graphDatas :

{

"词1": {"词2": "词1和词2的语义相似度", "词3": "词1和词3的语义相似度",...},

"词2": {"词1": "词2和词1的语义相似度", "词3": "词2和词3的语义相似度",...}

}

Dijkstra( graphDatas, key)：完成了任意一个key到图中任意一点最短距离的计算

输入： graphDatas , 词1

unprocessed = (词2，….词n)

shortest\_distances = {词1:0, 词n:最短路径数值}

path = {startNode:词1}

temPath =

返回shortestData = 键：词1一直到词n

值：{‘path’: 输入的词到任意一个词的路径

‘distance’：距离对应的距离

}

shortestDatas = 键：词1 其值是一个字典2。字典2的键是词2到词n,字典的每个键对应的值又是一个字典3，字典3的有两个键：path 和 distance。

值：{{词2：{‘path’:词1—>词2}，‘diatance’:数值}，{词3：{‘path’:词1—>词3}，‘distance’:数值}}}

intermediaryDegreeScore(word, shortestDatas)

输入：词1 shortestDatas

path= shortestDatas[m][k]['path'] 节点m 到节点k的最短路径

routes = path.split(‘->’) path切分之后取出路径上的每个节点，然后求节点的个数 routesNum = len(routes)

得分score = 1/ (routesNum – 1)

遍历所有m,k得到总分Score。

输出：居间度对应的数值。

interval字典 单词：居间度。

intermediaryDegreeDensity(fileName,path)

interval : 居间度字典

wordCount = len(interval)

计算居间度密度,开始时候对interval划分的区间个数：s=10。区间个数增长速度c = 5; 区间密度阈值：d = 0.8；对bc重新划分的最大次数 max = 6。

对interval 字典降序排列————sortedInterval

refinementBC( sortedInterval , s)

输入：降序排列的居间度字典，区间划分个数s

maxbc = sortedInterval[0][1] 最大居间度

minbc = sortedInterval[总数-1][1] 最小居间度

intervalScore = (maxbc-minbc)/s 居间度划分区间长度

intervalDensity 字典 保存区间 以及每个区间里面key。即可以求出每个划分区间的居间度密度。

maxratio = wordNum (对value.split(‘,’)划分之后求len())/wordCount。

返回 最大的maxratio, intervaDenstity 获取当前居间度密度最大值。

for 循环 loop=1 到 max :

s = s\*c

maxratio, intervalDensity = refinementBC(sortedInterval, s)

loop+=1

最后在居间度密度字典迭代完成之后，求每个单词对应的居间度密度：

for word in wordlist:  
intermediaryDensity[word] = wordNum / wordCount

最后返回intermediaryDensity 字典 ： word :居间度密度。