



PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Instituto de Ciências Exatas e de Informática

Nathan Gonçalves de Oliveira¹

Resumo

Em grafos direcionados há o fecho transitivo direto, representando o conjunto de vértices que podem ser alcançados a partir dele, e o fecho transitivo indireto, que é o conjunto de vértices que o alcançam. Será implementado métodos para identificar o fecho transitivo direto e indireto em alguns grafos, a partir de um método naive, e do método de Warshall. No final, haverá detalhes das implementações, experimentos e resultados obtidos.

¹Aluno de Graduação do curso de Engenharia de SoftwareAluno do Programa de Graduação em Engenharia de Software, Brasil – nathan.oliveira@sga.pucminas.br.

1 INTRODUÇÃO

Neste trabalho, serão implementados dois métodos para determinar fechos transitivos diretos e inversos em grafos direcionados: um método naive de busca por vértices e o método de Warshall. A base e a antibase de um grafo também serão determinadas usando essas estratégias. Experimentos serão realizados para avaliar o tempo médio gasto por cada estratégia em grafos aleatórios com diferentes tamanhos.

2 DESENVOLVIMENTO

O trabalho foi realizado utilizando a linguagem Java. Primeiramente foi realizado o método para a criação de grafos aleatórios. O usuário pode escolher o grafo de 100, 1.000, 10.000 ou 100.000 vértices. Após isso, o programa salva o grafo gerado em um arquivo .txt, apenas para conferência, com o nome de "GRAFOx.txt" (Este "x" é variável, e depende do número de vértices que o usuário escolheu). Após isso, o programa interage com o usuário novamente, perguntando se o mesmo gostaria de mostrar o fecho transitivo direto utilizando o método naive de busca em Largura ou o indireto, perguntando novamente o método que gostaria.

O método naive de Busca em Largura foi escolhido apenas por escolha do aluno, que possuía certa curiosidade em saber como este método funcionaria desde grafos pequenos até grandes, visto que a busca em largura utiliza uma abordagem de nível por nível, visitando todos os vértices a uma distância k antes de avançar para os outros vértices. Além disso, precisamos ter em mente a complexidade do método naive de Busca em Largura que é de $O(V + E)$, onde V é o número de vértices e E é o número de arestas, no pior caso, ou seja, quando o método passa por todos os outros vértices antes de chegar no final.

Este relatório em TeX foi realizado conforme o Modelo Canônico de Artigo do ICEI, da PUC Minas.

2.1 Método Naive - Busca em Largura - Breve explicação

A Busca em Largura é um algoritmo utilizado para percorrer ou buscar elementos em uma estrutura de dados em forma de grafo, como um grafo não direcionado ou um grafo direcionado. O algoritmo explora os vértices do grafo em camadas, começando pelo vértice inicial e expandindo para os vértices adjacentes antes de avançar para os vizinhos subsequentes. A BFS utiliza uma fila para armazenar os vértices a serem visitados, garantindo que os vértices adjacentes sejam visitados em ordem de proximidade ao vértice inicial. Esse algoritmo é amplamente utilizado para várias aplicações, como encontrar o caminho mais curto entre dois vértices, verificar a conexidade de um grafo, entre outros.

2.1.1 Método de Warshall - Breve explicação

O algoritmo de Warshall é um algoritmo utilizado para determinar o fecho transitivo de um grafo direcionado. O fecho transitivo de um grafo é o conjunto de todos os pares de vértices que possuem um caminho entre eles. O algoritmo recebe uma matriz de adjacência do grafo como entrada e realiza uma série de iterações para atualizar a matriz, de forma a incluir todas as arestas que indiretamente conectam dois vértices. Ele utiliza uma abordagem dinâmica, onde a matriz de adjacência é atualizada iterativamente com base nas conexões diretas e indiretas entre os vértices. O algoritmo de Warshall é eficiente para encontrar o fecho transitivo em um grafo e é amplamente utilizado em diversas aplicações, como análise de dependências, análise de fluxo em redes, entre outros.

3 RESULTADOS OBTIDOS

Abaixo há a discussão sobre os resultados obtidos durante a elaboração do trabalho, comparando o tempo de execução, consumo da CPU e memória.

Figura 1 – Eclipse em modo de espera

Processos					
Nome	Status	4% CPU	61% Memória	0% Disco	0% Rede
Aplicativos (4)					
> eclipse		0%	586,8 MB	0 MB/s	0 Mbps
> Gerenciador de Tarefas		0,4%	77,9 MB	0,1 MB/s	0 Mbps
> Google Chrome (43)		0%	1.909,8 MB	0 MB/s	0 Mbps
> Steam (32 bits) (9)		0,1%	356,1 MB	0 MB/s	0 Mbps

Foi utilizada a IDE Eclipse na elaboração dos códigos do trabalho. A CPU do sistema é o AMD Ryzen 5 5600X, com 16GB de memória RAM. Enquanto o Eclipse está em idle, ele consome 586MB. Consumo razoável, se formos considerar os outros programas que estão abertos, como a Steam.

Na execução do grafo de 100 vértices utilizando o método naive de Busca em Largura, a execução foi em aproximadamente 4 segundos. Já para o método de Warshall foi de 8 segundos. Se fôssemos seguir essa lógica, o consumo de memória deveria ser dobrado. Curiosamente, houve apenas um acréscimo de 22MB do uso de memória.

Processos		6%	63%	0%	0%
Nome	Status	CPU	Memória	Disco	Rede
Aplicativos (5)					
> eclipse		0,4%	622,0 MB	0,1 MB/s	0 Mbps
> Gerenciador de Tarefas		1,1%	84,0 MB	0 MB/s	0 Mbps
> Google Chrome (43)		0%	2.059,2 MB	0,1 MB/s	0 Mbps
> Paint		0%	43,5 MB	0 MB/s	0 Mbps
> Steam (32 bits) (9)		0,5%	353,7 MB	0 MB/s	0 Mbps

Figura 2 – Naive - 100 Vértices

Processos		8%	63%	1%	0%
Nome	Status	CPU	Memória	Disco	Rede
Aplicativos (5)					
> eclipse		1,7%	641,0 MB	0 MB/s	0 Mbps
> Gerenciador de Tarefas		1,1%	85,0 MB	0,1 MB/s	0 Mbps
> Google Chrome (45)		0%	2.070,2 MB	0,1 MB/s	0 Mbps
> Paint		0%	46,5 MB	0 MB/s	0 Mbps
> Steam (32 bits) (9)		0,7%	353,8 MB	0,1 MB/s	0 Mbps

Figura 3 – Warshall - 100 Vértices

Na execução do grafo de 1000 vértices, utilizando o método naive, o tempo de execução foi aumentado de forma gritante. A execução levou 97 segundos, ou 1 minuto e 37 segundos. O método de encontrar o fecho transitivo indireto foi o que mais levou tempo para ser executado. Nessa execução do algoritmo, foi consumido aproximadamente 800MB de memória. Agora, na execução do método de Warshall, a execução do fecho transitivo direto levou muito mais tempo do que a do método naive. A execução do método de Warshall foi iniciada às 19:36. Uma hora foi decorrida, e o programa ainda não tinha concluído a primeira tarefa de encontrar o fecho transitivo direto.

Figura 4 – Naive - 1000 Vértices

Processos		16%	63%	0%	0%
Nome	Status	CPU	Memória	Disco	Rede
Aplicativos (5)					
> eclipse (3)		12,1%	838,8 MB	0 MB/s	0 Mbps
> Gerenciador de Tarefas		0,5%	94,3 MB	0 MB/s	0 Mbps
> Google Chrome (43)		0,3%	2.102,0 MB	0,1 MB/s	0 Mbps
> Paint		0%	46,2 MB	0 MB/s	0 Mbps
> Steam (32 bits) (9)		0%	308,6 MB	0 MB/s	0 Mbps

O método naive para o grafo de 10.000 vértices. A busca do fecho transitivo direto foi realizada em 10 minutos, e utilizou quase 3GB de memória. Após mais 20 minutos, o programa realizou a tarefa de encontrar o fecho transitivo indireto, utilizando 3,5GB de memória nesse processo. Na tentativa de encontrar o fecho transitivo direto utilizando o método de Warshall, o programa foi iniciado,tendo um gasto de memória parecido com o do método naive de Busca em Largura. A execução foi iniciada às 19:30, e após decorridas 1h e 10min, o programa ainda estava executando o método para encontrar o fecho transitivo direto.

Figura 5 – Naive - 1000 Vértices

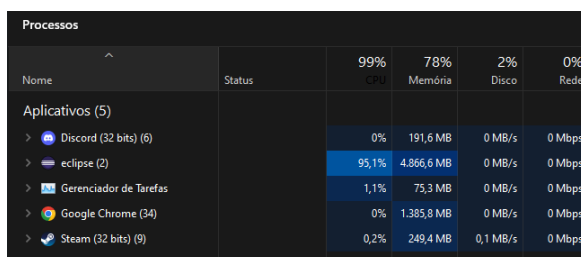
Processos		30%	62%	1%	0%
Nome	Status	CPU	Memória	Disco	Rede
Aplicativos (4)					
> eclipse (2)		13,6%	2.632,2 MB	31,4 MB/s	0 Mbps
> Gerenciador de Tarefas		1,7%	73,8 MB	0,1 MB/s	0 Mbps
> Google Chrome (27)		0%	1.020,8 MB	0,1 MB/s	0 Mbps
> Steam (32 bits) (9)		0%	426,7 MB	0 MB/s	0 Mbps

Agora a execução do grafo de 100.000 vértices foi desastrosa. O programa simplesmente não conseguiu ser executado. Tentei executar no Eclipse de minha máquina e no Replit (<https://replit.com/@nathanoli/Trabalho-GRAFOS?v=1>). Na minha máquina, foi utilizado incríveis 95,1% de uso de CPU, e aproximadamente, 4GB de memória foram utilizados. Houve o erro (java.lang.Out Of Memory Error: Java heap space), significando que o programa não tem memória suficiente disponível no heap para alocar um novo objeto ou armazenar os dados necessários para a execução do programa. Já na tentativa de executar pelo Replit, houve o mesmo erro, além de mostrar o alerta de que tanto a CPU, quanto a Memória estavam no limite.

3.0.1 Comentários

Podemos perceber que, para grafos relativamente pequenos (100 e 1000 vértices) o programa é executado de maneira relativamente rápida, tanto no método naive de Busca em Largura quanto para o método de Warshall. Já para o grafo de 10.000 vértices a situação muda um pouco. Para encontrar o fecho transitivo direto pelo método naive, o tempo de busca é relativamente aceitável, com cerca de 10 minutos. A tarefa inversa também leva um tempo aceitável. A tarefa de localizar o fecho transitivo direto no mesmo grafo de 10.000 vértices utilizando o método de Warshall porém não foi bem sucedida. Por algum motivo o programa demorou muito para realizar a busca do fecho transitivo direto, não finalizando a tarefa após uma hora.

Agora, a execução do fecho transitivo direto tanto para o método naive de busca em largura quanto para o método de Warshall, no grafo de 100.000 vértices não foi bem sucedida, pelo contrário, o programa nem conseguiu ser iniciado. Após digitar o número de vértices que o usuário deseja, o programa cria e salva o grafo em um arquivo .txt, apenas para conferência



Nome	Status	99% CPU	78% Memória	2% Disco	0% Rede
Discord (32 bits) (6)		0%	191,6 MB	0 MB/s	0 Mbps
eclipse (2)		95,1%	4.866,6 MB	0 MB/s	0 Mbps
Gerenciador de Tarefas		1,1%	75,3 MB	0 MB/s	0 Mbps
Google Chrome (34)		0%	1.385,8 MB	0 MB/s	0 Mbps
Steam (32 bits) (9)		0,2%	249,4 MB	0,1 MB/s	0 Mbps

Figura 6 – Eclipse - 100.000 Vértices

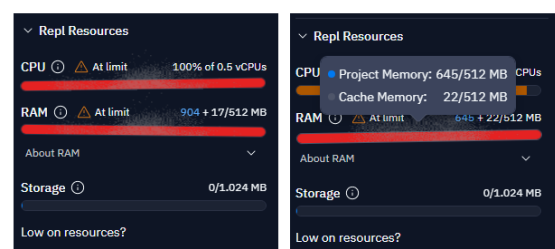


Figura 7 – Replit - 100.000 Vértices

4 CÓDIGOS

Figura 8 – Classe Main do trabalho

```

1 import java.util.ArrayList;
2 import java.util.Random;
3 import java.util.Scanner;
4
5 public class Main {
6
7     public static void main(String[] args) {
8
9         long start = System.currentTimeMillis();
10         Scanner scanner = new Scanner(System.in);
11         System.out.println("Digite o número de vértices do grafo (100, 1000, 10000, ou 100000): ");
12         int numVertices = scanner.nextInt();
13
14         Grafo grafo = new Grafo(numVertices);
15         grafo.determinaBase();
16         grafo.determinaAdjacencias();
17
18         Random random = new Random();
19         for (int i = 0; i < numVertices; i++) {
20             for (int j = i + 1; j < numVertices; j++) {
21                 if (random.nextBoolean()) {
22                     grafo.adicionaAresta(i, j);
23                 }
24             }
25         }
26
27         String mensagem = String.format("Grafo de %d vértices", numVertices);
28         System.out.println(mensagem + " criado com sucesso!");
29         grafo.salvaGrafo(nomeArquivo);
30
31         System.out.println("Digite o método para cálculo do fecho transitivo direto (1 - Naive, 2 - Marshall): ");
32         int opcao = scanner.nextInt();
33
34         System.out.println("Fecho transitivo direto");
35         ArrayList<Integer> fechoIndireto = new ArrayList<>(numVertices);
36         if (opcao == 1) {
37             for (int i = 0; i < numVertices; i++) {
38                 fechoIndireto(i) = grafo.determinaFechoTransitivoDiretoNaive(i);
39             }
40         } else if (opcao == 2) {
41             boolean[][] matrizAdjacenciaBoolean = new boolean[numVertices][numVertices];
42             for (int i = 0; i < numVertices; i++) {
43                 for (int j = 0; j < numVertices; j++) {
44                     matrizAdjacenciaBoolean[i][j] = grafo.temAresta(i, j);
45                 }
46             }
47             fechoIndireto = Marshall.determinaFechoTransitivoDiretoMarshall(matrizAdjacenciaBoolean);
48         } else {
49             System.out.println("Opção inválida");
50             scanner.close();
51             return;
52         }
53         imprimeLista(fechoIndireto);
54
55         System.out.println("Nome(s) do(s) grafo: " + grafo.determinaBase();
56         System.out.println("Adjacências: " + grafo.determinaAdjacencias());
57
58         scanner.close();
59         long end = System.currentTimeMillis();
60         long totalTime = end - start;
61         System.out.println("Tempo total de execução: " + totalTime/1000.0 + " segundos");
62
63     }
64
65     private static void imprimeLista(ArrayList<Integer> lista) {
66         for (int i = 0; i < lista.length; i++) {
67             System.out.print(i + " ");
68             for (int j = 0; j < lista.size(); j++) {
69                 System.out.print(lista.get(j) + " ");
70             }
71             System.out.println();
72         }
73     }
74 }

```

Figura 9 – Classe Grafo (Método Naive) do trabalho

[illegible]

Figura 10 – Classe Grafo (Método Naive) do trabalho

```

3 public class Marshall {
4
5     public static ArrayList<Integer>[] determinarFechoTransitivoDiretoMarshall(boolean[][] matrizAdjacencia) {
6         int numVertices = matrizAdjacencia.length;
7         ArrayList<Integer>[] fechosDiretos = new ArrayList[numVertices];
8
9         // Inicialização do fecho transitivo direto
10        for (int i = 0; i < numVertices; i++) {
11            fechosDiretos[i] = new ArrayList<Integer>();
12            for (int j = 0; j < numVertices; j++) {
13                if (matrizAdjacencia[i][j]) {
14                    fechosDiretos[i].add(j);
15                }
16            }
17        }
18
19        // Cálculo do fecho transitivo direto utilizando o algoritmo de Marshall
20        for (int k = 0; k < numVertices; k++) {
21            for (int i = 0; i < numVertices; i++) {
22                for (int j = 0; j < numVertices; j++) {
23                    if (!matrizAdjacencia[i][j] && matrizAdjacencia[i][k] && matrizAdjacencia[k][j]) {
24                        matrizAdjacencia[i][j] = true;
25                        fechosDiretos[i].add(j);
26                    }
27                }
28            }
29        }
30
31        return fechosDiretos;
32    }
33
34    public static ArrayList<Integer>[] determinarFechoTransitivoInversoMarshall(boolean[][] matrizAdjacencia) {
35        int numVertices = matrizAdjacencia.length;
36        ArrayList<Integer>[] fechosIndiretos = new ArrayList[numVertices];
37
38        // Inicialização do fecho transitivo indireto
39        for (int i = 0; i < numVertices; i++) {
40            fechosIndiretos[i] = new ArrayList<Integer>();
41            for (int j = 0; j < numVertices; j++) {
42                if (matrizAdjacencia[j][i]) {
43                    fechosIndiretos[i].add(j);
44                }
45            }
46        }
47
48        // Cálculo do fecho transitivo indireto utilizando o algoritmo de Marshall
49        for (int k = 0; k < numVertices; k++) {
50            for (int i = 0; i < numVertices; i++) {
51                for (int j = 0; j < numVertices; j++) {
52                    if (!matrizAdjacencia[i][j] && matrizAdjacencia[k][j] && matrizAdjacencia[i][k]) {
53                        matrizAdjacencia[i][j] = true;
54                        fechosIndiretos[i].add(j);
55                    }
56                }
57            }
58        }
59
60        return fechosIndiretos;
61    }
62 }

```

5 CONCLUSÃO

A partir da elaboração deste trabalho, pude perceber que, para grafos menores, basicamente qualquer método de busca pode ser usado. Já para grafos muito maiores, nem qualquer tipo de busca pode ser utilizado. Com o avanço da tecnologia, haverão métodos mais eficazes de realizar buscas em grafos muito grandes.

6 REFERÊNCIAS

BEAMER, Scott. Direction-optimizing breadth-first search. Disponível em: <<https://dl.acm.org/doi/10.5555/2388996.2389013>>. Acesso em 23 Abr. 2023.

GEEKSFORGEEEKS. Transitive closure of a graph. Disponível em <<https://www.geeksforgeeks.org/transitive-closure-of-a-graph/>>. Acesso em 30 Abr. 2023.

ICEI PUC Minas. Modelo Canônico de Artigo - ICEI PUC Minas. Disponível em: <<https://www.overleaf.com/latex/templates/modelo-canonical-de-artigo-icei-puc-minas/xbhfbmvvhjkv>>. Acesso em 30 Abr. 2023.

JUNIOR, Mário Sérgio Ferreira Alvim. Projeto e Análise de Algoritmos 2º Trabalho Prático. Disponível em <<https://homepages.dcc.ufmg.br/nivio/cursos/pa06/tp2/tp22/tp22.pdf>>. Acesso em 22 Abr. 2023.

SCIENCEDIRECT. Naïve Approach. Disponível em <<https://www.sciencedirect.com/topics/computer-science/naive-approach>> Acesso em 22 Abr. 2023.

WARSHALL, Stephen. A Theorem on Boolean Matrices. Disponível em <<https://dl.acm.org/doi/10.1145/321105.321107>>. Acesso em 30 Abr. 2023.