

# Text Retrieval - HW1

## 1)

To make the positional index support queries that require the words to be in the **same sentence**, we can add information about sentence numbers to the index.

Normally, a positional index looks like this:

term → docID : position1, position2, ...

To include sentence information, we can change it to this form:

term → docID : sentence\_index–position1, position2 | sentence\_index–position3, position4

- Each **sentence** is marked with its own index (1, 2, 3, ...).
- Inside each sentence, we list the **term positions**.
- The | **symbol** separates different sentences.

This way, when we search, we can easily check that two terms are in the same sentence (they have the same sentence index) and also check their distance in positions (for example the second constraint, at most 2 words apart).

**Note:** We continue numbering term positions across sentences (instead of restarting from 1 in each new sentence). This way, the same index can still support other types of queries that don't depend on the "same sentence" condition.

### Example:

Doc 1: S1: I<sup>1</sup> am<sup>2</sup> a<sup>3</sup> scientist<sup>4</sup>, and<sup>5</sup> I<sup>6</sup> am<sup>7</sup> currently<sup>8</sup> enrolled<sup>9</sup> to<sup>10</sup> IR<sup>11</sup> course<sup>12</sup>.

S2: I<sup>13</sup> am<sup>14</sup> a<sup>15</sup> student<sup>16</sup> at<sup>17</sup> Reichman<sup>18</sup> University<sup>19</sup>.

Doc 2: S1: I<sup>1</sup> was<sup>2</sup> a<sup>3</sup> student<sup>4</sup> and<sup>5</sup> scientist<sup>6</sup>.

### Postings list:

"I": 1:1–1,6 | 2–13; 2:1–1

"scientist": 1:1–4; 2:1–6

## 2)

We want to count how many times each term appears in each document, and then build an inverted index where for each term we store a list of pairs:

term → (docID1, tf), (docID2, tf)

So, in our example, the inverted index will look like:

"data": (1, 2), (2, 1)

"science": (1, 1), (2, 1)

"mining": (2, 1)

### How term frequency is used for ranking?

A simple ranking rule for a query  $q$  is:

$$\text{score}(d, q) = \sum_{t \in q} \text{tf}_{t,d}$$

For each document, we sum the **term frequencies** of all **query terms** that appear in that **document**.

Example for the query: "data science"

$$\text{score}(\text{doc1}, q) = \text{tf}(\text{"data"}, \text{doc1}) + \text{tf}(\text{"science"}, \text{doc1}) = 2 + 1 = 3$$

$$\text{score}(\text{doc2}, q) = \text{tf}(\text{"data"}, \text{doc2}) + \text{tf}(\text{"science"}, \text{doc2}) = 1 + 1 = 2$$

So, Doc1 would be ranked higher than Doc2 for the query "data science", because it contains "data" more times and is therefore considered more relevant to the query.

**Note:** In class we also saw a more advanced version that uses Weighting TF instead of regular TF, where term frequency is scaled as  $1 + \ln(\text{tf}_{t,d})$ . This gives less importance to very high frequencies while still rewarding words that appear more than once.

### 3)

**A)** We can treat each malware object as a structured document and build a **fielded inverted index**:

- **Type index** (exact match, categorical field) – the type is the dictionary key, and the postings are list of malware (doc) ids. For example:  
"virus" → mID1, mD2
- **Author index** (can be treated like text, or also as category) – the author is the dictionary key (term), and the postings are list of malware (doc) ids. For example:  
authorName → mID1, mD2, mID3
- **Description index** (standard text inverted index, possibly with positions and tf) – the term is the dictionary key, and the postings are list of pairs (malwareID, [positions]) or (malwareID, tf). For example:  
term → mID1: pos1, pos2; mID2: pos1;

**Overall:** We use three separate dictionaries – one for type, one for author, and one for description.

**B) First step:** at query time, we first merge (AND) the results from the **type** and **author** indexes to keep only the malware objects that match those fields.

For example:

Type = "virus" → m3, m5, m8, m11

Author = "darkcrew" → m5, m7, m8

Candidates = "virus"  $\cap$  "darkcrew" = m5, m8

**Note:** If the query doesn't specify a type or author, we just skip that filter.

**Second step:** then, among those filtered results, we use the **description** index to rank the documents based on textual relevance to the query terms.

For each document, we want to calculate the retrieval score, by summing up the weighted  $tf$  (using  $idf$ ) – for each term from the query. For example:

$$W_{t,d} = (1 + \ln (tf_{t,d})) \times \ln \left( \frac{N}{df_t} \right)$$
$$score(d, q) = \sum_{t \in q} W_{t,d}$$

Then, rank all candidate malware objects by this score in **descending** order.

The **final result** list contains only malware that:

- satisfies the Boolean conditions on **type** and **author**
- is ordered by how well its **description** matches the description part of the query.

**Note:** After filtering by type and author (the first step), we can also use for ranking the **cosine similarity** between the query description and each candidate's description (by representing them as vectors).