Projeto - Algoritmos e Estruturas de Dados II

Otimização de rotas para transporte de pacientes em hemodiálise utilizando o algoritmo do caixeiro viajante e grafos

Relatório do código do programa e instruções de uso

Docente:
Igor Medeiros Vanderlei

Discentes:
Fábio Guerra Medeiros Filho
Lucas Galdêncio Barbosa

Período 2022.1
Data: 12 de abril de 2023

O algoritmo utilizado para a implementação deste projeto é o algoritmo do Caixeiro Viajante para encontrar o menor caminho que passa por todos os vértices e retorna para o ponto inicial.

Nesse programa temos 2 Classes:

- ***VanHemodialise***
  - Classe que contém o algoritmo do caixeiro viajante e faz todo o cálculo do menor trajeto a ser percorrido pela van. Contém métodos que compõem o funcionamento do cálculo do dito trajeto e também o método ***main*** para a execução do programa.
  - A seguir, código da classe:

```
import java.util.ArrayList;

public class VanHemodialise {
   public static ArrayList<Integer> vanHemo(int[][] matrizDistancias, int pacienteInicial) {
      // Número de pacientes
      int n = matrizDistancias[0].length;
      // ArrayList para armazenar os pacientes visitados
      ArrayList<Integer> visitados = new ArrayList<>();
      // Adiciona o paciente inicial às visitados
      visitados.add(pacienteInicial);

      // Percorre os pacientes fazendo as escolhas
      int pacienteAtual = pacienteInicial;
      while (visitados.size() < n) {
         // Encontra o paciente mais próximo não visitado
         int pacienteProximo = -1;
         int distanciaMenor = Integer.MAX_VALUE;
         for (int i = 0; i < n; i++) {
            if (!visitados.contains(i) && matrizDistancias[pacienteAtual][i] < distanciaMenor) {
               pacienteProximo = i;
               distanciaMenor = matrizDistancias[pacienteAtual][i];
            }
         }
         // Verifica se encontrou um paciente próximo
         if (pacienteProximo != -1) {
            // Adiciona o paciente próxima aos visitados
            visitados.add(pacienteProximo);
            pacienteAtual = pacienteProximo;
         } else {
            // Não encontrou paciente próximo, finaliza o percurso
            break;
         }
      }

      // Adiciona o paciente inicial ao final do trajeto
      visitados.add(pacienteInicial);
      return visitados;
   }
```

```
    public static int calcularDistanciaTotal(int[][] matrizDistancias, ArrayList<Integer> pacientesVisitados) {
        int distanciaTotal = 0;
        int pacienteAnterior = pacientesVisitados.get(0);
        for (int i = 1; i < pacientesVisitados.size(); i++) {
            int pacienteAtual = pacientesVisitados.get(i);
            distanciaTotal += matrizDistancias[pacienteAnterior][pacienteAtual];
            pacienteAnterior = pacienteAtual;
        }
        return distanciaTotal;
    }

    public static void main(String[] args) {
        // Matriz de distâncias entre os pacientes, sendo nesse caso usado 7 pacientes e o vertice 0 sendo o hospital, ponto de partida e chegada do percurso.
        // Nesse mesmo programa, na classe GeradorMatriz pode ser criada uma nova matriz de forma aleatória e que satisfaça os requisitos do nosso caso de uso.
        int[][] distancias = {
                {0, 2, 5, 2, 3, 4, 5, 3},
                {2, 0, 3, 5, 2, 5, 2, 3},
                {5, 3, 0, 2, 5, 3, 2, 2},
                {2, 5, 2, 0, 3, 5, 3, 5},
                {3, 2, 5, 3, 0, 2, 5, 2},
                {4, 5, 3, 5, 2, 0, 3, 5},
                {5, 2, 2, 3, 5, 3, 0, 2},
                {3, 3, 2, 5, 2, 5, 2, 0}
        };

        // Nome dos vértices, sendo o primeiro sempre "Hospital" por ser o ponto de partida. Já os demais, os nomes dos pacientes.
        String[] nomesVertices = {"Hospital", "Beto", "Clara", "Daniel", "Eva", "Felipe", "Gabi", "Henrique"};

        // Ponto inicial, que no caso é o Hospital.
        int pacienteInicial = 0;

        // Encontra o trajeto da van
        ArrayList<Integer> trajeto = vanHemo(distancias, pacienteInicial);

        // Calcula a distância total percorrida
        int distanciaTotal = calcularDistanciaTotal(distancias, trajeto);

        // Imprime o trajeto
        System.out.println("Trajeto da van:");
        System.out.println("EX:  3: Felipe(5) -- '3' se refere ao numero da parada e '5' se refere ao vertice referente ao nome do paciente (ou hospital, quando '0').");
        for (int i=0; i < trajeto.size(); i++){
            int paciente = trajeto.get(i);
            String nomePaciente = nomesVertices[paciente];
            if (i == 0){
                System.out.printf("%d(Ponto de partida): %s(%d),  ", i, nomePaciente, paciente);
            } else if (i == trajeto.size()-1){
                System.out.printf("%d(Ponto final): %s(%d).\n", i, nomePaciente, paciente);
            } else{
                System.out.printf("%d: %s(%d),  ", i, nomePaciente, paciente);
            }
        }
        System.out.printf("Distância total percorrida: %d km.", distanciaTotal);
    }
}
```

A classe contém os métodos ***vanHemo***, ***calcularDistanciaTotal*** e ***main***.

- ***vanHemo***

  Recebe como parâmetros uma matriz de distâncias entre os pacientes e o índice do paciente inicial, e retorna um ArrayList contendo os índices dos pacientes em ordem de visita para a van.

  O método utiliza um laço while para percorrer os pacientes, a cada iteração encontrando o paciente mais próximo que ainda não foi visitado e adicionando-o ao ArrayList de visitados. Para encontrar o paciente mais próximo, o método percorre todos os pacientes que ainda não foram visitados e verifica qual tem a menor distância em relação ao paciente atual.

- ***calcularDistanciaTotal***

  Recebe como parâmetros a matriz de distâncias entre os pacientes e o ArrayList contendo os índices dos pacientes visitados pela van, realiza a soma dessas distâncias e retorna a distância total percorrida pela van nesse trajeto.

- ***main***

  Contém o código principal do programa, que cria uma matriz de distâncias entre pacientes, define o paciente inicial como o índice 0 (o hospital), chama o método ***vanHemo*** para encontrar o trajeto da van e o método ***calcularDistanciaTotal*** para calcular a distância total percorrida.

  O trajeto é impresso utilizando o ArrayList retornado pelo método ***vanHemo*** e um array com os nomes dos pacientes correspondentes a cada índice. A distância total percorrida é impressa utilizando o valor retornado pelo método ***calcularDistanciaTotal***.

- ***GeradorMatriz***
  - Classe que contém o algoritmo que gera uma matriz que venha a ser usada no grafo do algoritmo do caixeiro viajante na classe ***VanHemodialise***.
  - A seguir, código da classe:

```
import java.util.Random;

public class GeradorMatriz {
   public static void main(String[] args) {
      int[][] matriz = new int[8][8];
      Random rand = new Random();

      // Preenche a diagonal principal com zeros e preenche a matriz de forma simétrica
      for (int i = 0; i < matriz.length; i++) {
         for (int j = i + 1; j < matriz[0].length; j++) {
            int distancia = rand.nextInt(4) + 2;
            matriz[i][j] = distancia;
            matriz[j][i] = distancia;
         }
      }

      // Imprime a matriz gerada formatada
      System.out.println("Matriz gerada:");
      for (int i = 0; i < matriz.length; i++) {
         System.out.print("{");
         for (int j = 0; j < matriz[0].length; j++) {
            System.out.print(matriz[i][j]);
            if (j != matriz[0].length - 1) {
               System.out.print(", ");
            }
         }
         System.out.println("}");
      }
   }
}
```

A classe contém o método ***main***.

- ***main***

  Contém o código que realiza a criação de uma matriz que irá ser usada no programa principal.

  A matriz deve satisfazer as condições necessárias para o uso no programa, que são as seguintes: Como a matriz deve conter as distâncias entre os vértices, a matriz deve ser simétrica (para que a distância de ***i*** à ***j*** seja igual a distância de ***j*** à ***i*** e com a diagonal principal com valores 0 (sendo ***i = j,*** a distância de ***i*** ou ***j*** à si próprio deve ser 0).

  Como os pacientes estão localizados em endereços próximos, na mesma cidade, os valores das distâncias gerados que irão preencher a matriz serão aleatórios em um intervalo fechado de 2 à 5.

As instruções para o uso do programa são básicas e estão descritas a seguir:

- A princípio, o programa já estará com um modelo exemplo com uma matriz 8x8, ou seja, 1 hospital e 7 pacientes. E também, com os nomes, sendo o primeiro "Hospital" e os 7 demais, dos pacientes.
- O usuário poderá realizar a mudança da matriz com as distâncias dos pontos (hospital e pacientes) manualmente, podendo gerar uma nova matriz com as distâncias de forma aleatória na classe ***GeradorMatriz***, copiando e colando na ***main*** da classe ***VanHemodialise*** fazendo algumas mudanças básicas, somente adicionando vírgulas para que fique da mesma forma como padrão.
- Ao executar o programa, tudo será feito automaticamente, tudo será calculado e impresso.
- A seguir, um exemplo da saída do programa:
    - O resultado será impresso da seguinte forma:
      0(Ponto de partida): Hospital(0), 1: Beto(1), *2: Eva(4)*, 3: Felipe(5), 4: Clara(2), 5: Daniel(3), 6: Gabi(6), 7: Henrique(7), 8(Ponto final): Hospital(0).
      Distância total percorrida: 19 km.

      Explicando detalhadamente o formato da impressão:
      Onde vemos *2: Eva(4)*
      2 será o nº da parada do trajeto.
      Eva será o nome do paciente.
      4 será o índice do vértice no grafo.

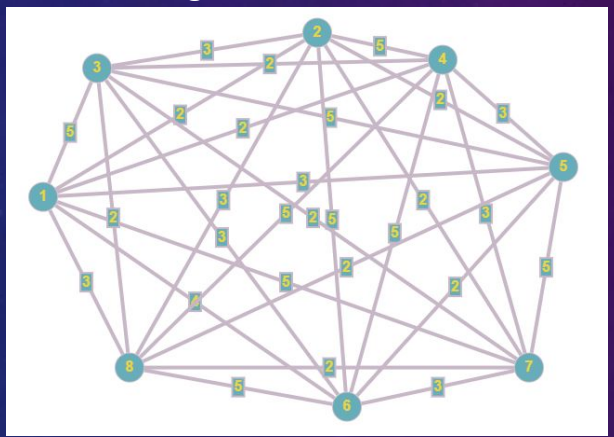
Para um melhor entendimento do trajeto e do grafo criado a partir da matriz, utilizamos a seguinte ferramenta contida nesse site: https://graphonline.ru/pt/

A seguir imagem de como ficará o grafo dada a seguinte matriz:

Matriz:

0, 2, 5, 2, 3, 4, 5, 3
2, 0, 3, 5, 2, 5, 2, 3
5, 3, 0, 2, 5, 3, 2, 2
2, 5, 2, 0, 3, 5, 3, 5
3, 2, 5, 3, 0, 2, 5, 2
4, 5, 3, 5, 2, 0, 3, 5
5, 2, 2, 3, 5, 3, 0, 2
3, 3, 2, 5, 2, 5, 2, 0

Grafo:



Vale lembrar que, no programa, enquanto o índice é 0, na representação gráfica, será 1. Pois na computação o primeiro índice sempre será o 0. Basta adicionarmos 1 unidade em relação ao índice passado na saída do programa.