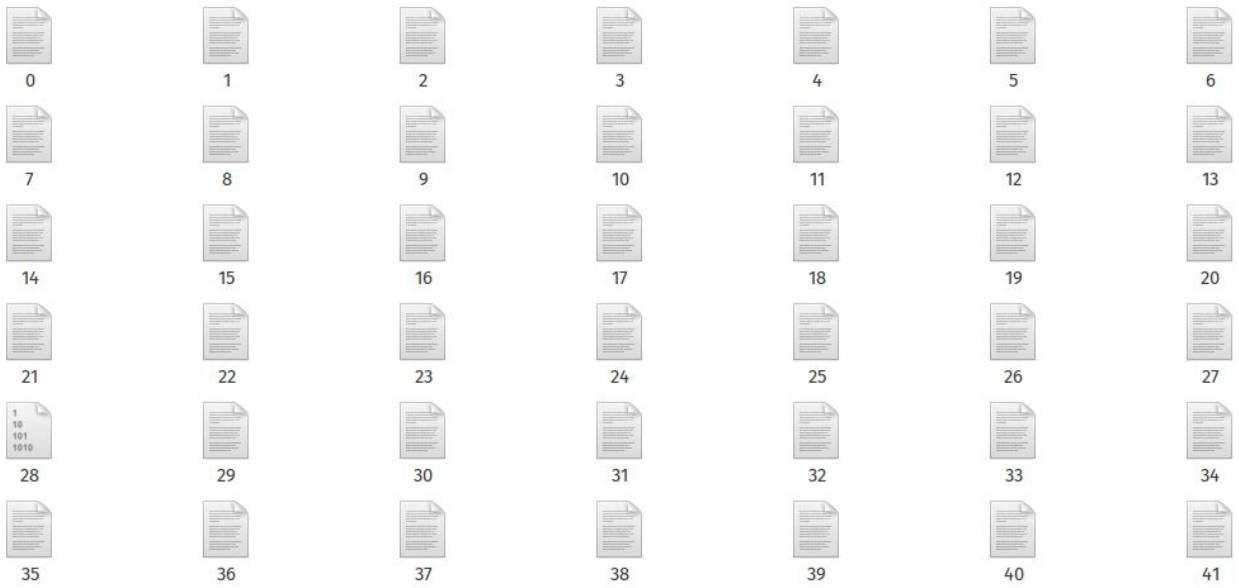
1 Premise

Search engines use different kinds 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 index of the document as shown below



2.2 Dictionary

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
...
```

2.3 Inverted Index

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

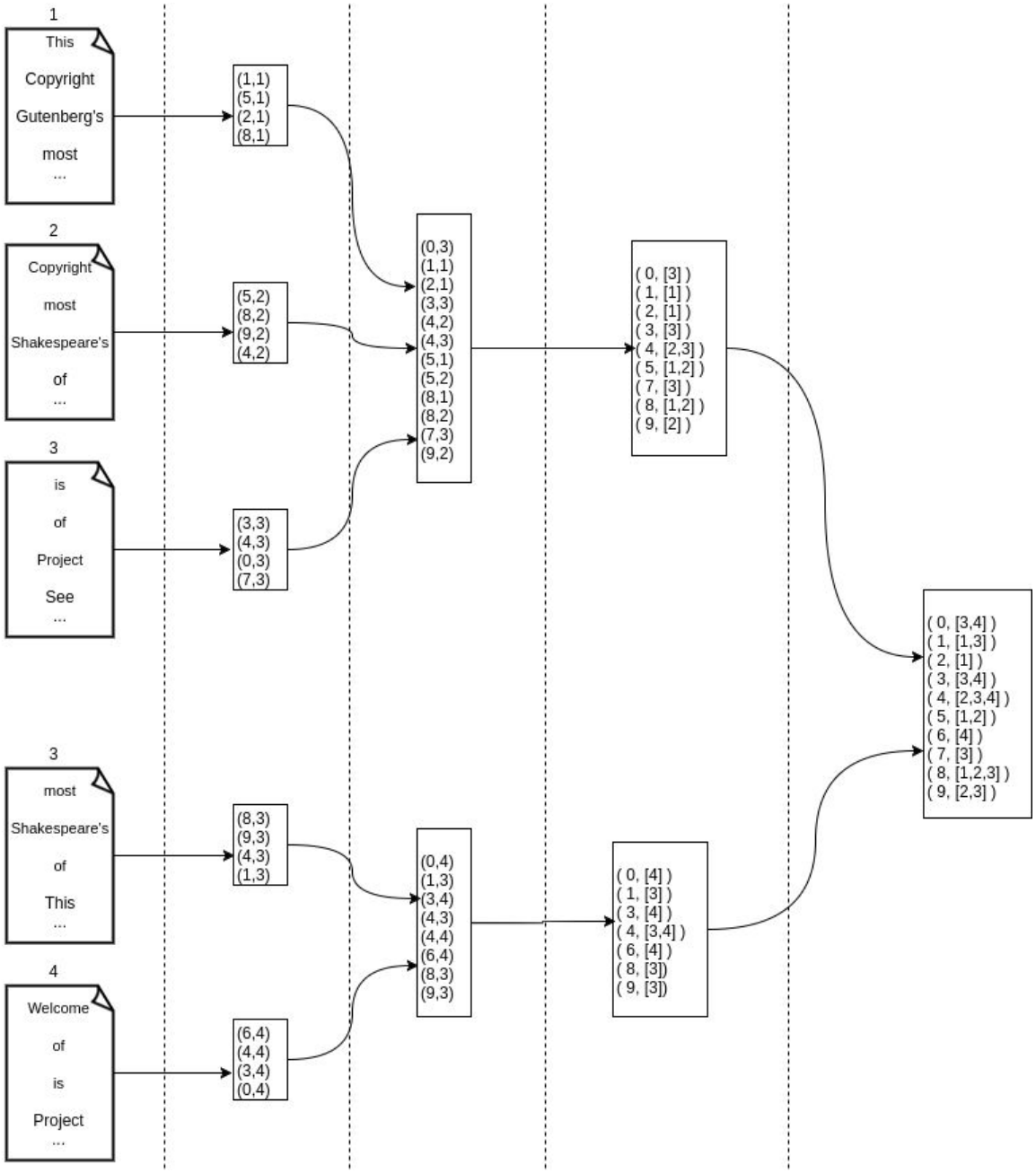
```
(0, [2,5,13,24,30])

(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])
...
```

2.4 Solution Approach

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 figure below)

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



Please Note: The index must be sorted by the words ids, and for every word id the matching list must be sorted by the document ids.
Checkout the associated algorithm - Blocked sort-based indexing

2.5 General Instructions

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

1. Implement a job to build the dictionary. 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.Use Spark for those jobs in the language you prefer.
3. Upload the solution to your personal github repository and send us the link within 7 days