1 Laboratory Description and Procedures

1.1 Learning Objectives

You should be able to do the following after completing this laboratory.

1. Understand how the Internet and the World Wide Web can be modeled as a graph and analyzed using its structural properties.

2. Understand the Python libraries for retrieving the text of Web pages and parsing them to extract content.

3. Understand the basics of how a search engine works.

1.2 Key Words

You should be able to define the following key words after completing this laboratory.

1. World Wide Web graph

2. pattern search

3. Knuth-Morris-Pratt algorithm

4. page rank

1.3 Scenario

In Lab 9, you learned how to implement a basic Web crawler and to construct a graphical representation of (a portion of) the World Wide Web. In this lab, you will combine that with a basic pattern search algorithm in order to develop a (very) rudimentary search engine. Note that real search engines are substantially more complex than what we will construct here. A real search engine works (such as Google) works approximately as follows:

- Crawl the Web as we did in Lab 9 and store the contents of each page.

- Create a graph consisting of all links between documents and compute the page ranks from that graph.
• For each link, store the anchor text (the text describing the link), as this may be more useful in determining what information a page contains than the page itself.

• Create a “lexicon” containing all possible search terms (mostly just dictionary words in various languages).

• Create unique IDs for each page and each word in the lexicon (using hashing).

• Create two indexes, the forward and inverted.
  – The forward index contains, for each document, a list of the IDs of all words in the document.
  – The inverted index contains a list of all documents in which each word occurs. Entries on the hit list include where in the document the word occurs, what its font size is and other relevant information.

When a user performs a search, the hit lists of each word in the query is searched for documents that also contain the other search terms. The documents are ranked according to a combination of:

• The page rank of the documents.

• A weighting of the importance of the search terms themselves (words that occur less frequently overall may be given higher weight since these are good “discriminators”).

• Other information about the search terms, such as the font size in the documents they appear in, whether they appear in anchor text, etc.

Putting together all of this information, we can return a list of the most relevant documents. Here, we will use a simplified model.

• We will only allow to search for single strings.

• We will create only the inverted index.

• We will return all matches and sort only by page rank.

1.4 Design and Analysis
You will use code similar to that from Laboratory 6 to build the inverted index as you crawl the Web and build your graph.

1.5 Program Specifications
Using your code from Laboratory 6 as a basis, implement a method for storing all the words appearing on a given Web page that will be called in the `parse()` method of the `Net` class.

After creating your inverted index, your program should prompt the user for a search term, find, pages containing that term and return the results (with some context), ranked by page rank.

1.5.1 Algorithms
The algorithm exemplified in this lab are graph search and hashing.
1.5.2 Data Structures

The data structures used in this lab are the graph data structure with adjacency lists and a hash table.

2 Laboratory Assignments

2.1 Programming and Analysis (60 points)

1. (30 points) Implement a method for adding the contents of a page to the inverted index.

2. (30 points) Implement a search method for the Net class that takes a search term as an argument and returns a list of hits using the inverted index. Your program should prompt the user for the search term in a loop so that you can search for more than one term without having to re-create the index (this is how search engines process queries so fast).

3. (20 points) Implement a print_results method that sorts the results by page rank and prints them out with some additional context, like this:

```
=================================================
"http://www.npr.org" (page rank 0.0694619930943 )
... When Danish Cows See Fresh Spring Pasture,...
=================================================
```

2.2 Follow-up Questions (20 points)

1. (10 points) Show the comparisons the string matcher makes for the pattern 0001 in the string 000010001010001 for all three algorithms we discussed (naive, Rabin-Karp, and KMP). For Rabin-Karp, use \( q = 11 \).

2. (10 points) How would you extend the Rabin-Karp method to the problem of searching a text string for the occurrence of any one of a given set of \( k \) patterns? Start by assuming all patterns have the same length. The generalize your solution so that patterns can have different lengths.