Universidade Federal de Itajubá – UNIFEI

Curso de Engenharia da Computação
ECOM02 – Teoria dos Grafos – Prof. Edmilson Marmo Moreira
Trabalho de Grafos com Implementação em PROLOG e HASKELL
Primeiro semestre de 2018

QUANTIDADE DE MEMBROS NO GRUPO:

O trabalho poderá ser realizado em grupos de, no máximo, 4 alunos.

FORMA DE APRESENTAÇÃO:

O trabalho deverá ser enviado através do SIGAA - *Sistema Integrado de Gestão de Atividades Acadêmicas* da UNIFEI.

Os códigos deste trabalho devem estar contidos em um único arquivo compactado denominado: `ECOM02.ZIP`

Haverá duas versões, uma na linguagem de programação Prolog e outra na linguagem de programação Haskell. O implementação de cada linguagem deve estar nos seguintes arquivos: `ECOPRO.PRO (.pl)` e `ECOHAS.hs`.

Os predicados da linguagem Prolog podem ser desenvolvidos em Turbo Prolog ou em um compilador do padrão ISO (como o GNU-Prolog).

DATA PARA ENTREGA E AVALIAÇÃO:

O data máxima para o envio do trabalho será: **03 de maio de 2018**.

No dia **04 de maio de 2018**, será realizado um teste para avaliar o conhecimento adquirido pelos alunos. O teste é individual e delimitará a nota máxima de cada membro do grupo.

A nota individual do trabalho somente poderá ultrapassar em 30% da nota obtida por cada aluno no teste do dia 03/05/2018.

Cada grupo tem até o dia **25 de abril de 2018** para enviar, por *e-mail*, a relação de alunos que farão parte do seu grupo.

IMPORTANTE: SEM ESTA INFORMAÇÃO, O ALUNO NÃO PODERÁ ENVIAR O TRABALHO E PERDERÁ OS PONTOS DESTA ATIVIDADE

DESCRIÇÃO DO TRABALHO:

Como explicado em sala de aula, o trabalho consiste na implementação de alguns métodos clássicos da Teoria dos Grafos através de predicados em Prolog e funções da linguagem Haskell.

- PARTE EM PROLOG:

Todas os predicados Prolog deverão ser desenvolvidos no mesmo arquivo.

A equipe poderá acrescentar predicados e declarações auxiliares, se for necessário. Entretanto, para que se mantenha o padrão que está sendo utilizado no curso, os grafos (independente de serem orientados ou não) deverão seguir a declaração:

```
Domains
  listaInt=integer*
  no=no(integer, listaInt)
  grafo=no*
```

Para que fique claro como os predicados deverão ser desenvolvidos, o exemplo a seguir ilustra a implementação de um método para verificar se um determinado grafo é regular.

```
Predicates
  grau(listaInt,integer).
  regular(grafo).
Clauses
  grau([],0).
  grau([_|Cauda],N):-
    grau(Cauda,N1),N=N+1.

  regular([_]).
  regular([no(_,Adj1),no(V,Adj2)|Cauda]):-
    grau(Adj1,T),grau(Adj2,T),
    regular([no(V,Adj2)|Cauda]).
```

A seguir serão apresentadas as descrições de cada método que deverá ser desenvolvido, com as respectivas assinaturas dos predicados.

1) Predicado para retornar a ordem de um grafo.

```
Predicates
  ordem(grafo,integer).
```

2) Predicado para retornar o tamanho de um grafo.

```
Predicates
  tamanho(grafo,integer).
```

3) Predicado para verificar se um grafo é Euleriano.

```
Predicates
  euleriano(grafo).
```

4) Predicado para encontrar o complemento de um grafo.

```
Predicates
  complemento(grafo,grafo).
```

5) Predicado para verificar se um grafo é um *multigrafo*.

```
Predicates
  multigrafo(grafo).
```

6) Predicado para retornar uma lista contendo o resultado de uma busca em profundidade a partir de um vértice dado.

```
Predicates
  buscaProf(grafo,integer,listaInt).
```

8) Predicado para identificar a quantidade de componentes conexas de um grafo.

```
Predicates
  componentesConex(grafo,integer).
```

- PARTE EM HASKELL:

Todas as funções deverão ser desenvolvidas no mesmo arquivo.

A equipe poderá acrescentar funções e declarações auxiliares, se for necessário. Entretanto, para que se mantenha o padrão que está sendo utilizado no curso, os grafos (independente de serem orientados ou não) deverão seguir a declaração:

```
type Vertice = (Int, [Int])
type Grafo = [Vertice]
```

Para que fique claro como as funções deverão ser desenvolvidas, o exemplo a seguir ilustra a implementação de uma função para verificar se um determinado grafo é regular.

```
grau :: [Int] -> Bool
grau [] = 0
grau (_:cauda) = 1 + grau(cauda)

regular :: Grafo -> Bool
regular [_] = True
regular ((v1,x):(v2,y):cauda) = grauVertice(x) == grauVertice(y) &&
                                regular((v2,y):cauda)
```

A seguir serão apresentadas as descrições de cada método que deverá ser desenvolvido, com as respectivas assinaturas dos predicados.

1) Função para retornar a ordem de um grafo.

```
Ordem :: Grafo -> Int
```

2) Função para retornar o tamanho de um grafo.

```
Tamanho :: Grafo -> Int
```

3) Função para verificar se um grafo é Euleriano.

```
Ordem :: Grafo -> Bool
```

4) Função para encontrar o complemento de um grafo.

```
Complemento :: Grafo -> Grafo
```

5) Função para verificar se um grafo é um *multigrafo*.

```
multigrafo :: Grafo -> Bool
```

6) Função para retornar uma lista contendo o resultado de uma busca em profundidade a partir de um vértice dado.

```
BuscaProf :: Grafo -> Int -> Grafo
```

7) Função para retornar uma lista contendo o resultado de uma busca em largura a partir de um vértice dado.

```
BuscaLarg :: Grafo -> Int -> Grafo
```