Universidade Federal Rural de Pernambuco

Disciplina: *Algoritmos em Grafos*
Professor: *Pablo Azevedo Sampaio*
Semestre: *2017.1*

Trabalhos Práticos da 1ª VA

Parte 1

Implementar uma estrutura de dados (pode ser uma classe) para representar grafos.

Requisitos:

1) Potencialmente, o grafo pode ser direcionado, com múltiplas arestas e loops, ou seja, um **pseudo-digrafo**. Grafos não-direcionados podem ser representados transformando todas as arestas em arcos simétricos.

2) Usar representação de listas de adjacências

3) Oferecer, pelo menos, as seguintes funcionalidades:

- Operação para **inicializar** o grafo com N vértices/nós, os quais são numerados de 0 a N-1
- Função para acrescentar um **arco** (x,y), onde x e y são vértices válidos
- Função para retornar os **sucessores** de um nó v (entrada)
- Testar se **existe ou não um arco** (x,y), para x e y dados como entradas
- Função para retornar o **nome** de um nó x dado
    - O nome pode ser formado apenas por letras (e não tem espaços)
- Ler um grafo de um arquivo texto conforme o formato descrito a seguir

Formato do arquivo:

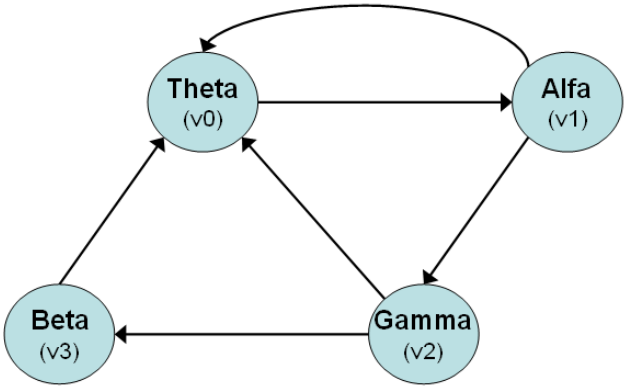
- A primeira linha indica a quantidade de nós N, onde os nós são identificados por um nome ou por um índice de 0 a N-1, conforme já indicado

- Em seguida, vem N linhas, onde cada linha seguinte se refere a um nó específico, na ordem de 0 a N-1
    - A primeira linha se refere ao nó 0, a segunda se refere ao nó 1, etc.

- Em cada linha referente a um nó **v**, são dadas estas informações, nesta ordem:
    - Uma string sem espaço em branco, formada apenas por letras, representando o **nome** (ou rótulo) do nó.
    - Em seguida, vem uma lista (possivelmente vazia) de inteiros representando os nós. Cada nó $\mathbf{u_i}$ presente nesta lista representa um sucessor de *v* (indicando a presença do arco $\mathbf{v} \rightarrow \mathbf{u_i}$)

- Exemplo:

  Representação visual de um digrafo, com os *nomes* dos vértices em negrito, e o número que o identifica entre parênteses:



  Arquivo, representando o grafo direcionado acima:

```
4
Theta 1
Alfa 0 2
Gamma 3 0
Beta 0
```

  (Para entender, compare o arquivo acima cuidadosamente com a explicação do formato de arquivo, dada na página anterior).

Parte 2

Esta parte consiste em implementar alguns testes de propriedades.

Usando a estrutura de dados da parte 1, vocês devem criar métodos (na própria classe ou em outra classe, como preferir) para, dado um grafo direcionado G de entrada, **testar estas propriedades**:

- Se **G** tem algum *loop*
- Se **G** tem arcos paralelos
- Se **G** é um grafo fracamente conectado.

(Fonte: Wikipedia)

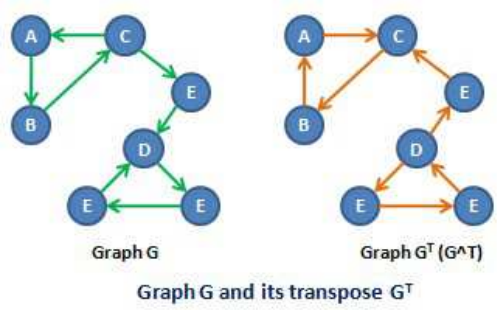
Cada um dos três problemas acima tem como entrada um grafo direcionado (digrafo) e dá saída binária (sim/não).

Parte 3

Fazer um dos dois:

1) Calcular o **transposto de G**, que é um grafo G^T tal que: para toda aresta *(x,y)* de G^T, existe uma aresta *(y,x)* em G e vice-versa (ou seja, G^T inverte as direções de todas as arestas de G). Retorne G^T como um grafo separado.

Exemplo de um grafo e seu transposto (do Wikipedia):

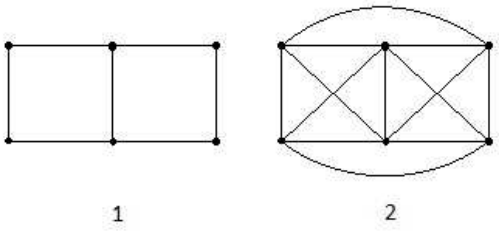


Graph G and its transpose G^T

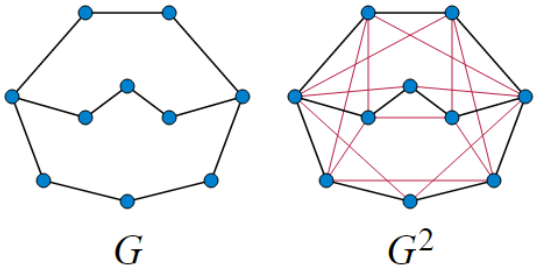
2) Calcular o **quadrado de G**, que é um grafo G^2 tal que: G^2 tem uma aresta *(x,y)* se e somente se *x* e *y* são diferentes e o menor caminho entre *x* e *y* tem, no máximo, 2 arestas (em outras palavras, se *y* é sucessor de *x* ou se *y* é o sucessor de um sucessor de *x*). Retorne G^2 como um grafo separado.

- Observe que G^2 não admite loops nem arestas paralelas

Exemplos de um grafo e seu quadrado:



(Internet – fonte desconhecida)



(Wikipedia)

Parte 4

Retornar a quantidade de componentes conectados de um digrafo qualquer. (Não precisa informar quais vértices compõem o componente).

- Entrada: (pseudo-)digrafo ***G***
- Saídas: um número inteiro (indicando a quantidade de componentes)