

Universidade do Estado do Amazonas

Escola Superior de Tecnologia

Data: 7 de março de 2017

Disciplina: Teoria dos Grafos

Professores: Elloá B. Guedes

Aluno:

WARM UP
SEQUÊNCIA DE GRAUS

1 Inscrição no Run.Codes

Você deve usar a plataforma Run.Codes para realizar as tarefas descritas neste projeto. Caso não tenha cadastro, primeiramente o faça em <http://run.codes/>. Utilize o seu e-mail institucional. Após a realização do cadastro, procure a turma ESTECP112 Teoria dos Grafos – Teoria dos Grafos 2017.1. O código para matrícula nesta turma é **9C5E**.

Para quem não conhece, o run.codes é uma plataforma automatizada para testes de entrada e saída. Cada aluno submete o seu código escrito na linguagem de programação determinada pelo professor da disciplina e este é submetido a um conjunto de testes previamente cadastrado pelo professor. A nota do aluno corresponde ao percentual de acertos nestes testes.

Algumas observações importantes. O aluno pode submeter quantas vezes quiser e efetuar diversos testes. A nota final não sofre influência do número de submissões, apenas do número de acertos ao final. Considere que seu programa recebe uma entrada de cada vez. Efetue testes em seu próprio computador antes de submeter ao run.codes, é uma forma mais simples de detectar o que pode estar havendo de errado com o seu código. Aproveite o tempo! Você tem até a data limite para submeter a versão final do seu código.

Do ponto de vista do professor, o run.codes é uma ferramenta excelente para detecção de plágio, evitando a cópia de resposta entre os estudantes. Para realizar esta tarefa, o run.codes utiliza a plataforma MOSS desenvolvida e mantida por Stanford.

2 Apresentação

Este é um problema de aquecimento para os exercícios de programação da disciplina. Todos devem fazê-lo para conhecer a plataforma run.codes e os detalhes de funcionamento, tais como entrada de dados, mensagem de erros, etc.

A *sequência de graus* de um grafo corresponde à listagem dos graus dos seus vértices em ordem crescente. O objetivo deste aquecimento é mostrar os passos preparatórios necessários para a obtenção da sequência de graus de um grafo em Java e Python, mostrando as classes consideradas e como capturar a entrada. Este tutorial servirá como base para a implementação de algoritmos da Teoria dos Grafos ao longo da disciplina.

3 Apresentando a Entrada

A primeira linha da entrada consiste em um número inteiro n que especifica a quantidade de nós que o grafo possui. As n linhas seguintes contém uma matriz de adjacência $n \times n$, que especifica as relações entre os nós dos grafos. Neste exemplo, as arestas não possuem peso. Os números que preenchem a matriz são binários, indicando se há aresta entre dois nós (1) ou não (0).

Para exemplificar, considere o exemplo a seguir:

```
5
0 1 0 1 0
1 0 1 1 0
0 1 0 1 1
1 1 1 0 1
0 0 1 1 0
```

É possível identificar que este grafo possui 5 vértices. Supõe-se que a linha ℓ_i da matriz de adjacência corresponde ao vértice $i - 1$ do grafo, em que $i = 0, 1, 2, \dots, 4$. Por exemplo, a linha ℓ_1 corresponde ao vértice 0, o qual possui uma aresta para o vértice 3. Como o grafo não é dirigido, observe que a matriz é simétrica. Note também que a contagem dos vértices começa com o número zero.

Seguindo a especificação dada nessa matriz, o grafo correspondente pode ser visto na Figura 1. Tente reproduzir esta representação a partir da matriz de adjacência.

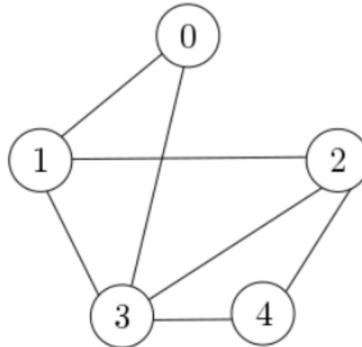


Figura 1: Grafo resultante da matriz de adjacência.

Quando o grafo é ponderado, ao invés de valores binários na matriz teremos os pesos das arestas correspondentes. Se o grafo for dirigido, a matriz não necessariamente é simétrica, pois pode existir uma aresta dirigida do nó i para o nó j , mas não do nó j para o nó i .

4 Implementando em Java

Para implementar esta tarefa na linguagem de programação Java vou utilizar a IDE Eclipse Mars e o Java versão 8. Todas esses recursos podem ser obtidos gratuitamente na internet.

Voltando ao problema, vamos modelar o grafo considerado na linguagem de programação a ser adotada na disciplina. Para tanto, vamos conceber classes adequadas. A classe Vértice (**Vertex**) possui um inteiro que é o rótulo (label) do vértice e uma lista de outros vértices. Alguns métodos irão nos auxiliar com as funcionalidades dessa classe, por exemplo, um método para indicar a quantidade de vizinhos e adicionar vizinho. Além disso, o método `toString` habilita uma visualização serializada das informações relevantes da classe.

```

1 import java.util.ArrayList;
2
3 public class Vertex {
4
5     private int label;
6     private ArrayList<Vertex> neighbours;
7
8     public Vertex(int label) {
9         this.label = label;
10        neighbours = new ArrayList<Vertex>();
11    }
12
13    public void addNeighbours(Vertex v){
14        neighbours.add(v);
15    }
16
17    public int countNeighbours(){
18        return neighbours.size();
19    }
20
21    public int getLabel(){
22        return this.label;
23    }
24
25    public String toString(){
26        String s = "";
27        for (Vertex vertex : neighbours) {
28            s += vertex.getLabel() + " ";
29        }
30
31        return s.trim();
32    }
33}
34

```

A classe Grafo (**Graph**), por sua vez, consiste em uma coleção de vértices. É possível adicionar vértices, pegar um vértice a partir do seu índice, dentre outros. Observe a implementação a seguir para esta classe.

```

1 import java.util.ArrayList;
2
3 public class Graph {
4
5     private ArrayList<Vertex> vertices;
6
7     public Graph(){
8         vertices = new ArrayList<Vertex>();
9     }
10
11    public void addVertex(Vertex v){
12        vertices.add(v);
13    }
14
15    public int size(){

```

```

16         return vertices.size();
17     }
18
19     public Vertex getVertex(int index){
20         return vertices.get(index);
21     }
22
23     public ArrayList<Vertex> getVertices(){
24         return vertices;
25     }
26
27 }

```

A partir de agora vamos combinar a utilização dessas duas classes junto ao programa principal (Main), o qual receberá dados da entrada do usuário segundo a formatação apresentada na seção anterior. Os comentários permitem um melhor entendimento do que está sendo feito a cada passo.

```

1  import java.util.Scanner;
2
3  public class Main {
4
5      public static void main(String[] args) {
6
7          Scanner in = new Scanner(System.in);
8          int countVertices = in.nextInt();
9          // Apenas para lidar com o \n da entrada após o primeiro inteiro
10         in.nextLine();
11
12
13         Graph myGraph = new Graph();
14         for (int i = 0; i < countVertices; i++){
15             // Criando todos os vértices do grafo
16             myGraph.addVertex(new Vertex(i));
17         }
18
19         // Vamos ler cada linha da entrada e criar a relação
20         //correspondente entre os vértices
21         for (int i = 0; i < countVertices; i++){
22
23             // Pegar o vértice da posição i para adicionar os vizinhos
24             Vertex vi = myGraph.getVertex(i);
25
26             // Leu a linha da entrada e converteu em uma lista
27             String[] line = in.nextLine().split("\\s+");
28
29             // Na iteração a seguir o j corresponde aos índices dos nós vizinhos
30             for (int j = 0; j < line.length; j++){
31                 // Pegou o j-ésimo valor da linha de entrada e converteu para inteiro
32                 int value = Integer.parseInt(line[j]);
33                 // Adicionou o v_j como vizinho do v_i
34                 if (value != 0){
35                     vi.addNeighbours(myGraph.getVertex(j));
36                 }
37             }
38         }
39
40         // Vamos agora imprimir todos os vértices e seus vizinhos
41         for (Vertex v : myGraph.getVertices()) {
42             System.out.println(v);
43         }
44     }
45 }

```

Ao informar a entrada do grafo exemplo, espera-se a seguinte saída para o código tal como está:

```
1 3  
0 2 3  
1 3 4  
0 1 2 4  
2 3
```

Essa saída pode ser interpretada como segue: o nó 0 possui os nós 1 e 3 como vizinhos, o nó 1 possui como vizinhos os nós 0, 2 e 3, e assim sucessivamente.

5 Implementando em Python

Utilizando Python 3, algumas estruturas de dados já disponíveis na linguagem simplificam consideravelmente a implementação. É o caso dos dicionários e das listas. Neste exemplo, vou utilizar um dicionário que mapeia o rótulo de um vértice para uma lista contendo os vértices adjacentes a ele. Observe que neste exemplo não vou utilizar aspectos da orientação a objeto, porém nem sempre isso é possível.

```
1 countVertices = int(input())  
2 # criando o dicionario  
3 graph = {}  
4  
5 for i in range(countVertices):  
6     # entrada convertida pra uma lista  
7     inputToList = [int(x) for x in input().split(" ")]  
8     adjacents = []  
9     # trocando os elementos pelos rotulos dos nos e eliminando  
10    # as entradas que sao iguais a zero  
11    for j in range(len(inputToList)):  
12        if inputToList[j] != 0:  
13            adjacents.append(j)  
14    # inserindo os vertices adjacentes a i no dicionario  
15    graph[i] = adjacents  
16  
17 print(graph)
```

6 Sua tarefa

Sua tarefa neste warm up é completar e modificar o código fornecido, em Python 3 ou Java, construindo o que for necessário para que a saída informe a sequência de graus do grafo fornecido como entrada. A sequência de graus consiste em uma sequência de números inteiros sem sinal separados por espaços em branco, em que os números são mostrados em ordem crescente. A sequência de graus corresponde aos graus dos vértices do grafo.

Para o exemplo fornecido como entrada, a saída é 2 2 3 3 4. Considere que apenas grafos válidos serão fornecidos como entrada. Lembre-se que vários grafos diferentes podem ser fornecidos pelo run.codes.

7 Links Úteis

- <http://www3.nd.edu/~kwb/nsf-ufe/1110.pdf>
- http://www.python-course.eu/graphs_python.php
- <http://visualgo.net/graphds>