Entrega de programación distribuida

Cano Jon, González Jorge, Lattes Adrián, Lobato Pablo

21/05/2021

Índice

Listings

1 Grafo multifichero:

En este ejercicio analizamos un grafo formado por multiples archivos y devolvemos todos sus triciclos.

1.1 Unión de rdds:

Primero unimos los rdd:

unión de rdd

```
62 def mixed(sc, rdds):
63     for rdd in rdds:
64         rdd.cache()
65     data = sc.union(rdds)
```

1.2 Triciclos

Dado el rdd con los nodos y sus listas de adyacencia considerando solo nodos posteriores definimos la función tag que genera la lista de 'exist' y 'pending' para un nodo:

tag

```
32 def tag(node_adjs): # Función iterativa, no perezosa
33     node = node_adjs[0]
34     adjs = list(node_adjs[1])
35     adjs.sort()
36     result = [((node, x), 'exists') for x in adjs]
37     for i in range(len(adjs)):
38         for j in range(i, len(adjs)):
39             result.append(((adjs[i], adjs[j]), ('pending', node)))
40     return result
```

Generamos las listas de adyacencia y generamos los triciclos:

tricycles

```
43 def tricycles(tags):
44     return tags\
45             .groupByKey()\
46             .filter(lambda x: len(x[1]) > 1 and 'exists' in x[1])\
47             .flatMap(
48                 lambda line: map(
49                     lambda x: (x[1], line[0][0], line[0][1]),
50                     filter(lambda x: not x == 'exists', line[1])
51                     )
52                 )
```

process data

```
55 def process_data(data):
56     edges = get_distict_edges(data)
57     node_adjs = get_node_adjs(edges)
58     tags = node_adjs.flatMap(tag)
59     return tricycles(tags)
```

2 Multiples grafos:

En este ejercicio cada fichero representa un grafo distinto.

2.1 Aristas:

Para analizar cada archivo como un grafo a parte cada arista contiene también el nombre del archivo al que pertenece:

Tuplemorph

```
20 tupleMorph = lambda tup, x : (tup[0], (tup[1], x))
```

2.2 Triciclos

Ahora realizamos un procedimiento análogo al del archivo anterior siempre teniendo en cuenta el archivo al que pertenece cada arista para que los triciclos esten compuestos por tres nodos del mismo grafo.

tag

```
34 def tag(node_adjs): # Función iterativa, no perezosa
35     node = node_adjs[0]
36     adjs = list(node_adjs[1])
37     adjs.sort()
38     result = [((node, x), (graph, 'exists')) for (x, graph) in adjs]
39     for i in range(len(adjs)):
40         for j in range(i, len(adjs)):
41             if adjs[j][1] == adjs[i][1] and not adjs[i][0] == adjs[j][0]: #Los nodos aparecen en el mismo grafo
42                 result.append(((adjs[i][0], adjs[j][0]), (adjs[i][1], ('pending', node))))
43     return result
```

tricycles

```
66 def tricycles(tags):
67     return tags\
68             .groupByKey()\
69             .filter(lambda x: len(x[1]) > 1 and findExists(x[1]))\
70             .map(construct_tricycles)
```

process data

```
77 def process_data(data):
78     edges = get_distict_edges(data).cache()
79     node_adjs = get_node_adjs(edges).cache()
80     tags = node_adjs.flatMap(tag).cache()
81     return tricycles(tags)
```