Topics: Hadoop MapReduce, Collaborative Editing, and CRDT

This assignment consists of five problems.

Submission Deadlines

- **Problems 1 to 3**: Submit by **February 28th**.
- **Problems 4 and 5**: Submit by **March 24th**.

General Guidelines

- Submit a well-structured, concise report ensuring clarity and readability.
- Provide clear execution instructions for the code.
- Late submissions are subject to penalties as per LMS guidelines.
- Plagiarized or inauthentic work will receive no marks.
- Upload a single zip file named after your roll number, containing:
  - Source code (excluding datasets).
  - Report with relevant screenshots.
  - Runtime analysis and system configuration details for all problems.
  - Execution instructions for each problem in separate files: `problem_X.txt`.
- Only one group member should submit the assignment.
- Groups may be required to demonstrate their solutions to a TA. All members must participate, and individual assessments may be conducted.

Problem 1. (8 marks)
A **Conflict-Free Replicated Data Type (CRDT)** is a state-based object that satisfies the following properties:

- The **merge** method is **associative**: For any three state-based objects x, y, z, the following holds:
$$\text{merge}(\text{merge}(x, y), z) = \text{merge}(x, \text{merge}(y, z)).$$
- The **merge** method is **commutative**: For any two state-based objects x and y, we have:
$$\text{merge}(x, y) = \text{merge}(y, x).$$
- The **merge** method is **idempotent**: For any state-based object x,
$$\text{merge}(x, x) = x.$$
- Every **update** method is **increasing**: Let x be an arbitrary state-based object, and let $y = \text{update}(x, ...)$ be the result of applying an arbitrary update to x. The update method is increasing if:
$$\text{merge}(x, y) = y.$$

Consider the following **M-Counter**, a state-based object that represents a counter which can be incremented but not decremented.

M-Counter Description

- The internal state of an **M-Counter** replicated on n machines is an n-length array of non-negative integers.
- The **query** method returns the sum of all elements in the n-length array.
- The **add(x)** update method, when invoked on the i-th server, increments the i-th entry of the array by x. For example:
  - Server 0 increments the 0th entry of the array.
  - Server 1 increments the 1st entry of the array, and so on.
- The **merge** method performs a pairwise maximum of the two arrays.

The following Python implementation of an **M-Counter** demonstrates these properties:

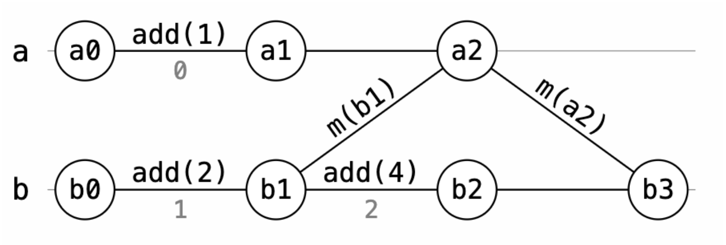
```python
class GCounter(object):
  def __init__(self, i, n):
    self.i = i # server id
    self.n = n # number of servers
    self.xs = [0] * n

  def query(self):
    return sum(self.xs)

  def add(self, x):
    assert x >= 0
    self.xs[self.i] += x

  def merge(self, c):
    zipped = zip(self.xs, c.xs)
    self.xs = [max(x, y) for (x, y) in zipped]
```

1. Do you think the **M-Counter** state-based object (SBO) qualifies as a CRDT? Justify your answer by considering the CRDT properties. (3 marks)
2. Consider the following state diagram and populate the state table with the fields: *state*, *query*, and *history*. (Refer to the lecture slides on LMS for an explanation of the notations used in the diagram.) (3 marks)
3. Should CRDTs guarantee **formal correctness** (e.g., convergence and consistency) or ensure that their behavior aligns with intuitive application semantics? Discuss the trade-offs between correctness and application semantics. (2 marks)



Problem 2. (10 marks)

In this problem, you are required to develop a MapReduce program to process Wikipedia articles and simulate a collaborative editing scenario.

The program should read each article from the Wikipedia dump available on the LMS: `Wikipedia-EN-20120601_ARTICLES.tar.gz`. For each word in the article, the mapper should emit a key-value pair, where the key represents the document ID, and the value is a pair containing the index position of the word in the document and the word itself.

The reducer will group the incoming key-value pairs based on the document ID and emit key-value pairs in the form:

```
(index, (doc-ID, word))
```

You should consider using at least three reducers in your deployment.

If you observe the output of the reducer, it simulates a collaborative editing environment. If each document is thought of as a different version or snapshot of the same document, with the document ID acting as the snapshot ID, then each key-value pair emitted by the reducer would indicate the index position and the respective changes that occurred at that index across different timestamps.

To develop the code, you may use the smaller dataset `Wikipedia-50-ARTICLES.tar` shared on the LMS. Be mindful that processing the full Wikipedia dump will require significant time; plan accordingly to ensure efficient execution of the MapReduce job.

Problem 3. (10 marks)

In this problem, you will write another MapReduce program that processes the output file generated by your previous program.

Write a mapper that reads from the file containing the index, document ID, and word, and emits key-value pairs of the form:

```
(index, (timestamp, word))
```

Here, `index` serves as the key, while the value is a pair containing the document ID, timestamp, and word. The timestamp should be derived from the document ID and converted into the standard Java timestamp format.

Write a reducer that groups the key-value pairs based on the index and emits the word associated with the maximum timestamp. Compare the output of the reducer with the actual Wikipedia article that has the highest document ID (i.e., the most recent version of the file).

If the generated file does not match the actual Wikipedia article, identify the reasons for the discrepancies. Implement another MapReduce function with the necessary modifications to address these issues. Finally, discuss how the modified MapReduce job produced the correct output.

Note: For Problem-3, submit both the MapReduce programs and their respective output files. Additionally, you may write a simple program in your preferred programming language to compare the output with the Wikipedia article that has the maximum timestamp value in the filename—submit that program as well.

Problem 4. Co-occurring word matrix generation: Pairs & Stripes and local aggregation **14 Points**

a Develop a MapReduce program aimed at identifying the top 50 most frequently occurring words from the provided Wikipedia dump located on LMS (`Wikipedia-EN-20120601_ARTICLES.tar.gz`). Utilize the *pairs* approach for this task. Exclude non-stop words while computing the frequency of occurrence. Utilize the distributed caching option within Hadoop, demonstrated in `WordCount2.java`, to eliminate all stop-words in the mapper. (3 marks)

b Write another MapReduce program, employing the pairs approach, to construct the *co-occurring word matrix* based solely on the frequent words identified in the previous question. Consider a word distance of d when determining co-occurring pairs. Report the runtime for $d = \{1, 2, 3, 4\}$. (3 marks)

c Develop a *stripe* algorithm to create the co-occurring word matrix. Again, report the runtime for $d = \{1, 2, 3, 4\}$. (3 marks)

e Additionally, implement local aggregations separately at the Map-class level and the Map-function level, and compare their performance. Conduct this comparison for parts (b) and (c). Once again, the runtime must be reported for $d = \{1, 2, 3, 4\}$. (5 marks)

Note that these programs are expected to require a considerable amount of time to execute on your local system. It is advisable to plan your experiments in advance. Ensure that adequate screenshots are provided in the report to demonstrate execution time, among other relevant metrics. Mere submission of code and a few sample runtimes does not guarantee full marks.

Problem 5. Indexing Documents via Hadoop **12 Points**

a Implement a pair of a `Map` and a `Reduce` function which, for each distinct term that occurs in any of the text documents in `Wikipedia-EN-20120601_ARTICLES.tar.gz`, counts the number of distinct documents in which the term appears. We will call this value the Document Frequency (DF) of that term in the entire set of Wikipedia articles. Store the resulting DF values of all terms in a single TSV file with the following schema:

```
TERM<tab>DF
```

While generating the output in the above format, consider filtering out all terms that belong to the stopwords.txt file shared on LMS. (You may perform this filter operation in your map method.) Identify the top 100 terms with a high document frequency. Use those terms alone for the next sub problem.

Hint: Also here consider the Porter stemmer that is available from `opennlp-tools-1.9.3.jar` *on LMS. After importing the library with* `import opennlp.tools.stemmer.PorterStemmer;` *and creating an instance of* `PorterStemmer stemmer = new PorterStemmer();` *use* `stemmer.stem(token);` *for stemming an input token.* **6 Points**

b Implement another pair of a `Map` and a `Reduce` function which, for each document in `WikipediaEN-20120601_ARTICLES.tar.gz`, first counts the number of occurrences of each distinct term within the given document. We will call this value the Term Frequency (TF) of that term in the given document. Implement a stripes algorithm here as we are dealing with just 100 terms.

In a second step, your combined MapReduce function should multiply the TF value of each such term with the inverse of the logarithm of the normalized DF value calculated by the previous MapReduce function, i.e., SCORE = TF$\times$log(10000/DF+1) for each combination of a term and a document. You may cache the former TSV file with the DF values by adding it via `Job.addCacheFile(<path-to-DF-file>)` in the driver function of your MapReduce program. The `Map` class should then load this file upon initialization into an appropriate main-memory data structure.

The result of this MapReduce function should be a single TSV file that has the same schema as the file we used in the previous exercise sheets: **6 Points**

```
ID<tab>TERM<tab>SCORE
```

Hint: You may use either the given document URL's or simple integer id's for the `ID` *field of this TSV file (as long as they uniquely identify the original text article).*