CS 276 / LING 286 Spring 2013

Programming Assignment 1

Due: Thursday, April 18, 2013, 4 p.m. PDT

Overview

In this programming assignment, you will be applying knowledge that you have learned from Lecture 1-4 to build a simple indexing and retrieval system for a search engine. This assignment should be done in teams of two or individually. More specifically, it involves the following tasks:

1. Build an uncompressed index over a corpus of webpages from the stanford.edu domain (c.f. Sec. Corpus), and implement retrieval for Boolean conjunctive queries.
2. Build a compressed index over the same corpus, using variable length encoding and implement Boolean conjunctive queries.
3. Write up a report describing your program and answer a set of questions.
4. For extra credit, you are encouraged to implement other compression algorithms (e.g., gamma-encoding)

See the 'Submission' section for details on how to submit the assignment. The output of your code will be auto-graded so it is important to follow all the input/output instructions.

Corpus

The corpus you will be working with for this PA contains webpages from the stanford.edu domain. The data and starter code is available to download from the Coursera

website under the Programming Assigments section. The total size of the corpus is about 170MBs. There are 10 sub-directories (named 0-9) under the data directory. Each file under the sub-directory is the content of an individual webpage. You should assume that the individual file names are unique within each sub-directory, but not necessarily the case across sub-directories (i.e., the full path of the files are unique). All the HTML info has been stripped off from those webpages, so they only contain space-delimited words. You should not re-tokenize the words. Each consecutive span of non-space characters consists of a word token in the corpus.

Task 1: Building an uncompressed index and retrieval (50%)

The first part of this assignment is to build an inverted index of this corpus, and implement Boolean conjunctive queries. In particular, you need to implement the blocked sort-based indexing (BSBI) algorithm. You should treat each sub-directory as a block, and only load one block in memory at a time when you build your index (note: you will be penalized if you construct the index by loading blocks of larger than one directory in memory at once). Note that we are abstracting away from the operating systems meaning of blocks and by a block we mean a sub-directory

You can use any programming language that you are comfortable with. If Python is your choice of language, then you can find some sample skeleton code in the 'skeleton_code' directory inside the code provided to you.

All of your submitted code for this part of the PA should reside in a directory called "task1". Under "task1" directory, you must supply two shell scripts, named "index.sh" and "query.sh". Each script should invoke the necessary program to perform the indexing and retrieval tasks. Sample index.sh and query.sh scripts to invoke index and query Python programs can be found in the task1 directory. If you change these scripts, make sure that they can be invoked from the parent directory (i.e. "sh task1/index.sh *[input arguments - see below]*" and "sh task1/query.sh *[input arguments - see below]*" should work).

Indexing

The index.sh script should take two command-line arguments:

1. input_data_dir: a string valued argument, describing the directory location of the input corpus data

2. output_index_dir: a string valued argument. This is the location of the output directory containing the generated index. You should assume the output directory does not exist yet.

In addition, the index.sh script should **only** print the total number of files in the corpus to stdout. You can freely write to stderr for your own purpose, we will not be collecting stderr outputs.

At grading time, we will be invoking the index.sh script in the following way:
sh task1/index.sh input_data_dir output_index_dir

We will then collect timing statistics on how long the indexing takes, and the size of your generated index. We will also verify that the total number of documents you output matches the expected count.

Boolean conjunctive retrieval

The query.sh will take one command line argument, which is the directory location of the index you built. Your retrieval program should read from stdin a sequence of word tokens separated by space, which forms the Boolean conjunctive query. There could be any non-zero number of query terms. Your program should print to stdout **only** the list of documents containing the query terms, one document file name on each line, sorted in lexicographical order. Again, you are free to write anything to stderr. The document file name should only include the subdirectory name and the file name, e.g.,
0/crypto.stanford.edu_
0/crypto.stanford.edu_DRM2002_
1/crypto.stanford.edu_cs142_
1/crypto.stanford.edu_cs155_hw_and_proj_

If the conjunctive query has no results (could also be caused by any of the terms in the query is not being found in the corpus), your program should output "no results found" to stdout.

It is important to make sure that you are **not loading the whole index into memory**. The whole point of indexing is that you can seek to the corresponding index position of a query term on the disk, and just read into memory the posting list of that term, without having to load the full index.

You are required to **order the terms by postings list length** to optimize query performance. You will be penalized if you do not implement this optimization.

We will be invoking your script with the following command:
sh task1/query.sh index_dir <query.txt
where query.txt contains a list of query terms, all in one line.

Toy dataset

It is usually a good idea to verify your programs output on a small toy dataset first during development. We have provided you with a small toy dataset and a set of sample queries and outputs, under the 'toy_example' directory.

There is a script "grader.sh" under the same toy example directory. It is similar to the auto-grader script we use for final evaluation. You should be able run it to verify your program meets our input/output requirements.

Also, a set of development queries and their outputs are given at the 'dev_queries' and 'dev_output' directories.

Grading

To ensure your index is built correctly, you will be tested on 20 Boolean conjunctive queries of one or multiple terms. For those queries, you will get 1.5% of the final grade for each query you answer correctly, for a total of 30% of your grade. You will earn another 20% if none of the following penalties apply to your program. You will be penalized by

1. 5% if your program does not load one block at a time in indexing (see our notes at the beginning of Task 1 description).
2. 5% if you do not order the query terms by postings list length in retrieval
3. 4% if your retrieval program loads the whole index into memory rather than simply loading the postings lists of just the query terms.
4. 3% if the total number of files your indexer outputs does not match the correct count.
5. 3% if the timing statistics of your retrieval algorithm are way out of the norm.

The maximum combined grades for this task is 50%.

Task 2: Building a compressed index and retrieval (30%)

In the second task, you should build a compressed index using gap encoding with variable byte encoding for each gap. The input/output format and code structure is the same as Task 1, but your code should reside in directory "task2". We will also use an auto-grader to build a compressed index with your program, and run a set of queries. Similar to Task 1, we will be invoking your scripts with 'sh task2/index.sh [input arguments]' and 'sh task2/query.sh [input arguments]'

Grading

Similar to Task 1, you will be tested on 20 conjunctive queries (1% for each query you answer correctly, for a total of 20%). And you will earn another 10% if none of the following penalties apply:

1. 5% if your compression algorithm does not achieve a reduction in index size.
2. 5% if your query response time with the compressed index is way out of norm.

The combined total grade of this task is 30%. Note that if your program does not implement the variable length encoding compression algorithm, you will receive **no credit** for this task.

Task 3: Report (20%)

Please write up a 1-2 page report and submit alongside with your code. It should contain the following sections:

1. Briefly describe how your program is structured and the key steps in your indexing and retrieval algorithms. Make sure you report statistics on the size of the index (compressed and uncompressed), statistics of retrieval time for the development queries. (5%)

2. Answer each of the following questions: (5% each)

   a) In this PA we asked you to use each sub-directory as a block and build index for one block at a time. Can you discuss the tradeoff of different sizes of blocks? Is there a general strategy when we are working with limited memory but want to minimize indexing time?

   b) Is there a part of your indexing program that limits its scalability to larger datasets? Describe all the other parts of the indexing process that you can optimize for indexing time/scalability and retrieval time.

   c) Any more ideas of how to improve indexing or retrieval performance?

Grading

Your write-up will contribute to 20% of your final grade.

Extra credit: experiment with additional compression methods (15%)

Implement one more index compression method (e.g., gamma-encoding), and supply your code in the same structure as Task 1 and 2 under directory "extra_credit". Similar to Tasks 1 and 2, we will be invoking your scripts with 'sh extra_credit/index.sh *[input arguments]*' and 'sh extra_credit/query.sh *[input arguments]*'

Also include in the report a discussion of the space/speed tradeoff of this additional compression method, in comparison to Task 2 and uncompressed index.

Grading

You will be tested on 20 queries, you will receive 0.5% for each query you answer correctly. Rest of the 5% of grade will depend on the compression you are able to achieve over variable length encoding compression in Task 2. You will be penalized if the retrieval timings are abnormally long. We will inspect your code for implementation correctness if necessary.

Submission

We will use Coursera for grading and collecting the assigment. To submit the assignment, run 'python submit.py'. You need to submit each task individually. Before submitting, make sure to write your and your partner's (if any) SUNet ID in the 'people.txt' file (one ID in one line). The report should be in the same directory with name 'report.pdf'. Note that all file names are in lowercase. Make sure to check the Coursera website for the feedback for the tasks a short time after submitting them.

You also need to submit the code in a single .zip file, with filename identical to your and your partner's (if any) Stanford SUNetIDs concatenated by underscore. E.g., for students Jason Smith with Stanford ID jsmith and his partner Laura Page with Stanford ID lpage, their submission should be file jsmith_lpage.zip. If you are working alone on the assignment, the file name will just be [Your SUNetID].zip. The zipped file should be submitted via Coursera under Programming Assignments section. It should be submitted for the part 'Code' (use the Output Submission option and ignore the Additional Submission option). The unzipped jsmith_lpage.zip file should be a single directory 'jsmith_lpage' with the following directory structure:

```
jsmith_lpage
├── task1
│   ├── index.sh
│   ├── query.sh
│   └── <any additional files your program needs>
├── task2
│   ├── index.sh
│   ├── query.sh
│   └── <any additional files your program needs>
├── report.pdf
├── people.txt
└── extra_credit (optional)
    ├── index.sh
    ├── query.sh
    └── <any additional files your program needs>
```

Please make sure to follow the above directory structure.

Before you submit your assignment and code, it will be a good idea to run the 'grader.sh' script on the toy data set, and also make sure you get the same output on the development queries (under directory dev_queries) as the sample out (under directory dev_output). Note that the indexes for Task 1, Task 2 and Extra Credit have to be constructed from scratch (that is, they are not built upon indexes from other tasks). Also, remember that indexes in Task 2 and Extra Credit will need to be stored in binary format.

If the submission script gives an error, your code has not been submitted (regardless of the 'accepted' status message you get at the end.) If the starter code and/or submission script does not work on your shell/operating system, you can use the Stanford servers (for ex, corn.stanford.edu).

Make sure you reserve enough time for the submission script to run and submit the assignment (that is, do not wait till the last minute). Coursera will record a late day even if you submit 5 mins late.