Análise de complexidade de um algoritmo de sobreposição de arvores ternarias

Cezar Soares[1], **Fernando Trindade**[1]

[1]Faculdade de Informática
Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Av. Ipiranga, 6681 – Partenon – Porto Alegre – RS – Brasil

Resumo. *O objetivo deste trabalho é determinar a complexidade de um algoritmo que permite fazer a sobreposição entre arvores ternarias utilizando os nodos centrais. Na teoria a complexidade do algorítimo vai depender dos valores de entrada e de como estão distribuídos em forma de matriz e pode ser descrita como*

$$O(N)$$

.

1. Introdução

O objetivo deste trabalho é determinar a complexidade de um algoritmo que permite a sobreposição de arvores binarias transformando as em ternarias baseando nos nodos centrais. A tarefa proposta apresenta três etapas: (a) desenvolver uma implementação do algoritmo, (b) determinar a complexidade do algoritmo utilizado um contador de iterações implementado no método de sobreposição das arvores, (c) demonstrar através de gráfico e tabela de desempenho do algoritmo implementado. O desafio foi proposto pela SBC em 2016 letra F ao qual a solução envolvia fundir arvores[de Computação 2016].

A conclusão é de que a complexidade depende dos valores de entrada (N - total de nodos das arvores e S - valor de saída esperado apos a sobreposição das arvores) e da distribuição dos nodos em forma de uma matriz pode ser descrita como

$$O(N)$$

ou seja quanto maior o tamanho da entrada e o valor esperado da saída sera levemente aumentado o numero de iterações.

2. Metodologia

Foi realizada a codificação em linguagem Java para a criação de tabela e gráfico para analise de desempenho e complexidade do algorítimo incluindo testes unitários utilizando a ferramenta JUnit para verificação de funcionamento do algorítimo.

3. Implementação da codificação

A implementação foi desenvolvida na linguagem Java e sera demostrado logo abaixo.

4. Algoritmo

Algorítimo utilizado para gerar os cálculos de complexidade.

```
1 Algorítimo implementado para função de sobreposição das arvores, ou seja
   ,algorítimo principal que calcula o maior caminho central :
   [soares and Trindade 2017].
2 Algorítimo principal que calcula o maior caminho central : public int
   getMaiorCaminhoCentral()  add(2); int contadorInicial =
   getCaminhoCentralRaiz(); add(4); if (contadorInicial ==
   matrizArvore.length - 1)  add(1); return contadorInicial;  add(4); int
   contadorCaminho = 1; int contadorNodo = 0; numeroNodo++;
3 while (contadorNodo ¡ matrizArvore.length)  add(2); while
   (matrizArvore[numeroNodo][2] != 0)  add(4);
4 contadorCaminho++; numeroNodo = matrizArvore[numeroNodo][2]; add(4);
   while (true)
5 if (matrizArvore[numeroNodo][2] != 0)  add(4); contadorCaminho++;
   numeroNodo = matrizArvore[numeroNodo][2]; add(5);  else break;  if
   (matrizArvore[numeroNodo][3] != 0  contadorInicial ¡ contadorCaminho)
   add(6); contadorInicial = contadorCaminho; numeroNodo++;
   contadorCaminho = 1; add(4);  if (matrizArvore[numeroNodo][3] == 0
   contadorInicial ¡ contadorCaminho)  add(6); contadorInicial =
   contadorCaminho; add(1);  if (matrizArvore[numeroNodo][3] != 0
   contadorInicial ¿ contadorCaminho)  add(6); numeroNodo++;
   contadorCaminho = 1; add(3);  contadorNodo = numeroNodo; add(1);
   contadorNodo++; contadorCaminho = 1; numeroNodo = contadorNodo;
   add(4);  add(1);
6 return contadorInicial;
```

Gráfico para analise de complexidade (Figura 1).

No gráfico de dispersão foi utilizado a linha de tendencia linear, demostrando também o R, o Y para verificação do calculo de complexidade.

4.1. Análise

Utilizando de uma tabela Determinamos a classe de complexidade, utilizando a análise dos dados de saída, contadores no código implementado.

Tabela de Analise		
N - Total de nodos	S - Tamanho da saída	Contador de iteração
14	11	104
8	6	104
5	3	118
2	1	118
3	2	118
10	8	213
20	18	368

Analisando a tabela podemos ver um crescimento linear das iterações, onde além de depender do total de nodos das arvores, depende também da distribuição dos nodos

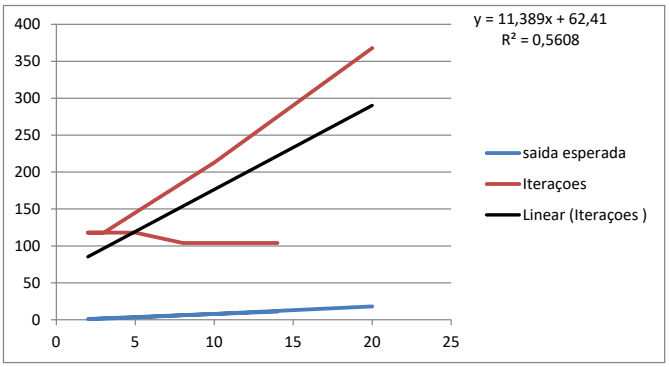

Figura 1. Gráfico de dispersão

Figura 2. Tabela de analise

centrais nela, mesmo para os casos de grandes entradas de N nodos na arvore, o que pesou mais foi a sua destruição.

5. Conclusão

Concluímos que apesar de depender do tamanho das arvores canhotas e destras e do maior caminho central a ser buscado o nosso algorítimo manteve o grau de complexidade de

$$O(N)$$

no pior caso.O fato de ser limitado aos nodos centrais para que haja a sobreposição influenciou de forma significativamente nas iterações visto que dependia muito da distribuição dos mesmos. Pudemos observar que mesmo em entadas(Tamanho das arvores semelhantes) porem com distribuição diferente dos nodos centrais em alguns casos houve um aumento e em outros uma leve diferença para menos no numero de iterações pra geração de uma arvore ternaria. o grau de complexidade somente pode ser reduzido com uma formula matemática especifica que talvez possa chegar ao grau de complexidade menor ou uma boa otimização do nosso código [Cormen et al. 2001].

Referências

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to algorithms*, volume 6. MIT press Cambridge.

de Computação, S. B. (2016). Maratona de programação 2016. http://maratona.ime.usp.br/hist/2016/primeira-fase/maratona.pdf.

soares, C. and Trindade, F. (2017). Trabalho final de alpro iii. https://github.com/FernandoDevGit/Trabalho-Final-Alpro$_{I}II.git$.