GROUP NUMBER: 16

GROUP MEMBERS: Matteo Manzini, Jacopo Onorati

AVAILABLE INPUTS: Input files are available in the hdfs file system: /data/BDC2425/artificial1M7D100K.txt and /data/BDC2425/artificial4M7D100K.txt

PART 1: The goal of this test is to assess the scalability of the standard and fair implementations. The test must be performed on file artificial4M7D100K.txt. However, if your implementation is slow (i.e., taking more than 10 minutes for the slowest run), you can use the smaller file artificial1M7D100K.txt. You must use the following parameters: L=16, K=100, M=10.

Name of used file: /data/BDC2425/artificial1M7D100K.txt

SCALABILITY WITH RESPECT TO NUMBER OF EXECUTORS			
Number of executors	**Spark Lloyd's implementation**	**MRFairLloyd**	**MRComputeFairObjective**
2	11838 ms	604852 ms	51899 ms
4	6728 ms	318888 ms	26744 ms
8	5177 ms	155414 ms	13682 ms
16	3364 ms	85151 ms	7008 ms

General hints:

- Remember that Spark uses the lazy evaluation for constructing an RDD. Therefore, be sure to include an action on the final RDD when you take running times.
- Any used RDD in your program should be cached.
- Do not include the reading of the input in your running times.

PART 2: Describe the program GxxGEN.java or GxxGEN.py that you have implemented for point 5 of the specifications. Include a brief high-level description of your program and the constraints, if any, on the input parameters (e.g., the minimum number of points *N*).

The generator we implemented (file G16GEN.py) spawns points composed of two coordinates and a label 'A' or 'B' representing the demographic group the point belongs. First, the method generate_circles() creates groups of only B-labeled points. Each group is randomly placed on the plane and the points are enclosed in an ideal circle (circle() method). The points are distribuited randomly and uniformly in the circle.
For each circle of B points generate_circles() method calls genA() method in order to spawn randomly and uniformly A points enclosed in an ideal circle smaller than first one and enclosed in it. A points will be less than half of B points (see "*CONSTRAINTS*" section). The sum of each circle of Bs and the corresponding circle of As makes a cluster of points. The entire set of points is divided almost equally among the different clusters (when the division of N by K is not perfect some clusters will have a point more in order to distribuite the remainder).
The idea of making such clusters is to highlight the differences between the Standard K-Means clustering and Fair K-means performed on the same dataset.
The circles are, in most cases, well-separated, therefore Standard K-Means is likely to return clusters which are very similar to the original circles (including both A and B points) that make up our dataset. As a result, MRComputeFairObjective() tends to return a poor score, since A and B points are likely to be unbalanced within each cluster.
In contrast, Fair K-Means will "break" the circles in an attempt to return clusters for which the two demographic groups are balanced causing the MRComputeFairObjective() to give us a better value.

CONSTRAINTS

- Even division of N by K must be > 4 because each cluster must have at least 2 A points, therefore the number of B points must be more than 5 by construction.
- The number of A points generated for each circle must be a random integer between 1 and max_A where max_A = (points per cluster)*0.4