Linguagem para manipulação de grafos

André Levi Zanon - 619922;

Rodrigo Ferrari de Souza - 619795;

Victor Hugo Domingos de Abreu - 619841;

Victor Santos Ferrari - 552437.

Visão Geral

- Projeto de linguagem de programação para manipulação de grafos.
- Criação de Grafos:
  - Inserção de vértices e arestas;
  - Atribuição de valores;
- Aplicação de algoritmos clássicos da Teoria dos Grafos:
  - Dijkstra;
  - Busca em Largura;
  - Busca em Profundidade;
- Auxílio ao aprendizado de conceitos da Teoria dos Grafos;

Gramática Final

```
grammar Grafos;

@members{...}

algoritmo : 'declaracoes' declaracao 'inicio' codigo 'fim' ;

declaracao : variavel declaracao | ;

variavel : tipo IDENT mais_variavel ;

mais_variavel : ',' IDENT mais_variavel | ;

tipo :  'grafo'     |
        'vertice'   |
        'int'       |
        'vetor'     ;

codigo : (instrucoes)* ;
```

Gramática Final

```
                //cria aresta em um grafo, delimitado por dois veritices e um peso
instrucoes :    'aresta' '(' pa_grafo=IDENT ',' pa_vertice1=IDENT ',' pa_vertice2=IDENT ',' a_int=INTEIRO')'
                // remove de um grafo o vertice
                | 'remove_vert' '(' pr_grafo=IDENT ',' pr_vertice= IDENT ')'
                //define-se em um grafo, o custo para chegar no vertice
                | 'set_custo_para_vertice' '(' ps_grafo=IDENT ',' ps_vertice=IDENT ',' int_ou_ident ')'
                //imprime uma variavel ou cadeia
                | 'imprime' '(' print_grafo=IDENT ')'
                | 'listar' '(' var_ou_cadeia ')'
                | 'empilha' '(' pem_vetor=IDENT ',' int_ou_ident ')'
                | 'enfileira' '(' pen_vetor=IDENT ',' int_ou_ident ')'
                //lacos e comandos condicionais
                | 'se' expressao 'entao' codigo senao_opcional 'fim_se'
                | 'para' para=IDENT 'em' vetores_para 'faca' codigo 'fim_para'
                | 'enquanto' expressao 'faca' codigo 'fim_enquanto'
                //chamada de atribuicao
                | atribuicao=IDENT '<-' expressao
                //funcao dijktra pre-implementada, retorna um grafo
                | 'dijkstra' '(' pd_grafo=IDENT ',' pd_vertice=IDENT ')'
                //funcao prim pre-implementada, retorna um grafo
                | 'prim' '(' pp_grafo=IDENT ',' pp_vertice=IDENT ')'
                //funcao dfs pre-implementada, retorna um grafo
                | 'dfs' '(' pdfs_grafo=IDENT ',' pdfs_vertice=IDENT ')'
                //funcao bfs pre-implementada, retorna um grafo
                | 'bfs' '(' pbfs_grafo=IDENT ',' pbfs_vertice=IDENT ')'
                | instrucoes_com_retorno
                | instrucoes_de_vetores
                ;
```

Gramática Final

```
int_ou_ident : INTEIRO | IDENT;

instrucoes_com_retorno :
                //retorna em um grafo o peso de uma aresta
                'get_peso' '(' pgp_grafo=IDENT ',' pgp_vertice1=IDENT ',' pgp_vertice2=IDENT ')'
                //retorna o custo para chegar em um vertice
                | 'get_custo_para_vertice' '(' pgc_grafo=IDENT ',' pgc_vertice=IDENT')'
                //retorna a quantidade de vertices em um grafo
                | 'qtde_vert' '(' pqv_grafo=IDENT ')' //grafo
                //retorna o vertice desempilhado
                | 'desempilha''(' pdem_vetor=IDENT ')' //vetor
                //retorna o vertice desenfileirado
                | 'desenfila' '(' pden_vetor=IDENT ')' //vetor
                //funcao que retorna o tamanhao de um vetor
                | 'tamanho_vetor' '('ptam_vetor=IDENT')' //vetor
                ;

instrucoes_de_vetores :
                //retorna um vetor de vizinhos a partir de um grafo e um vertice
                'vizinhos' '(' pv_grafo=IDENT ',' pv_vertice=IDENT ')'  //grafo,vertice
                //retorna um vetor de todos os vertices do grafo
                | 'vertices' '(' pver_grafo=IDENT ')' //grafo (retorna lista com vertices)
                ;
```

Gramática Final

```
vetores_para : instrucoes_de_vetores | IDENT ;

var_ou_cadeia : CADEIA | IDENT ;

senao_opcional : 'senao' codigo | ;

expressao : exp_aritmetica op_opcional;

exp_aritmetica : termo outros_termos;

outros_termos : (op_adicao termo)*;

termo : fator outros_fatores;

outros_fatores: (op_multiplicacao fator)* ;

fator : parcela outras_parcelas;

outras_parcelas : ('%' parcela)*;

parcela : IDENT | INTEIRO | instrucoes_com_retorno | instrucoes_de_vetores |'(' expressao ')';

op_opcional : op_relacional exp_aritmetica | ;

op_relacional : '=' | '<>' | '>=' | '<=' | '>' | '<';

op_multiplicacao : '*' | '/';
```

Gramática Final

```
op_adicao : '+' | '-';


/*Tokens: */
IDENT      : ('a'..'z' | 'A'..'Z' | '_') ('a'..'z' | 'A'..'Z' | '0'..'9' | '_')*;

INTEIRO    : ('+' | '-')? ('0'..'9')('0'..'9')* | 'INFINITO' | 'infinito';

CADEIA     : '"' ~('\n' | '\r' | '\'' | '\t')* '\'' | '"' ~('\n' | '\r' | '"' | '\t')* '"';

COMENTARIO : '{' ~('\n'|'}')* '}' {skip();};

WS         :    (' ' | '\t' | '\r' | '\n') {skip();};

/* Tokens errados: para tratarmos os mesmos foram setados tipos especificos(setType(int)) para cada um
 * assim para os tokens nao declarados foi setado o inteiro ERRO_TOKEN que possui por valor 10 e para os
 * os comentarios incompletos foi declarado ERRO_COMENT com o valor 11. Ambos os valores foram escolhidos
 * de forma parcial
 */
NAO_DECL   : ('@'| '|' | '!' | '"') {setType(ERRO_TOKEN);};

COMENTARIO_INCOMPLETO : '{' ~('\n'|'}')*'\n' {setType(ERRO_COMENT);};
```

Análise Semântica

Geração de Código

Teste - Sintático

- Variável não declarada:

Entrada

```
declaracoes:
    grafo g
    vertice v1, v2, v3
    int valor

inicio

    empilha(1, v1)

    aresta(g, v1, v2, 1)
    aresta(g, v2, v3, 1)
    aresta(g, v1, v3, 1)
fim
```

Saída

```
Linha 8: erro sintatico proximo a 1
Fim da compilacao
```

Teste - Semântico

- Repetição de declaração de variável;
- Variável e identificador não declarado;
- Custo de aresta e vértice não pode ser negativo;

Entrada

```
declaracoes:
    grafo g
    vertice v1, v2, v3,
        g

inicio

    valor <- valor + 1
    empilha(v4, 1)
    aresta(g, v1, v2, -1)
    aresta(g, v2, v3, 1)
    aresta(g, v1, v3, 1)

    set_custo_para_vertice(g, v1, -1)
fim
```

Saída

```
Linha 4: identificador g ja declarado anteriormente
Linha 8: identificador valor nao declarado
Linha 8: identificador valor nao declarado
Linha 9: identificador v4 nao declarado
Linha 10: peso nao pode ser negativo do identificador -1
Linha 14: peso nao pode ser negativo do identificador -1
Fim da compilacao
```

Teste – Geração de Código

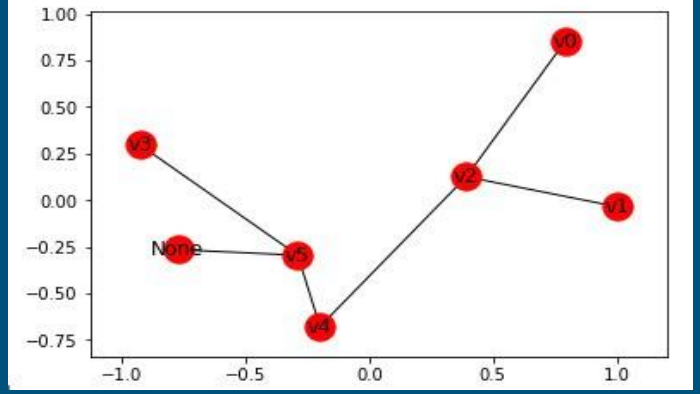
- Dijkstra

Entrada

```
declaracoes:
grafo g1

vertice v0
vertice v1
vertice v2
vertice v3
vertice v4
vertice v5

inicio
    aresta(g1, v0, v1, 1)
    aresta(g1, v0, v2, 1)
    aresta(g1, v0, v5, 6)
    aresta(g1, v1, v2, 1)
    aresta(g1, v2, v3, 4)
    aresta(g1, v2, v4, 1)
    aresta(g1, v3, v4, 2)
    aresta(g1, v4, v5, 1)
    aresta(g1, v3, v5, 1)

    dijkstra(g1,v5)

fim
```

Saída – arquivo .png

```
predecessor de  v0 e =  v2
predecessor de  v1 e =  v2
predecessor de  v2 e =  v4
predecessor de  v5 e =  None
predecessor de  v3 e =  v5
predecessor de  v4 e =  v5
```



Teste – Geração de Código

Saída – código em Python

```
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
gl = nx.Graph()
gl.add_edge('v0', 'v1', weight = 1)
gl.add_edge('v0', 'v2', weight = 1)
gl.add_edge('v0', 'v5', weight = 6)
gl.add_edge('v1', 'v2', weight = 1)
gl.add_edge('v2', 'v3', weight = 4)
gl.add_edge('v2', 'v4', weight = 1)
gl.add_edge('v3', 'v4', weight = 2)
gl.add_edge('v4', 'v5', weight = 1)
gl.add_edge('v3', 'v5', weight = 1)

CopiaGrcopy0 = nx.Graph()
CopiaGrcopy0 = gl

def Dijkstra(G, CopiaG, R1):
    for i in G.nodes():
        G.node[i]['peso'] = np.inf
    G.node[R1]['peso'] = 0
    Predecessor = {}

    for i in G.nodes():
        Predecessor[i] = None

    while G.number_of_nodes() > 0:
        pesoMinimo = np.inf
```

```
    for i in G.nodes():
        if G.node[i]['peso'] <= pesoMinimo:
            verticePesoMinimo = i
            pesoMinimo = G.node[i]['peso']

    for i in G.neighbors(verticePesoMinimo):
        if G.node[i]['peso']>G.get_edge_data(verticePesoMinimo, i)['weight']+G.node[verticePesoMinimo]['peso']:
            G.node[i]['peso'] =G.get_edge_data(verticePesoMinimo, i)['weight']+G.node[verticePesoMinimo]['peso']
            Predecessor[i] = verticePesoMinimo
    G.remove_node(verticePesoMinimo)

for key in Predecessor:
    print("predecessor de ", key, "e = ", Predecessor[key])

vertice = []
arestas = []

for key in Predecessor:
    vertice.append(key)

for key in Predecessor:
    arestas.append((key, Predecessor[key]))
```

```
vertice = []
arestas = []

for key in Predecessor:
    vertice.append(key)

for key in Predecessor:
    arestas.append((key, Predecessor[key]))

newGdij = nx.Graph()
newGdij.add_nodes_from(vertice)
newGdij.add_edges_from(arestas)
pos = nx.spring_layout(newGdij, k = 1, iterations=10)
nx.draw_networkx(newGdij, pos)
plt.show()


Dijkstra(gl, CopiaGrcopy0, 'v5')

#Fim da compilacao
```

Dificuldades

- Adicionar a implementação dos algoritmos na geração de código;
- Criação da gramática;