Instituto de Ciências Matemáticas e de Computação

Departamento de Ciências de Computação

SCC0503 - Algoritmos e Estruturas de Dados II

Relatório Exercício 05

Alunos: João Otávio da Silva, Leonardo Gonçalves Chahud

Professor: Leonardo Tórtoro Pereira

Julho

2022

Conteúdo

1 Introdução

O objetivo principal do exercício é descobrir o vértice mais central, mais periférico e, por fim, o vértice mais distante do vértice mais periférico. Analogamente, isso significa descobrir, de acordo com a contextualização do exercício feita sobre empresa "Braba Log", a melhor cidade para estabelecer um centro de distribuição de mercadorias (vértice mais central), a cidade mais periférica e a cidade mais distante da cidade mais periférica, respectivamente. Por fim, os nós são tratados como cidades e as arestas, como rodovias.

2 Desenvolvimento

2.1 Dígrafo

A representação utilizada foi a implementação de dígrafo com lista de adjacência, a mesma vista em em aula, `DigraphList`. Com uma única alteração no dado presente nos vértices, que será citada posteriormente.

- **Vértice:** Elemento fundamental do grafo, possui um dado, no caso desse exercício, uma *quest*.
- **Aresta:** Conexão entre dois vértices, possui a referência para o vértice de destino e um peso, no caso desse exercício, todos os pesos são 1.
- **Lista de Vértices:** Uma lista que contém as referências para todos os vértices pertencentes ao grafo.
- **Lista de Adjacência:** Uma lista onde para cada vértice há outra lista, contendo referências para todos as arestas que saem do mesmo.

2.2 Alteração no Vértice

Anteriormente havia uma *string* representando o vértice, agora, cada vértice passa a ter uma instância de uma classe arbitrária que implementa a interface `Data`. Essa, por sua vez, possuí a assinatura de duas funções:

- `stringToTerminal:` Exige uma *string* que será utilizada para impressões no terminal, por exemplo, o caminho das travessias.
- `stringToGraphviz:` Exige uma *string* que será utilizada para impressão da imagem do grafo pela biblioteca *graphviz*. Idealmente é uma representação mais mais concisa do dado, por exmplo, o título da *quest*.

```java
public interface Data {
    String stringToTerminal();
    String stringToGraphviz();
}
```

Figura 1: Interface Data

Dessa forma, qualquer classe externa pode utilizar essa representação de grafo, bastando para isso, implementar as duas funções da interface.

2.3 Caminhos mais Curtos

O algoritmo usado para o cálculo dos caminhos mais curto foi o *Floyd Warshall*. Foi utilizada a implementação do algoritmo feito através da herança da interface `TraversalStrategy` feita em aula.

O algoritmo gera uma matriz onde o elemento `[i][j]` corresponde à menor distância entre o *i-ésimo* e o *j-ésimo* vértice.

```java
@Override
public void traverseGraph(Vertex source) {
    for (int i = 0; i < getGraph().getNumberOfVertices(); i++) {
        for (int j = 0; j < getGraph().getNumberOfVertices(); j++) {
            Vertex origin = getGraph().getVertices().get(i);
            Vertex destination = getGraph().getVertices().get(j);
            if (getGraph().edgeExists(origin, destination)) {
                distanceMatrix[i][j] = getGraph().getDistance(origin, destination);
            }
            else {
                distanceMatrix[i][j] = Float.POSITIVE_INFINITY;
            }
        }
    }
    for (int k = 0; k < getGraph().getNumberOfVertices(); k++) {
        for (int i = 0; i < getGraph().getNumberOfVertices(); i++) {
            for (int j = 0; j < getGraph().getNumberOfVertices(); j++) {
                double newDistance = distanceMatrix[i][k] + distanceMatrix[k][j];
                if (newDistance < distanceMatrix[i][j]) {
                    distanceMatrix[i][j] = newDistance;
                }
            }
        }
    }
    printDistanceMatrix();
}
```

Figura 2: Algoritmo Floyd Warshall

2.4 Excentricidade

```java
public double getVertexEccentricity(Vertex vertex) {
    int vertexIndex = graph.getVertices().indexOf(vertex);
    double eccentricity = 0;
    boolean foundTheFirst = false;
    for (int i = 0; i < graph.getNumberOfVertices(); i++) {
        if (i != vertexIndex) {
            if (distancesMatrix[i][vertexIndex] > eccentricity || !foundTheFirst) {
                eccentricity = distancesMatrix[i][vertexIndex];
                foundTheFirst = true;
            }
        }
    }
    return eccentricity;
}
```

Figura 3: Excentricidade

A excentricidade de um vértice `v` é o máximo entre as menores distâncias entre o vértice `v` e os vértices que alcançam ele. Ou seja:

`max(distancesMatrix[i][vIndex])`

2.5 Vértice Central e Periférico

- **Vértice mais central:** vértice com a menor excentricidade
- **Vértice mais periférico:** vértice com a maior excentricidade.

```java
public Vertex centralVertex() {
    Vertex centralVertex = null;
    double minimumEccentricity = Double.POSITIVE_INFINITY;
    for (Vertex currentVertex : graph.getVertices()) {
        double currentVertexEccentricity = getVertexEccentricity(currentVertex);
        if (currentVertexEccentricity < minimumEccentricity) {
            centralVertex = currentVertex;
            minimumEccentricity = currentVertexEccentricity;
        }
    }
    return centralVertex;
}

public Vertex peripheralVertex() {
    Vertex peripheralVertex = null;
    double maximumEccentricity = Double.NEGATIVE_INFINITY;
    for (Vertex currentVertex : graph.getVertices()) {
        double currentVertexEccentricity = getVertexEccentricity(currentVertex);
        if (currentVertexEccentricity > maximumEccentricity) {
            peripheralVertex = currentVertex;
            maximumEccentricity = currentVertexEccentricity;
        }
    }
    return peripheralVertex;
}
```

Figura 4: Vértice Central e Periférico

2.6 Leitura e Inserção

```java
private void readVertices(int numberOfVertices) {
    for (int i = 0; i < numberOfVertices; i++) {
        String line = in.nextLine();
        Point point = splitPoint(line);
        Vertex newVertex = new Vertex(point);
        pointVertexMap.put(point, newVertex);
        graph.addVertex(newVertex);
    }
}

private void readEdges(int numberOfEdges) {
    for (int i = 0; i < numberOfEdges; i++) {
        String line = in.nextLine();
        String[] pointsString = splitConnection(line);
        Point pointA = splitPoint(pointsString[0]);
        Point pointB = splitPoint(pointsString[1]);
        graph.addEdge(pointVertexMap.get(pointA), pointVertexMap.get(pointB),
                Point.euclideanDistance(pointA, pointB));
    }
}

private Point splitPoint(String line) {
    String[] coordinates = line.split( regex: ",");

    return new Point(Double.parseDouble(coordinates[0]), Double.parseDouble(coordinates[1]));
}

private String[] splitConnection(String line) {
    return line.split( regex: ":");
}
```

Figura 5: Leitura de Vértices e Arestas

1. É lido um inteiro n correspondente ao número de vértices.
2. São lidos e inseridos n pontos, representando os nós (cidades).
3. É lido um inteiro m correspondente ao número de arestas.
4. São lidos e inseridos m pares de pontos, representando as arestas (rodovias). Os pesos das arestas são a distância euclidiana entre os pontos.

O método de inserção de aresta, precisa da referência para os vértices para ser executado, porém, a entrada é recebida do usuário na forma de par de pontos. Portanto, a fim de facilitar a chamada desse método, durante a inserção dos vértices, são guardadas em um *HashMap* as relações `[Ponto:Vértice]`.

As linhas são lidas inteiras como uma *string*, depois são divididas (de acordo com os caracteres ',' e ':') e convertidas em *doubles*.

3 Resultados

Caso de Teste	Vértice mais central	Vértice mais periférico	Vértice mais distante do vértice mais periférico
1	(8.0, 10.0)	(3.0, 0.0)	(20.0, 1.0)
2	(8.0, 8.0)	(15.0, 15.0)	(13.0, 0.0)
3	(6.0, 2.0)	(0.0, 0.0)	(9.0, 10.0)
4	(3.0, 6.0)	(9.0, 10.0)	(1.0, 1.0)

Os resultados obtidos, mostram as coordenadas cartesianas correspondentes aos vértices (cidades) de interesse.

- **Vértice mais central:** vértice de menor excentricidade.
- **Vértice mais central:** vértice de maior excentricidade.

No contexto desse exercício, o vértice mais central pode fundamentar a escolha de uma cidade para ser construído um centro de distribuição. Escolhendo a cidade mais central, a distância de uma cidade ao centro de distribuição, no pior caso, é minimizada.