# Project 6: Test Search Engine
# Cloud Computing
# Spring 2017

Professor Judy Qiu

## Goal

After having familiarized yourself with the "HBase Building an Inverted Index" homework and "PageRank algorithms" homework, you are ready to use these applications to test the search engine function from the packaged executable.

## Deliverables

Zip your source code, library, and results in a file named username@test-search-engine.zip. Please submit this file to the Canvas Assignments page.

## Evaluation

The point total for this project is 6, where the distribution is as follows:

- Completeness of your code (5 points)
- Correct output (1 points)

## Search Engine Implementation

Before we test the search engine, we need to write the PageRank output to the HBase clueWeb09PageRankTable.

```
$ export HADOOP_CLASSPATH='/root/software/hbase-0.94.7/bin/hbase classpath'
$ hadoop jar /root/software/hadoop-1.1.2/lib/cglHBaseMooc.jar iu.pti.hbaseapp.clueweb09.
    PageRankTableLoader /root/MoocHomeworks/HBaseInvertedIndexing/resources/en0000-01and02.
    docToNodeIdx.txt /root/MoocHomeworks/HBaseInvertedIndexing/resources/en0000-01
    and02_reset_idx_and_square_pagerank.out
```

Now, combined with "Building an Inverted Index", we have built three database tables on HBase:

- clueWeb09DataTable
- clueWeb09IndexTable
- clueWeb09PageRankTable

The data-flow of the program is shown in Figure 1.
You need to complete the following code before you can run the search engine:

```
$ vim src/iu/pti/hbaseapp/clueweb09/SearchEngineTester.java
```

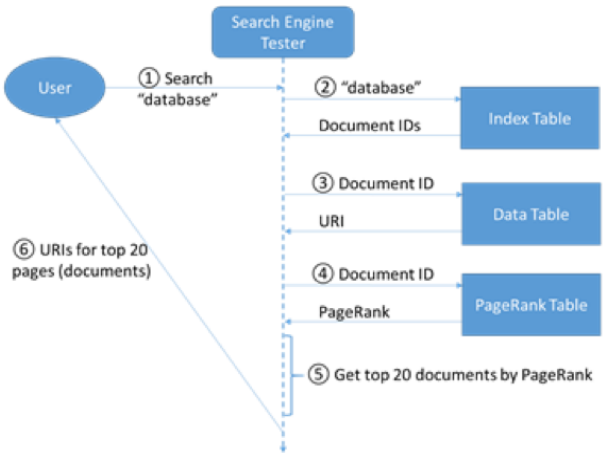


Figure 1: Dataflow for searching keyword “database” among the constructed databases

```
public static void searchKeyword(String keyword) throws Exception {
  Configuration hbaseConfig = HBaseConfiguration.create();
  HTable dataTable = new HTable(hbaseConfig, Constants.CW09_DATA_TABLE_BYTES);
  HTable indexTable = new HTable(hbaseConfig, Constants.CW09_INDEX_TABLE_BYTES);
  HTable prTable = new HTable(hbaseConfig, Constants.CW09_PAGERANK_TABLE_BYTES);

  int topCount = 20;
  // this is the heap for storing the top 20 ranked pages
  PriorityQueue<PageRecord> topPages = new PriorityQueue<PageRecord>(topCount);

  // get the inverted index row with the given keyword
  keyword = keyword.toLowerCase();
    byte[] keywordBytes = Bytes.toBytes(keyword);
  Get gIndex = new Get(keywordBytes);
    Result indexRow = indexTable.get(gIndex);

    // loop through the document IDs in the row. Recall the schema of the
    clueWeb09IndexTable:
    // row key: term (keyword), column family: "frequencies", qualifier: document ID, cell
    value: term frequency in the corresponding document
  int pageCount = 0;
    for (KeyValue kv : indexRow.list()) {
        String pageDocId = null;
        int freq = 0;
        String pageUri = null;
        float pageRank = 0;

      // Write your codes for the main part of implementation here
        // Step 1: get the document ID of one page, as well as the keyword's frequency in
    that page
    // Step 2: get the URI of the page from clueWeb09DataTable
    // Step 3: get the page rank value of this page from clueWeb09PageRankTable
      // End of your code

        // Use the heap to select the top 20 pages according to page rank
      PageRecord page = new PageRecord(pageDocId, pageUri, pageRank, freq);
      if (topPages.size() < topCount) {
      topPages.offer(page);
      }
      else {
      PageRecord head = topPages.peek();
      if (page.pageRank > head.pageRank) {
        topPages.poll();
        topPages.offer(page);
      }
```

```
      }

      pageCount++;
      if (pageCount % 100 == 0) {
      System.out.println("Evaluated " + pageCount + " pages.");
      }
  }
    System.out.println("Evaluated " + pageCount + " pages.");
  dataTable.close();
  indexTable.close();
  prTable.close();

  System.out.println("Evaluated " + pageCount + " pages in total. Here are the top 20 pages
    according to page ranks:");
  Stack<PageRecord> stack = new Stack<PageRecord>();
  while (topPages.size() > 0) {
    stack.push(topPages.poll());
  }
  while (stack.size() > 0) {
    PageRecord page = stack.pop();
    System.out.println("Document ID: " + page.docId + ", URI: " + page.URI + ", page rank: "
     + page.pageRank + ", word frequency: "
      + page.termFreq);
  }
}
```

## Compile and Run the Program

```
$ cd /root/MoocHomeworks/HBaseInvertedIndexing/
$ vim src/iu/pti/hbaseapp/clueweb09/SearchEngineTester.java
$ cd /root/MoocHomeworks/HBaseInvertedIndexing/
$ ant
$ cp /root/MoocHomeworks/HBaseInvertedIndexing/dist/lib/cglHBaseMooc.jar /root/software/
    hadoop-1.1.2/lib/
```

Now you can test the functionality of the search engine by running the program with keywords.

```
$ cd /root/software/hadoop-1.1.2/
$ ./bin/hadoop jar lib/cglHBaseMooc.jar  iu.pti.hbaseapp.clueweb09.SearchEngineTester search
    -keyword snapshot
$ ./bin/hadoop jar lib/cglHBaseMooc.jar  iu.pti.hbaseapp.clueweb09.SearchEngineTester get-
    page-snapshot 00000113548 |  grep snapshot
```

## What?s next?

Congratulations, you have finished the search engine project!