

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`.

```
1 $ export HADOOP_CLASSPATH='/root/software/hbase-0.94.7/bin/hbase_classpath'
2 $ hadoop jar /root/software/hadoop-1.1.2/lib/cglHBaseMoo.jar iu.pti.hbaseapp.clueweb09.
   PageRankTableLoader /root/MoochHomeworks/HBaseInvertedIndexing/resources/en0000-01and02.
   docToNodeIdx.txt /root/MoochHomeworks/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:

```
1 $ vim src/iu/pti/hbaseapp/clueweb09/SearchEngineTester.java
```

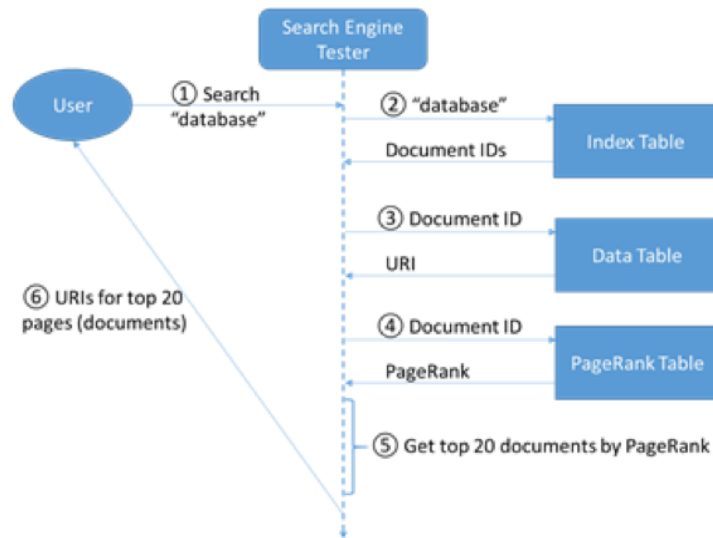


Figure 1: Dataflow for searching keyword “database” among the constructed databases

```

1 public static void searchKeyword(String keyword) throws Exception {
2     Configuration hbaseConfig = HBaseConfiguration.create();
3     HTable dataTable = new HTable(hbaseConfig, Constants.CW09_DATA_TABLE_BYTES);
4     HTable indexTable = new HTable(hbaseConfig, Constants.CW09_INDEX_TABLE_BYTES);
5     HTable prTable = new HTable(hbaseConfig, Constants.CW09_PAGERANK_TABLE_BYTES);
6
7     int topCount = 20;
8     // this is the heap for storing the top 20 ranked pages
9     PriorityQueue<PageRecord> topPages = new PriorityQueue<PageRecord>(topCount);
10
11     // get the inverted index row with the given keyword
12     keyword = keyword.toLowerCase();
13     byte[] keywordBytes = Bytes.toBytes(keyword);
14     Get gIndex = new Get(keywordBytes);
15     Result indexRow = indexTable.get(gIndex);
16
17     // loop through the document IDs in the row. Recall the schema of the
18     // clueWeb09IndexTable:
19     // row key: term (keyword), column family: "frequencies", qualifier: document ID, cell
20     // value: term frequency in the corresponding document
21     int pageCount = 0;
22     for (KeyValue kv : indexRow.list()) {
23         String pageDocId = null;
24         int freq = 0;
25         String pageUri = null;
26         float pageRank = 0;
27
28         // Write your codes for the main part of implementation here
29         // Step 1: get the document ID of one page, as well as the keyword's frequency in
30         // that page
31         // Step 2: get the URI of the page from clueWeb09DataTable
32         // Step 3: get the page rank value of this page from clueWeb09PageRankTable
33         // End of your code
34
35         // Use the heap to select the top 20 pages according to page rank
36         PageRecord page = new PageRecord(pageDocId, pageUri, pageRank, freq);
37         if (topPages.size() < topCount) {
38             topPages.offer(page);
39         }
40         else {
41             PageRecord head = topPages.peek();
42             if (page.pageRank > head.pageRank) {
43                 topPages.poll();
44                 topPages.offer(page);
45             }
46         }
47     }
48 }

```

```

43     }
44
45     pageCount++;
46     if (pageCount % 100 == 0) {
47         System.out.println("Evaluated " + pageCount + " pages.");
48     }
49 }
50 System.out.println("Evaluated " + pageCount + " pages.");
51 dataTable.close();
52 indexTable.close();
53 prTable.close();
54
55 System.out.println("Evaluated " + pageCount + " pages in total. Here are the top 20 pages
    according to page ranks:");
56 Stack<PageRecord> stack = new Stack<PageRecord>();
57 while (topPages.size() > 0) {
58     stack.push(topPages.poll());
59 }
60 while (stack.size() > 0) {
61     PageRecord page = stack.pop();
62     System.out.println("Document ID: " + page.docId + ", URI: " + page.URI + ", page rank: "
        + page.pageRank + ", word frequency: "
63         + page.termFreq);
64 }
65 }

```

Compile and Run the Program

```

1 $ cd /root/MoocHomeworks/HBaseInvertedIndexing/
2 $ vim src/iu/pti/hbaseapp/clueweb09/SearchEngineTester.java
3 $ cd /root/MoocHomeworks/HBaseInvertedIndexing/
4 $ ant
5 $ 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.

```

1 $ cd /root/software/hadoop-1.1.2/
2 $ ./bin/hadoop jar lib/cglHBaseMooc.jar iu.pti.hbaseapp.clueweb09.SearchEngineTester search
    -keyword snapshot
3 $ ./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!