Relatório Segundo Trabalho de PAA 2013-2

Chrystinne Fernandes, Daltro Gama, Luiz Aguiar

20 de Novembro de 2013

Introdução

Para implementar a modelagem do problema do quebra-cabeça de 8 como enunciado, foi utilizada a linguagem de programação Java 1.7. Os tempos que serão apresentados no relatório dizem respeito à execução do código Java em uma máquina com processador Intel Core i7 2.7GHz, 8Gb de RAM 1.333MHz DDR3 e 4Mb de cache L3 rodando MAC OSX 10.9.

Estruturas de Dados Utilizadas

Aproveitando que a linguagem escolhida para a implementação foi a linguagem Java, a estrutura de dados utilizada foi implementada utilizando os artifícios próprios da linguagem: Cada nó é modelado como uma classe Java, e cada objeto nó leva em si a sua lista de adjacências, que aponta diretamente para a respectiva instância, sem indireções.

A configuração é representada como um array de nove bytes, onde cada byte pode ter o valor de 0 até 9, onde o zero indica o espaço que está vazio e pode ser movido.

Foi interessante ter uma forma eficiente de se localizar a instância de um nó de uma configuração arbitrária na memória, inclusive para efeito de construir o grafo de forma eficiente. Para isto, o conjunto global dos nós do grafo poderiam ser armazenados em um array de 9! elementos. O problema aqui é o de que a busca por um nó de uma dada configuração se faz ineficiente, pois seria O(n) em um array de 9! posições.

Para resolver esta questão, foi calculado um número inteiro que represente unicamente cada configuração possível do jogo, chamado "id"do nó do grafo. Caso a configuração seja {1,2,3,4,5,6,7,8,0} (*a solução do quebra-cabeça*), o id do nó é 123.456.780, que é facilmente comportado num número inteiro de 32 bits. Este id, então, é usado como chave na implementação padrão do Java para tabela hash java.util.HashMap(), que provê gravação e recuperação de pares chave-valor, dada sua chave.

Tarefa 1

Na tarefa 1, pedia-se para montar o grafo e, com sua instância, calcular o número de componentes conexas, sendo que para cada componente conexa fornecer o número de nós e arestas.

O tempo de execução do algoritmo de montagem do grafo, onde são construídas todas as permutações para a configucação do jogo e os links entre os nós são devidamente criados, levou um tempo médio de 1.2 segundos na máquina com as configurações já descritas.

Uma vez montado o grafo, foi executado um procedimento de DFS para identificar o número de componentes conexas, assim como contar quantos nós e arestas cada componente possui.

Convenientemente, a DFS implementada já atende também a tarefa 3. O tempo de execução desta DFS foi de 1.3 segundoss na máquina com as configurações já descritas.

A conclusão do cálculo das componentes conexas é a de que o grafo está dividido em duas componentes conexas de igual tamanho. Ambas possuem 181.440 nós e 241.920 arestas.

É possível justificar que o grafo se divide em duas componentes de igual tamanho. Vamos considerar, para isto, o número de inversões em uma configuração. Por exemplo, a configuração {1,2,3,4,5,6,7,8,0} não possui inversões. Se movimentarmos o 8 para a direita, obteremos {1,2,3,4,5,6,7,0,8}, o que introduz duas inversões. Sempre que uma movimentação é feita, duas inversões são feitas, de forma que o número de inversões de qualquer configuração que possa ser atingida da configuração inicial {1,2,3,4,5,6,7,8,0} será par. Este conjunto formará a componente conexa que possui a configuração inicial. A outra componente conexa é a que possui configurações com números ímpares de inversões.

Tarefa 2

O grafo do jogo em questão possui duas configurações de cujas distâncias mínimas para a solução são máximas: A configuração {6,4,7,8,5,-,3,2,1} (*O '-' indica o espaço vazio*) e a {8,6,7,2,5,4,3,-,1}.

Também podemos verificar que não existe caminhos para chegar a configuração pretendida usando certas configurações de saída, como por exemplo a configuração {1,3,2,4,5,6,7,8,-} ou {8,6,7,5,4,3,2,1,-} para a componente conexa de numeros de inversões par, confirmando assim a existência de 2 componentes conexas.

Utilizamos uma BFS para descobrirmos as máximas distâncias mínimas para se chegar a configuração pretendida, a BFS foi executada com sucesso, tendo como tempo de execução o tempo médio de 0.2 segundos as configurações da máquina especificada acima. Concluimos que o total de "jogadas"para atingirmos a configuração pretendida (*partindo de uma configuração válida para a componente conexa*) foi de 32 passos com a complexidade de tempo analisada, para a execução da BFS, da ordem de O(m + n) sendo n o número de vértices e m o número de arestas do grafo.

A estrutura de fila foi utilizada para que fosse assegurada a ordem de chegada dos vértices, proporcianando assim, que as visitas a cada vértice fossem realiazadas atraves da ordem de chegada na fila, e assegurando que um vétice já visitado não entrasse novamente na fila.

Abaixo toda o caminho gerado à partir da segunda configuração, até a solução final, passo a passo:

1:

8	6	7
2	5	4
3		1

2:

8	6	7
2	5	4
3	1	

3:

8	6	7
2	5	
3	1	4

4:

8	6	
2	5	7
3	1	4

5:

8		6
2	5	7
3	1	4

6:

	8	6
2	5	7
3	1	4

7:

2	8	6
	5	7
3	1	4

8:

2	8	6
3	5	7
	1	4

9:

2	8	6
3	5	7
1		4

10:

2	8	6
3		7
1	5	4

11:

2	8	6
	3	7
1	5	4

12:

2	8	6
1	3	7
	5	4

13:

2	8	6
1	3	7
5		4

14:

2	8	6
1	3	7
5	4	

15:

2	8	6
1	3	
5	4	7

16:

2	8	6
1		3
5	4	7

17:

2		6
1	8	3
5	4	7

18:

2	6	
1	8	3
5	4	7

19:

2	6	3
1	8	
5	4	7

20:

2	6	3
1		8
5	4	7

21:

2		3
1	6	8
5	4	7

22:

	2	3
1	6	8
5	4	7

23:

1	2	3
	6	8
5	4	7

24:

1	2	3
5	6	8
	4	7

25:

1	2	3
5	6	8
4		7

26:

1	2	3
5	6	8
4	7	

27:

1	2	3
5	6	
4	7	8

28:

1	2	3
5		6
4	7	8

29:

1	2	3
	5	6
4	7	8

30:

1	2	3
4	5	6
	7	8

31:

1	2	3
4	5	6
7		8

32:

1	2	3
4	5	6
7	8	

Tarefa 3

A identificação dos pontos de articulação do grafo pôde ser gerada simultaneamente ao cálculo das componentes conexas, como já descrito na sessão Tarefa 1. O tempo médio de execução da DFS modificada foi de 1.3 segundos.