**Definir tokens**

PATH : ^(.+)/([^/]+) //para diretorios

VARIABLE : [a-z][a-zA-Z0-9]

FLOAT : [\-\+]?(\d\*[.])?\d””

GRAPH: “Graph”

INT\_TYPE: “Int”

FLOAT\_TYPE: “Float”

NODE\_TYPE: “Node”

EDGE\_TYPE: “Edge”

INTEGER : ^[-+]?\d+

STRING = “\”[] ?????

**2. Syntax Definition**

**2.1 Creation of Graphs**

**//Criação de um grafo**

Graph graph = new Graph(PATH);

**2.3 Consulta do grafo**

**//Consulta do grafo (usar gather no gremlin para retornar lista)**

**2.3.1 List of nodes and edges**

Node[] nodes = graph.nodes(); //todos os nós

Node[] nodes = graph.nodes().key(name=”ze”)

// Verificar como pode ser definida a key pelo user

Edge[] edges = graph.edges(). //todos as arestas

Edge[] edges = graph.edges.label(“knows”);// arestas que tenham a label “knows”

**2.3.2 Get a node**

Node n = graph.node(ID); ID = INTEGER

Node n = nodes[INTEGER]

**2.3.3 Get an edge**

Edge e = graph.edge(ID);

Edge e = edges[INTEGER];

**2.3.4 Node Properties**

Int i = n.id //identification of node

Edge[] edges = n.in; // list of incoming edges

Edge[] edges = n.out; //list of outcoming edges

Edge[] edges = n.(out|in).label(“knows”); //with label

n.key

Key key k;

k.name

k.value

---TODO--- Nós podem ter keys

**2.3.5 Edge Properties**

n = e.in; //devolve nó que entra

n = e.out; //devolve nó que chega

Float f = e.weight; //peso da aresta

Int i = e.id; //identification of edge

**//Search algorithms**

graph.shortestPath(node1, node2); //utiliza algoritmo shortest path

graph.shortestPath(node1, node2, [option1,option2,...]);

graph.DepthSearch(node1, node2);

graph.BreadthSearch(node1,node2);