背景：

Antlr将js代码转化为语法树并存在token流中。 树中的每个 节点node 都有其对应的interval

interval.a 即为该节点在token流中的起始位置 （左端点）

interval.b 即为该节点在token流中的终止位置 （右端点）

于是将token流对应在坐标轴并将其展开为树可理解为线段树

于是可以断定 父亲点的区间 覆盖 孩子节点的区间，且 父节点的len >孩子节点的len

【1，100】

100】【20，40】

【40，100】

【10，20】

【1，10】

【1，20】

【40，100】

【20，40】

同时，antlr的Visitor遍历机制为先根遍历，在Visitor 遍历的同时对节点进行编号.因其先根遍历机制，可以确定:

父亲节点的编号(node\_parent.id) < 孩子节点的编号(node\_son.id)

为读取节点信息，构建节点类

Node：

String type

Integer L （左端点）

Integer R （右端点）

Integer id

Integer len ///// len = R-L

并将所有节点存储于列表 list中，Node list[树的所有节点个数]

以邻接表的形式存储该树，树的节点类型为

Tree\_node:

Integer id

ArrayList<Integer> son\_list;

Tree\_node Tree\_node\_list[树的所有节点个数]

算法描述：

首先对list中节点排序

list.sort(cmp= list\_cmp)

def list\_cmp(node\_a,node\_b):

if node\_a 的len 与 node\_b 的len不等:

len小的放前面

else: 若相等

id大的放前面

接下来从头开始以 每个节点为起点(start\_node) 与其后面节点(temp\_node) 作比较。

当遇到一个区间可以覆盖起始节点的节点 （temp\_node.L <= start\_node.L and temp\_node.R >= start\_node.R）则 该节点为起始节点的父节点 ，将该 起始节点id 加入到父亲节点的son\_list 中。

**for i in range(len(list)):**

**for k in range(i+1,len(list)):**

**if list[i].L >= list[k].L and list[i].R <= list[k].R:**

**Tree\_list[list[ k ].id].node\_list.append(list[i].id)**

**break**