School of Computing and Information Systems
The University of Melbourne
COMP90049 Knowledge Technologies (Semester 2, 2017)
Workshop exercises: Week 6

1. What are the four tasks into which a Web-scale information retrieval engine is usually divided? Briefly summarise each one.

   - **Crawling**: finding and downloading as many documents as we can from the web (hopefully all of them, although this isn't possible in practice)
   - **Parsing**: turning each document into a list of tokens (or terms), probably by removing page metadata, case folding, stemming, etc.
   - **Indexing**: building an inverted index out of all of the tokens in our downloaded document collection. (We stop worrying about the original documents at this point.)
   - **Querying**: after the previous three steps have been completed (off-line), we are ready to accept user queries (on-line), in the form of keywords, that we tokenise (in a similar manner to our document collection) and then apply our querying model (e.g. TF-IDF) based on the information in the inverted index, to come up with a document ranking
   - (Optionally) **Add-ons**: Extras to change the above ranking, based on ad-hoc application of certain factors, for example, PageRank, HITS, click-through data, zones, anchor text, etc.

2. When parsing Web pages:

   (a) What is "tokenisation"? What are some common problems that arise in tokenisation of English text? What about other languages?

   - Tokenisation is the act of turning the raw text of a document into tokens that we will use to compare against the (similarly tokenised) query, for example, by removing meta-data and punctuation, folding case, canonicalising for dialect differences, and so on.
   - In English, this might correspond to our notion of "words," although this is less well defined for languages with different approaches to morphology and syntax (many Aboriginal North American languages), and those that don't use whitespace in the same way (Mandarin, for example).
   - Even in English, there are many issues: for example, contractions — is `can't` one token or two? The lecture slides discuss many examples of this.
   - In Japanese, tokenisation was so bad for such a long time, that keyword search as we know it today was mostly useless, and up until recently, topic based approaches were more popular.

   (b) What is "stemming"? How might it be different in languages other than English?

   - The lecture slides discuss stemming — the process of removing morphological affixes to arrive a sort-of "base form" for a token — pretty thoroughly.
   - The problem, of course, is that different languages use different approaches for stemming, because the affixes to be removed are different in, say, German, than they are in English. This means that when you wish to perform stemming in the tokenisation processing for the document, you typically need to know the language in which the document was written. Language identification is potentially not as easy as it seems, and is still a productive research area. For example, see:

     Timothy Baldwin and Marco Lui (2010) Language Identification: The Long and the Short of the Matter, In Proceedings of Human Language Technologies: The 11th Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL HLT 2010), Los Angeles, USA, pp. 229–237.

3. Assume that we have crawled the following "documents":

```
1) The South Australian Tourism Commission has defended a marketing strategy
which pays celebrities to promote Kangaroo Island tourism to their followers
on Twitter.
2) Mr O'Loughlin welcomed the attention the use of Twitter had now attracted.
3) Some of the tweeting refers to a current television advertisement promoting
Kangaroo Island.
4) Those used by the Commission have included chef Matt Moran, TV performer
Sophie Falkiner and singer Shannon Noll.
5) He said there was nothing secretive about the payments to celebrities to
tweet the virtues of a tourism destination.
6) Marketing director of SA Tourism, David O'Loughlin, said there was no ethical
problem with using such marketing and it might continue to be used.
7) Depending on their following, celebrities can be paid up to $750 for one
tweet about the island.
```

- Parse each document into terms.

Let's consider a term to be a token with whitespace (\b) on either side. Let's also strip punctuation and fold case. Maybe we might stem the tokens as well and apply other canonicalisation processes.

- Construct an inverted index over the documents, for (at least) the terms `and`, `australia`, `celebrity`, `commission`, `island`, `on`, `the`, `to`, `tweet`, `twitter`

Assuming we've stemmed the tokens in the document collection, an inverted index for these terms might look like the following, using $(d, f_{d,t})$ pairs:

`and`	→	(4,1)	→	(6,1)								
`australia`	→	(1,1)										
`celebrity`	→	(1,1)	→	(5,1)	→	(7,1)						
`commission`	→	(1,1)	→	(4,1)								
`island`	→	(1,1)	→	(3,1)	→	(7,1)						
`on`	→	(1,1)	→	(7,1)								
`the`	→	(1,1)	→	(2,2)	→	(3,1)	→	(4,1)	→	(5,2)	→	(7,1)
`to`	→	(1,2)	→	(3,1)	→	(5,2)	→	(6,1)	→	(7,1)		
`tweet`	→	(3,1)	→	(5,1)	→	(7,1)						
`twitter`	→	(1,1)	→	(2,1)								

Although the document weights will depend on which model we are using, it's a reasonable idea to store the raw term frequencies at this point, as $f_{d,t}$ will probably be a component in $w_{d,t}$ later. (Although, given that almost all of the frequencies are 1, it might be easier to have an implicit representation!)

- Using the vector space model and the cosine measure, rank the documents for the query `commission to island on twitter`

  (a) Using the weighting function $w_{d,t} = f_{d,t} \times \frac{N}{f_t}$

    – For each term in each document, we need to find its weight according to this model. The term frequencies ($f_{d,t}$) have been discussed above; we also need the document frequencies (f_t) for each term.

    – To find the (inverse) document frequency of a term, we simply count up the number of documents in which the term appears; this can be done by inspection of the lists above. For example, the document frequency of `and` is 2 because it appears in documents 4 and 6; the document frequency of `australia` is 1 because it only appears in document 1 (although this could depend on how we pre-process terms like *SA*); and so on.

    – The other item of information we require is N, the number of documents in the collection: 7.

– We can now substitute into the formula to find the weights of the terms within the documents (we'll assume that the weights of terms not present in a document are all 0):

$$\begin{aligned}
w_{1,\text{and}} &= 0 \\
w_{2,\text{and}} &= 0 \\
w_{3,\text{and}} &= 0 \\
w_{4,\text{and}} &= 1 \times \frac{7}{2} = 3.5 \\
w_{5,\text{and}} &= 0 \\
w_{6,\text{and}} &= 1 \times \frac{7}{2} = 3.5 \\
w_{7,\text{and}} &= 0
\end{aligned}$$

And so on. I'll include a term–document matrix representation of this data for convenience below[1].

Doc	`and`	`aus`	`cel`	`com`	`isl`	`on`	`the`	`to`	`twe`	`twi`	Length
1	0	7	2.3	3.5	2.3	3.5	1.2	2.8	0	3.5	10.3
2	0	0	0	0	0	0	2.3	0	0	3.5	4.2
3	0	0	0	0	2.3	0	1.2	1.4	2.3	0	3.8
4	3.5	0	0	3.5	0	0	1.2	0	0	0	5.1
5	0	0	2.3	0	0	0	2.3	2.8	2.3	0	4.9
6	3.5	0	0	0	0	0	0	1.4	0	0	3.8
7	0	0	2.3	0	2.3	3.5	1.2	1.4	2.3	0	5.6

– With a cosine similarity model, we find the cosine of the angle between the vector specified by the query, compared with the angle specified by the vector specified by each document. What is the vector specified by the query here? Well, we aren't given an explicit model in this case, but we'll assume that it's binary: the weight is 1 where the term appears in the query, and 0 otherwise:

$$q \quad : \quad \langle 0,0,0,1,1,1,0,1,0,1 \rangle$$

– We can now plug these into the cosine formula:

$$\begin{aligned}
\cos(d_1, q) &= \frac{\langle 0,7,2.3,3.5,2.3,3.5,1.2,2.8,0,3.5 \rangle \cdot \langle 0,0,0,1,1,1,0,1,0,1 \rangle}{|\,\langle 0,7,2.3,3.5,2.3,3.5,1.2,2.8,0,3.5 \rangle\,| \cdot |\,\langle 0,0,0,1,1,1,0,1,0,1 \rangle\,|} \\
&= \frac{0 \times 0 + 7 \times 0 + 2.3 \times 0 + 3.5 \times 1 + 2.3 \times 1 + 3.5 \times 1 + 1.2 \times 0 + 2.8 \times 1 + 0 \times 0 + 3.5}{\sqrt{0^2 + 7^2 + 2.3^2 + 3.5^2 + \cdots}\sqrt{0^2 + 0^2 + 0^2 + 1^2 + 1^2 + 1^2 + \cdots}} \\
&\approx \frac{15.6}{(10.3)(2.2)} \approx 0.69
\end{aligned}$$

– You should notice that there are a few places we can speed up this calculation:
  * The length of the query is always the same: $\sqrt{5} \approx 2.2$
  * The length of the documents can be pre-calculated (they don't depend on the query): I've indicated them in the table above
  * Words where the weight in the query is 0 (i.e. words that don't appear in the query) don't affect the dot product, consequently, I'll only represent the slice through the vector space for the terms which actually appear in the query (`commission`, `island`, `on`, `to`, `twitter`) in the calculations below — although note that the document length calculations still take into account all of the terms!

[1] In reality, we would never want to calculate this exhaustively, because there are too many terms in the collection; instead, we only worry about the terms that are actually in the query, as well as the (fixed) length of the vectors.

$$\begin{aligned}
\cos(d_2,q) &= \frac{\langle 0,0,0,0,3.5\rangle \cdot \langle 1,1,1,1,1\rangle}{\mid \langle 0,0,0,0,0,0,2.3,0,0,3.5\rangle \mid \cdot \mid \langle 1,1,1,1,1\rangle \mid} \\
&\approx \frac{3.5}{(4.2)(2.2)} \approx 0.38 \\
\cos(d_3,q) &= \frac{\langle 0,2.3,0,1.4,0\rangle \cdot \langle 1,1,1,1,1\rangle}{\mid \langle 0,0,0,0,2.3,0,1.2,1.4,2.3,0\rangle \mid \cdot \mid \langle 1,1,1,1,1\rangle \mid} \\
&\approx \frac{3.7}{(3.8)(2.2)} \approx 0.44 \\
\cos(d_4,q) &= \frac{\langle 3.5,0,0,0,0\rangle \cdot \langle 1,1,1,1,1\rangle}{\mid \langle 3.5,0,0,3.5,0,0,1.2,0,0,0\rangle \mid \cdot \mid \langle 1,1,1,1,1\rangle \mid} \\
&\approx \frac{3.5}{(5.1)(2.2)} \approx 0.31 \\
\cos(d_5,q) &= \frac{\langle 0,0,0,2.8,0\rangle \cdot \langle 1,1,1,1,1\rangle}{\mid \langle 0,0,2.3,0,0,0,2.3,2.8,2.3,0\rangle \mid \cdot \mid \langle 1,1,1,1,1\rangle \mid} \\
&\approx \frac{2.8}{(4.9)(2.2)} \approx 0.26 \\
\cos(d_6,q) &= \frac{\langle 0,0,0,1.4,0\rangle \cdot \langle 1,1,1,1,1\rangle}{\mid \langle 3.5,0,0,0,0,0,0,1.4,0,0\rangle \mid \cdot \mid \langle 1,1,1,1,1\rangle \mid} \\
&\approx \frac{1.4}{(3.8)(2.2)} \approx 0.17 \\
\cos(d_7,q) &= \frac{\langle 0,2.3,3.5,1.4,0\rangle \cdot \langle 1,1,1,1,1\rangle}{\mid \langle 0,0,2.3,0,2.3,3.5,1.2,1.4,2.3,0\rangle \mid \cdot \mid \langle 1,1,1,1,1\rangle \mid} \\
&\approx \frac{7.2}{(5.6)(2.2)} \approx 0.58
\end{aligned}$$

– All of the documents have non-zero similarity, so they all get returned, and the order (from greatest estimated relevance to least) is: {1,7,3,2,4,5,6}.

– Notice also that each cosine calculation involves the same division by the length of the query (in this case, $\sqrt{5}$) — since all we care about is the order of the documents and not the actual score, we can safely leave this term out (but it isn't the cosine anymore, so I'll call it S below).

(b) Using the weighting functions $w_{d,t} = 1 + \log_2 f_{d,t}$ and $w_{q,t} = \log_2(1 + \frac{N}{f_t})$

– We have explicit weights for both the documents (TF) and the query (IDF) in this model; since we multiply the two terms together in the dot product anyway, this doesn't end up changing very much.

– What happens with the document weights? Well, let's recall that all of the term frequencies are 1, except for a few 2s (for `the` and `to`) — $1 + \log_2(1) = 1 + 0 = 1$; $1 + \log_2(2) = 1 + 1 = 2$. Consequently, the document weights are equivalent to the term frequencies for this collection. (Although, higher values would have been reduced in magnitude using this model.)

– What about the query weights? Well, we can just examine the document frequencies for the five terms in the query, and substitute to find:

$$q : \langle 2.2, 1.7, 2.2, 1.3, 2.2\rangle$$

Now, the cosine calculations look similar:

$$\begin{aligned}
S(d_1,q) &= \frac{\langle 1,1,1,2,1\rangle \cdot \langle 2.2,1.7,2.2,1.3,2.2\rangle}{\mid \langle 0,1,1,1,1,1,1,2,0,1\rangle \mid} \\
&\approx \frac{10.9}{\sqrt{11}} \approx 3.3 \\
S(d_2,q) &= \frac{\langle 0,0,0,0,1\rangle \cdot \langle 2.2,1.7,2.2,1.3,2.2\rangle}{\mid \langle 0,0,0,0,0,0,2,0,0,1\rangle \mid} \\
&\approx \frac{2.2}{\sqrt{5}} \approx 0.98 \\
S(d_3,q) &= \frac{\langle 0,1,0,1,0\rangle \cdot \langle 2.2,1.7,2.2,1.3,2.2\rangle}{\mid \langle 0,0,0,0,1,0,1,1,1,0\rangle \mid} \\
&\approx \frac{3.0}{\sqrt{4}} \approx 1.5 \\
S(d_4,q) &= \frac{\langle 1,0,0,0,0\rangle \cdot \langle 2.2,1.7,2.2,1.3,2.2\rangle}{\mid \langle 1,0,0,1,0,0,1,0,0,0\rangle \mid} \\
&\approx \frac{2.2}{\sqrt{3}} \approx 1.3 \\
S(d_5,q) &= \frac{\langle 0,0,0,2,0\rangle \cdot \langle 2.2,1.7,2.2,1.3,2.2\rangle}{\mid \langle 0,0,1,0,0,0,2,2,1,0\rangle \mid} \\
&\approx \frac{2.6}{\sqrt{10}} \approx 0.82 \\
S(d_6,q) &= \frac{\langle 0,0,0,1,0\rangle \cdot \langle 2.2,1.7,2.2,1.3,2.2\rangle}{\mid \langle 1,0,0,0,0,0,0,1,0,0\rangle \mid} \\
&\approx \frac{1.3}{\sqrt{2}} \approx 0.92 \\
S(d_7,q) &= \frac{\langle 0,1,1,1,0\rangle \cdot \langle 2.2,1.7,2.2,1.3,2.2\rangle}{\mid \langle 0,0,1,0,1,1,1,1,1,0\rangle \mid} \\
&\approx \frac{5.2}{\sqrt{6}} \approx 2.1
\end{aligned}$$

- The document ranking has changed a little bit this time (because the relative differences between common terms and rare terms is smaller due to the log): {1,7,3,4,2,6,5}
- Also note that some of the values are greater than 1, because I left out the division by the length of the query (hence S and not cos), which gives a factor of $\sqrt{19.1}$ in the denominator — it doesn't change the order though.

• Suppose there is an accumulator limit of 2 and that query terms are processed in order of decreasing rarity — are the same top two documents found? What are the similarities of the top two documents?

  - We're using a limiting approach toward accumulator usage: we have two accumulators, which I'm going to call A_1 and A_2 (which will be associated with their corresponding documents $A_{d,1}$ and $A_{d,2}$).
  - We'll process terms in order of decreasing rarity, which means that the rarest terms are processed first, down to the commonest terms — in our TF-IDF model, this means that the terms with the greatest $w_{q,t}$ value will be the rarest, and so on. (We'll use the model from part (b) above.)
  - The terms `commission`, `on`, and `twitter` are tied for the highest weight (they each occur twice in the collection). In practice, we'd probably do some kind of tie-break — say, in lexical order, or reverse lexical order, whichever allows faster processing, or maybe chosen randomly — in this case, let's consider what happens for each of these terms:

    (a) If we process `commission` first:

      ∗ The inverted list for `commission` contains document 1 and document 4.

∗ We first look at document 1, where `commission` occurs once. There's a free accumulator (A_1), so we update it with this document ($A_{d,1} = 1$) and increment the (initially zero) value with the dot-product component for this term:

$$\begin{aligned} A_1 &= A_1 + w_{q,\text{com}} \times w_{d_1,\text{com}} \\ &= 0 + 2.170 \times 1 \\ &= 2.170 \end{aligned}$$

∗ Next, we look at document 4, which also has one `commission`. There's a free accumulator (A_2), so we update it with this document ($A_{d,2} = 4$), and:

$$\begin{aligned} A_2 &= A_2 + w_{q,\text{com}} \times w_{d_4,\text{com}} \\ &= 0 + 2.170 \times 1 \\ &= 2.170 \end{aligned}$$

∗ If there were more documents, we'd be out of luck, because we've run out of accumulators. (But some models let us over-write accumulators if the score for this term would out-weigh the lowest accumulator value; for example, if the next document had two `commissions`, its accumulator value would be 4.340 and that document would replace, say, document 4.)
∗ We now process one of `on` or `twitter`. In this case, it doesn't matter which order we consider them, so let's do `on` first.
∗ The inverted list for `on` contains document 1 and document 7.
∗ We first look at document 1, with one `on`, for which there is an accumulator (A_1). We update its value with the dot-product component for this term:

$$\begin{aligned} A_1 &= A_1 + w_{q,\text{on}} \times w_{d_1,\text{on}} \\ &= 2.170 + 2.170 \times 1 \\ &= 4.340 \end{aligned}$$

∗ Next, we have document 7. There's no accumulator for that document, so we ignore it. (But again, some models will calculate its dot-product and replace the lowest-valued accumulator if this term would be better.)
∗ The next best term is `twitter`, which has documents 1 and 2 in its inverted list (both with a frequency of 1).
∗ For document 1:

$$\begin{aligned} A_1 &= A_1 + w_{q,\text{twi}} \times w_{d_1,\text{twi}} \\ &= 4.340 + 2.170 \times 1 \\ &= 6.510 \end{aligned}$$

∗ For document 2, there is no accumulator.
∗ The next best term is `island`, with documents 1, 3, and 7 in its list (all with frequency 1). We update document 1 again, and ignore the other two, because they don't have accumulators:

$$\begin{aligned} A_1 &= A_1 + w_{q,\text{isl}} \times w_{d_1,\text{isl}} \\ &= 6.510 + 1.737 \times 1 \\ &= 8.247 \end{aligned}$$

∗ The final term is `to`, with document 1 twice, document 3 once, document 5 twice, and documents 6 and 7 once. Again, document 1 is the only one with an accumulator:

$$\begin{aligned} A_1 &= A_1 + w_{q,\text{isl}} \times w_{d_1,\text{isl}} \\ &= 8.247 + 1.263 \times 2 \\ &= 10.773 \end{aligned}$$

- Note that the document 5 would replace document 4 at this point ($1.263 \times 2 = 2.526 > 2.170$) if our model allows it.
- The final step is to divide through by the length of the documents that correspond to accumulators:

$$\begin{aligned}
A_1 &= \frac{A_1}{W_{d_1}} \\
&= \frac{10.773}{\sqrt{11}} \approx 3.248 \\
A_2 &= \frac{A_2}{W_{d_4}} \\
&= \frac{2.170}{\sqrt{3}} \approx 1.253 \\
(\text{ or } A_2 &= \frac{A_2}{W_{d_5}} \text{, if we'd replaced 4 with 5} \\
&= \frac{2.526}{\sqrt{10}} \approx 0.799\ )
\end{aligned}$$

- We'd then return the documents from our accumulators from highest weight to lowest, namely 1 then 4 (or 1 then 5).

(b) What if we'd started with `on`?

- The inverted list for `on` contains document 1 ($A_{d,1}$) and document 7 ($A_{d,2}$):

$$\begin{aligned}
A_1 &= A_1 + w_{q,\text{on}} \times w_{d_1,\text{on}} \\
&= 0 + 2.170 \times 1 \\
&= 2.170 \\
A_2 &= A_2 + w_{q,\text{on}} \times w_{d_7,\text{on}} \\
&= 0 + 2.170 \times 1 \\
&= 2.170
\end{aligned}$$

- The next two terms are `commission` and `twitter`: both of them have document 1 in their inverted list, and other documents that don't have accumulators:

$$\begin{aligned}
A_1 &= A_1 + w_{q,\text{com}} \times w_{d_1,\text{com}} \\
&= 2.170 + 2.170 \times 1 \\
&= 4.340 \\
A_1 &= A_1 + w_{q,\text{twi}} \times w_{d_1,\text{twi}} \\
&= 4.340 + 2.170 \times 1 \\
&= 6.510
\end{aligned}$$

- `island` has both 1 and 7, as well as 3, for which there is no accumulator:

$$\begin{aligned}
A_1 &= A_1 + w_{q,\text{isl}} \times w_{d_1,\text{isl}} \\
&= 6.510 + 1.737 \times 1 \\
&= 8.247 \\
A_2 &= A_2 + w_{q,\text{isl}} \times w_{d_7,\text{isl}} \\
&= 2.170 + 1.737 \times 1 \\
&= 3.907
\end{aligned}$$

- `to` has both 1 (twice) and 7 (once), as well as other documents without accumulators:

$$\begin{aligned}
A_1 &= A_1 + w_{q,\text{to}} \times w_{d_1,\text{to}} \\
&= 8.247 + 1.263 \times 2 \\
&= 10.773 \\
A_2 &= A_2 + w_{q,\text{to}} \times w_{d_7,\text{to}} \\
&= 3.907 + 1.263 \times 1 \\
&= 5.170
\end{aligned}$$

∗ (Note that none of the other documents will surpass 5.170 this time.) We normalise:

$$\begin{aligned}
A_1 &= \frac{A_1}{W_{d_1}} \\
&= \frac{10.773}{\sqrt{11}} \approx 3.248 \\
A_2 &= \frac{A_2}{W_{d_7}} \\
&= \frac{5.170}{\sqrt{6}} \approx 2.111
\end{aligned}$$

∗ This time, we return document 1, followed by document 7.

(c) And if we'd started with `twitter`? Well, the calculations end up looking a lot like those of `commission` above:

∗ The inverted list for `twitter` contains document 1 ($A_{d,1}$) and document 2 ($A_{d,2}$):

$$\begin{aligned}
A_1 &= A_1 + w_{q,\text{on}} \times w_{d_1,\text{on}} \\
&= 0 + 2.170 \times 1 \\
&= 2.170 \\
A_2 &= A_2 + w_{q,\text{on}} \times w_{d_2,\text{on}} \\
&= 0 + 2.170 \times 1 \\
&= 2.170
\end{aligned}$$

∗ The next terms are `commission` and `on`: both have document 1, neither has document 2:

$$\begin{aligned}
A_1 &= A_1 + w_{q,\text{com}} \times w_{d_1,\text{com}} \\
&= 2.170 + 2.170 \times 1 \\
&= 4.340 \\
A_1 &= A_1 + w_{q,\text{on}} \times w_{d_1,\text{on}} \\
&= 4.340 + 2.170 \times 1 \\
&= 6.510
\end{aligned}$$

∗ `island` has document 1, as well as other documents without accumulators:

$$\begin{aligned}
A_1 &= A_1 + w_{q,\text{isl}} \times w_{d_1,\text{isl}} \\
&= 6.510 + 1.737 \times 1 \\
&= 8.247
\end{aligned}$$

∗ `to` has document 1 (twice), no document 2, and other documents without accumulators:

$$\begin{aligned}
A_1 &= A_1 + w_{q,\text{isl}} \times w_{d_1,\text{isl}} \\
&= 8.247 + 1.263 \times 2 \\
&= 10.773
\end{aligned}$$

        * (Again, document 5 might replace document 2 at this point ($2.526 > 2.170$).) We normalise as usual:

$$\begin{aligned}
A_1 &= \frac{A_1}{W_{d_1}} \\
&= \frac{10.773}{\sqrt{11}} \approx 3.248 \\
A_2 &= \frac{A_2}{W_{d_2}} \\
&= \frac{2.170}{\sqrt{5}} \approx 0.970 \\
(\text{ or } A_2 &= \frac{A_2}{W_{d_5}} \text{ , if we'd replaced 2 with 5} \\
&= \frac{2.526}{\sqrt{10}} \approx 0.799\ )
\end{aligned}$$

        * And this time, we return 1, followed by 2 (or 1 followed by 5).
    - So what? Well, obviously the return set is different because we can only return 2 documents, as opposed to all of the documents from the approach without accumulators above.
    - Note also the different ordering; we still find that document 1 is the top document, but whether we find the second best document (7) depends on the order in which we process the terms. In some models, we actually end up returning the **worst** document (5).
    - Even if we'd allowed **three** accumulators, we'd end up with one of the following document orderings: {1,7,4}, {1,7,2}, {1,7,5}, {1,4,2}, {1,4,5}, {1,2,5} — depending on the order that we process terms, whether we allow for replacements, and how we break ties.
    - We'd need **four** accumulators to guarantee that document 7 is returned as a result, regardless of the order that we process the terms.

4. What is "phrase querying"? What extra information would you need to store in your inverted index to accomplish phrase querying? How much extra space would that require? Indicate how you would use that information to execute a phrasal query like `to the commission` on the data set above.

    - Phrase querying is where we treat the query as a phrase, which is to say, that we care about the *order* of terms in the query — in particular, we wish to return documents where the terms occur one after each other, in the order that they occur in the query. (You've probably tried this in Google or other search engines by wrapping your query in quotation marks " ".)
    - For a search engine to process queries of this type, the positions of each instance of the term within the document needs to be stored (in the inverted index). If we're going to use a 4-byte unsigned integer for each term position, then our inverted index will be about the same size as our original document collection! (However, there are some compression tricks that we can use.)
    - When processing our inverted index to search for results to the query, we find the document set which contains all of the terms in the query, and then check the corresponding positions of the query terms in each document, to see if they have occurred in order (the positions are sequential). (Note that we will need to decode the list of positions that we discussed encoding above.) The brute force approach is pretty slow, but there are a couple of ways that we can make it faster, for example, by starting with the term that occurs the least frequently within a given document.

5. When evaluating a query, why should we begin with the rarest terms (those with greatest $w_{q,t}$)? Is this more true for ranked or Boolean querying?

    - This is far more true for Boolean querying, where we are typically performing set intersections. By starting with the rarest terms, we can typically reduce the comparisons because each intersection guarantees that the resulting set is as small or smaller than the sets under consideration. If we started with common terms, we would do a lot of comparisons for documents that are going to be pruned in for the rare terms anyway.
    - For ranked querying, it can still be true for certain approximate methods. But conceptually, if we are just filling in accumulators for each document from the inverted index as we process terms, we don't get any benefit from starting with the high-weight terms. (Unless the number of accumulators we're using is really small, in which case we get some weird effects in missing good documents, as shown above.)

6. What is "link analysis"? Briefly describe the PageRank algorithm and the rationale behind it.

    - The terms in the document aren't the only factor for determining which documents are relevant — for example popular documents are often more relevant than unpopular documents. One way of approximating this is through **click-through** data, especially if we've already processed a given query before: if someone has clicked though to a certain document, it's more likely to be relevant than the documents that they didn't click on.
    - Link analysis is another method for approximating this: basically people tend to provide hypertext links on their webpages to useful documents. If a document has many (good) incoming links, it is more likely to be relevant than a document with few (good) incoming links.
    - The lecture slides has some material on PageRank (the pseudocode has been added to the lecture slides as an appendix of-sorts), and the reading is quite thorough. PageRank is quite complicated, but briefly, we set up a network where each page has "credits" associated with it. Then we have an iterative algorithm where pages transfer their credits to the pages to which they have outgoing links, and receive credits from pages with links pointing to them. (There's also another term for "random teleports", where pages transfer a portion of their credits to every other page evenly). After a number of iterations, the pages with a large number of credits tend to be better than documents with few credits, and we might use this to alter our document ranking somewhat.