Projeto e Análise de Algoritmos
Trabalho Prático - Grafos

Raquel Yuri da Silveira Aoki (201566918)

[1]Universidade Federal de Minas Gerais (UFMG)
Departamento de Ciência da Computação

raquel.aoki@dcc.ufmg.br

1. Introdução

Este relatório é referente ao 2º Trabalho Prático da disciplina de Projeto e Análise de Algoritmos de 2015/2. O problema a ser resolvido é o Quebra-cabeça das N pastilhas (N-puzzle), em que o tabuleiro tem $N^{1/2}$x$N^{1/2}$ espaços, N-1 pastilhas e 1 espaço em branco. O objetivo do jogo é colocar o espaço em branco na primeira posição seguido pelas demais peças em ordem crescente de grandeza, sendo o único movimento permitido o de deslizar uma peça para o espaço em branco. Considerando um N-pastilhas com n=3, o estado final procurado seria:

	1	2
3	4	5
6	7	8

Sabe-se que nem todas as configurações de entrada possuem solução. Portanto, o algoritmo deve identificar se o arquivo dado como *input* tem solução, e caso tenha, encontra-lá.

Na Seção 2 será discutida a modelagem do problema, sua solvabilidade e as heurísticas utilizadas. A Seção 3 apresenta um pseudocódigo do algoritmo utilizado para resolver o problema e na Seção 4 é feita sua análise de complexidade. A análise experimental realizada é apresentada na Seção 5; o uso e a compilação na Seção 6. As conclusões a certa do trabalho são feitas na Seção 7, e por fim, são apresentadas as referências do trabalho.

2. Modelagem

O Quebra-cabeça das N pastilhas será modelado com um grafo, onde cada estado é considerado um vértice e os estados alcançáveis são aqueles à um movimento do espaço vazio.

A solução ótima é o menor caminho entre o estado inicial e o estado final/objetivo. O caminho é denominado como a sequência de movimentos/ações que o espaço vazio faz até chegar na configuração objetivo.

Essa busca da solução no grafo/árvore foi feita com o A* que busca pelo melhor estado momentâneo, ou seja, aquele a um menor custo a partir do estado inicial. O A* é ótimo, pois se a solução existe, ele garantidamente encontrará o menor caminho até o estado final como provado em [2].

Mais sobre o N-pastilha e sobre o algoritmo A* pode ser visto em [4] e [1].

2.1. Solvabilidade

Como citado, sabe-se que nem todas as soluções iniciais possuem soluções. Existe uma maneira de verificar se um estado qualquer possui solução ou não, como pode ser visto em maiores detalhes em [6].

Considerando o tabuleiro linearizado, ocorre uma inversão quando uma peça de valor X precede uma peça de valor menor que o seu. Somando-se todas as inversões do tabuleiro inicial e considerando que a primeira linha é a 0:

- Tabuleiro dimensão impar: tem solução sempre que o número de inversões é par;
- Tabuleiro dimensão par:
  - tem solução se o número de inversões é par e o espaço _ está em uma linha par;
  - tem solução se o número de inversões é impar e o espaço _ está em uma linha impar;

Portanto, ao ler a configuração inicial, o primeiro passo é avaliar se possui solução. Se não tiver, o programa imprimi no *output* a frase "Sem Solução", se tem, ele inicia o algoritmo para procurar o melhor caminho até o estado final.

2.2. Heurísticas

Cada estado intermediário entre a configuração inicial e a final possui um custo, e é o seu cálculo que se encontra a principal diferença entre A* e as demais abordagens para resolver esse problema. O A* considera em sua função de custo além do custo real para se chegar à esse estado corrente a partir do inicio, um valor dado pela heurística utilizada.

$$f(n) = g(n) + h(n)$$

em que f(n) é o custo, g(n) o custo real para se chegar a esse estado e h(n) o custo dado pela heurística.

Várias heurísticas podem ser usadas, mas elas são consideradas admissíveis somente se $h(n) \leq g(n)$, ou seja, qualquer movimento pode, na melhor das hipóteses, mover uma peça para um passo mais perto do objetivo e consistentes se respeitam a desigualdade do triângulo $h(n) \leq custo(n, a, n') + h'(n)$.

Tanto na contagem das inversões quanto nas heurísticas, a peça _ não é considerada nos cálculos. As 3 heurísticas utilizadas foram:

2.2.1. Distância de Manhattan

Consiste na soma da quantidade de passos verticais e horizontais que uma peça precisa para chegar ao seu estado final considerando que existe somente essa peça no tabuleiro. Para exemplificar, considere a Tabela 1: as peças 1, 2, 5, 7 e 8 estão nos seus lugares finais; a peça 3 precisa de 1 movimento vertical para chegar ao seu lugar certo; a peça 4 precisa de 1 vertical e 1 horizontal e peça 6 precisa de um passo vertical; portanto, DM = 4.

Essa é uma heurística admissível pois cada peça será movida pelo menos a quantidade de passos entre ela e sua posição final, então $h(n) \leq g(n)$.

Tabela 1. Exemplo

4	1	2
6		5
3	7	8

Cada peça pode ter os seguintes estados após um movimento: ou ela aproximou da sua posição final ou ela afastou. Caso ela se aproxime da sua posição final, $h'(n)+1 = h(n)$ e sendo o custo do movimento 1, então $h(n) \leq 1 + h'(n) + 1$; por outro lado, se ela se afastar da sua posição final $h'(n) - 1 = h(n)$ e $h(n) \leq 1 + h'(n) - 1$. Em ambos os casos a desigualdade do triângulo é respeitada e portanto conclui-se que essa é uma heurística consistente.

2.2.2. Distância de Hamming

Consiste na soma de peças fora do seu local final. No estado mostrada na Tabela 1, as peças 3, 4 e 6 estão fora do lugar, portanto, DH=3.

A Distância de Hamming é uma heurística admissível pois o total de movimentos para ordenar as peças corretamente é pelo menos o número de peças fora do lugar, já que cada peça precisa ser movida pelo menos uma vez,logo, $h(n) \leq g(n)$.

Ela também é consistente. Cada peça possui três estados possíveis: caso não tenha saído sido movida, $h(n) = h'(n)$ e adicionando o custo, $h(n) \leq 1+h'(n)$; caso tenha ido para seu lugar final, $h(n) = 1+h'(n)$ e adicionando o custo tem-se $h(n) \leq 1+h'(n)+1$; e caso tenha saído do seu lugar final $h(n) = h'(n) - 1$ e acrescentando o custo $h(n) \leq 1 + h'(n) - 1$. Como a desigualdade triangular não foi infringida em nenhum dos casos, conclui-se que ela é consistente.

2.2.3. Distância Euclidiana

Considerando que a posição final de uma dada peça é (x1,y1) e que ela atualmente está em (x2,y2), o valor da heurística seria $DE = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$. Por exemplo, a peça 4 do Tabuleiro 1 está na posição (0,0) e sua posição final é a (1,1), então $DE_4 = \sqrt{(1 - 0)^2 + (1 - 0)^2}$ Repete-se a mesma fórmula para todas as peças e ao final soma-se.

Pode-se provar que essa heurística é admissível a partir do fato que a distância de Manhattan foi admissível. Relembrando o Teorema de Pitágoras, sabe-se que $c^2 = a^2+b^2$ o que implica $c = \sqrt{a^2 + b^2} \leq a+b$. A Distância de Manhattan fornece exatamente $a+b$, que pelo Teorema de Pitágoras é maior que $c = h_E(n)$, sendo $h)E(n)$ o valor da heurística da Distância Euclidiana. Logo, como sabe-se que a propriedade $h_M(n) \leq g(n)$ é válida para a Distância de Manhattan e $h_E(n) \leq h_M(n)$, tem-se $h_E(n) \leq g(n)$.

Sua consistência pode ser provada considerando os estados possíveis de sua peça. Se ele chegou ao estado final, quer dizer que no estado $h(n)$ ele estava há no máximo um passo vertical ou horizontal e portanto, $h(n) = \sqrt{1} = 1$ e $h'(n) = 0$, então $h(n) = 1 \leq 1 + h'(n)$; no segundo estado considerando que ela estava no seu lugar final e afastou 1

unidade, $h(n) = 0 \leq 1 + h'(n)$, sendo que $h'(n) = 1$. Portanto, a desigualdade triangular é válida e essa é uma heurística consistente.

2.3. Iterações

Os custos dos estados não visitados mas já descobertos são guardados em uma fila. Ao longo das iteração, o algoritmo escolhe o próximo estado com o menor custo de acordo com a função heurística utilizada.

Um *hash* é utilizado para guardar os estados já visitados. A *key* é um número que corresponde ao estado já visitado (Ex.: A sequência de estados 1 2 3 /4 5 6/7 8 _ vira o número 123456780.) e a essa key é associado o menor custo já observado desse estado, lembrando que o custo é uma soma do valor dado pela heurística e da distância do estado corrente ao estado inicial.

3. Algoritmo

A lógica utilizada pelo algoritmo foi a seguinte:

```
struct tree_node;
final // estado objetivo
heu // numero da heuristica utilizada
n // tamanho do tabuleiro

while(stack.size()>0) // fila de possiveis estados tem elementos
    curr = min(stack);
    inserir(curr, final, n,heu)
        // dentro da funcao inserir
        novo tree_node* t1,t2, t3,t4
        movendo(curr,t1, 'u'); movendo(curr,t2, 'r')
        movendo(curr,t3, 'l'); movendo(curr,t4, 'd')

        t1->cost = heu(t1,final)+t1->distancia
        t2->cost = heu(t2,final)+t2->distancia
        t3->cost = heu(t3,final)+t3->distancia
        t4->cost = heu(t4,final)+t4->distancia
        t1->parent = t2->parent = curr
        t3->parent = t4->parent = curr

        bool sucesso = false
        testasucesso(t1,final);     testasucesso(t2,final)
        testasucesso(t3,final);     testasucesso(t4,final)

        if(sucesso)
            print(caminho); esvazia(stack)
        else
            valido(t1) stack.push_back(t1)
                se nao delete(t1)
```

```
valido(t2) stack.push_back(t2)
    se nao delete(t2)
valido(t3) stack.push_back(t3)
    se nao delete(t3)
valido(t4) stack.push_back(t4)
    se nao delete(t4)
```

A função *movendo* reorganiza a sequência dos elementos de acordo com o movimento quando ele é possível de ser feito. Quando o movimento não é válido, como por exemplo quando o _ está na linha superior e é dado o movimento "acima", é retornado um vetor vazio como sua nova configuração. As próximas funções retornam um valor *default* quando ele é vazio. Em *heu* é calculado o valor da heurística para essa nova configuração. A função *testasucesso* testa se a solução foi obtida, se sim ele *printa* o caminho com a função *print* e esvazia a fila para terminar o *loop*, se não, caso o movimento seja válido, ele acrescenta o caminho na fila, se não, o caminho é deletado.

4. Análise Complexidade

A complexidade depende da heurística utilizada, mas aqui será feita a análise somente do pior caso.

4.1. Complexidade de tempo

A complexidade do A* é exponencial $O(b^d) = O(3^d)$, em que b é o fator de ramificação e d é a profundidade da árvore. O fator de ramificação do algoritmo desenvolvido é 3, pois dado um movimento ele pode fazer no máximo outros 3, já que um deles voltará para o estado anterior. Embora tenha complexidade exponencial, boas heurísticas diminuem significativamente esse custo.

Como pode ser visto em [5], com a utilização de boas heurísticas e certas restrições, a complexidade pode ser polinomial.

4.2. Complexidade de espaço

Sua complexidade de espaço é exponencial $O(b^d)$, com $b = 3$, devido a necessidade de guardar os nós até encontrar o estado final.

Portanto, embora o A* funcione muito bem para o N-pastilha de dimensão 3, para certas configurações de dimensão 4 ele pode exaurir a memória do computador antes de obter a solução. Mais cobre sua complexidade de espaço pode ser visto em [3].

5. Análise Experimental

Como mostrado na Seção 2, todas as heurísticas são consistentes, e portanto, levarão a solução ótima. Entretanto, como dão custos diferentes para um mesmo estado, a ordem de expansão pode ser diferente e nesse caso, a quantidade de estados expandidos pode diferir.

Uma heurística é considerada melhor que outra se encontra o resultado com um menor número de expansão de estados, o que demanda menos tempo e memória. Abaixo para o caso de um N-puzzle de dimensão 3 tem-se uma comparação dos resultados das 3 heurísticas utilizadas:

Tabela 2. Análise Experimental N-puzzle dimensão 3

Estado Inicial				D. Manhattan	D. Hamming	D. Euclidiana
8	1	3	Movimentos	25	25	25
6	7		Tempo	0.142s	2010.232s	3.976s
5	2	4	Expansão	3319	93210	19598
2	4	6	Movimentos	26	26	26
8		1	Tempo	0.194s	7,701s	1.305s
3	5	7	Expansão	2681	17198	7627
2		6	Movimentos	27	27	27
7	8	1	Tempo	0.240s	16.454s	1.629s
5	4	3	Expansão	2469	20200	8736
8	7	6	Movimentos	28	28	28
5	4	3	Tempo	0.036s	21.770s	2.301s
2	1		Expansão	357	21894	8686

Como a Tabela 2 mostra, a heurística que apresentou os melhores resultados foi a D.Manhattan, pois foi a que precisou expandir o menor número de estados para obter a solução, o que reflete diretamente no seu tempo. Já a com os piores resultados foi a D. Hamming, pois devido ao grande número de estados expandidos, ela teve um maior consumo de tempo e memória.

São exemplos de estados sem solução os estados iniciais presentes na Tabela 3.

Tabela 3. Estados sem solução N-puzzle dimensão 3

Estado Inicial 1			Estado Inicial 2			Estado Inicial 3		
1	2	3		3	6	2		6
6	5	4	1	4	7	7	8	1
	7	8	2	5	8	5	3	4

Para dimensões maiores, a qualidade das heurísticas da Distância de Hamming e Distância Euclidiana cai muito, principalmente quando o caminho até a solução final têm mais que 20 passos. Foram feitas análises com N-pastilhas de dimensão 4 utilizando a Distância de Manhattan das configurações iniciais mostradas na Tabela 4.

Tabela 4. Análise Experimental N-puzzle dimensão 4

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

O Estado Inicial 1 tem solução com 35 movimentos, levou 9.035s e expandiu 24156 estados. Já o Estado Inicial 2 levou 2.278s, expandiu 12898 estados e encontrou a

solução final com 28 movimentos. A Tabela 5 mostra dois estados iniciais sem solução de dimensão 4.

Tabela 5. Estados sem solução N-puzzle dimensão 4

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

A análise experimental foi feita no computador "jumento"da sala 3017, com 3.9GB de memória RAM, processador Intel(R) Core(TM)2 Duo CPU E7200 @2.53 GHz, com o sistema operacional Ubuntu 10.04.

6. Uso e compilação

Conforme descrito no enunciado do trabalho prático, os programas podem ser compilados através de *bash compile.sh* e executado com *bash execute.sh input.txt* , sendo que no input.txt é dado o estado inicial. Como saída, tem-se o arquivo output.txt, que na primeira linha mostra a quantidade de movimentos e nas demais o caminho feito.

O programa foi testado no computador "jumento"disponível do laboratório da pós graduação na sala 3017. O compilador utilizado foi o g++ (no computador testado o gcc não estava disponível).

Os programas fazem isso das bibliotecas include iostream, fstream, sstream, vector, math.h, iterator, map e algorithm.

7. Conclusão

A proposta do 2º Trabalho Prático era criar um algoritmo para obter a solução do problema da N pastilha de maneira ótima usando o A*. Como pode ser visto na modelagem e no algoritmo desenvolvido nas Seções 2 e 3, essa meta foi atingida.

Na Seção 4 mostra-se que a complexidade de tempo e de espaço do algoritmo no pior caso é exponencial. A complexidade de tempo é reduzida com o uso de boas heurísticas, mas a complexidade de espaço não, o que pode exaurir a memória do computador antes de obter o caminho ótimo para o estado final. Para dimensão 3 não houveram problemas com a memória e os resultados foram obtidos de maneira relativamente rápida, mas para dimensões maiores ou iguais que 4 quando a configuração inicial demandava uma expansão muito grande dos estados esses problemas passaram a ser observados.

Uma comparação entre as heurísticas utilizadas é mostrada na Seção 5. Nessa seção foi possível constatar que a melhor heurística é a Distância de Manhattan, a segunda melhor é a Distância Euclidiana e a com pior desempenho a Distância de Hamming. Além disso, quanto maior a dimensão do puzzle, mais demorado se torna achar sua resolução quando ela existe.

De modo geral, acredito que esse trabalho prático possui grande relevância por tratar de um problema de complexidade exponencial e que fazem o uso de heurísticas. Em

particular, faz notar como é importante a administração da memória para que o algoritmo exaurir a memória do computador o mais tarde possível para conseguir obter a solução de problemas com caminhos longos/profundos.

Referências

[1] R. B. Games. Introduction to a*, disponıvel em: http://www.redblobgames.com/pathfinding/a-star/introduction.html. 2004.

[2] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[3] R. E. Korf. *Artificial intelligence search algorithms*. Chapman & Hall/CRC, 2010.

[4] D. Nayak. *Analysis and Implementation of Admissible Heuristics in 8 Puzzle Problem*. PhD thesis, 2014.

[5] J. Pearl. Heuristics: intelligent search strategies for computer problem solving. 1984.

[6] M. Ryan. Software workshop java, solvability of the tiles game, disponıvel em: http: // www. cs. bham. ac. uk/ mdr. 2004.