13 14 6 12 11 15 0 10 9 8 7

12 1 5 4 9 14 6 10 3 15 13 2 11 8 0 7

0 4 6 13 2 5 3 7 1 11 14  8 12 15 9 10

2 3 4 5 1 13 14 6 0 11 15 7 12 10 9 8

2 4 5 6 0 3 14 13 1 11 9 7 12 15 10 8

12 9 7 2 10 14 0 3 13 4 1 8 6 11 5 15

14 13 7 8 5 2 15 3 12 6 4 9 10 1 0 11

3 11 10 2 8 6 12 1 13 4 7 0 14 5 15 9

3 0 2 10 11 5 14 6 9 7 15 4 8 1 13 12

11 1 5 0 8 13 3 9 15 12 4 6 14 2 10 7

* **Heurística 1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Estado Inicial | Nº Movimento | Tempo (ms) | Iterações | Memória |
| 2 3 4 5 1 13 14 6 12 11 15 0 10 9 8 7 | 8 | 15 | 9 |  |
| 12 1 5 4 9 14 6 10 3 15 13 2 11 8 0 7 |  |  |  |  |
| 0 4 6 13 2 5 3 7 1 11 14 8 12 15 9 10 |  |  |  |  |
| 2 3 4 5 1 13 14 6 0 11 15 7 12 10 9 8 | 13 | 16 | 28 |  |
| 2 4 5 6 0 3 14 13 1 11 9 7 12 15 10 8 | 28 | 1746 | 114294 |  |
| 12 9 7 2 10 14 0 3 13 4 1 8 6 11 5 15 |  |  |  |  |
| 14 13 7 8 5 2 15 3 12 6 4 9 10 1 0 11 |  |  |  |  |
| 3 11 10 2 8 6 12 1 13 4 7 0 14 5 15 9 |  |  |  |  |
| 3 0 2 10 11 5 14 6 9 7 15 4 8 1 13 12 |  |  |  |  |
| 11 1 5 0 8 13 3 9 15 12 4 6 14 2 10 7 |  |  |  |  |

* **Heurística 2**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Estado Inicial | Nº Movimento | Tempo (ms) | Iterações | Memória |
| 2 3 4 5 1 13 14 6 12 11 15 0 10 9 8 7 |  |  |  |  |
| 12 1 5 4 9 14 6 10 3 15 13 2 11 8 0 7 |  |  |  |  |
| 0 4 6 13 2 5 3 7 1 11 14 8 12 15 9 10 |  |  |  |  |
| 2 3 4 5 1 13 14 6 0 11 15 7 12 10 9 8 |  |  |  |  |
| 2 4 5 6 0 3 14 13 1 11 9 7 12 15 10 8 |  |  |  |  |
| 12 9 7 2 10 14 0 3 13 4 1 8 6 11 5 15 |  |  |  |  |
| 14 13 7 8 5 2 15 3 12 6 4 9 10 1 0 11 |  |  |  |  |
| 3 11 10 2 8 6 12 1 13 4 7 0 14 5 15 9 |  |  |  |  |
| 3 0 2 10 11 5 14 6 9 7 15 4 8 1 13 12 |  |  |  |  |
| 11 1 5 0 8 13 3 9 15 12 4 6 14 2 10 7 |  |  |  |  |

* **Heurística 3**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Estado Inicial | Nº Movimento | Tempo (ms) | Iterações | Memória |
| 2 3 4 5 1 13 14 6 12 11 15 0 10 9 8 7 |  |  |  |  |
| 12 1 5 4 9 14 6 10 3 15 13 2 11 8 0 7 |  |  |  |  |
| 0 4 6 13 2 5 3 7 1 11 14 8 12 15 9 10 |  |  |  |  |
| 2 3 4 5 1 13 14 6 0 11 15 7 12 10 9 8 |  |  |  |  |
| 2 4 5 6 0 3 14 13 1 11 9 7 12 15 10 8 |  |  |  |  |
| 12 9 7 2 10 14 0 3 13 4 1 8 6 11 5 15 |  |  |  |  |
| 14 13 7 8 5 2 15 3 12 6 4 9 10 1 0 11 |  |  |  |  |
| 3 11 10 2 8 6 12 1 13 4 7 0 14 5 15 9 |  |  |  |  |
| 3 0 2 10 11 5 14 6 9 7 15 4 8 1 13 12 |  |  |  |  |
| 11 1 5 0 8 13 3 9 15 12 4 6 14 2 10 7 |  |  |  |  |

* **Heurística 4**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Estado Inicial | Nº Movimento | Tempo (ms) | Iterações | Memória |
| 2 3 4 5 1 13 14 6 12 11 15 0 10 9 8 7 |  |  |  |  |
| 12 1 5 4 9 14 6 10 3 15 13 2 11 8 0 7 |  |  |  |  |
| 0 4 6 13 2 5 3 7 1 11 14 8 12 15 9 10 |  |  |  |  |
| 2 3 4 5 1 13 14 6 0 11 15 7 12 10 9 8 |  |  |  |  |
| 2 4 5 6 0 3 14 13 1 11 9 7 12 15 10 8 |  |  |  |  |
| 12 9 7 2 10 14 0 3 13 4 1 8 6 11 5 15 |  |  |  |  |
| 14 13 7 8 5 2 15 3 12 6 4 9 10 1 0 11 |  |  |  |  |
| 3 11 10 2 8 6 12 1 13 4 7 0 14 5 15 9 |  |  |  |  |
| 3 0 2 10 11 5 14 6 9 7 15 4 8 1 13 12 |  |  |  |  |
| 11 1 5 0 8 13 3 9 15 12 4 6 14 2 10 7 |  |  |  |  |

* **Heurística 5**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Estado Inicial | Nº Movimento | Tempo (ms) | Iterações | Memória |
| 2 3 4 5 1 13 14 6 12 11 15 0 10 9 8 7 |  |  |  |  |
| 12 1 5 4 9 14 6 10 3 15 13 2 11 8 0 7 |  |  |  |  |
| 0 4 6 13 2 5 3 7 1 11 14 8 12 15 9 10 |  |  |  |  |
| 2 3 4 5 1 13 14 6 0 11 15 7 12 10 9 8 |  |  |  |  |
| 2 4 5 6 0 3 14 13 1 11 9 7 12 15 10 8 |  |  |  |  |
| 12 9 7 2 10 14 0 3 13 4 1 8 6 11 5 15 |  |  |  |  |
| 14 13 7 8 5 2 15 3 12 6 4 9 10 1 0 11 |  |  |  |  |
| 3 11 10 2 8 6 12 1 13 4 7 0 14 5 15 9 |  |  |  |  |
| 3 0 2 10 11 5 14 6 9 7 15 4 8 1 13 12 |  |  |  |  |
| 11 1 5 0 8 13 3 9 15 12 4 6 14 2 10 7 |  |  |  |  |

1. **Comparações por tabelas**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | **Iterações** | | | | |
|  | Solução ótima  (movimentos) | Heurística 1 | Heurística 2 | Heurística 3 | Heurística 4 | Heurística 5 |
| Caso 1: | 8 | 9 | 14 | 12 |  | 12 |
| Caso 2: | 36 |  |  | 20.719 |  | 20.688 |
| Caso 3: | 41 |  |  | 225.989 |  | 225.863 |
| Caso 4: | 13 | 26 | 360 | 24 |  | 24 |
| Caso 5: | 28 | 68.211 | 680.819 | 931 |  | 930 |
| Caso 6: | 46 |  |  | 326.802 |  | 326.734 |
| Caso 7: | 48 |  |  | 529.322 |  | 529.308 |
| Caso 8: |  |  |  |  |  |  |
| Caso 9: |  |  |  |  |  |  |
| Caso 10: | 42 |  |  | 337.133 |  | 336.912 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | **Uso de memória** | | | | |
|  | Solução ótima  (movimentos) | Heurística 1 | Heurística 2 | Heurística 3 | Heurística 4 | Heurística 5 |
| Caso 1: | 8 |  |  |  |  |  |
| Caso 2: | 36 |  |  |  |  |  |
| Caso 3: | 41 |  |  |  |  |  |
| Caso 4: | 13 |  |  |  |  |  |
| Caso 5: | 28 |  |  |  |  |  |
| Caso 6: | 46 |  |  |  |  |  |
| Caso 7: | 48 |  |  |  |  |  |
| Caso 8: |  |  |  |  |  |  |
| Caso 9: |  |  |  |  |  |  |
| Caso 10: | 42 |  |  |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | **Tempo de processamento (segundos)** | | | | |
|  | Solução ótima  (movimentos) | Heurística 1 | Heurística 2 | Heurística 3 | Heurística 4 | Heurística 5 |
| Caso 1: | 8 |  |  |  |  |  |
| Caso 2: | 36 |  |  |  |  |  |
| Caso 3: | 41 |  |  |  |  |  |
| Caso 4: | 13 |  |  |  |  |  |
| Caso 5: | 28 |  |  |  |  |  |
| Caso 6: | 46 |  |  |  |  |  |
| Caso 7: | 48 |  |  |  |  |  |
| Caso 8: |  |  |  |  |  |  |
| Caso 9: |  |  |  |  |  |  |
| Caso 10: | 42 |  |  |  |  |  |