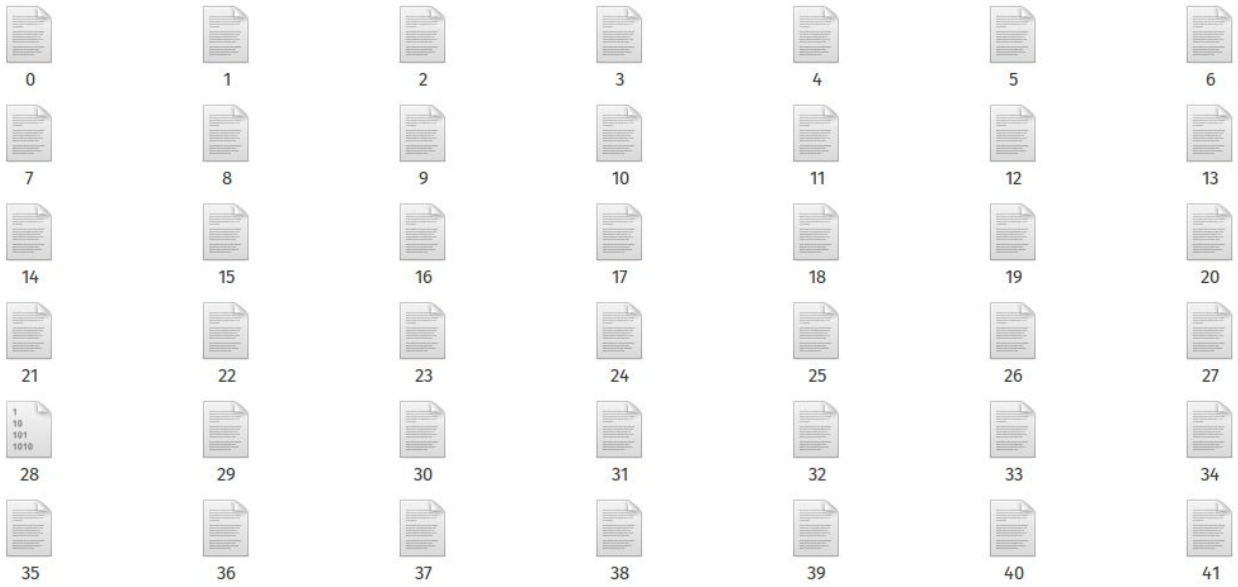
Inverted Full-Text Index

Create an application that reads a collection of Documents, and produces an inverted index that gives, for every word, the list of documents it appears in.

Details

We have a collection of N documents. In our case, we have one file per document and the name of a file is simply the indices of the document as shown on the figure below -



We want a dictionary that matches every word from the documents with a unique id. See the figure below for a sample.

```
Project         0
This            1
Gutenberg's     2
is              3
of              4
Copyright       5
Welcome         6
See             7
most            8
Shakespeare's   9
...
```

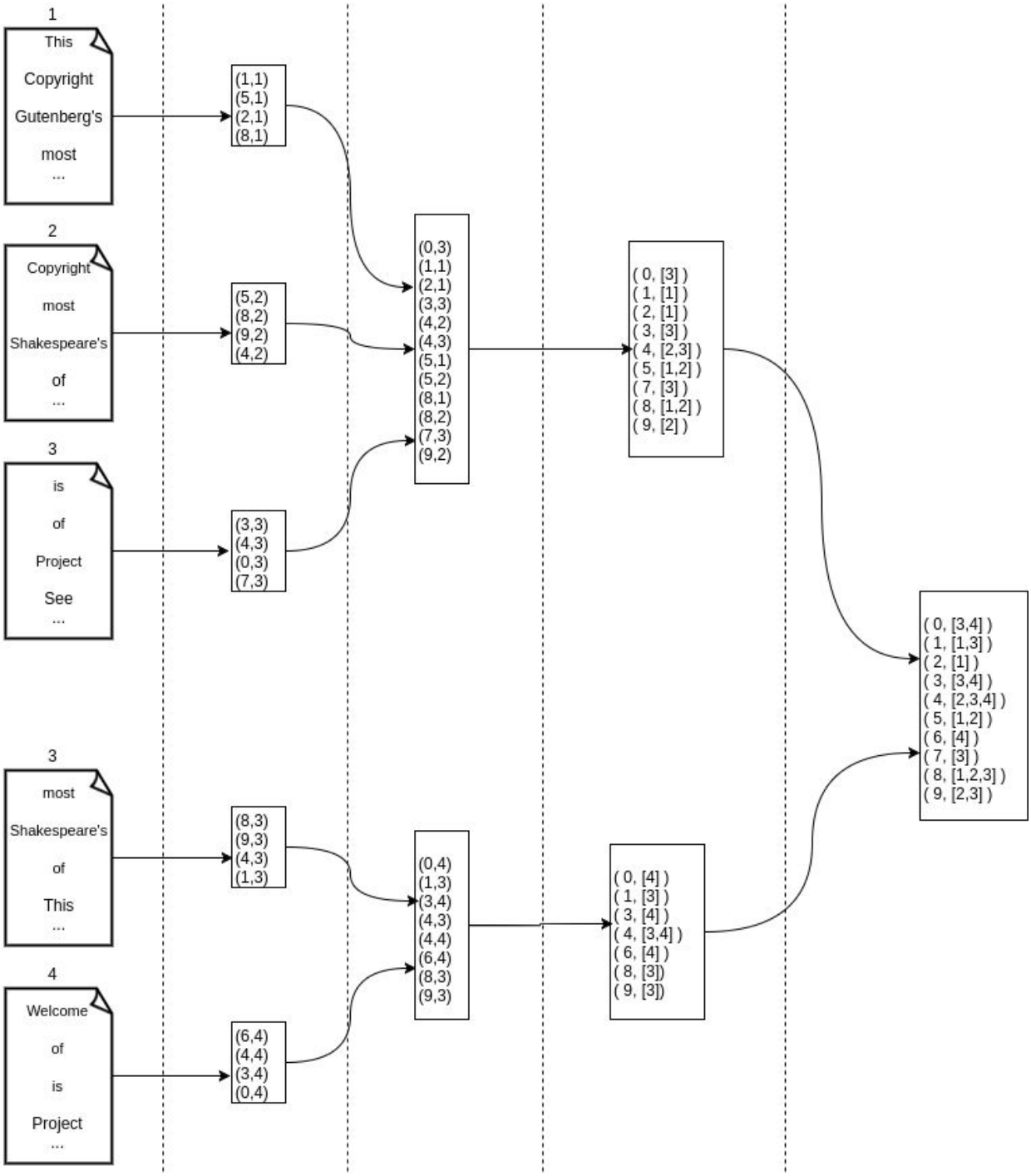
Using both the dataset and the dictionary we can build an inverted index that gives, for every word, the list of documents it appears in. See the figure below -

```
(0, [2,5,13,24,30])

(1, [1,2,4,23,44])

(3, [5,9,24,44])

(4, [2,3])

(5, [3,6,9,10,33])

(6, [5,44])

(7, [30,40])

(8, [1,4,7,35])

(9, [16,22])

(10, [13,16,17,28,34])

(11, [1,9])
...
```

These are the 4 steps of the algorithm.

1. Read the documents and collect every pair (word_id, doc_id)
2. Sort those pairs by word_id and by doc_id
3. For every word_id, group the pairs so you have its list of documents
4. Merge the intermediate results to get the final inverted index

See the figure below -


1
This
Copyright
Gutenberg's
most
...
2
Copyright
most
Shakespeare's
of
...
3
is
of
Project
See
...
3
most
Shakespeare's
of
This
...
4
Welcome
of
is
Project
...
(1,1)
(5,1)
(2,1)
(8,1)
(5,2)
(8,2)
(9,2)
(4,2)
(3,3)
(4,3)
(0,3)
(7,3)
(8,3)
(9,3)
(4,3)
(1,3)
(6,4)
(4,4)
(3,4)
(0,4)
(0,3)
(1,1)
(2,1)
(3,3)
(4,2)
(4,3)
(5,1)
(5,2)
(8,1)
(8,2)
(7,3)
(9,2)
(0,4)
(1,3)
(3,4)
(4,3)
(4,4)
(6,4)
(8,3)
(9,3)
(0, [3])
(1, [1])
(2, [1])
(3, [3])
(4, [2,3])
(5, [1,2])
(7, [3])
(8, [1,2])
(9, [2])
(0, [4])
(1, [3])
(3, [4])
(4, [3,4])
(6, [4])
(8, [3])
(9, [3])
(0, [3,4])
(1, [1,3])
(2, [1])
(3, [3,4])
(4, [2,3,4])
(5, [1,2])
(6, [4])
(7, [3])
(8, [1,2,3])
(9, [2,3])

Deliverables

The solution should work on a massive dataset. The algorithm should be able to run on a distributed system so we are not limited by the amount of storage, memory, and CPU of a single machine.

The index must be sorted by the word_ids, and for every word_id the matching list must be sorted by the ids of the documents.

You can find sample documents here.

You can use Python, Java, Golang, Rust or Scala. Please do NOT use any big-data frameworks like Spark, Flink, Hadoop, etc.

Write proper tests and documentation. Deliver your source code in a public Github repo. Include Dockerfile and Docker compose configurations (if required) for build and test.

References

https://nlp.stanford.edu/IR-book/html/htmledition/blocked-sort-based-indexing-1.html