PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
BACHARELADO EM ENGENHARIA DE SOFTWARE
DISCIPLINA DE ALGORITMOS E ESTRUTURAS DE DADOS II

Algoritmo de Menor Caminho em Grafo
"As Contas Conjuntas"

GABRIEL FERREIRA KURTZ
gabriel.kurtz@acad.pucrs.br

30 de outubro de 2017

Abstract

This article demonstrates an algorithm that can find the shortest path in a graph where some nodes are linked, representing a set of bank accounts with multiple owners. A transfer must be made between two persons, but the path between two accounts must use always accounts that are co-owned by the same person.

It was developed as the second evaluated task for the Algorithms II class.

1. INTRODUÇÃO

Para este trabalho, deve-se solucionar um problema onde é necessário transferir dinheiro entre as contas bancárias de duas pessoas. No entanto, supõe-se que o governo cobre uma taxa sempre que a conta do destinatário for de uma titularidade diferente da origem.

Felizmente, todas as contas utilizadas no exemplo são conjuntas, sendo possível evitar o pagamento desta taxa fazendo com que o dinheiro chegue ao indivíduo do destino utilizando diversas transferências entre contas que possuem um titular em comum.

O problema pode ser interpretado como um grafo, onde os correntistas são nodos e as contas conjuntas representam arestas. Para reproduzir esta estrutura de grafos, foi desenvolvido um algoritmo na linguagem Java, que será apresentado ao longo do trabalho.

2. OBJETOS

Há basicamente dois objetos relevantes neste problema: as Pessoas, representando os nodos, e as Contas, que são as arestas que unem Pessoas. Cada um destes objetos recebeu sua própria classe no algoritmo, com os atributos necessários. A classe dos nodos recebeu os seguintes atributos:

```
public class Pessoa {
	public String nome;
	public HashSet<String> conjuntas;
	public String anterior;
	public int distancia;

	public Pessoa(String nome) {
		this.nome = nome;
		this.conjuntas = new HashSet<String>();
		this.anterior = "";
		this.distancia = -1;
	}
	public void addConjunta(String nome) {
		conjuntas.add(nome);
	}
```

- nome(String): É o nome da pessoa.
- conjuntas (HashSet<String>): É um conjunto dos nomes das pessoas com quem o objeto possui contas conjuntas, ou seja, seus vizinhos de arestas.
- anterior(String): Permite gravar, em um caminhamento de qualquer espécie, qual o nodo anterior ao objeto em uma sequencia que busca o menor caminho, da mesma forma como o algoritmo de Dijkstra por exemplo.
- distancia(int): No tipo de caminhamento que foi realizado, o atributo distancia é apenas uma marca que indica se aquele nodo já foi visitado ou não. Optamos por mantê-lo como int para ter a flexibilidade de implementar outros tipos de caminhamentos na solução do problema caso fosse necessário.

O único método desta classe, addConjunta, serve para tornar o código um pouco mais legível quando é necessário adicionar novas pessoas ao HashSet de vizinhos (conjuntas).

Já a classe das arestas, chamada de Conta, recebeu os seguintes atributos:

- numero(int): O número da conta corrente.
- titulares(HashSet<String>): Os titulares daquela conta, ou seja, os nodos que são unidos por aquela aresta. Optamos por gravar esta informação no formato String em vez de utilizar diretamente as iterações de Pessoa por acreditar que torna o algoritmo mais leve, além de aumentar a independência entre as classes. A partir da String do nome, é possível buscar o objeto Pessoa em um dicionário.

```
public class Conta {
	public int numero;
	public HashSet<String> titulares;

	public Conta(int numero) {
		this.numero = numero;
		titulares = new HashSet<String>();
	}
	public void addTitular(String nome) {
		titulares.add(nome);
	}
```

Novamente, o único método serve para acrescentar elementos ao HashSet da classe sem tornar o código excessivamente poluído.

O problema de encontrar o menor caminho entre a origem e o destino da transferência poderia ser resolvido somente com a classe Pessoa, mas neste caso perder-se-ia a informação do número da Conta, que identifica cada aresta, e foi solicitado no enunciado. Esta foi a única função da classe Conta na realidade.

3. ARQUIVOS DE ENTRADA DE DADOS

A entrada de dados, em formato de texto, possui na primeira linha um número que indica o total de contas correntes naquele caso. Em seguida, um conjunto de linhas onde o primeiro elemento de cada linha é o número da conta, seguida dos nomes de ambos os correntistas (cada conta possui exatamente dois correntistas).

```
1  100
2  54149 José_Antônio Gabriel
3  88122 Lucas Marcelo
4  47206 Alves Melo
5  84608 Dias Lúcia
6  24112 Marco_Antônio Alexandre
7  44701 Andréia Pinto
8  64377 Guilherme Marisa
9  15168 André_Luiz Vagner
10 67487 Maria_de_Lurdes Vagner
11 33571 Gonçalves Ferreira
12 38259 Vagner Maria_Helena
13 24631 Lima Dias
14 13009 Rosane Vera_Lúcia
15 60089 Tiago Beatriz
```

Figura 1: Exemplo de entrada de dados

A última linha não representa uma conta e, portanto, não possui número. Ela indica o nome do correntista da origem da transferência e quem é o destinatário.

Para ler os dados, o algoritmo utilizou o BufferedReader e o Scanner da linguagem java.

4. ESTRUTURAS DE DADOS

O objeto Pessoa, com indicação dos nodos vizinhos no atributo conjuntas, já chega bem próximo de criar uma estrutura encadeada que realize o caminhamento que desejamos. O único detalhe é que a informação dos vizinhos foi criada com o formato String. Assim, cada objeto pode ser criado e manipulado integralmente, sem que existam ainda os nodos vizinhos. No entanto, para atingir estes vizinhos no caminhamento, é necessário converter esta informação do nome em formato String no nodo correspondente.

Foi utilizado um dicionário no formato HashMap<String, Pessoa> para completar esta estrutura:

```
HashMap<String, Pessoa> pessoas = new HashMap<String, Pessoa>();
```

Cada nome está mapeado ao objeto daquela Pessoa.

Além disso, ao final do algoritmo, será necessário converter cada aresta utilizada em uma informação que apresente o número da conta e seus titulares. Para possibilitar esta consulta, foi criado um HashSet<Conta> com todas as contas:

```
HashSet<Conta> contas = new HashSet<Conta>();
```

Também é necessário gravar os nomes da origem e do destino da transferência:

```
String nome1="", nome2="";
```

Com esta estrutura é possível ler e armazenar todos os dados do problema:

```
try (BufferedReader br = Files.newBufferedReader(input,
Charset.forName("utf8")))
{
	String linha = br.readLine();
	Scanner sc = new Scanner(linha).useDelimiter(" ");

	int totalContas = sc.nextInt();

	while((linha = br.readLine()) != null) {
		sc = new Scanner(linha).useDelimiter(" ");

		//Populando set de contas
		if(sc.hasNextInt()) {
			Conta contaNova = new Conta(sc.nextInt());
			contaNova.addTitular(sc.next());
			contaNova.addTitular(sc.next());
			contas.add(contaNova);
		}
		//Gravando nome de pessoas da transferencia
		else {
			nome1 = sc.next();
			nome2 = sc.next();
}	}	}
```

Primeiramente, o HashSet de Contas é populado com todas as contas do arquivo de origem, gravando ambos os titulares no objeto Conta correspondente, que em seguida é adicionado ao Set.

Quando chega na última linha do arquivo, os nomes de origem e destino são gravados nas variáveis nome1 e nome2.

É possível iterar o HashSet de Contas e a partir dele criar os objetos dos Nodos e também o seu dicionário:

```
for(Conta c : contas) {
	//Iterando titulares da Conta
	for(String titular : c.titulares) {
		//Se for a primeira conta da pessoa, cria a pessoa no dicionario e
acrescenta conta conjunta
		if(!pessoas.containsKey(titular)) {
			Pessoa novaPessoa = new Pessoa(titular);
			for(String titular2 : c.titulares) {
				if(titular2 != titular) {
					novaPessoa.addConjunta(titular2);
				}
			}
			pessoas.put(titular, novaPessoa);
		}
		//Se já tiver conta, simplesmente acrescenta a conjunta
		else {
			for(String titular2 : c.titulares) {
				if(titular2 != titular) {
				pessoas.get(titular).addConjunta(titular2);
				}
			}
		}
	}
}
```

Ao iterar as Contas, buscando os nomes de todas as pessoas do exercício, para cada nome encontrado há duas alternativas, dependendo se o objeto daquela Pessoa já foi criado ou não.

Caso o objeto ainda não exista, ele é criado e é indicado o coproprietário da conta no seu atributo conjuntas, e posteriormente este objeto é acrescentado ao dicionário pessoas.

Se o objeto já existe, basta localizá-lo no dicionário e acrescentar o nome do nodo vizinho(coproprietário).

5. CAMINHAMENTO

Neste momento, temos todos os dados dos arquivos lidos e armazenados. Pode-se iniciar a preparação para o caminhamento que buscará o menor caminho para a transferência.

Optamos por realizar um caminhamento “por bolhas”, e não por profundidade, para evitar a redundância no algoritmo. Partindo do indivíduo destino da transferência, marcamos ele como distância 1 (na verdade estamos utilizando o atributo distância apenas como uma marca que indica se o algoritmo já foi visitado ou não: o atributo inicia em -1, e qualquer número positivo indica que o nodo já foi visitado). A partir daí, visitamos todos os vizinhos do nodo inicial e marcamos como distância 2. O mesmo ocorre com os vizinhos destes sucessivamente, até que se encontre o nodo da origem da transferência.

Para organizar a ordem das visitas, foi utilizada uma fila do tipo ArrayDeque do Java. Passamos os nomes da origem e do destino para objetos Pessoa e então acrescentamos o destino como o primeiro elemento da fila:

```
ArrayDeque<Pessoa> fila = new ArrayDeque<Pessoa>();
Pessoa pessoa1 = pessoas.get(nome1);
Pessoa pessoa2 = pessoas.get(nome2);

pessoa2.distancia = 1;
fila.add(pessoa2);
```

Agora basta iniciar o caminhamento:

```
lacoCaminha:
while(!fila.isEmpty()) {
      Pessoa p = fila.remove();

      for(String v : p.conjuntas) {
            Pessoa vizinho = pessoas.get(v);
            if(vizinho.distancia<0) {
                  vizinho.distancia = p.distancia + 1;
                  vizinho.anterior = p.nome;
                  fila.add(vizinho);
                  if(vizinho == pessoa1) break lacoCaminha;
            }
      }
}
```

Enquanto houver Pessoas na fila, o algoritmo pega a primeira pessoa da fila (a que está lá a mais tempo) e itera todos os vizinhos deste nodo. Para cada vizinho, sua distância para o nodo inicial é gravada como a distância do nodo anterior acrescida de 1. Grava-se o nome do nodo anterior como o que foi retirado por último da fila, que é o nodo que conduz até este vizinho. Por fim, acrescenta-se todos os vizinhos à fila, de modo a garantir que todos os vizinhos da bolha seguinte sejam visitados após o final da bolha atual.

Quando encontra-se o nodo que corresponde à pessoa de origem da transferência (pessoa1), pode-se encerrar este laço pois já se conhece todo o caminho que liga pessoa1 e pessoa2.

Basta, portanto, percorrer este caminho agora partindo da origem (pessoa1) até o destino e imprimir no terminal as informações de cada conta percorrida para satisfazer o enunciado. A informação da conta é buscada no HashSet contas, procurando contas cujos titulares contenham os nomes do nodo atual e do anterior.

Para isso, basta saltar de nodo para nodo utilizando a informação do nome guardado no atributo anterior, buscando o nodo correspondente a este nodo:

```
Pessoa aux1 = pessoa1;
Pessoa aux2 = pessoas.get(pessoa1.anterior);
while(aux1 != pessoa2) {
	lacoConta:
	for(Conta c : contas) {
		if(c.titulares.contains(aux1.nome) &&
c.titulares.contains(aux2.nome)) {
			System.out.println(c.numero + " - " + aux1.nome + " - " +
aux2.nome);
			aux1 = aux2;
			aux2 = pessoas.get(aux1.anterior);
			break lacoConta;
		}
	}
}
```

É fundamental interromper o laço marcado com a label lacoConta com o comando break ao localizar a conta desejada pois, ao encontrar a conta do nodo atual e do anterior, ele atualiza os auxiliares que contém o nodo atual e o anterior para os próximos valores antes de checar o while que repete a iteração. Portanto, ele seguiria pesquisando no mesmo HashSet com os novos valores até encerrar todas as contas. Caso na ordem do HashSet haja uma outra conta em que um dos titulares seja a pessoa2, pesquisaria posteriormente se o segundo titular é o aux2, que está apontado para null. Isto vai gerar um erro de Null Pointer, pois a pessoa2 não possui nodo anterior: ela é a primeira do caminhamento. Com o comando break, forçamos a conferência do while todas as vezes que se encontra uma conta, garantindo que o algoritmo não buscará o anterior nulo da pessoa2.

6. CASOS DE TESTE

As figuras a seguir demonstram os resultados dos testes do algoritmo com os casos propostos no exercício. O resultado do terminal demonstra o caminho que deve ser percorrido para transferir o valor desejado entre as contas utilizando somente contas conjuntas que possuam titulares em comum. Foi indicado também o tempo total de execução do algoritmo, que começa a contar depois que se informa o nome do arquivo com os dados de origem.

```
--- Iniciando Cronômetro ---
Total de Contas Correntes: 100
--- Inicializando Estrutura de Contas para arquivo caso01 ---
Pessoas: 116   ---   Contas: 100
Origem: Mara   ---   Destino: Cristina
----------------------------------------
47898 - Mara - Fernandes
79130 - Fernandes - Cristina
--- Tempo Total de Execução: 0.03 segundos ---
```

Figura 2: Caso de teste caso01

```
--- Iniciando Cronômetro ---
Total de Contas Correntes: 200
--- Inicializando Estrutura de Contas para arquivo caso02 ---
Pessoas: 160   ---   Contas: 200
Origem: Aline   ---   Destino: Freitas
----------------------------------------
6628 - Aline - Luis
53358 - Luis - Alexandre
96296 - Alexandre - Mara
53454 - Mara - Ana
36509 - Ana - José_Carlos
42335 - José_Carlos - Oliveira
67643 - Oliveira - Morais
32390 - Morais - Luciano
83075 - Luciano - Freitas
--- Tempo Total de Execução: 0.05 segundos ---
```

Figura 3: Caso de teste caso02

```
--- Iniciando Cronômetro ---
Total de Contas Correntes: 300
--- Inicializando Estrutura de Contas para arquivo caso03 ---
Pessoas: 162   ---   Contas: 300
Origem: Sabrina   ---   Destino: Adão
----------------------------------------
60809 - Sabrina - Marques
61227 - Marques - Melo
50188 - Melo - Terezinha
66406 - Terezinha - Adão
--- Tempo Total de Execução: 0.06 segundos ---
```

Figura 4: Caso de teste caso03

```
--- Iniciando Cronômetro ---
Total de Contas Correntes: 400
--- Inicializando Estrutura de Contas para arquivo caso04 ---
Pessoas: 172   ---   Contas: 400
Origem: Santos   ---   Destino: Ribeiro
-----------------------------------------
70409 - Santos - Caroline
6116 - Caroline - Ana_Maria
86391 - Ana_Maria - Natália
57631 - Natália - Ribeiro
--- Tempo Total de Execução: 0.07 segundos ---
```

Figura 5: Caso de teste caso04

```
--- Iniciando Cronômetro ---
Total de Contas Correntes: 500
--- Inicializando Estrutura de Contas para arquivo caso05 ---
Pessoas: 171   ---   Contas: 500
Origem: Antônio_Carlos   ---   Destino: Thiago
-----------------------------------------
1706 - Antônio_Carlos - João_Batista
62833 - João_Batista - Vagner
6121 - Vagner - Thiago
--- Tempo Total de Execução: 0.07 segundos ---
```

Figura 6: Caso de teste caso05

```
--- Iniciando Cronômetro ---
Total de Contas Correntes: 600
--- Inicializando Estrutura de Contas para arquivo caso06 ---
Pessoas: 172   ---   Contas: 600
Origem: Rosane   ---   Destino: Andréa
-----------------------------------------
66726 - Rosane - Karla
99220 - Karla - Luis
11871 - Luis - Andréa
--- Tempo Total de Execução: 0.07 segundos ---
```

Figura 7: Caso de teste caso06

```
--- Iniciando Cronômetro ---
Total de Contas Correntes: 700
--- Inicializando Estrutura de Contas para arquivo caso07 ---
Pessoas: 172   ---   Contas: 700
Origem: Pereira   ---   Destino: Caio
-----------------------------------------
56208 - Pereira - Andréa
71249 - Andréa - Oliveira
22521 - Oliveira - Caio
--- Tempo Total de Execução: 0.08 segundos ---
```

Figura 8: Caso de teste caso07

```
--- Iniciando Cronômetro ---
Total de Contas Correntes: 800
--- Inicializando Estrutura de Contas para arquivo caso08 ---
Pessoas: 172   ---   Contas: 800
Origem: Thiago   ---   Destino: Ribeiro
----------------------------------------
26875 - Thiago - Paulo_Roberto
58025 - Paulo_Roberto - Marli
5145 - Marli - Ribeiro
--- Tempo Total de Execução: 0.09 segundos ---
```

Figura 9: Caso de teste caso08

```
--- Iniciando Cronômetro ---
Total de Contas Correntes: 900
--- Inicializando Estrutura de Contas para arquivo caso09 ---
Pessoas: 172   ---   Contas: 900
Origem: José   ---   Destino: Mateus
----------------------------------------
13581 - José - Felipe
53354 - Felipe - Leonardo
84812 - Leonardo - Mateus
--- Tempo Total de Execução: 0.1 segundos ---
```

Figura 10: Caso de teste caso09

```
--- Iniciando Cronômetro ---
Total de Contas Correntes: 1000
--- Inicializando Estrutura de Contas para arquivo caso10 ---
Pessoas: 172   ---   Contas: 1000
Origem: Manoel   ---   Destino: Rosa
----------------------------------------
27284 - Manoel - Souza
25482 - Souza - Rosa
--- Tempo Total de Execução: 0.09 segundos ---
```

Figura 11: Caso de teste caso10

7. CONCLUSÃO

O algoritmo está realizando o que espera-se dele em um tempo bastante aceitável. Era um exercício um pouco mais simples do que o primeiro da disciplina, e acreditamos ter concluído com sucesso. Novamente houve boas oportunidades de aprendizado solucionando este problema.

ANEXO I – CÓDIGO DO ALGORITMO EM JAVA

```
//---Classe AsContasConjuntas
import java.io.BufferedReader;
import java.io.IOException;
import java.nio.charset.Charset;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayDeque;
import java.util.Deque;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Scanner;

public class AsContasConjuntas {
        public static void main(String args[]) {

                Scanner in = new Scanner(System.in);

                System.out.print("--- Pontificia Universidade Catolica do Rio Grande do Sul ---"
                                + "\n--- Bacharelado em Engenharia de Software ---"
                                + "\n--- Disciplina de Algoritmos e Estruturas de Dados II ---"
                                + "\n--- Prof. Joao Batista Oliveira ---"
                                + "\n--- Gabriel Ferreira Kurtz ---"
                                + "\n\n------------------------------------------------------------------------"
                                + "\n--- ALGORITMO DE MENOR CAMINHO EM ESTRUTURA DE CONTAS CONJUNTAS ---"
                                + "\n------------------------------------------------------------------------"
                                + "\n\nInsira o nome do arquivo de dados a ser processado:"
                                + "\n> ");

                String arquivo = in.nextLine();
                Path input = Paths.get(arquivo);

                long startTime = System.nanoTime();

                System.out.println("\n--- Iniciando Cronômetro ---");

                HashSet<Conta> contas = new HashSet<Conta>();
                HashMap<String, Pessoa> pessoas = new HashMap<String, Pessoa>();
                ArrayDeque<Pessoa> fila = new ArrayDeque<Pessoa>();
                String nome1="", nome2="";

                try (BufferedReader br = Files.newBufferedReader(input, Charset.forName("utf8")))
                {
                        String linha = br.readLine();
                        Scanner sc = new Scanner(linha).useDelimiter(" ");

                        int totalContas = sc.nextInt();
                        System.out.println("Total de Contas Correntes: " + totalContas);

                        while((linha = br.readLine()) != null) {
```

```
                        sc = new Scanner(linha).useDelimiter(" ");

                        //Populando set de contas
                        if(sc.hasNextInt()) {
                                Conta contaNova = new Conta(sc.nextInt());
                                contaNova.addTitular(sc.next());
                                contaNova.addTitular(sc.next());
                                contas.add(contaNova);
                        }
                        //Gravando nome de pessoas da transferencia
                        else {
                                nome1 = sc.next();
                                nome2 = sc.next();
                        }
                }
        }
        catch (IOException x) {
                System.err.format("Erro de E/S: %s%n", x);
        }

        //Iterando set de Contas
        for(Conta c : contas) {
                //Iterando titulares da Conta
                for(String titular : c.titulares) {
                        //Se for a primeira conta da pessoa, cria a pessoa no dicionario e acrescenta
conta conjunta
                        if(!pessoas.containsKey(titular)) {
                                Pessoa novaPessoa = new Pessoa(titular);
                                for(String titular2 : c.titulares) {
                                        if(titular2 != titular) {
                                                novaPessoa.addConjunta(titular2);
                                        }
                                }
                                pessoas.put(titular, novaPessoa);
                        }
                        //Se já tiver conta, simplesmente acrescenta a conjunta
                        else {
                                for(String titular2 : c.titulares) {
                                        if(titular2 != titular) {
                                                pessoas.get(titular).addConjunta(titular2);
                                        }
                                }
                        }
                }
        }

        System.out.println("--- Inicializando Estrutura de Contas para arquivo " + arquivo + " ---"
                        + "\nPessoas: " + pessoas.size() + "   ---   Contas: " + contas.size()
                        + "\nOrigem: " + nome1 + "   ---   Destino: " + nome2
                        + "\n-----------------------------------------");

        Pessoa pessoa1 = pessoas.get(nome1);
```

```
                Pessoa pessoa2 = pessoas.get(nome2);

                pessoa2.distancia = 1;

                fila.add(pessoa2);

                lacoCaminha:
                while(!fila.isEmpty()) {
                        Pessoa p = fila.remove();

                        for(String v : p.conjuntas) {
                                Pessoa vizinho = pessoas.get(v);
                                if(vizinho.distancia<0) {
                                        vizinho.distancia = p.distancia + 1;
                                        vizinho.anterior = p.nome;
                                        fila.add(vizinho);
                                        if(vizinho == pessoa1) break lacoCaminha;
                                }
                        }
                }

                Pessoa aux1 = pessoa1;
                Pessoa aux2 = pessoas.get(pessoa1.anterior);
                while(aux1 != pessoa2) {
                        lacoConta:
                        for(Conta c : contas) {
                                if(c.titulares.contains(aux1.nome) && c.titulares.contains(aux2.nome)) {
                                        System.out.println(c.numero + " - " + aux1.nome + " - " + aux2.nome);
                                        aux1 = aux2;
                                        aux2 = pessoas.get(aux1.anterior);
                                        break lacoConta;
                                }
                        }
                }

                long endTime = System.nanoTime();
                double duration = ((endTime*1.0 - startTime)/1000000000);
                duration = (Math.round(duration*100)/100.0);
                System.out.println("--- Tempo Total de Execução: " + duration + " segundos ---");

        }
}
```

```
//---Classe Pessoa
import java.util.HashSet;

public class Pessoa {

	public String nome;
	public HashSet<String> conjuntas;
	public String anterior;
	public int distancia;

	public Pessoa(String nome) {
		this.nome = nome;
		this.conjuntas = new HashSet<String>();
		this.anterior = "";
		this.distancia = -1;
	}

	public void addConjunta(String nome) {
		conjuntas.add(nome);
	}
}
```

```
//---Classe Conta
import java.util.HashSet;

public class Conta {

	public int numero;
	public HashSet<String> titulares;

	public Conta(int numero) {
		this.numero = numero;
		titulares = new HashSet<String>();
	}

	public void addTitular(String nome) {
		titulares.add(nome);
	}

}
```