PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS E INFORMÁTICA
UNIDADE EDUCACIONAL PRAÇA DA LIBERDADE
Bacharelado em Engenharia de Software

Anderson Barbosa Coutinho
Henrique Alberone Nunes Alves Ramos
Mateus Santos Fonseca
Yan Max Rodrigues Sette Pinheiro

Trabalho final – Algoritmos Computacionais em Grafos

Belo Horizonte
2019

1. Introdução

Este trabalho busca, através de aplicações de conceitos aprendidos na disciplina Algoritmos Computacionais em Grafos, resolver um dado problema.

O problema, resumidamente, consiste em alocar N alunos a K professores, formando grupos de pesquisa onde os alunos participantes possuam um trabalho semelhante entre si. Neste problema, o número de professores é menor do que o de aluno e o grau de relacionamento entre os alunos era a dissimilaridade entre as suas áreas de pesquisa.

O desenvolvimento deste trabalho deu-se, primeiramente, pelas fases de compreensão do problema e modelagem do problema como um grafo de forma manual, após estas duas etapas, foram feitas as leituras das entradas, estruturação do modelo de solução, implementação de algoritmos que compõem a solução e execução de testes.

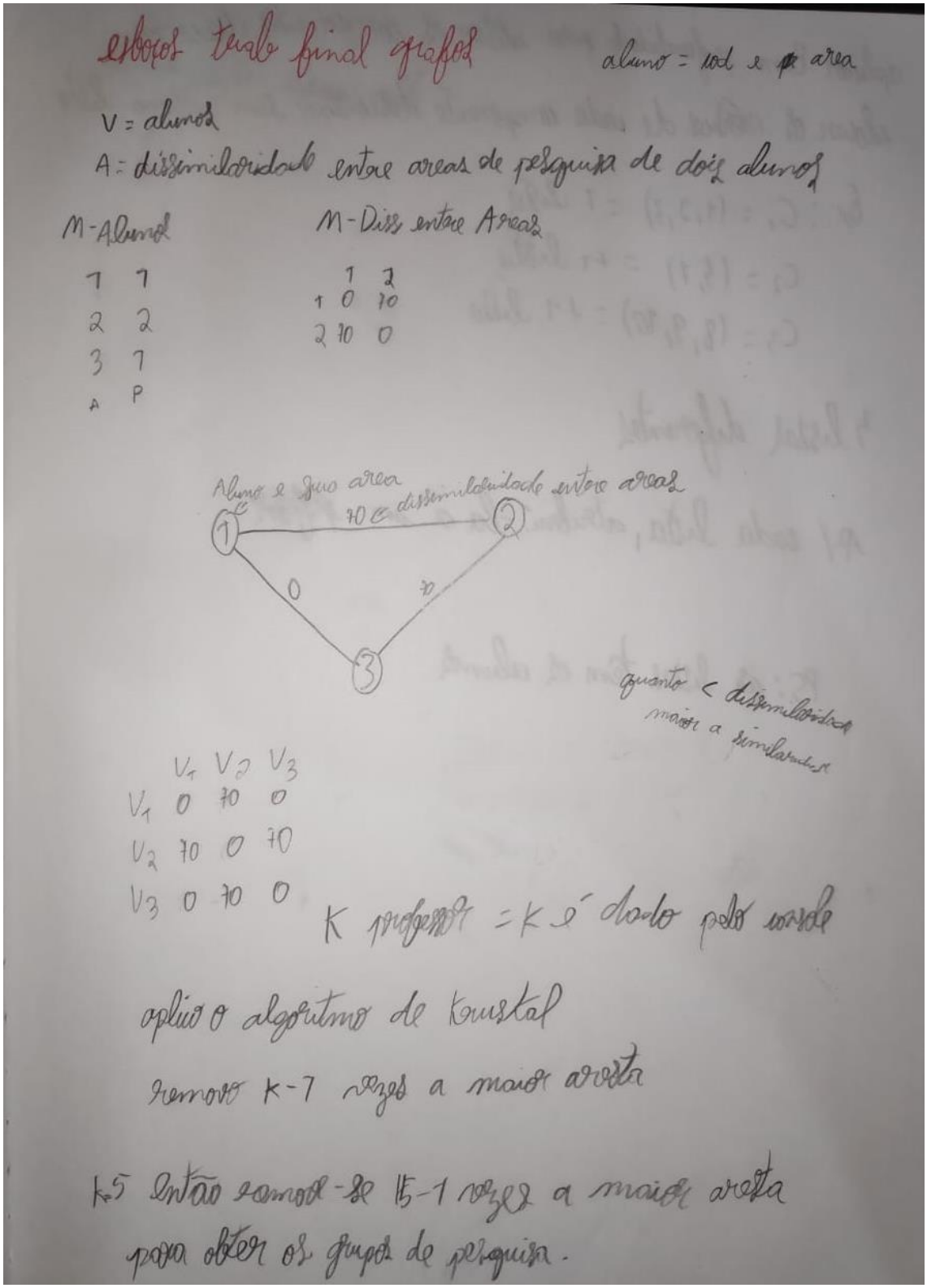
2. Desenvolvimento

A compreensão do problema deste trabalho foi feita conjuntamente com a modelagem manual do grafo, para que, uma vez que o problema fosse compreendido, a solução encontrada utilizasse conceitos de grafos.

Antes de qualquer implementação ser realizada, o problema foi modelado manualmente e rascunhos foram utilizados para o auxílio desta modelagem. Um exemplo de rascunho estará disponível nesta seção. Nesta fase, após análise contínua dos integrantes do grupo, toda a solução foi modelada. No grafo modelado, os vértices são os alunos e as arestas são a dissimilaridade entre as áreas de pesquisa dos alunos. (Nesta etapa também foi percebido que o grafo sempre seria completo).

Após a etapa de compreensão e modelagem, a implementação foi iniciada. O primeiro artefato produzido foi a leitura dos arquivos “Aluno-Pesquisa” e “Matriz de Dissimilaridade”. Esta etapa foi a primeira a ser implementada justamente para facilitar o teste enquanto era desenvolvida a solução, uma vez que, para alterar os valores de entrada, era só alterar os arquivos.

Exemplo de rascunho utilizado na fase inicial da modelagem:



Tendo concluída a fase de leitura de arquivos, deu-se início a estruturação do modelo de solução e implementação dos algoritmos que compõem a solução. Para isso, os seguintes passos foram seguidos na implementação:

1. Definição da forma de representação do grafo. A estrutura de representação de grafos escolhida foi a matriz de adjacência. Com isso, o grafo era preenchido de acordo com os arquivos lidos;
2. Foi verificada a necessidade de se obter a Árvore Geradora Mínima deste grafo, portanto a aplicação de um algoritmo que gerasse a AGM deveria ser aplicada (Kruskal ou Prim), nesta solução, utilizamos o algoritmo de Prim;
3. Após a geração da AGM, era necessário retirar K – 1 vezes (onde K é quantidade de professores) a aresta de maior dissimilaridade, uma vez que, para formar os grupos de pesquisa, eram necessários K componentes em um grafo desconexo. O motivo pelo qual a aresta de maior dissimilaridade deve ser retirada dá-se no problema, uma vez que é buscado formar grupos de pesquisa que possuam alunos com a menor dissimilaridade entre suas áreas de pesquisa;
4. Uma vez gerado o grafo desconexo com os K componentes, resta apenas associar estes K componentes à K grupos de pesquisas diferentes, e mostrar estes grupos de pesquisa gerados no console.

Após a finalização do desenvolvimento, alguns testes foram utilizados para verificar se a solução estava funcionando. Os testes utilizados estão disponíveis na seção de testes realizados.

3. Testes realizados

Alguns testes para a verificação da solução foram realizados. Dentre eles:

Teste 1:

Dissimilaridade:
0 70 50 30
70 0 60 50
50 60 0 40
30 50 40 0

Alunos:
01 01
02 02
03 01
04 03
05 04

Para estes arquivos lidos, as saídas devem ser:

Para K = 2:

Output:
Grupo 1: Aluno 1, Aluno 3, Aluno 5, Aluno 4,
Grupo 2: Aluno 2,

Para K = 3:

Output:
Grupo 1: Aluno 1, Aluno 3, Aluno 5,
Grupo 2: Aluno 2,
Grupo 3: Aluno 4,

Para K = 4:

Output:
Grupo 1: Aluno 1, Aluno 3,
Grupo 2: Aluno 2,
Grupo 3: Aluno 4,
Grupo 4: Aluno 5,

Teste 2.

Dissimilaridade:
0 70 50
70 0 60
50 60 0

Alunos:
01 01
02 02
03 01
04 03

Para estes arquivos lidos, as saídas devem ser:

Para K = 2

Output:
Grupo 1: Aluno 1, Aluno 3, Aluno 4,
Grupo 2: Aluno 2,

Para K = 3

Output:
Grupo 1: Aluno 1, Aluno 3,
Grupo 2: Aluno 2,
Grupo 3: Aluno 4,

Teste 3.

Dissimilaridade:
0 70
70 0

Alunos:
01 01
02 02
03 01

Para estes arquivos lidos, a saída deve ser:

Para K = 2

Output:
Grupo 1: Aluno 1, Aluno 3,
Grupo 2: Aluno 2,

4. Conclusão

Este trabalho foi fundamental para compreendermos os conceitos e aplicações da disciplina de Algoritmos Computacionais em Grafos.

Tendo em vista a complexidade do trabalho e dos algoritmos, diversas fontes foram consultadas para conseguirmos implementar a solução. As referências estão comentadas nos códigos, quando explicitadas pelo autor.

O trabalho, inicialmente, foi muito complicado de se desenvolver, uma vez que uma maior abrangência da matéria era requerida para seu desenvolvimento. Tendo em vista que os integrantes do grupo tentaram solucionar este trabalho cedo demais, alguns conceitos ainda não haviam sido vistos e não foi possível desenvolvê-lo. Após o término das matérias necessárias para completar este trabalho, o grupo retornou a desenvolver, e, naturalmente, os problemas foram solucionados.

Finalmente, é importante frisar que, apesar de ser um trabalho extenso e complexo, ele é fundamental para, não só compreender os conceitos, necessidades e aplicações de grafos, como para aprender mais sobre a área também, pois, como dito anteriormente, várias pesquisas foram realizadas, e, com isso, houve muito aprendizado.

5. Código da solução

Classe Principal

```
package application;

import java.util.Scanner;

import utilities.Algoritmo;

public class Principal {

	public static void main(String[] args) {

		/*
		 * Comentadas estão as formas de leitura manuais das PATHs dos arquivos a serem
		 * lidos. Não comentadas e defaults estão a leitura dos arquivos
		 * entradaAlunoPesquisa e entradaMatrizDissimilaridade
		 */

		Scanner reader = new Scanner(System.in);
		// System.out.println("Qual diretorio do arquivo que relaciona Aluno com
		// Pesquisa?");
		// String alunoPesquisaPath = reader.nextLine();
		String alunoPesquisaPath = "entradaAlunoPesquisa.txt";

		// System.out.println(
		// "Qual o diretorio do arquivo que fornece a matriz de dissimilaridade entre as
		// ï¿½reas de pesquisa");
		// String matrizDissimilaridadePath = reader.nextLine();
		String matrizDissimilaridadePath = "entradaMatrizDissimilaridade.txt";

		Algoritmo.rodar(alunoPesquisaPath, matrizDissimilaridadePath);

		reader.close();

	}

}
```

Classe LeitorArquivo

```
package utilities;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

import entities.Aluno;

public class LeitorArquivo {

	public static Aluno[] getAlunoPesquisa(String alunoPesquisaPath) {
		int qntLinhas = getQuantidadeLinhas(alunoPesquisaPath);
		Scanner leitor = null;
		Aluno[] vetorAlunoPesquisa = new Aluno[qntLinhas];

		int i = 0;
		try {
			leitor = new Scanner(new File(alunoPesquisaPath));
		} catch (FileNotFoundException e) {
			e.getStackTrace();
		}

		/*
		 * Para cada linha do arquivo, cria um aluno x com seu codigo e
		disciplina e
		 * coloca no vetorALunoPesquisa
		 */
		while (leitor.hasNextLine()) {
			String[] linhaAtual = leitor.nextLine().split(" ");
			Aluno x = new Aluno(Integer.parseInt(linhaAtual[0]),
Integer.parseInt(linhaAtual[1]));
			vetorAlunoPesquisa[i] = x;
			i++;
		}

		return vetorAlunoPesquisa;
	}

	public static int[][] getMatrizDissimilaridade(String
matrizDissimilaridadePath) {
		int qntLinhas = getQuantidadeLinhas(matrizDissimilaridadePath);
		int matrizAdjascencia[][] = new int[qntLinhas][qntLinhas];

		int i = 0, j = 0;
		Scanner leitor = null;

		try {
			leitor = new Scanner(new File(matrizDissimilaridadePath));
		} catch (FileNotFoundException e) {
			e.getStackTrace();
		}
		while (leitor.hasNextLine()) {
			String[] linhaAtual = leitor.nextLine().split(" ");
			while (j < qntLinhas) {
				matrizAdjascencia[i][j] =
Integer.parseInt(linhaAtual[j]);
				j++;
```

```
                }
                i++;
                j = 0;
            }

            return matrizAdjascencia;
        }

        public static int getQuantidadeLinhas(String path) {
            int qntDeLinhas = 0;
            Scanner leitor = null;

            try {
                leitor = new Scanner(new File(path));
            } catch (FileNotFoundException e) {
                e.getStackTrace();
            }
            while (leitor.hasNextLine()) {
                qntDeLinhas++;
                leitor.nextLine();
            }
            return qntDeLinhas;
        }

}
```

Classe Aluno

```java
package entities;

public class Aluno {

	int codigoAluno;
	int areaPesquisa;

	public Aluno(int codigoAluno, int areaPesquisa) {
		this.codigoAluno = codigoAluno;
		this.areaPesquisa = areaPesquisa;
	}

	public int getCodigoAluno() {
		return codigoAluno;
	}

	public void setCodigoAluno(int codigoAluno) {
		this.codigoAluno = codigoAluno;
	}

	public int getAreaPesquisa() {
		return areaPesquisa;
	}

	public void setAreaPesquisa(int areaPesquisa) {
		this.areaPesquisa = areaPesquisa;
	}

}
```

Classe Algoritmo

```
package utilities;

import java.util.Scanner;

import entities.Aluno;

public class Algoritmo {

	public static void rodar(String alunoPesquisaPath, String
matrizDissimilaridadePath) {

		Aluno alunoPesquisa[] =
LeitorArquivo.getAlunoPesquisa(alunoPesquisaPath);
		int dissimilaridadePesquisa[][] =
LeitorArquivo.getMatrizDissimilaridade(matrizDissimilaridadePath);
		int qntAlunos = alunoPesquisa.length;
		int qntProfessores = 0;

		do {
			System.out.println("Insira a quantidade de professores:
");
			Scanner leitor = new Scanner(System.in);
			qntProfessores = leitor.nextInt();
		} while (qntProfessores > qntAlunos || qntProfessores <= 0);


		if (qntProfessores == 1) {
			System.out.print("Grupo 1: ");
			for (int i = 1; i <= qntAlunos; i++) {
				System.out.print("Aluno " + i + ", ");
			}
			System.exit(1);
		}

		int[][] grafoAlunos = new int[qntAlunos][qntAlunos];

		/*
		 * Neste grafo, os vértices são os alunos e as arestas são a
		 * dissimilaridade entre as areas de pesquisa dos alunos
conectados
		 *
		 * Eh necessario decrementar 1 no getAreaPesquisa pois a
pesquisa retornada no
		 * get ï¿½ correspondente ao valor do get - 1 na matriz de
dissimilaridade.
		 * EXEMPLO: aluno.getAreaPesquisa[0] = 1, porem na matriz, este
valor 1 ï¿½
		 * representado na posicao 1-1 que ï¿½ igual a 0;
		 */

		for (int i = 0; i < grafoAlunos.length; i++) {
			for (int j = 0; j < grafoAlunos.length; j++) {
				grafoAlunos[i][j] =
dissimilaridadePesquisa[alunoPesquisa[i].getAreaPesquisa() -
1][alunoPesquisa[j]
						.getAreaPesquisa() - 1];
			}
```

```
            }

            printMatrizes(alunoPesquisa, dissimilaridadePesquisa,
grafoAlunos);

            // executar o algoritmo de kruskal para obter a AGN
            int[][] matrizResultado = Prim.prim(grafoAlunos);

            /**
             * ImpressÃ£o da matrizResultado com a resposta do Algoritmo de
Prim
             */

            System.out.println("\n");
            System.out.println("------------ AGM ------------");
            for (int contadorHorizontal = 0; contadorHorizontal <
matrizResultado[0].length; contadorHorizontal++) {
                  for (int contadorVertical = 0; contadorVertical <
matrizResultado[0].length; contadorVertical++) {


      System.out.print(matrizResultado[contadorHorizontal][contadorVertical]
+ "\t");
                  }

                  System.out.println();
            }

            // para k-1 professores, retirar a maior aresta existente (com
maior
            // dissimilaridade)
            int[][] matrizFinal = retirarAresta(qntProfessores,
matrizResultado);

            System.out.println("\n\n");
            // mostrar os grupos de pesquisa formados
            Graph.addMatrixToEdge(matrizFinal);
      }

      private static int[][] retirarAresta(int qntProfessores, int[][]
matrizResultado) {
            int maiorAresta = 0;
            int x = 0;
            int y = 0;

            // quantidade do k - 1
            for (int k = 1; k < qntProfessores; k++) {
                  // encontrar maior aresta
                  for (int i = 0; i < matrizResultado.length; i++) {
                        for (int j = 0; j < matrizResultado.length; j++) {
                              if (maiorAresta <= matrizResultado[i][j]) {
                                    maiorAresta = matrizResultado[i][j];
                                    x = i;
                                    y = j;
                              }
                        }
                  }
                  // remover maior aresta
                  matrizResultado[x][y] = -1 - k;
```

```
                        matrizResultado[y][x] = -1 - k;

                        // zerar maior aresta antes de nova execucao
                        maiorAresta = 0;
                  }

                  System.out.println("\n\n");
                  System.out.println("------------ Componentes ------------");
                  for (int contadorHorizontal = 0; contadorHorizontal <
matrizResultado[0].length; contadorHorizontal++) {
                        for (int contadorVertical = 0; contadorVertical <
matrizResultado[0].length; contadorVertical++) {


      System.out.print(matrizResultado[contadorHorizontal][contadorVertical]
+ "\t");
                        }

                        System.out.println();
                  }
                  return matrizResultado;

      }

      private static void printMatrizes(Aluno[] alunoPesquisa, int[][]
dissimilaridadePesquisa, int[][] grafoAlunos) {

                  System.out.println("Vetor aluno-pesquisa:");
                  for (Aluno aluno : alunoPesquisa) {
                        System.out.println(aluno.getCodigoAluno() + " " +
aluno.getAreaPesquisa());
                  }

                  System.out.println("\n");

                  System.out.println("Matriz de dissimilaridades entre
pesquisas:");
                  for (int i = 0; i < dissimilaridadePesquisa.length; i++) {
                        for (int j = 0; j < dissimilaridadePesquisa.length; j++) {
                              System.out.print(dissimilaridadePesquisa[i][j] +
"\t");
                        }
                        System.out.println();
                  }

                  System.out.println("\n");

                  System.out.println("Grafo final (V = Alunos e E =
dissimilaridade entre areas de pesquisas entre os alunos):");
                  for (int i = 0; i < grafoAlunos.length; i++) {
                        for (int j = 0; j < grafoAlunos.length; j++) {
                              System.out.print(grafoAlunos[i][j] + "\t");
                        }
                        System.out.println();
                  }
      }

      public static boolean contains(final int[] array, final int v) {
```

```
		boolean result = false;
		for (int i : array) {
			if (i == v && v != 0) {
				result = true;
				break;
			}
		}

		return result;
	}

}
```

Classe Graph

```
package utilities;

//Java program to print connected components in
//an undirected graph
import java.util.LinkedList;

class Graph {
	// A user define class to represent a graph.
	// A graph is an array of adjacency lists.
	// Size of array will be V (number of vertices
	// in graph)
	int V;
	LinkedList<Integer>[] adjListArray;

	// constructor
	Graph(int V) {
		this.V = V;
		// define the size of array as
		// number of vertices
		adjListArray = new LinkedList[V];

		// Create a new list for each vertex
		// such that adjacent nodes can be stored

		for (int i = 0; i < V; i++) {
			adjListArray[i] = new LinkedList<Integer>();
		}
	}

	// Adds an edge to an undirected graph
	void addEdge(int src, int dest) {
		// Add an edge from src to dest.
		adjListArray[src].add(dest);

		// Since graph is undirected, add an edge from dest
		// to src also
		adjListArray[dest].add(src);
	}

	void DFSUtil(int v, boolean[] visited) {
		// Mark the current node as visited and print it
		visited[v] = true;
		System.out.print("Aluno " + (v + 1) + ", ");
		// Recur for all the vertices
		// adjacent to this vertex
		for (int x : adjListArray[v]) {
			if (!visited[x])
				DFSUtil(x, visited);
		}

	}

	void connectedComponents() {
		// Mark all the vertices as not visited
		boolean[] visited = new boolean[V];
		int x = 1;
		for (int v = 0; v < V; ++v) {
```

```
                if (!visited[v]) {
                        // print all reachable vertices
                        // from v
                        System.out.print("Grupo " + x + ": ");
                        x++;
                        DFSUtil(v, visited);
                        System.out.println();
                }
        }
    }

    public static void addMatrixToEdge(int[][] matrix) {
        Graph g = new Graph(matrix.length);
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix.length; j++) {
                if (matrix[i][j] > -1) {
                    g.addEdge(i, j);
                }
            }
        }
        g.connectedComponents();
    }
}
```

Classe Prim

```
package utilities;
/*
Algoritmo de Prim
Autor:
    VojtÄ›ch JarnÃ-k(1930) e Robert C. Prim(1957)
Colaborador:
    Filipe Saraiva (filip.saraiva@gmail.com)
Tipo:
    graph
Descriçãoo:
    O Algoritmo de Prim é um algoritmo em grafos clássico que determina a
    Ãrvore geradora mínima de um grafo conectado não-orientado.
Complexidade:
    O(nÂ²)
Dificuldade:
    medio
Referências:
    [1] http://en.wikipedia.org/wiki/Prim%27s_algorithm
    [2] http://pt.wikipedia.org/wiki/Algoritmo_de_Prim
    [3] http://www-b2.is.tokushima-u.ac.jp/~ikeda/suuri/dijkstra/Prim.shtml
Licenca:
    GPLv3

---------- Pequenas modificações realizadas por: Yan Max ----------
*/

import java.util.ArrayList;

public class Prim {

      /**
       * ImplementaÃ§Ã£o do Algoritmo de Prim
       */
      public static int[][] prim(int[][] matrizSistema) {

            /**
             * ArrayList para guardar os vÃ©rtices jÃ¡ verificados pelo
Algoritmo de Prim
             */
            ArrayList<Boolean> verticesVerificados = new
ArrayList<Boolean>();

            /**
             * ArrayList para guardar as distÃ¢ncias relativas para cada
vÃ©rtice em cada
             * iteraÃ§Ã£o do Algoritmo de Prim
             */
            ArrayList<Integer> distanciaRelativa = new ArrayList<Integer>();

            /**
             * ArrayList unidimensional que guarda os nÃ³s vizinhos de cada
nÃ³ do grafo da
             * Ã¡rvore final produzida pelo Algoritmo de Prim
             */
            ArrayList<Integer> nosVizinhos = new ArrayList<Integer>();

            /**
```

```
             * InicializaÃ§Ã£o de variÃ¡veis
             */
            for (Integer contador = 0; contador < matrizSistema[0].length;
contador++) {
                  verticesVerificados.add(false);
                  nosVizinhos.add(0);
                  distanciaRelativa.add(Integer.MAX_VALUE);
            }

            distanciaRelativa.set(0, new Integer(0));

            /**
             * DefiniÃ§Ã£o do ponto que serÃ¡ a raiz da Ã¡rvore resultante
             */
            Integer pontoAvaliado = 0;

            /**
             * Estrutura para execuÃ§Ã£o das iteraÃ§Ãµes do Algoritmo de
Prim
             */
            for (Integer contadorPontosAvaliados = 0;
contadorPontosAvaliados < matrizSistema[0].length; contadorPontosAvaliados++)
{
                  for (Integer contadorVizinhos = 0; contadorVizinhos <
matrizSistema[0].length; contadorVizinhos++) {

                        /**
                         * Verifica se o nÃ³ a ser avaliado nesta
iteraÃ§Ã£o jÃ¡ foi avaliado
                         * anteriormente; se sim, passa para a prÃ³xima
iteraÃ§Ã£o
                         */
                        if ((verticesVerificados.get(contadorVizinhos)) ||
(contadorVizinhos == pontoAvaliado)) {
                              continue;
                        }

                        /**
                         * Duas comparaÃ§Ãµes aqui:
                         *
                         * 1Âª - Verifica se na matrizSistema hÃ¡ algum
valor na coluna que seja >= 0.
                         * Caso afirmativo, significa que hÃ¡ uma aresta
entre estes dois pontos do
                         * grafo.
                         *
                         * 2Âª - Verifica se o peso da aresta entre os dois
nÃ³s Ã© menor que a atual
                         * distanciaRelativa do nÃ³ vizinho.
                         *
                         * Caso correto, a distanciaRelativa do nÃ³ vizinho
ao que estÃ¡ sendo avaliado
                         * no momento serÃ¡ atualizada pelo valor dessa
nova distancia avaliada atÃ© o
                         * pontoAvaliado.
                         */

                        if ((matrizSistema[pontoAvaliado][contadorVizinhos]
>= 0)
```

```
                                        &&
((matrizSistema[pontoAvaliado][contadorVizinhos] < distanciaRelativa
                                                        .get(contadorVizinhos))))
{

                                if
(matrizSistema[pontoAvaliado][contadorVizinhos] == 0) {

      matrizSistema[pontoAvaliado][contadorVizinhos] = 1;
                                }

                                distanciaRelativa.set(contadorVizinhos,
matrizSistema[pontoAvaliado][contadorVizinhos]);

                                nosVizinhos.set(contadorVizinhos,
pontoAvaliado);

                        }
                    }

                    /**
                     * Marca o vÃ©rtice de pontoAvaliado como um vÃ©rtice jÃ¡
verificado
                     */
                    verticesVerificados.set(pontoAvaliado, true);

                    /**
                     * PreparaÃ§Ã£o para seleÃ§Ã£o do prÃ³ximo vÃ©rtice a ser
avaliado
                     */
                    pontoAvaliado = new Integer(0);
                    Integer distanciaComparada = new
Integer(Integer.MAX_VALUE);

                    /**
                     * SeleÃ§Ã£o do prÃ³ximo vÃ©rtice a ser avaliado
                     */
                    for (Integer contador = 1; contador <
verticesVerificados.size(); contador++) {

                        /**
                         * Se o vertice a ser verificado jÃ¡ foi verificado
anteriormente (true) passa
                         * Ã  prÃ³xima iteraÃ§Ã£o.
                         */
                        if (verticesVerificados.get(contador)) {
                            continue;
                        }

                        /**
                         * Se a distÃ¢ncia relativa desse ponto for menor
que a do ponto avaliado
                         * assume-se esse novo ponto como o ponto avaliado.
                         *
                         * Ao final da iteraÃ§Ã£o, serÃ¡ selecionado o
ponto com menor distÃ¢ncia
                         * relativa.
                         */
```

```
                        if (distanciaRelativa.get(contador) <
distanciaComparada) {
                              distanciaComparada =
distanciaRelativa.get(contador);
                              pontoAvaliado = contador;
                        }

                  }

            }

            int[][] matrizResposta = new
int[matrizSistema[0].length][matrizSistema[0].length];

            /*
             * Inicializar matrizResposta com -1
             */
            for (int i = 0; i < matrizSistema[0].length; i++) {
                  for (int j = 0; j < matrizSistema[0].length; j++) {
                        matrizResposta[i][j] = -1;
                  }
            }
            /**
             * CriaÃ§Ã£o da matrizResposta com a Ã¡rvore resultante do
Algoritmo de Prim
             */
            for (int contador = 1; contador < nosVizinhos.size();
contador++) {
                  matrizResposta[contador][nosVizinhos.get(contador)] =
matrizSistema[contador][nosVizinhos.get(contador)];
                  matrizResposta[nosVizinhos.get(contador)][contador] =
matrizResposta[contador][nosVizinhos.get(contador)];
            }

            return matrizResposta;
      }
}
```