# TriangleCountingProblem_Assignment3

April 10, 2017

## 1 Triangle Counting Problem | BDA Assignment #2

### 1.1 Developer: Anshul Khantwal, MT16010, IIIT Delhi

### 1.2 NodeIterator Algorithm

```
In [11]: from pyspark.sql import SQLContext
         edges = sc.parallelize([(1,2),(1,3),(2,3),(3,4),(3,5),(4,5)])
         vectices = sc.parallelize([(1,),(2,),(3,),(4,),(5,),(6,)])
```

The implementation of the algorithm is as follows:

```
Round 1: Generate the possible length two paths in the graph by pivoting on every
node in parallel.
Round 2: Check which of the length two paths generated in Round 1 can be closed by
an edge in the graph and count the triangles accordingly.
```

```
In [12]: def findTriangles(edges):

             def preprocessEdges(x):
                 if x[0]<x[1]:
                     return (x[0],x[1])
                 else:
                     return (x[1],x[0])

             edges = edges.map(preprocessEdges)


             def mapper1(x):
                 if x[0]<x[1]:
                     return (x[0],[x[1]])
                 else:
                     return (x[1],[x[0]])
             output_map1 = (edges.map(mapper1).filter(lambda x:x!=None)
                 .reduceByKey(lambda x,y:x+y))


             def reducer1(x):
```

```
                output = []
                for a in range(0,len(x[1])):
                    for b in range(a+1,len(x[1])):
                        output.append(((x[1][a],x[1][b]),[x[0]]))
                return output
            output_reducer1 = output_map1.flatMap(reducer1)
            output_reducer2 = edges.map(lambda x: ((x[0],x[1]),["*"]))
            output_reducer2 = output_reducer2.union(output_reducer1)
            output = output_reducer2.reduceByKey(lambda x,y:x+y).collect()

            def generateTriplets(x):
                output = []
                for tupples in x:
                    vertex_list = tupples[1]
                    if "*" in vertex_list and len(vertex_list)!=1:
                        vertex_list = set(vertex_list)-{"*"}
                        for vertex in vertex_list:
                            output.append((tupples[0][0],tupples[0][1],vertex))

                return output

            return len(generateTriplets(output))
```

```
In [13]: def driverNodeIteratorAlgorithm(edges):
             return findTriangles(edges)
```

```
In [14]: #Driver program to count the triangles
         output1 = driverNodeIteratorAlgorithm(edges)
         print("No. of Triangles:\t",output1)
```

```
No. of Triangles:        2
```

### 1.3 Partition Algorithm

The algorithm works by partitioning the graphs into overlapping subsets so that each triangle is present in at least one of the subsets. Given such a partition, we can then use any sequential triangle counting algorithm as a black box on each partition, and then simply combine the results.

```
In [15]: def driverPartitionAlgorithm(edges):
             p = 4

             def mapper(x):
                 i = int(x[0]) % p
                 j = int(x[1]) % p

                 output = []

                 for a in range(0,p):
```

```
            for b in range(a+1,p):
                for c in range(b+1,p):
                    if {i,j}.issubset({a,b,c}):
                        (output.append((str(a)+" "+str(b)+" "+str(c),
                                        [(x[0],x[1])])))
    return output
mapper_output = (edges.flatMap(lambda x:mapper(x))
    .reduceByKey(lambda x,y: x+y))

def reducer(edge_list):
    no_triangles = 0


    def findTriangles(edges):
        import networkx as nx
        G=nx.Graph()
        for x in edges:
            G.add_edge(x[0],x[1])
        result=[]
        done=set()    #
        for n in G:
            done.add(n)    #
            nbrdone=set()    #
            nbrs=set(G[n])
            for nbr in nbrs:
                if nbr in done:    #
                    continue    #
                nbrdone.add(nbr)    #
                for both in nbrs.intersection(G[nbr]):
                    if both in done or both in nbrdone:    #
                        continue    #
                    result.append( (n,nbr,both) )
        return result

    triangles = findTriangles(edge_list)


    def weightedCount(x):
        u = int(x[0]) % p
        v = int(x[1]) % p
        w = int(x[2]) % p

        z = 1
        if u==v and v==w:
            z = (u*(u-1)/2) + u*(p-u-1) + ((p-u-1)*(p-u-2)/2)
        elif u==v or v==w or u==w:
            z = p-2
        z = 1/z
```

```
                    #return (str(x[0])+" "+str(x[1])+" "+str(x[2]),z)
                    return z

                for tri in triangles:
                    no_triangles +=  weightedCount(tri)
                return ("*",no_triangles)

            reducer_output = mapper_output.map(lambda x:reducer(x[1]))
            return reducer_output.values().sum()
```

```
In [16]: #Driver program to count the triangles
         output2 = driverPartitionAlgorithm(edges)
         print("No. of Triangles:\t",output2)
```

```
No. of Triangles:         2.0
```

### 1.4 Comparing results of both the algorithms

```
In [17]: if output1 == output2:
             print("Both had converged to same number of Triangles.")
```

```
Both had converged to same number of Triangles.
```

```
In [19]: import time
         currentMilliTime = lambda: int(round(time.time() * 1000))

         t1 = currentMilliTime()
         driverNodeIteratorAlgorithm(edges)
         t2 = currentMilliTime()
         time1 = t2-t1

         t1 = currentMilliTime()
         driverPartitionAlgorithm(edges)
         t2 = currentMilliTime()
         time2 = t2-t1
```

```
In [20]: print("NodeIterator Algorithm's Execution Time: \t {0} milliseconds.".form
         print("Partition Algorithm's Execution Time: \t\t {0} milliseconds.".forma
```

```
NodeIterator Algorithm's Execution Time:          12359 milliseconds.
Partition Algorithm's Execution Time:                 4304 milliseconds.
```