CMPUT 291 Wi17 - File and Database Management

Mini-project 2 spec

CMPUT291 - Winter 2017
Mini Project II

(group project)

***Due**: April 3rd at 5pm*

Clarifications:

You are responsible for monitoring the course discussion forum in eclass and this section of the project specification for more details or clarifications. No clarification will be posted after*5pm on April 2nd*.

- **March 14**. Each project group can have 2-3 members.

Introduction

The goal of this project is to teach the concept of working with data in the physical layer. This is done by building an information retrieval system, using the Berkeley DB library for operating on files and indices. Your job in this project is to write programs that keep data in files and maintain indices that provide basic searches over data. 80% of the project mark would be assigned to your implementation which would be assessed in a demo session, and is further broken down to three phases with 10% of the mark allocated for Phase 1, 5% of the mark for Phase 2, and 65% for Phase 3. Another 15% of the mark will be assigned for the documentation and quality of your source code and for your design document. 5% of the mark is assigned for your project task break-down and your group coordination.

Group work policy

You will be doing this project with one other partner from the 291 class. Your group information from mini-project 1 is copied to mini-project 2 groups page on the assumption that you would be working in the same group. If you decide to change groups, please notify the instructor. It is assumed that both group members contribute somewhat equally to the project, hence they would receive the same mark. In case of difficulties within a group and when a partner is not lifting his/her weight, make sure to document all your contributions. If there is a break-up, each group member will get credit only for his/her portion of the work completed (losing the mark for any work either not completed or completed by the partner). For the same reason, a break-up should be your last resort.

Task

You are given a data file, which you will use to construct your indices. Here is a small data file with only 10 records and here is one with 1000 records. The data includes tweets from the twitter network; each tweet record consists of an id, a date when the tweet is posted, the tweet text, and name, location, description and url of the poster. The records are formatted in xml and have a start tag <status> and and an end tag </status>; each record has a newline character at the end. Fields inside records are also formatted similarly with respective tags. Your job is to create indices, following Phases 1 and 2, and use those indices to process the queries in Phases 3.

Phase 1: Preparing Data Files

Write a program that reads tweets records in an xml format from standard input and produces 3 files as follows:

terms.txt: This file includes terms extracted from tweet text, name and location; for our purpose, suppose a term is a consecutive sequence of alphanumeric or underscore '_' characters, i.e *[0-9a-zA-Z_]*. The format of the file is as follows: for every term *T* in the text of a tweet with id *I*, there is a row in this file of the form *t-T':I* where *T'* is the lowercase form of *T*. For every term *T* in the name field of a tweet with id *I*, there is a row in this file of the form *n-T':I*, and for every term *T* in the location field of a tweet with id *I*, there is a row in this file of the form *l-T':I*. Ignore all special characters coded as*&#number;* such as &#29987; which represents産. Also ignore terms of length 2 or less. Also you notice that all terms are turned to lowercase before being written out. Here are the respective files for our input files with 10 records and 1000 records.

dates.txt: This file includes one line for each tweet in the form of *d:I* where d is the date at which the tweet is created and *I* is the tweet id. Here are the respective files for our input files with 10 records and 1000 records.

tweets.txt: This file includes one line for each tweet in the form of *I:rec* where *I* is the tweet id and rec is the full tweet record in xml. Here are the respective files for our input files with 10 records and 1000 records.

These files can also be found at directory*~drafiei/291/pub* on the lab machines. In the same directory, you would also find larger size files (with 10k records) that you may want to use in the testings of your programs.

Phase 2: Building Indexes

Sort the files built in Phase 1 using the Linux *sort* command; pass the right option to the sort command to keep only the unique rows (see the man page for sort). You can keep the sorted data under the same file names or pass sorted records to stdout so they can be piped to your loading program (as described next). Suppose the sorted files are named as before (to simplify our presentation here). Given the sorted files *terms.txt*, *dates.txt* and *tweets.txt*, create the following three indexes: (1) a hash index on tweets.txt with tweet ids as keys and the full tweet record as data, (2) a B+-tree index on terms.txt with terms as keys and tweet ids as data, (3) a B+-tree index on dates.txt with dates as keys and tweet ids as data. You should note that the keys in all three cases are the character strings before colon ':' and the data is everything that comes after the colon. Use the *db_load* command to build your indexes. db_load by default expects keys in one line and data in the next line. Also db_load treats backslash as a special character and you want to avoid backslash in your input. Here is a simple Perl script that converts input records into what db_load expects and also removes backslashes. Your program for

Phase 2 would produces three indexes which should be named *tw.idx*, *te.idx* and *da.idx* respectively corresponding to indexes 1, 2, and 3, as discussed above.

In addition to *db_load*, you may also find *db_dump* with option p useful as you are building and testing the correctness of your indexes.

Phase 3: Data Retrieval

Given the index files *tw.idx*, *te.idx*, and *da.idx* created in Phase 2 respectively on tweet ids, terms and dates, write a program that processes queries as follows. Each query returns the full record of the matching tweets, with record id first, followed by the rest of the fields formatted for output display, which should be readable and not in xml. Here are some examples of queries:

1. *text:german*
2. *name:german*
3. *location:german*
4. *german*
5. *text:german%*
6. *date:2011/01/01*
7. *date>2011/01/01*
8. *date<2011/01/01*
9. *text:german date>2011/01/01*

The first query returns all record that have *german* in their tweet text; similarly the second and the third queries return all records that have the term in their name and location fields respectively. The fourth query returns all records that have the term in tweet text, name or location fields. The fifth query returns all records that have a term starting with *german* in their tweet text. The sixth, seventh and eight queries respectively return tweet records that are created on, after, and before 2011/01/01. The ninth query returns all records that have the term german in tweet text and the tweet record is created after 2011/01/01.

More formally, each query defines some conditions that must be satisfied by text, name, location and created_at fields of the matching records. A condition can be either an exact match or a partial match; for simplicity, partial matches are restricted to prefix matches only (i.e. the wild card % can only appear at the end of a term). All matches are case-insensitive, hence the queries "German", "german", "gErman" would retrieve the same results; for the same reason the extracted terms in previous phases are all stored in lowercase. Matches on dates can be exact (as in query 6) or range searches (as in queries 7 and 8). Matches on terms can be exact (as in queries 1-4) or partial (as in query 5). A query can have multiple conditions (as in query 9) in which case the result must match all those conditions. Here is a grammar for the queries.

Testing

At demo time, your code will be tested under a TA account. You will be given the name of a data file, and will be asked (1) to prepare terms.txt, dates.txt, tweets.txt, (2) build Berkeley DB indexes tw.idx, te.idx and da.idx, and (3) provide a query interface, which will allow us to test your system for Phase 3. We typically follow a 5 minutes rule, meaning that you are expected to prepare the data and build your indices in Phases 1 and 2 in less than 5 minutes; if not, you may lose marks and we may have to use our own indexes, in which case you would lose the whole mark for Phases 1 and 2.
The demo will be run using the source code submitted and nothing else. Make sure your submission includes every file that is needed. There will be a limited time for each demo. Every group will book a time slot convenient to all group members to demo their projects. At demo time, all group members

must be present. The TA will be asking you to perform various tasks and show how your application is handling each task. A mark will be assigned to your demo on the spot after the testing.

Important note: You can make no assumption on the size of the input file (and any of the files and indexes that are built from the input file). We will be using a relatively large file for testing and you don't want your program to break down in our tests, resulting in a poor mark! That said, you can make very few assumptions (such as the inverted list of a term can fit in main memory) and state them in your report.

Instructions for Submissions

Your submission includes (1) the source code for phases 1, 2 and 3 and any makefile or script that you may need to compile your code, and (2) a short report. Your source code must include at least three programs, i.e. one for each phase. Your program for Phase 3 would implement a simple query interface either in your favorite programming language (e.g. Python, C or C++, Java). For phases 1 and 2, you can have more than one program (for example, in Python, C or C++, Java, Perl) and can make use of any Unix command and scripting language (e.g. Perl, bash) that runs under Linux on lab machines, as long as you clearly document in your report how and in what sequence the programs or commands should run. The source code is submitted as follows:

- Create a single gzipped tar file with all your source code and additional files you may need for your demo. Name the file *prj2code.tgz*.
- Submit your project tarfile in the project submission site by the due date at the top of this page.

Your report must be type-written and submitted in hardcopy at the designated drop boxes located on the first floor of CSC building, across from the room 1-45 before the due date. Your report cannot exceed 3 pages in length plus one additional cover page.**Your project report is due on Thursday April 6th, at 10am,** and no late submission is allowed for the project report. **All project members are expected to submit their code online but only one copy of the project report will be submitted.**

The report should include (a) a general overview of your system with a small user guide, (b) a description of your algorithm for evaluating queries, in particular evaluating queries with multiple conditions and wild cards and range searches and an analysis of the efficiency of your algorithm, (c) your testing strategy, and (d) your group work break-down strategy. The general overview of the system gives a high level introduction and may include a diagram showing the flow of data between different components; this can be useful for both users and developers of your application. The user guide should have instructions for running your code for phases 1, 2 and 3. The testing strategy discusses your general strategy for testing, with the scenarios being tested and the coverage of your test cases. The group work strategy must list the break-down of the work items among partners, both the time spent (an estimate) and the progress made by each partner, and your method of coordination to keep the project on track. The design document should also include any assumption you have made or any possible limitations your code may have.

Last modified: Wednesday, 5 April 2017, 8:30 AM