Local space analysis MRPrintStatistics function

The **MRPrintStatistics** function takes in input the set *U,* which is the set of the points, and *C*, which is the set of the centroids. We aimed at printing the triplets (c_i, NA_i, NB_i) for $1 \leq i \leq k$, where c_i is the i-th centroid in *C*, and NA_i and NB_i are the numbers of points of *A* and *B*, respectively, in the cluster *i* centered in c_i.

In the **MRPrintStatistics** function we implemented a one-round MapReduce algorithm, exploiting the RDD that we created earlier in the main function and that we split into *L* partitions.

Map Phase

In the Map Phase, we mapped each point of the RDD through the function *min_cluster (point, C)* which takes in input the set of centroids C and a tuple which contains the pair (p, g_p), where $p \in \mathbb{R}^D$ is a point, and $g_p \in \{A, B\}$ is a character representing its demographic group.

This function computes the **square distances** from the point and each centroid given in input, then computes the **index of the cluster the point belongs to** (the one with the minimum distance from the centroid) and returns a tuple containing the index we just computed and the demographic group the point belongs to (A or B) e.g. *(1, A).*

Local space analysis (Map Phase)

The local memory of the map phase is $O(D * K)$ where D is the dimension of each point and K the number of centroids, since the function calculates the minimum distance between each point and the K centroids.

Shuffle + grouping phase

We grouped the output of the map phase by the cluster indexes, obtaining a tuple which contains the index of the cluster and an iterable object with the labels of the demographic groups. e.g. *(1, [A, B, A, A, B]).*

Reduce phase

For every cluster, we counted the number of occurrences of the characters A and B returning them as a dictionary. e.g. *(1, {'A': 22, 'B': 57'}).*

Local space analysis (Reduce Phase)

Since we partitioned the data into *L* partitions, in the worst-case scenario (which is when all the points in the selected partition belong to the same cluster) the maximum amount of main memory required by the invocation of this function requires $O\left(\frac{N}{L}\right)$ space where *N* = |*U*|.

Local space of the function

The overall complexity of this MapReduce algorithm is $O\left(D * K + \frac{N}{L}\right) = O\left(\frac{N}{L}\right)$ which is sublinear for L > 1. The quantity D*K is a constant since it does not depend on the input size, so it can be neglected.

Therefore, the complexity meets the design goals for a MapReduce algorithm.