School of Computing and Information Systems
The University of Melbourne
COMP90042 Web Search and Text Analysis (Semester 1, 2019)

Workshop exercises: Week 3

Discussion

1. Data compression of a **postings list** in an inverted index can help reduce space usage of the index.

   (a) What is the intuition behind compression algorithms used for postings list compression? Why do they work?

   - Data compression seeks to reduce space usage by removing redundancy from the data. As a general principle, we want to spend fewer bits representing objects that occur often. For example, a regular ASCII symbols always 8 bits. However, in English text symbols such as ′ ′ or ′e′ occur much more often. Thus, when storing English text we would like to represent frequent symbols with less than 8 bits. On the other hand symbols that occur infrequently (such as ′q′ or ′z′) could even be represented using more than 8 bits!
   - In terms of postings list compression we generally have two sequences of integers associated with each term. The ids of documents containing that term (in the range [1,N] where N is the number of documents in the collection) and the frequencies with which each term occurs in a specific document.
   - The list of document identifiers is generally stored as an increasing sequence of integers often containing large numbers. Frequencies are generally small (a given word only occurs a few times in each document).
   - Instead of storing the increasing list of document identifiers we instead store the *gaps* between subsequent document ids. This transforms the postings lists data into a set of integers consisting of mostly small values. Thus, we seek to create integer compression schemes which are able to store small numbers efficiently, while spending more bits on large numbers (which occur infrequently).

   (b) What is **Variable Byte Compression** and how does it compress an integer?

   - Variable Byte compression is an integer encoding scheme which allows storing small numbers in fewer bytes (compared to using a fixed 32/64 bits per integer).
   - Conceptually it chunks up the binary representation of a number X into 7 bit chunks. Each chunk is stored in an individual byte where the top bit is used to indicate if the current byte stores a complete number or additional bytes have to processed.
   - Small numbers (less than 128) can be stored in one byte only. In practice this corresponds to 95% of all postings that have to be stored in an inverted index.

(c) Determine the values of integers X and Y that were encoded as the byte sequence [52,34,147,42,197] using the Variable Byte algorithm described in the lecture slides 10.

- It works as follows:
  i. Read byte 52 which corresponds to `00110100`.
  ii. The top bit is not set so we read the next byte $34 =$ `00100010`.
  iii. Again the top bit is not set. We read the next byte $147 =$ `10010011`.
  iv. Now the top bit is set, we have read all the bytes to decode the first number.
  v. We wrote the least significant 7 bits first so the 7 bits coming from 52 correspond to the left-most 7 bit chunk of the final number: `xxxxxxx|xxxxxxx|0110100`
  vi. Next, the 7 bits extracted from 34 are added: `xxxxxxx|0100010|0110100`
  vii. Finally, the last 7 bits are added to the bit representation: `0010011|0100010|0110100`, which is 315700.
  viii. The second number is decoded the same way as $42 =$ `00101010`, $197 =$ `11000101` which is `1000101|0101010` = 8874.
- Interestingly, Vbyte codes are self-delimiting, meaning we don't have to store how many numbers there are or where they start and end.

2. Algorithms such as WAND help speed up query processing.

(a) What is the intuition behind WAND? What is the output produced by WAND?

- In practice, search engines (and users!) are only interested in the first $10/100$ documents so evaluating everything is very inefficient. Even if a single evaluation of one document is fast, for a large number of documents, searching would still take seconds which is unacceptable for most users.
- WAND is a top-K query processing algorithm. Thus, it returns only the top-K highest scoring documents.
- As no complete ranking of all documents is required, we can skip evaluating documents that would/could not enter the final top-K results list.

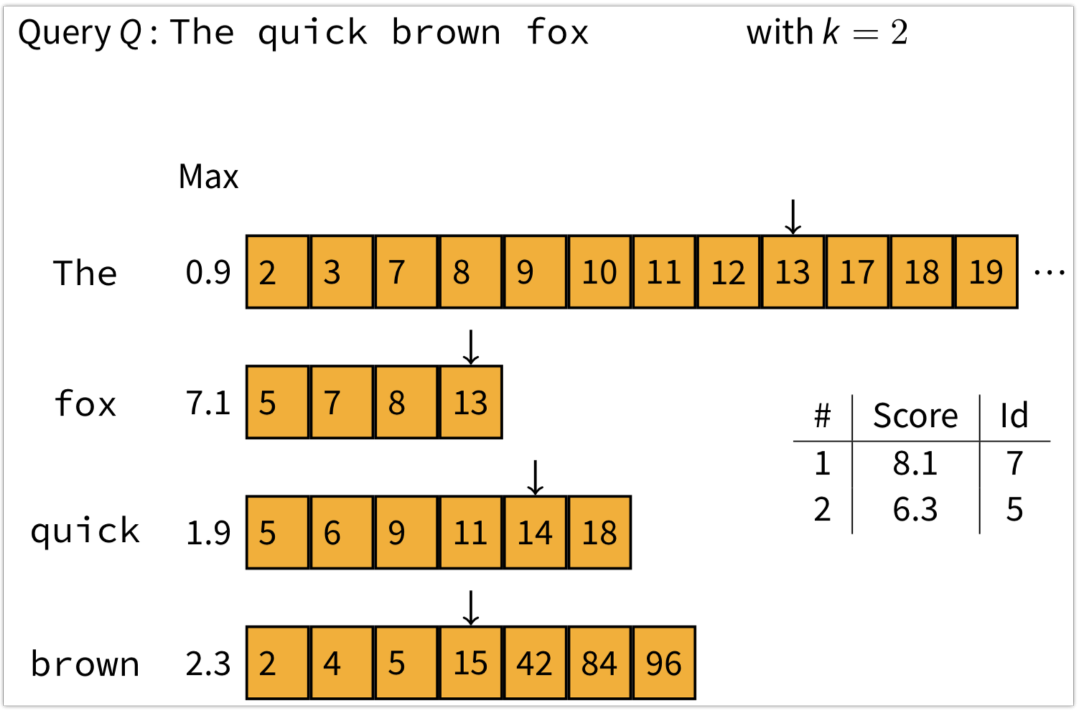
(b) What extra information is stored for each term to allow algorithms like WAND to skip evaluating documents? How is it computed? What restriction does it place on the query process?

- For each term in the collection one additional floating point number called the *maximum contribution* is stored.
- The maximum contribution is used to overestimate the score any term of the query can have on the total score of a document. Thus, if a document score including these overestimated contributions is LESS than the lowest scoring term in our current top-K list, it can never be part of the final top-K result set. Therefore we do not have to evaluate it.
- The maximum contribution for each term is computed by computing the similarity metric for a query that contains only that term for all documents in the index (at construction time!). For example, to compute the maximum contribution for the term "house" we run the query "house"

and compute the score for all documents. The highest score is then stored as the maximum contribution the term "house" can have to the score of ANY document. Note that this introduces a dependency between the similarity metric (e.g. BM25) and the index which is stored on disk. Thus, we cannot use a different metric at query time.

- Additionally, we would want the underlying postings list representation to support the `GEQ` operation which allows us to efficiently skip over documents without decompressing large parts of the index.

(c) Assume Document 13 has just been evaluated. In the setting below, what is the next document that will be evaluated?



Query Q: The quick brown fox with $k = 2$
Max
The 0.9 2 3 7 8 9 10 11 12 13 17 18 19 ···
fox 7.1 5 7 8 13
quick 1.9 5 6 9 11 14 18
brown 2.3 2 4 5 15 42 84 96
# Score Id
1 8.1 7
2 6.3 5

- 13 is evaluated and might enter the top-K list.
- After 13 is evaluated the list for the term `fox` is finished. Thus, it can no longer contribute to the score of any document.
- To decide which is the next *potential* document to be evaluated, we (1) sort the list by increasing current id and (2) sum up the maximum contributions of the lists (top to bottom) until the sum is larger than the smallest score in our top-K list.
- However, the maximum scores for the remaining lists (fox is finished) is 0.9, 1.9 and $2.3 = 0.9 + 1.9 + 2.3 = 5.1$. So, even if 13 does not get added to the top-K list, the minimum score in the heap is already larger than the score of ANY remaining document we have not evaluated. Thus, the WAND algorithm finishes and we return the top-K result list.

3. Discuss the concepts of **query expansion** and **relevance feedback** and how they are related.

- Query expansion improves query *recall* by dealing with the *vocabulary mismatch* problem where terminology in the query and terminology in the document collection don't match (e.g. poison vs toxin).
- Relevance feedback is one way to expand/reformulate the query by incorporating feedback into the query process. There are many different ways to incorporate and retrieve relevance feedback (discussed below).
- However, there are other query expansions methods such as using a thesaurus or wordnet which do not require relevance feedback.

4. Discuss and give an example of the following forms of relevance feedback.

   (a) User relevance feedback

   (b) Pseudo relevance feedback

   (c) Indirect relevance feedback

   (a) User feedback comes directly from the user issuing the query. This might be in the form of clicking on a result or swiping left/right on your phone. In the lecture we looked at Pinterest where we could refine the results returned for the query "watch" by clicking on watches we like. HOW this feedback is incorporated into the search process depends. One way could be to do query expansion.

   (b) Pseudo relevance feedback (or sometimes called blind feedback) is an automated system that does not require any actions by the user. It is based on the assumption that after running the original user query, the top documents will likely be relevant. So we analyze these documents to extract additional informative/important terms from these documents. Different methods can be used to determine these additional terms. We might look at the topic models of these documents to determine additional terms. Another methods might look at the frequency of terms in the documents relative to their occurrence in the overall collection. For example, in the top documents of the query "melbourne public transport", the terms "tram" or "bus" will likely occur with higher frequency than what we would expect from other documents in the collection. Thus, these terms might be candidates for query expansion. If these terms are included in a secondary query, we would be able to find documents talking about "melbourne trams" even though they might not mention "public transport". Thus, the *recall* would increase.

   (c) User feedback is problematic because users might be reluctant to provide manual feedback. Pseudo relevance feedback has the drawback that no actual feedback from any human is incorporated in the process. Indirect user feedback seeks to incorporate implicit feedback from users. That might be statistics about what pages users visit after being displayed a results for a certain query. Thus, the feedback would not be from the current user seeking information, but other users who searched for the same query in the past. These information are collected in large quantities in query click logs at most major web search engines and e-commerce platforms. Note that these information might not be used to perform query expansion but rather re-rank documents directly based on this click feedback.

5. Is it possible to perform **query expansion** without **relevance feedback** and vice versa? Discuss!

    (a) Query expansion can be performed using global resources such as WordNet which do not require relevance feedback.

    (b) Relevance feedback from a user (direct or indirect) can be incorporated without performing query expansion. This might be in the form of re-ranking existing documents or discovering documents that are similar to documents (the "more like this" button) to the ones users consider relevant.