Universidade Federal de São Carlos
Construção de Compiladores 2

LGraph : linguagem para manipulação de grafos

Wilton Vicente Gonçalves da Cruz RA: 586889
Guilherme Lemos RA: 587010
Wellyson Freitas RA: 587125
Vitor Machado Cid RA: 564478
Ciência da Computação
2° semestre de 2016

Sumário

1. Visão geral

A linguagem de programação “LGraph” foi desenvolvida com o propósito de auxiliar na criação e manipulação de grafos. Desta forma, buscou-se através de aspectos sintático e semânticos oferecer ao utilizador uma linguagem intuitiva e familiar a área de grafos, fazendo em uma linha de código o que necessitaria de várias linhas em outras linguagens.

LGraph contém tipos básicos como inteiros, reais e cadeia de caracteres, além de tipos especifícos como “nodes” e “edges”, os principais componentes de um grafo, além do tipo “graph”. A criação de um grafo é facilitada com a necessidade apenas de se informar os nós e arestas do mesmo. São oferecidos comandos para atualizar um grafo e encontrar métricas sobre um grafo e salvar os resultados em um arquivo.

É dada também a possibilidade de se criar nós com atributos para um grafo e, através de uma estrutura de repetição percorrer os nós de um grafo (com ou sem atributos) e, utilizando de estruturas condicionais, fazer operações entre estes vértices e outras variáveis.

Na sequência deste manual serão apresentados as regras sintáticas, léxicas e semânticas de LGraph. Serão dados exemplos para facilitar o entendimento de tais regras.

2. Aspectos sintáticos

Um programa em LGraph deve, obrigatoriamente, iniciar com a palavra-chave “begin” e terminar com a palavra-chave “end”.

2.1 Declaração de variáveis

Após a inserção de “begin”, deve vir a área de declaração de variáveis, área esta iniciando com a palavra-chave “data”. Uma variável deve ser um identificador começando obrigatoriamente com uma letra e sendo uma combinação de letras, dígitos e “_”. Por exemplo:

var1 => permitido 1var => não permitido

Variáveis de mesmo tipo podem ser declaradas conjuntamente, devendo ser separadas por vírgulas. Escrita(s) a(s) variável(is) deve ser informado seu tipo. Assim, deve colocar “:” seguido do tipo (falaremos de tipos mais abaixo). Exemplo:

```
var : int
aux1, aux2 : string
```

No caso de se declarar uma variável do tipo “nodes” com atributos, deve-se informar entre parênteses os identificadores dos atributos seguidos de “:” e de seu tipo, separados por vírgulas, tipo este que pode ser “int”, “float” ou “string” apenas. Exemplo:

```
nos : nodes(at1 : int, at2: float, at3 : string)
```

2.2 Tipos

São seis os tipos de variáveis fornecidos por LGraph. Três tipos são básicos e populares. São eles o “int”, que corresponde a um número inteiro, “float”, um número real e “string”, uma cadeia de caracteres. Além disso, são suportados mais três tipos familiares ao domínio de grafos. O tipo

“nodes”, que representa um conjunto de nós de um grafo, o tipo “edges”, as arestas do grafo e o tipo “graph”, o grafo em si.

Tipos
int
float
string
nodes
edges
graph

Para o tipo “int” são suportados números inteiros com e sem sinal. Números reais também podem ser positivos ou negativos, porém com a presença de um “.” representando casas decimais. Exemplos:

234 => “int” 345.43 => “float”

O tipo “graph” corresponde a um identificador que segue as convenções já citadas anteriormente. O tipo “nodes” tem uma estrutura especial. Caso ele seja um conjunto de nós sem atributos, sua estrutura sintática consiste em uma lista com string, int ou float separados por vírgulas e casa um representando um nó. Por exemplo, uma variável “nodes” com nós identificados por “A”, 1 , “B” seria assim:

varNodes = [“A”, 1, “B”]

Porém, o tipo “nodes” pode ter atributos nos nós. Assim, sua estrutura sintática seria uma lista de tuplas, onde cada tupla conteria o identificador do nó (string, int ou float) seguido de seus atributos previamente declarados. Sua sintaxe seria assim:

varNodesAtributos = [(idNo1, atrib1 = valor1, atrib2 = valor2,….) , (idNo2, atrib1 = valor1, atrib2 = valor2, ….) , ….]

Onde idNo1, …. , idNoJ seriam os identificadores de cada nó, ou seja, uma string, um inteiro ou um valor real. Já atrib1, …. , atribJ seriam os atributos do tipo “nodes” declarado, seguido de seus valores de inicialização de acordo com o tipo de cada um deles. Por exemplo, caso fosse uma declarada “varNodesAtributos”:

varNodesAtributos : nodes(peso:int , filho : string, valor:int)

Uma possível inicialização para “varNodesAtributos” seria:

varNodesAtributos = [(“A”,peso = 1, filho = “next”, valor = 4) , (“G”, peso = 1, filho = “left”, valor = 234)]

O tipo “edges” também tem uma sintaxe específica. Ele consiste também em uma lista de tuplas. Cada tupla deve ter três elementos. Os dois primeiros devem ser os identificadores de nós que ligam a aresta. Podem ser de tipo “int”, “string” ou “float”. O terceiro elemento deve ser um valor inteiro positivo ou negativo que represente o peso desta aresta. Um detalhe é que se for usado

um identificador de nó não existente no grafo, automaticamente o mesmo será inserido. A sintaxe geral é essa:

edges = [(idNo, idNo, peso), (idNo,idNo, peso) , …….]

Por exemplo, caso eu queira uma aresta que ligue um nó “X” a um nó “Y” com peso 0, outra que ligue “Y” a “Z” com peso -1 e outra que ligue “X” a 1 com peso 5, teríamos:

edgesGrafo1 = [(“X”, “Y”, 0) , (“Y”, “Z”, -1) , (“X”, 1, 5)]

2.3 Comando para inicializar grafo

A linguagem “LGraph” fornece um comando para inicialização de um grafo. Sua sintaxe é que segue.

create graph idGrafo (nodes = nosGrafo , edges = arestasGrafo)

Onde “idGrafo” corresponde a um identificador de um grafo previamente declarado com tipo “graph”, “nosGrafo” uma variável de tipo “nodes”, com ou sem atributos, e “arestasGrafo” de tipo “edges”, ambas previamente declaradas.

2.4 Comando para atualizar grafo

Um comando para atualizar um grafo já inicializado também é fornecido. Ele permite que adicione-se novas arestas e nós ao grafo. Sua sintaxe geral:

update graph idGrafo with (nodes = nosGrafo, edges = arestasGrafo)

Onde “idGrafo” corresponde a um identificador de um grafo previamente declarado com tipo “graph”, “nosGrafo” uma variável de tipo “nodes”, com ou sem atributos, e “arestasGrafo” de tipo “edges”, ambas previamente declaradas.

2.5 Comando para ler grafo a partir de arquivo externo

A linguagem “LGraph” permite que, ao invés de se criar um grafo manualmente através do comando “create”, seja possível obter um grafo a partir de um arquivo externo com formato específico. Será dedicada uma seção ao longo deste manual para tal comando e suas restrições. Sua sintaxe seria a seguinte:

read graph idGrafo from pathArquivo

Onde “idGrafo” é o identificador de grafo de tipo “graph” previamente declarado e “pathArquivo” é uma variável tipo “string” com o caminho completo do arquivo ou uma “string” diretamente. Exemplo:

```
read graph grafo1 from “home/fulano/grafo.paj”
caminho = “home/fulano/grafo.paj”
read graph grafo1 from caminho
```

2.6 Comando para plotar grafo

“LGraph” possui um comando simples para plotar gráfico em um arquivo de imagem “png”. Para plotar um gráfico basta usar o comando “plot” que possui a seguinte sintaxe:

plot graph idGrafo

Onde “idGrafo” é a varável de tipo “graph” previamente declarada.

2.7 Comando para encontrar métricas

Um comando para encontar métricas em um grafo ou em cada vértice de um grafo foi criado. Mais detalhes serão dados na sequência deste manual. Sua sintaxe geral é:

find metrica of graph idGrafo ou **find metrica of vertex idVertice in graph idGrafo**

Onde “metrica” corresponde ao nome da métrica a ser calculada (quais estão disponíveis serão dadas na sequência), “idGrafo” é a variável de tipo “graph” previamente declarada e “idVertice” um identificador para um vértice do grafo.

2.8 Comando para imprimir

O comando “print” permite imprimir o valor de uma variável declarada de tipo “string”, “int” ou “float, apenas, além de uma cadeia de caracteres diretamente. A impressão é feita em um arquivo de saída “print.txt”. Apenas um elemento pode ser impresso por vez. A sintaxe geral é a que segue. Ele pode estar em todo o programa, porém dentro de um “foeach” pode apenas estar dentro de uma condicional “if-else”.

print(ident) ou print(“texto”)

2.9 Comando de atribuição

A atribuição de uma variável por outra ou por um valor é feita com o operador “=”. A sintaxe é:

idVar = atribuido (variável ou valor, por exemplo, 123)

2.10 Comando de repetição para percorrer grafo

Um comando de repetição “foreach” permite percorrer cada vértice de um dado grafo. Dentro deste comando são permitidos comandos condicionais “if-else”. Sua sintaxe geral é:

foreach vertex idVertice in idGrafo
begin

comandos condicionais

end

Onde “idVertice” é um identificador para os vértices a serem percorridos, não precisando ser declarado, porém com nome diferente de outras variáveis e “idGrafo” o identificador de tipo “graph” previamente declarado. Para acessar o valor de um atributo de um vértice, caso ele exista,

deve-se sar o “.”. Por exemplo. Caso o vértice do “loop” seja identificado por “v” e ao menos um nó do grafo tenha um atributo “peso”, para acessá-lo deve-se usar “v.peso”.

2.11 Comando condicional “if-else”

Um comando condicional “if-else” segue o padrão usual de outras linguagens de programação. Sua sintaxe é:

```
if( expressão_booleana1 op_logico expressão_booleana2 op_logico…… expressao_booleanaN)
begin

        comandos impressão ou atribuição

end
else
begin

        comandos impressão ou atribuição

end
```

Onde “expressao_booleana” corresponde a uma expressão relacional ou de igualdade (falaremos mais abaixo) e “op_logico” são operadores lógicos “AND” e “OR” (falaremos depois). A estrutura de “else” é opcional.

2.12 Expressões booleanas

Um expressão booleana consiste em um expressão cuja análise retorna valor verdadeiro ou falso. Em “LGraph” estas expressões podem ser relacionais ou de igualdade. No caso de expressões relacionais os operandos podem de tipo “int” ou “float” apenas, sejam valores diretos como “123” ou através de variáveis declaradas. Os operadores relacionais são “<”, “>” , “<=” ou “>=”. Por exemplo,

```
var : int
var > 1 => expressão relacional válida !

“ser” > “se” => expressão relacional inválida !
```

No caso de expressões de igualdade os operadores são “==” ou “!=” (diferente) . Além de valores inteiros e reais são permitidas cadeias de caracteres.

```
str : string
str = “o”
str == “ok” => expressão de igualdade válida!

g1 : graph
g1 > 1 => expressão relacional inválida! (tipo graph não permitido)
```

Combinações permitidas de tipos nas expressões serão mostradas em aspectos semânticos.

2.13 Operadores lógicos

Os operadores lógicos a serem usados no comando “if” entre expressões booleanas são o “E” e o “OU”. Em “LGraph” são representados por “AND” e “OR”, respectivamente.

2.14 Comentários

Comentários podem ser feitos usando “/*” para abrí-los e “*/” para fechá-los.

3. Aspectos semânticos

Nesta seção serão apresentados aspectos semânticos de “LGraph” em cada um dos comandos possíveis. Serão apresentadas as regras semânticas e erros associados.

3.1 Declaração de variáveis

Exceto o identificador informado no comando “foreach” para o vértice, todas as outras variáveis devem ser declaradas previamente na seção “data” do programa. Caso seja utilizada uma variável não declarada, o compilador apresentará erro semântico de variável não declarada. Exemplo:

var1,var2:int
var1 = 1 => OK!
var4 = 5 => ERRO! Variável “var4” não declarada!

3.2 Atribuição de variáveis

A atribuição de variáveis é feita usando o operador “=” como falado. Só é permitida atribuição de tipos iguais. Por exemplo, valores numéricos inteiros com variáveis de tipo “int”, valores reais com variáveis tipo “float”, cadeia de caracteres com variáveis tipo “string” e entre variáveis de mesmo tipo.

var1 : int, var2 : string
var1 = “ola” => ERRO! Atribuição de cadeia de caracteres em variável de tipo “int”

var2 = 1 => OK! Variável de tipo “int” recebe valor numérico 1.

3.3 Comando CREATE

A variável de tipo “graph” que deve ser passada no comando “CREATE” deve ter sido declarada. Caso não tenha sido declarada ou não seja de tipo “graph” o compilador acusará erro. As variáveis de inicialização de nós e arestas também devem ter sido declaradas e serem de tipos “nodes” ou “nodes_com_atributos” e “edges”, respectivamente. Caso contrário, o compilador acusará erro. Caso tente-se criar um grafo já criado anteriormente com o comando “CREATE”, ocorrerá erro também. Exemplo:

create graph grafo1 (nodes = n , edges = e) => OK!
create graph grafo1 (nodes = n1 , edges = e1) => ERRO! “grafo1” já criado!

No processo de criação de grafo, as variáveis “nodes” e “edges” não necessitam terem sido inicializadas. Porém, os vértices indicados nas arestas devem estar obrigatoriamente nos nós adicionados, ao contrário ocorrerá erro de execução do “script” python gerado. Assim, no momento

em que se cria um grafo com nós sem atributos, apenas nós sem atributos poderão ser adicionados com “update”, e vice-versa.

3.4 Comando UPDATE

Caso o identificador do grafo passado no comando “update” se refira a uma variável de tipo não “graph” ou não tenha sido declarada, ocorrerá um erro. Além disso, caso tente-se atualizar um grafo que ainda não foi criado com “create” também ocorrerá erro. Além disso, caso o grafo a ser atualizado tenha sido criado com vértices sem atributos, na atualização apenas vértices sem atributos poderão ser adicionados, e o contrário também é verdadeiro. Se isso não ocorrer, um erro semântico será emitido.

3.5 Comando PLOT

O comando “plot” apenas exige um identificador para o grafo a ser plotado. Assim, a variável deve ser de tipo “graph” e deve ter sido declarada. O grafo correspondente deve também ter sido inicializado com “create”, senão ocorrerá erro. Porém, o arquivo “.png” a ser gerado será o referente ao último comando “plot” executado, por isso, faz sentido apenas usar uma vez o comando “plot” para cada grafo.

3.6 Comando FIND

O comando “find” encontrar uma métrica de um grafo ou de um vértice de tal grafo e imprime os resultados num arquivo “metricasIdGraf.txt”. Assim, caso o identificador de grafo passado como parâmetros não seja de tipo “graph” ou não tenha sido declarado será emitido erro. Além disso, o nome da métrica ser encontrada deve ser uma das disponíveis e que serão mostradas ainda neste manual. Caso deseje-se encontrar a métrica de um vértice específico, deve-se passar uma “string” com seu identificador. Por exemplo,

find degree of vertex “A” in graph grafo1

deseja-se encontar a métrica “degree” no vértice “A” do grafo1. Caso não exista o vértice no grafo, não será dado erro, apenas será impresso com valor de métrica vazio.

Outra observação consiste no fato de que algumas métricas só se aplicam a grafo ou a vértice. Assim, caso deseje-se encontrar a métrica para um objeto inválido, um erro será emitido.

3.7 Comando FOREACH

O comando comando “foreach” recebe um identificador que represetará os vértices de um dado grafo. O identificador passado não necessita ter sido definido na seção “data”, apenas deve seguir as convenções léxicas para identificador e não ter sido utilizado em comandos “foreach” anteriores. O identificador do grafo a ser percorrido deve ser uma variável do tipo “graph” e ter sido previamente declarada.

Caso o grafo a ser percorrido não tenha sido inicializado com “create”, também será emitido erro.

3.8 Comando IF-ELSE

O comando “if-else” consiste em uma estrutura condicional. A expressão booleana em “if” deve ser uma combinação de expressões relacionais, de igualdade e lógicas. Caso uma variável não declarada seja usada na construção da expressão booleana, um erro semântico será gerado.

3.9 Expressões relacionais

Expressões relacionais podem ter operadores “>” , “>=” , “<” e “<=”. Os operandos devem ser apenas números inteiros, reais e variáveis de tipo “int” e “float”. Assim, caso isto não aconteça, um erro semântico será emitido. Expressões relacionais só podem ser usadas dentro de estruturas “IF”.

3.10 Expressões de igualdade

Expressões de igualdade podem ter operadores “==” e “!=” (diferente). Estas comparações podem ser feitas com combinações de números inteiros, reais, cadeias de caracteres e variáveis de tipo “string”, “int” e “float”.

3.11 Comando PRINT

Caso deseje-se imprimir uma variável, esta deve ser de tipo “int”, “float” ou “string” ou o vértice de “foreach”. Caso isto não aconteça, um erro semântico será emitido, assim como se esta variável não tiver sido previamente declarada.

3.12 Comando READ

Caso o endereço do arquivo de grafo seja passado através de uma variável do tipo “string”, esta deve ser de tipo “string” e deve ter sido previamente declarada. Além disso, caso o caminho do arquivo passado seja inválido ou o arquivo esteja com problemas ou não seja um arquivo das extensões a serem posteriormente explicadas, nenhum erro será emitido, cabendo ao usuário ficar atento a isso.

4. Leitura de grafos a partir de arquivos

A leitura de grafos a partir de arquivos só será permitida para extensões “.paj”. Porém, nenhum verificação será feita quanto a isso. Além disso, nem todos os arquivos “.paj” possuem o mesmo padrão, podendo ocorrer caracteres inválidos. Assim, cabe ao usuário ficar atento a isso, visto que o arquivo gerado em “Python” não será executado e nada será criado na pasta de saída do compilador.

Serão fornecidos alguns “datasets” “.paj” para fins de teste.Estes testes estão em pasta nomeada “DataSetsPaj”.

5. Calculando métricas

As métricas fornecidas por “LGraph” por enquanto se baseiam em grau e conectividade. Na tabela abaixo o nome da métrica, em que objetos se aplicam e o que significam. Na chamada do comando “find” será gerado um arquivo “txt” com o identificador do grafo unido com “Metricas”. Por exemplo, caso queira se achar uma métrica no grafo ou no vértice de “grafo1”, então será gerado o arquivo “grafo1Metricas.txt” com esta e outras possíveis métricas.

Métrica	Objeto	Significado
degree	Grafo ou vértice	Grau de todos os vértices de um grafo ou do vértice especificado.
degree_centrality	Grafo ou vértice	Centralidade de grau de todos os vértices do grafo ou do vértice especificado.
average_node_connectivity	Grafo	Média de conectividade de nós do grafo especificado.
edge_connectivity	Grafo	Média de conectividade das arestas do grafo especificado
node_connectivity	Grafo	Média de conectividade dos nós do grafo especificado.

Buscou-se fornecer métricas que não fazem distinção entre grafos direcionados. Em “LGraph” não é possível criar grafos direcionados. Porém, podem ocorrer problemas no cálculo de métricas, apesar de todas elas terem sido intensamente testadas.

6. Usando o compilador

6.1 Requisitos

Antes de usar o compilador LGraph existem alguns requisitos:

1) Instalação do módulo “networkx” para manipulação de grafos pelo código Python gerado.
2) Leitor de arquivo texto simples.
3) Leitor de imagens “png”.
4) Máquina virtual JAVA instalada para execução de arquivo “.jar”.

6.2 Conteúdo

O compilador “LGraph” consiste em um arquivo “.jar”. Junto a ele são fornecidos alguns recursos. São fornecidos conjunto de dados de formato “.paj” na pasta “DataSetsPaj”, arquivos exemplos de programas em “LGraph” , casos de teste com e sem erros, além deste manual de instruções.

6.3 Executando o compilador

A execução do compilador “LGraph” se dá por linha de comando. Com a máquina virtual JAVA previamente instalada e endereçada, deve seguir a seguinte sintaxe para execução do compilador “LGraph” com um arquivo de entrada com o programa a ser compilado:

java -jar [path]LGraph.jar [path]arquivoEntrada.lgraph

O valor de “path” no comando acima corresponde ao caminho completo do arquivo “Lgraph.jar” e do arquivo do programa a ser compilado “arquivoEntrada.lgraph”. Para facilitar, sugere-se executar o compilador dentro do diretório do “.jar” e/ou do arquivo de entrada. A extensão do arquivo de entrada pode ser “.lgraph”, “.txt”, sem nenhuma extensão, com a única restrição de ser um arquivo texto.

6.4 Saída do compilador

Após a execução do compilador, será gerado um diretório “LGraph” no diretório atual do terminal no momento em que o compilador foi executado. Dentro deste diretório estarão os resultados do processo de compilação.

Caso ocorreu erro de execução, léxico, sintático ou semântico, apenas um arquivo “erro.txt” será gerado no diretório “LGraph”. Caso não ocorram erros, será gerado um arquivo “Python” que será executado automaticamente e gerará arquivos de acordo com os comandos do programa. Por exemplo, caso ocorra comandos “plot” arquivos de imagem “.png” serão gerados. Caso sejam calculadas métricas arquivos texto serão gerados, bem como comando “print”.

7. Exemplos

Os exemplos a serem aqui mostrados estão disponíveis no diretório “Exemplos” .

7.1 Criando grafo com nós sem atributos e plotando

O programa abaixo declara três variáveis: grafo de tipo “graph”, nos de tipo “nodes” e arestas de “edges”. Correspondem ao grafo, nós e arestas, respectivamente. Posteriormente, são inicializadas variáveis com os identificadores dos nós, as arestas que os ligam e plota-se o grafo criado.

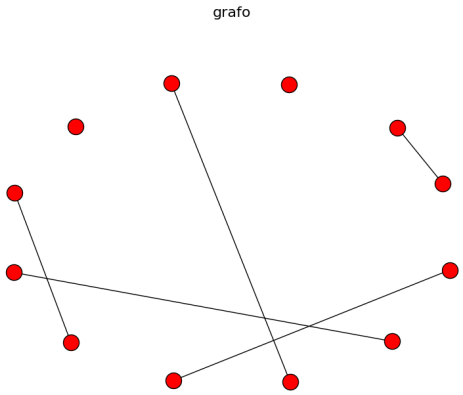
```
begin
    data grafo: graph,
        nos : nodes,
     arestas : edges

    nos = ["A", "B", 1, 2, 4, 5, "C", "D", 8.9, "O", "ex", "9"]
    arestas = [("A","B",0), ("9",5,1), ("C", 1, 0), ("ex",8.9,0), (4,2,0)]
    create graph grafo (nodes = nos, edges = arestas)

    plot graph grafo

end
```

No diretório “LGraph” foi gerado o arquivo “Python”, o arquivo de “print” sem nada, visto que não há comandos de “print” e a imagem “.png” correspodente ao grafo plotado.



7.2 Criando um grafo de nós com atributos e plotando

O programa abaixo declara uma variável de tipo “graph”, outras de tipo “nodes” , porém comatributos “peso” e “filho”, tipos “int” e “string”, além da variável arestas de “tipo” edges. De posse destas inicializadas, cria o grafo. Posteriormente o plota.

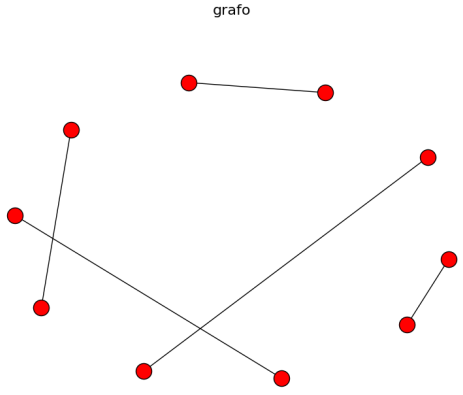
```
begin
        data grafo: graph,
            nos : nodes(peso:int, filho:string),
         arestas : edges

        nos = [("A",peso=1,filho="aroldo"), ("C",peso=45,filho="karl"), (1,peso=3,filho="lenin"),
        ("B",peso=4,filho="trump")]
        arestas = [("A","B",0), ("9",5,1), ("C", 1, 0), ("ex",8.9,0), (4,2,0)]
        create graph grafo (nodes = nos, edges = arestas)

        plot graph grafo

end
```

Uma observação é que na declaração das arestas são colocados nós que não foram informados na variável “nos”. Porém, devido a flexibilidade da linguagem Python, o nó inserido automaticamente, porém sem atributos. O gráfico gerado está abaixo.

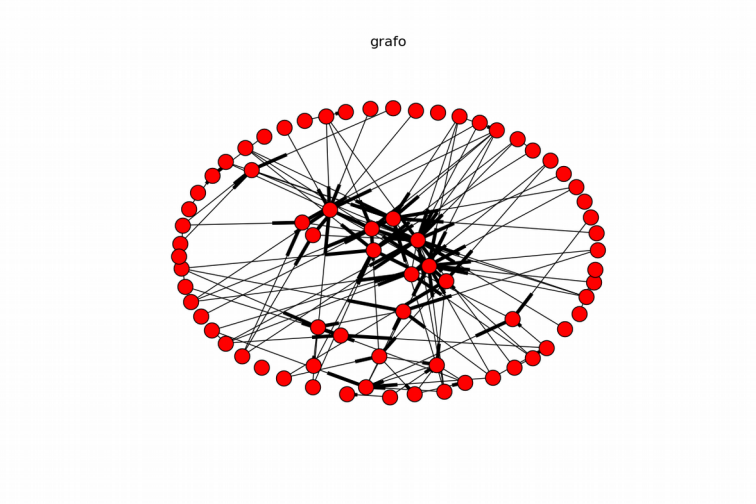


7.3 Lendo um grafo a partir de um arquivo e plotando

No programa abaixo, é declarada uma variável “graph” para conter o grafo. No comando “read” é lido do arquivo especificado no “path” o grafo a ser guardado em “grafo”. Posteriormente , é pedido para se plotar este grafo.

```
begin
    data grafo: graph
    read graph grafo from file "/home/wilton/Documents/CC2/T3/DataSetsPaj/futebol.paj"
    plot graph grafo

end
```



7.4 Calculando métricas em grafo

Na variável “grafo” é lido o grafo de um arquivo. Usando o comando “find” encontra-se métrica de grau “degree” que retorna o grau de todos os nós, além da conectividade das arestas do grafo. Estas métricas são salvas em um arquivo “grafoMetricas.txt”, arquivo este na pasta “Exemplo4” da pasta “Exemplos”.

```
begin
    data grafo: graph
    read graph grafo from file "/home/wilton/Documents/CC2/T3/DataSetsPaj/futebol.paj"
    find degree of graph grafo
    find edge_connectivity of graph grafo

end
```

7.5 Calculando métrica em um vértice de um grafo

O programa abaixo cria um grafo “grafo” com alguns nós sem atributos e algumas arestas. Posteriormente, encontra a métrica de um vértice específico, no caso, do “A”. Um arquivo de saída “grafoMetricas.txt” é gerado apontando como grau de “A” o valor dois.

```
begin
    data grafo: graph, nos:nodes, arestas:edges
```

```
    nos = ["A","B","C","D","E","F"]
    arestas = [("A","B",0), ("C","F",1), ("D","E",8), ("A","F",0)]
    create graph grafo (nodes = nos, edges = arestas)
    find degree of vertex "A" in graph grafo
end
```

7.6 Percorrendo vértices com atributos de um grafo

O programa abaixo percorre os vértices de “grafo”. Os nós inseridos em “grafo” tem um atributo “cor” de tipo “string”. Assim, quando o vértice possuir a cor “azul” será impressa uma mensagem “vertice com cor azul”.

```
begin
    data grafo: graph, nos:nodes(cor:string), arestas:edges
    nos = [("A",cor="azul"),("B",cor="vermelho"),("C",cor="roxo"),("D",cor="amarelo"),
    ("E",cor="cinza"),("F",cor="branco")]
    arestas = [("A","B",0), ("C","F",1), ("D","E",8), ("A","F",0)]
    create graph grafo (nodes = nos, edges = arestas)

    foreach vertex v in grafo
    begin
        if(v.cor=="azul")
        begin
           print("vertice com cor azul")
        end

    end
end
```

7.7 Atribuição de variáveis e impressão

O programa abaixo declara duas variáveis de tipo “int”, uma de tipo “float” e outra de tipo “string”. Atribui a “inteiro” um valor inteiro, a “real” um valor real, “cadeia” uma cadeis de caracteres e a “inteiro2” o valor em “inteiro”. Posteriormente imprime tais valores. A variável “inteiro2” passa a ter valor de “inteiro”.

```
begin
    data inteiro,inteiro2:int, real:float, cadeia:string

    inteiro = 2
    real = 3.45
    cadeia = "ola mundo"
    inteiro2 = inteiro
    print(inteiro)
    print(real)
    print(cadeia)
    print(inteiro2)
end
```

7.8 Encontrando maior peso entre vértices com atributos em um grafo

No programa abaixo é criado um grafo com vértices que possuem um atributo “peso” de tipo “float”. A variável “maior” , também de tipo “float”, guardará o maior peso encontrado nos vértice de “grafo”. Fora do “loop”, ela será impressa.

```
begin

    data grafo: graph, nos:nodes(peso:float), arestas:edges, maior:float

    nos = [("A",peso=1.3),("B",peso=4.5),("C",peso=6.9),("D",peso=7.5),("E",peso=1.7),
("F",peso=7.9)]
    arestas = [("A","B",0), ("C","F",1), ("D","E",8), ("A","F",0)]
    create graph grafo (nodes = nos, edges = arestas)
    maior = 0.0

    foreach vertex v in grafo
    begin
        if(v.peso>=maior)
        begin
           maior = v.peso
        end

    end
    print(maior)
end
```

A saída no arquivo “print.txt” foi 7.9.

7.9 Exemplo completo

O programa abaixo busca unir todos os comandos de “LGraph”. Ele está todo comentado e une os conceitos já apresentados em exemplo anteiores. Como já dito anteriormente, os resultados da compilação deste e outros exemplo encontra-se no diretório “Exemplos”.

```
begin
    data grafo,grafo1:graph, path:string, nos,nosUP:nodes(peso:int, cor:string), arestas:edges,
maior:int

    /* Caminho arquivo de .paj */
    path = "/home/wilton/Documents/CC2/T3/G19-LGraph/DataSetsPaj/teste.paj"

    /* Leitura do grafo para var grafo */
    read graph grafo from file path

    /* Encontrando metricas em grafo */
    find degree of graph grafo
    find edge_connectivity of graph grafo
```

```
/* Percorrendo vertices de grafo e imprimindo seus ids */
foreach vertex v in grafo
begin
        if(1==1)
        begin
                print(v)
        end

end

/* Plotando grafo */
plot graph grafo

/* Criando grafo com atributos manualmente */

/* Vertices */
nos = [("A",peso=1,cor="vermelho"), ("B",peso=9,cor="roxo"),
("C",peso=10,cor="azul"), ("D",peso=3,cor="amarelo"), ("U",peso=2,cor="rosa")]

/* Arestas: devem conter vertices em nos */
arestas = [("A","B",0), ("C","D",1)]

/* Criacao do grafo grafo1 */
create graph grafo1 (nodes=nos, edges=arestas)

/* Plotando grafo1 */
plot graph grafo1

/* Variavel maior auxiliar para encontrar maior peso */
maior = 0

/* Encontra vertice cor azul e maior peso*/
foreach vertex j in grafo1
begin
        if(j.cor=="azul")
        begin
                print("Vertice com cor azul:")
                print(j)
        end
        if(j.peso >= maior)
        begin
                maior = j.peso
        end
end

/* Imprimir maior peso */
print("O maior peso é:")
print(maior)
```

```
        /* Atualizando vertices e arestas de grafo1 */
        nosUP = [("E",peso=90,cor="purpura"),("H",peso = 2, cor="marrom")]

        arestas = [("E","A",0),("H","C",9)]

        update graph grafo1 with (nodes = nosUP, edges = arestas)

        plot graph grafo1

        maior = 0

        /* Encontra vertice cor azul e maior peso*/
        foreach vertex k in grafo1
        begin
                if(k.cor=="azul")
                begin
                        print("Vertice com cor azul:")
                        print(k)
                end
                if(k.peso >= maior)
                begin
                        maior = k.peso
                end
        end

        /* Imprimir maior peso */
        print("O maior peso apos update é:")
        print(maior)

end
```