CSCI 6370 IR and Web Search
ASSIGNMENT 3
Due is 06/22/2020 23:59
Ulvi Bajarani
Student ID 20539914
E-mail: ulvi.bajarani01@utrgv.edu

Questions and Answers:

Problem 1. Assume that a document corpus follows Zipf's Law. Show that in this corpus, the number of words with frequency 1 is $N/2$ statistically, where N is the total number of words in the corpus.

Answer 1.
By Zipf, the number of words appearing n times has the rank r_n equal to

$$r_n = \frac{AN}{n}$$

, where A — the constant, N — total number of the words.
Knowing that the word r_n appearing n times less than r_1, and r_{n+1} appearing $n+1$ times less than r_1, we can calculate the number of words appearing exactly n times by

$$I_n = r_n - r_{n+1} = \frac{AN}{n} - \frac{AN}{n+1} = \frac{AN}{n(n+1)}$$

By the formula, we can see that if $n = 1$, the number of words that is appeared exactly once is directly dependent to the half of the total number of words, $N/2$. In other words, if we don't count the A, $I_n \sim N/2$.

Problem 2. Suppose that you own a web search service. You would like to allow advertisers to bid for "keywords". That is, ads for highest bidders will be displayed

when user query contains a purchased keyword. Since you the popularity of search keywords meets Zipf's Law, can you explain a meaningful way to set the base prices for keywords to help you make lots of profits?

Answer 2.

Assume a keyword price is P_n. Since that the search keywords meets Zipf's Law, it is better to have the logic in Amazon.com: set high prices for frequent words, and low prices for rare words. In other words, the dependence should be $P_n \sim \frac{1}{r_n}$.

Problem 3. Suppose that you need to implement a web search service – Fishing Guide for the Gulf of Mexico. As a part of this project, you need to collect and index web pages pertaining to fishing in the Gulf of Mexico. Explain how you may use the topic directed crawling to help your implementation.

Answer 3. The fish-search algorithm, despite its some limitations, might be used for topic directed crawling:

- Get as Input parameters, the initial node, the width (width), depth (D) and size (S) of the desired graph to be explored, the time limit, and a search query;
- Set the depth of the initial node as depth = D, and Insert it into an empty list;

- While the list is not empty, and the number of processed nodes is less than S, and the time limit is not reached:
  1. Pop the first node from the list and make it the current node;
  2. Compute the relevance of the current node;
  3. If depth > 0:
     1. If current_node is irrelevant to the query,
        Then:
        - For each child, child_node, of the first width children of current_node:
          - Set potential_score(child_node) = 0.5;
        - For each child, child_node, of the rest of the children of current_node:
          - Set potential_score(child_node) = 0;

        Else:
        - For each child, child_node, of the first (a * width) children of current_node (where a is a pre-defined constant typically set to 1.5):
          - Set potential_score(child_node) = 1;
        - For each child, child_node, of the rest of the children of current_node:
          - Set potential_score(child_node) = 0;
     2. For each child, child_node, of current_node:
        - If child_node already exists in the priority list,
          Then:

1. Compute the maximum between the existing score in the list to the newly computed potential score;
2. Replace the existing score in the list by that maximum;
3. Move child_node to its correct location in the sorted list if necessary;
Else:

- Insert child_node at its right location in the sorted list according to its potential_score value;

3. For each child, child_node, of current_node:

- Compute its depth, depth(child_node), as follows:

1. If current_node is relevant to the query,
Then: Set depth(child_node) = D;
Else: depth(child_node) = depth(current_node) - 1;
2. If child_node already exists in the priority list,
Then:
   1. Compute the maximum between the existing depth in the list to the newly computed depth;
   2. Replace the existing depth in the list by that maximum.

- EndWhile;