**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.

O computador utilizado nos testes possui as seguintes características relevantes:

* Processador: Intel Core 2 Duo E7500 2,93GHz
* Memória RAM: 4GB total, com limite de 2GB para os testes.
* Sistema Operacional: Windows 10 64 bits

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**

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 *TreeSet* 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 e não permite chave repetida. Assim permite que elementos sejam inseridos, removidos e acessados em uma complexidade de tempo média de O(1).
* ***TreeSet:*** 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 inserção, remoção e acesso a elementos tem complexidade O(log n) sendo que a remoção do menor elemento tem complexidade O(1).

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

Os estados iniciais:

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

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

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

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

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

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

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

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

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

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

**Observação**: Casos que ficaram em branco nos testes seguintes não foram possíveis de se resolver pela limitação da memória.

* **Heurística 1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Estado Inicial | Nº Movimentos | Tempo (ms) | Iterações | Memória (MB) |
| **Caso 1:** | 8 | 3 | 9 | 7,8 |
| **Caso 2:** | 36 | - | - | - |
| **Caso 3:** | 41 | - | - | - |
| **Caso 4:** | 13 | 3 | 26 | 8,0 |
| **Caso 5:** | 28 | 731 | 68.221 | 158,8 |
| **Caso 6:** | 46 | - | - | - |
| **Caso 7:** | 48 | - | - | - |
| **Caso 8:** | 54 | - | - | - |
| **Caso 9:** | 52 | - | - | - |
| **Caso 10:** | 42 | - | - | - |

* **Heurística 2**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Estado Inicial | Nº Movimentos | Tempo (ms) | Iterações | Memória (MB) |
| **Caso 1:** | 8 | 3 | 14 | 7,9 |
| **Caso 2:** | 36 | - | - | - |
| **Caso 3:** | 41 | - | - | - |
| **Caso 4:** | 13 | 3 | 360 | 12,1 |
| **Caso 5:** | 28 | 13.177 | 680.819 | 753,2 |
| **Caso 6:** | 46 | - | - | - |
| **Caso 7:** | 48 | - | - | - |
| **Caso 8:** | 54 | - | - | - |
| **Caso 9:** | 52 | - | - | - |
| **Caso 10:** | 42 | - | - | - |

* **Heurística 3**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Estado Inicial | Nº Movimentos | Tempo (ms) | Iterações | Memória (MB) |
| **Caso 1:** | 8 | 3 | 12 | 8,0 |
| **Caso 2:** | 36 | 313 | 15.721 | 61,72 |
| **Caso 3:** | 41 | 3.749 | 168.491 | 450,2 |
| **Caso 4:** | 13 | 4 | 23 | 8,45 |
| **Caso 5:** | 28 | 57 | 595 | 17,5 |
| **Caso 6:** | 46 | 3.536 | 187.867 | 446,75 |
| **Caso 7:** | 48 | 10.826 | 301.679 | 622,9 |
| **Caso 8:** | 54 | 50.500 | 1.874.175 | 2.338,45 |
| **Caso 9:** | 52 | - | - | - |
| **Caso 10:** | 42 | 7.064 | 337133 | 532,7 |

* **Heurística 4  
  Pesos:   
  p1 = 0,03  
  p2 = 0,02  
  p3 = 0,95**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Estado Inicial | Nº Movimentos | Tempo (ms) | Iterações | Memória (MB) |
| **Caso 1:** | 8 | 3 | 12 | 8,34 |
| **Caso 2:** | 36 | 422 | 21.351 | 107,77 |
| **Caso 3:** | 41 | 4.234 | 227.504 | 486,93 |
| **Caso 4:** | 13 | 4 | 24 | 8,65 |
| **Caso 5:** | 28 | 62 | 931 | 28,28 |
| **Caso 6:** | 46 | 6.907 | 342.862 | 591,06 |
| **Caso 7:** | 48 | 14.032 | 579.277 | 782,6 |
| **Caso 8:** | 54 | - | - | - |
| **Caso 9:** | 52 | - | - | - |
| **Caso 10:** | 42 | 6.828 | 337.684 | 592,24 |

* **Heurística 5**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Estado Inicial | Nº Movimentos | Tempo (ms) | Iterações | Memória (MB) |
| **Caso 1:** | 8 | 3 | 12 | 8,0 |
| **Caso 2:** | 36 | 387 | 20.688 | 66,6 |
| **Caso 3:** | 41 | 5.903 | 225.863 | 462,0 |
| **Caso 4:** | 13 | 4 | 24 | 8,2 |
| **Caso 5:** | 28 | 71 | 930 | 20,6 |
| **Caso 6:** | 46 | 7.191 | 326.734 | 533,1 |
| **Caso 7:** | 48 | 15.705 | 529.308 | 743,5 |
| **Caso 8:** | 54 | - | - | - |
| **Caso 9:** | 52 | - | - | - |
| **Caso 10:** | 42 | 7.278 | 336.912 | 533,4 |

1. **Algoritmo em C++**

O algoritmo implementado em C++ foi exatamente o mesmo que em Java, utilizando as mesmas estruturas equivalentes de C++.

Para o conjunto de abertos foi utilizado o *std::set* que é equivalente ao *TreeSet* do Java, e para o conjunto de fechados foi utilizado o *std::unordered\_map* que é equivalente ao HashMap do Java:

* ***std::unordered\_map*:** Não são ordenados em nenhuma ordem particular dos elementos e internamente são organizados em *buckets* dependendo do valor hash. Possuem complexidade de tempo média O(1), ou seja, constante, para inserção, remoção e acesso.
* ***std::set:*** É implementado utilizando uma árvore binária de busca, e são ordenados de acordo com o método de comparação da classe dos objetos do conjunto. A complexidade de tempo para inserção, remoção e acesso é de O(log n) e constante para extrair o elemento de menor valor.

**Observação**: Casos que ficaram em branco nos testes seguintes não foram possíveis de se resolver pela limitação da memória.

* **Heurística 1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Estado Inicial | Nº Movimentos | Tempo (ms) | Iterações | Memória (MB) |
| **Caso 1:** | 8 | 1 | 9 | 0,4 |
| **Caso 2:** | 36 | - | - | - |
| **Caso 3:** | 41 | - | - | - |
| **Caso 4:** | 13 | 2 | 27 | 0,4 |
| **Caso 5:** | 28 | 18.528 | 130.133 | 54,6 |
| **Caso 6:** | 46 | - | - | - |
| **Caso 7:** | 48 | - | - | - |
| **Caso 8:** | 54 | - | - | - |
| **Caso 9:** | 52 | - | - | - |
| **Caso 10:** | 42 | - | - | - |

* **Heurística 2**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Estado Inicial | Nº Movimentos | Tempo (ms) | Iterações | Memória (MB) |
| **Caso 1:** | 8 | 2 | 22 | 0,4 |
| **Caso 2:** | 36 | - | - | - |
| **Caso 3:** | 41 | - | - | - |
| **Caso 4:** | 13 | 63 | 500 | 0,7 |
| **Caso 5:** | 28 | 137.350 | 935.879 | 360,9 |
| **Caso 6:** | 46 | - | - | - |
| **Caso 7:** | 48 | - | - | - |
| **Caso 8:** | 54 | - | - | - |
| **Caso 9:** | 52 | - | - | - |
| **Caso 10:** | 42 | - | - | - |

* **Heurística 3**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Estado Inicial | Nº Movimentos | Tempo (ms) | Iterações | Memória (MB) |
| **Caso 1:** | 8 | 1 | 12 | 0,4 |
| **Caso 2:** | 36 | 3.161 | 22.405 | 10,3 |
| **Caso 3:** | 41 | 37.386 | 247.978 | 108,5 |
| **Caso 4:** | 13 | 3 | 26 | 0,4 |
| **Caso 5:** | 28 | 158 | 1.176 | 1,0 |
| **Caso 6:** | 46 | 37.368 | 255.513 | 103,0 |
| **Caso 7:** | 48 | 79.466 | 529.056 | 214,7 |
| **Caso 8:** | 54 | 593.261 | 3.792.797 | 1.557,5 |
| **Caso 9:** | 52 | 358.398 | 2.305.614 | 928,7 |
| **Caso 10:** | 42 | 42.288 | 286.658 | 117,0 |

* **Heurística 4  
  Pesos:   
  p1 = 0,03  
  p2 = 0,02  
  p3 = 0,95**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Estado Inicial | Nº Movimentos | Tempo (ms) | Iterações | Memória (MB) |
| **Caso 1:** | 8 | 1 | 12 | 0,4 |
| **Caso 2:** | 36 | 4.109 | 22.298 | 10,4 |
| **Caso 3:** | 41 | 32.312 | 193.914 | 86,5 |
| **Caso 4:** | 13 | 3 | 26 | 0,4 |
| **Caso 5:** | 28 | 187 | 1.176 | 1,0 |
| **Caso 6:** | 46 | 45.844 | 282.553 | 114.3 |
| **Caso 7:** | 48 |  |  |  |
| **Caso 8:** | 54 |  |  |  |
| **Caso 9:** | 52 |  |  |  |
| **Caso 10:** | 42 | 45.506 | 276.507 | 115.4 |

* **Heurística 5**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Estado Inicial | Nº Movimentos | Tempo (ms) | Iterações | Memória (MB) |
| **Caso 1:** | 8 | 1 | 12 | 0,4 |
| **Caso 2:** | 36 | 3.175 | 22.457 | 10,2 |
| **Caso 3:** | 41 | 32.868 | 222.786 | 96,7 |
| **Caso 4:** | 13 | 3 | 26 | 0,4 |
| **Caso 5:** | 28 | 163 | 1.176 | 1,0 |
| **Caso 6:** | 46 | 33.556 | 230.094 | 90,5 |
| **Caso 7:** | 48 | 93.216 | 618.431 | 251,4 |
| **Caso 8:** | 54 | 600.220 | 3.816.967 | 1.531,4 |
| **Caso 9:** | 52 | 455.203 | 2.908.226 | 1.221,0 |
| **Caso 10:** | 42 | 43.887 | 294.976 | 121,0 |

1. **Conclusão**

Com relação às linguagens de programação utilizadas podemos perceber que o algoritmo escrito em Java foi mais rápido em tempo de processamento do que o algoritmo escrito em C++, porém a memória utilizada é muito maior se comparada ao que o C++ utilizou, e por isso, utilizando o algoritmo em Java não foi possível resolver os Casos Teste 8 e 9, pois esses são casos que levam muitas iterações e por isso a memória disponível (2GB) para os testes não foi suficiente. Já em C++ esses casos foram resolvidos.

Em relação ao algoritmo A\* e com o uso de uma heurística apropriada, podemos notar que a solução ótima é dada relativamente rápida, mas o uso de memória para problemas mais complexos é elevado.

Observando as cinco heurísticas utilizadas podemos perceber que a terceira heurística obtém resultados mais satisfatórios, por ser mais eficiente em termos de uso de memória. As duas heurísticas iniciais além de não resolver a maioria dos Casos Testes devido à limitação de memória, não auxilia na melhora de desempenho ao ser utilizadas em conjuntos, como podemos observar nos resultados quinta heurística.