Exercise: Analysis of a File Containing City Population Data

Tasks

Perform the following tasks:

1. **Read the file "cities.csv" with SCALA**:
   - The file can be read with the command `sc.textFile("cities.csv")`, where `sc` is the Spark context.
   - Use the command `sc.textFile("cities.csv").take(1001)` to load the first 1001 lines and inspect the data.
2. **Count the lines of the result**:
   - Count the number of lines in the file to verify that the data has been loaded correctly. This can be done using the `.count()` function.
3. **Filter the file to remove the header**:
   - Filter the file to remove the headline (the first row), which contains the column names.
   - Hint: use the `filter` function and `!line.startsWith("City")` to exclude the header line.
4. **Map the file to split each line into fields**:
   - Use a `map` function to split each line into its respective fields (i.e., columns).
   - Each field corresponds to the columns "City", "Country", and "Population".
   - For this task, you can use `line.split(",")` to separate the fields by commas.
5. **Retrieve the population column**:
   - Use a `map` function to retrieve the field corresponding to "Population".

- Convert the population field, which is a string, to an integer using `.toInt`.
- The population field is the third column (index `2`).

6. **Filter the collection of population values**:
   - Filter the population collection to retain only records where the population is greater than 0.
   - The final collection will be called `collectPopulations`.
7. **Use a MapReduce function to compute the total number of cities**:
   - Code a MapReduce function to compute the total number of cities in `collectPopulations`.
   - The result will be stored in the variable `countCities`.
8. **Use a reduce function to compute the total sum of city populations**:
   - Code a reduce function to compute the total population of all cities.
   - The result will be stored in the variable `totalPopulation`.
9. **Use a reduce function to compute the maximum population of a city**:
   - Code a reduce function to compute the maximum population of a city.
   - The result will be stored in the variable `maxPopulation`.
   - What can you conclude from this result?
10. **Compute and print on screen**:
    - Compute and display the following results:
      - Total population
      - Number of cities
      - Maximum city population
      - Average city population

Expected Scala Code

Here's a general idea of how your code could look. Adjust it according to the exact structure of the `cities.csv` file.

```
// Step 1: Read the file
val file = sc.textFile("cities.csv")

// Step 2: Count the number of lines
val lineCount = file.count()
println(s"Number of lines: $lineCount")

// Step 3: Filter to remove the header
val dataWithoutHeader = file.filter(line => !line.startsWith("City"))

// Step 4: Split each line into fields (City, Country, Population)
val columns = dataWithoutHeader.map(line => line.split(","))

// Step 5: Retrieve the Population field (third column) and convert it to an integer
val populations = columns.map(fields => fields(2).toInt)

// Step 6: Filter to keep only populations greater than 0
val collectPopulations = populations.filter(pop => pop > 0)

// Step 7: Compute the total number of cities
val countCities = collectPopulations.count()

// Step 8: Compute the total population
val totalPopulation = collectPopulations.reduce(_ + _)

// Step 9: Compute the maximum population of a city
val maxPopulation = collectPopulations.reduce((a, b) => Math.max(a, b))

// Step 10: Compute the average population
val averagePopulation = totalPopulation / countCities

// Print the results
println(s"""
  | Total number of cities: $countCities
  | Total population: $totalPopulation
  | Maximum city population: $maxPopulation
  | Average city population: $averagePopulation
""".stripMargin)
```

Dataset Structure

The dataset (`cities.csv`) is assumed to have the following columns:

```
City, Country, Population
New York, USA, 8419600
```

```
Los Angeles, USA, 3980400
Paris, France, 2148000
...
```

This exercise is a good parallel to the `les-arbres.csv` problem but works with city population data, and it introduces similar MapReduce operations to compute counts, sums, maximums, and averages.