1 Context

Aemon the Castle Black's maester is a close advisor of Jeor Mormont, the Lord Commander. He is also the Castle Black's library director. This library has hundreds of thousands of books and some of them are so rare you can not even find them in the Citadel.

Searching information in this huge books collection is getting more and more challenging for Aemon as the years pass. He decided to ask Sam, his favorite trainee for some help to find a way to easily **search information in documents from keywords**. Then Sam asked his friend Jon Snow who always liked information search and data engineering.

Jon Snow decided to use vector representation of documents. Search engines use different kind of document representations and one of them is the vector representation. It gives the ability to directly use mathematical tools such as distance, similarity and dimension reduction.

Our challenge is not about those mathematical tools but is about building an **inverted index** of the documents to speed up calculations. Indeed those calculations are often based on dot products that consume a lot of CPU and memory. Having an inverted index simplifies those dot products.
We want to write an efficient implementation to build an inverted index of a large collection of documents.

2 Algorithm

2.1 Documents

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 indice of the document as shown on the figure 1.

Figure 1: Document collection

2.2 Dictionnary

We want a dictionnary that matches every word from the documents with a unique id. See the figure 2 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
...
```

Figure 2: Dictionnary example

2.2.1 Inverted index

Using both the dataset and the dictionnary we can build an inverted index that gives, for every word, the list of documents it appears in. See the figure 3.

```
(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])
...
```

Figure 3: Inverted index example

We want a solution that works on a massive dataset. Our algorithm has to 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.

These are the 4 steps of the algorithm. See also the figure 4.

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

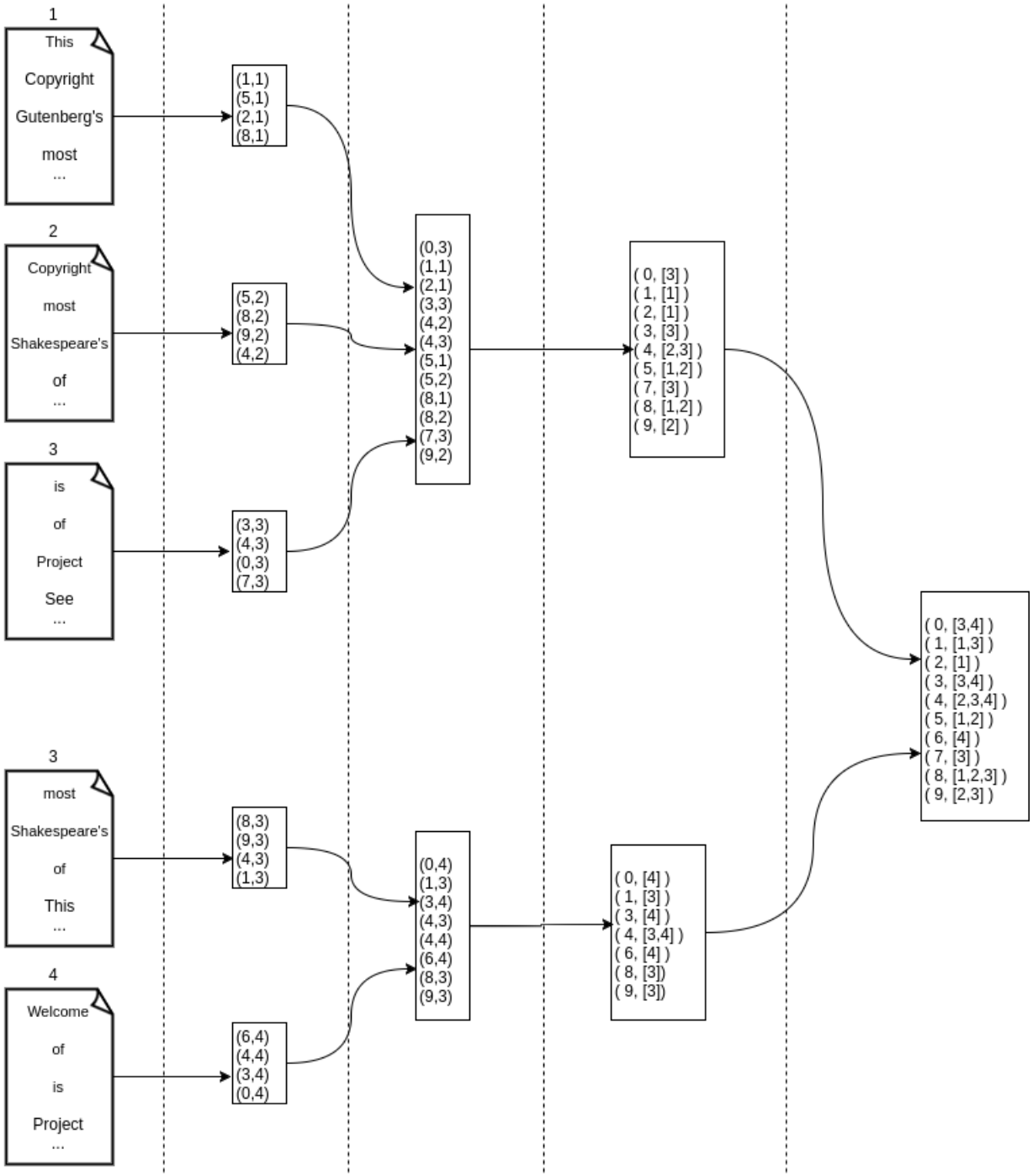

Figure 4: Algorithm steps

NB: the index must be sorted by the words ids, and for every word id the matching list must be sorted by the documents ids.
Associated algorithm: Blocked sort-based indexing

3 Questions

You will find the dataset of documents in the *dataset* folder of the repository.

1. Implement a job to build the dictionnary. Every line of the output file will be the word and its id. In the documents of the given dataset the words are separated by one or multiple spaces.

2. Implement one or multiple jobs to build the inverted index.

~~You may use Hadoop or Spark for those jobs, and the language you prefer.~~