

Assignment #5 (practice only – not graded)

Question 1. Bloom Filters

- (a) Suppose you have a Bloom Filter consisting of an m -bit array and k hash functions with values ranging from 0 to $m - 1$. What is the false positive rate after n objects have been inserted into the Bloom Filter?
- (b) What are the ratios of m to n if you want to achieve a false positive rate of 0.1% with one, two, and three hash functions?
- (c) Suppose you wish to use 12 bits per object for the Bloom Filter. What is the false positive rate if five hash functions are used?
- (d) Suppose you have a Bloom Filter with three hash functions and a bit array of length 13. The three hash functions h_1 , h_2 , and h_3 produce the following values for the three objects to be inserted into the Bloom Filter.

$$\begin{array}{lll} h_1(object_1) = 11 & h_2(object_1) = 3 & h_3(object_1) = 12 \\ h_1(object_2) = 9 & h_2(object_2) = 11 & h_3(object_2) = 1 \\ h_1(object_3) = 3 & h_2(object_3) = 8 & h_3(object_3) = 12 \end{array}$$

You have learnt that a Bloom Filter provides only two functions, `insert(h)` and `exist(h)`, where h is the hash value provided by a hash function.

- (i) What is the state of the Bloom Filter after inserting the three objects? (Show the values in the bit array.) How many times will the `insert()` function be called?
- (ii) Suppose another (fourth) object produces the hash values of 3, 12, and 9 for the hash functions h_1 , h_2 , and h_3 respectively. What will the `exist()` function return if the other three objects have already been inserted? Does the fourth object look as if it exists in the Bloom Filter? Why or why not?
- (iii) In general, if an object does not exist in the Bloom Filter, what will the `exist()` function return, and what does the answer mean?

Question 2. Top- k Queries

- (a) Suppose you have two lists of ranked pages, denoted as L_1 and L_2 . Each entry of L_i ($i = 1, 2$) is in the form of $(did_p, v_{p,i})$, where did_p is a document ID used to identify a page p on the web, and $v_{p,i}$ is the cosine measure value of the page p with respect to L_i . The two lists are sorted in descending order by the $v_{p,i}$ values, as follows.

$$\begin{array}{llllllll} L_1 & (17, 0.53) & (77, 0.51) & (91, 0.47) & (88, 0.46) & (52, 0.40) & (47, 0.38) & (12, 0.29) & (13, 0.27) \\ L_2 & (47, 0.30) & (13, 0.28) & (91, 0.27) & (77, 0.26) & (12, 0.25) & (88, 0.22) & (17, 0.22) & (52, 0.19) \end{array}$$

Assume L_1 and L_2 are the lists for the terms “panda” and “pangolin”, and you would like to search for pages that match both terms. Show how you can use Fagin’s Algorithm (FA) and the Threshold Algorithm (TA) from class to determine the top-2 results, i.e., pages with the best two values of $v_{p,1} + v_{p,2}$.

- (b) Does TA perform better than FA in the above example? Why or why not?

Question 3. Recall and Precision

- (a) Define “recall” and “precision” in one sentence each.
- (b) Suppose you have a collection of 1000 documents. Among these 1000 documents, 89 are relevant to the query “pangolin”. Suppose you have a search engine which returns only the top-10 results to each user query. Of the top-10 results returned by the search engine for the query “pangolin”, 3 of them are not relevant, while 7 are relevant. What is the precision of the search engine for the query “pangolin”? What is the recall? Does it actually make sense to talk about recall in this case? Discuss.

Question 4. Recrawling

Suppose you have a search engine that covers a domain containing a total of 100 million URLs (pages), e.g., the `hk` domain for Hong Kong. We make the simplifying assumption that the URLs stay the same (i.e., no new pages are created under new URLs) but that the content available under each URL (the page) can change due to updates. Suppose you have a web crawler and internet connection that allow you to recrawl up to 20 million pages per day.

Your goal is to recrawl each page periodically in a way that minimizes the number of outdated pages in your collection. Page updates are assumed to occur at random according to the following model, where the collection contains three types of pages: 20 million pages of type *A*, which have an 50% chance of being updated on any particular day, 20 million pages of type *B*, which have a 10% chance of being updated on any particular day, and 60 million pages of type *C*, which have only a 1% chance of being updated on any particular day.

For simplicity, you may assume in the following that all the 10 million pages that can be crawled on each day are crawled exactly at midnight, and that we are interested in the number of outdated pages that exist right after this occurs. This way, you do not have to worry about when exactly during the day a page is updated or recrawled.

- (a) Suppose you decide to recrawl all pages at the same speed, so that every page is visited every 5 days. What is the expected number of pages in your collection that are outdated at any point in time (i.e., that have changed since you last crawled them)? Or suppose every day you recrawl 10,000,000 pages of type *A*, 8,000,000 of type *B*, and 2,000,000 pages of type *C*. What is the expected number of pages that are outdated under this policy? Is it better or worse than the first policy? (Show how to derive formulas for estimating the number of outdated pages, and give at least an approximation of the resulting percentages.)
- (b) Discuss other possible recrawling strategies for this scenario. How should you allocate your crawling capacity among the pages? Can you come up with a better allocation that further reduces the number of outdated pages? Would you use most of your crawling capacity to crawl pages from class *A*, or class *B*, or class *C*, or in what combination? Discuss.