DSCI 551 – HW4

(Spark Dataframe, Indexing, and Query Execution)

(Fall 2020)

100 points, Due 11/11, Wednesday

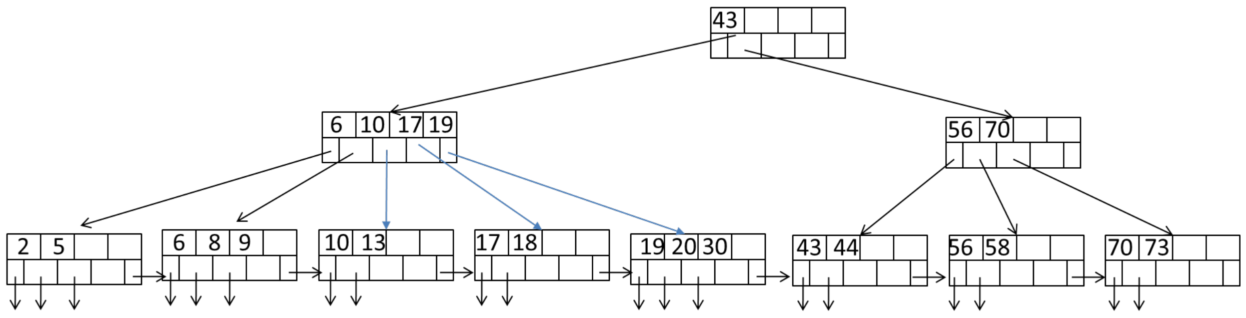
1. [30 points] Consider the same JSON files for the world database that you used in your lab 3 task. Write Spark Python code to implement the same set of queries. Name your code q1.py, q2.py, …, q5.py. For each query, **include a screenshot showing the query results in your report**. You can assume that the 3 JSON files, e.g., country.json, are stored in the same folder where your code will be executed.

   Execution format: spark-submit q1.py
   spark-submit q2.py
   spark-submit q3.py
   …

   1. Find out which countries in Europe have a GNP between 100,000 and 500,000 (inclusive). Output **name** of countries only.
   2. Find names of countries and their capital cities for all countries in "North America". Sort by country name (ascending) and return top 10 results.
   3. Find names of counties and their official languages for all countries in "North America". Sort the results by country name (ascending) and return top 10.
   4. For each continent, compute the average life expectancy of all countries whose GNP > 10000 in the continent.
   5. Find out for each continent, the number of countries where French is an official language.

Please include all your work in question 2 and 3 in your report.

2. [30 points] Consider the following B+tree for the search key "age". Suppose the degree d of the tree = 2, that is, each node (except for root) must have at least two keys and at most 4 keys.



a. Describe the process of finding keys for the query condition “age >= 10 and age <= 60”. How many blocks I/O’s are needed for the process? **<u>Please show your steps and describe in detail.</u>**
b. Draw the B+tree after inserting 74, 75, 76 into the tree. Only need to show the final tree after 3 insertions.
c. Draw the tree after deleting 58 from the original tree.

For question b and c, you can use online tool (‘https://projects.calebevans.me/b-sketcher/’) and select Tree Type: B+tree to draw your results, or manually draw the tree and attach the pictures of your results.

3. [40 points] Consider natural-joining tables R(a, b) and S(a,c). Suppose we have the following scenario.
   i. R is a clustered relation with 500 blocks and 10,000 tuples
   ii. S is a clustered relation with 1000 blocks and 20,000 tuples
   iii. S has a clustered index on the join attribute a
   iv. V(S, a) = 20 (recall that V(S, a) is the number of distinct values of a in S
   v. 102 pages available in main memory for the join.
   vi. Assume the output of join is given to the next operator in the query execution plan (instead of writing to the disk) and thus the cost of writing the output is ignored.

   Describe the steps (including input, output, and their sizes at each step) for each of the following join algorithms. What is the total number of block I/O’s needed for each algorithm? Which algorithm is most efficient?

   a. Nested-loop join with R as the outer relation
   b. Sort-merge join (assume only 100 pages used for sorting and 101 pages for merging). When the number of runs of a relation is too large for merging, the runs will be further merged first. Select the relation with a larger number of runs for further merging if both have too many runs.
   c. Partitioned-hash join (assume 101 pages used in partitioning of relations and no hash table used for lookup in joining buckets)
   d. Index join (ignore the cost of index lookup)

<u>Submission:</u>

Please use **<u>python 3</u>** and test your scripts in question 1 on **<u>ec2</u>** to make sure they can work.

Please submit a zip file named Firstname_Lastname_hw4.zip including the following files:

- A PDF file(Firstname_Lastname_hw4_report.pdf) document showing:
  - screenshots of results for all 5 sub questions in question 1 (only results, no need to attach your cocd)
  - results of question 2

    - results of question 3
- A folder(Firstname_Lastname_hw4_1) containing your Spark Python codes for 5 sub questions in question 1: q1.py, q2.py,…, q5.py