

Mini-project 2 Report

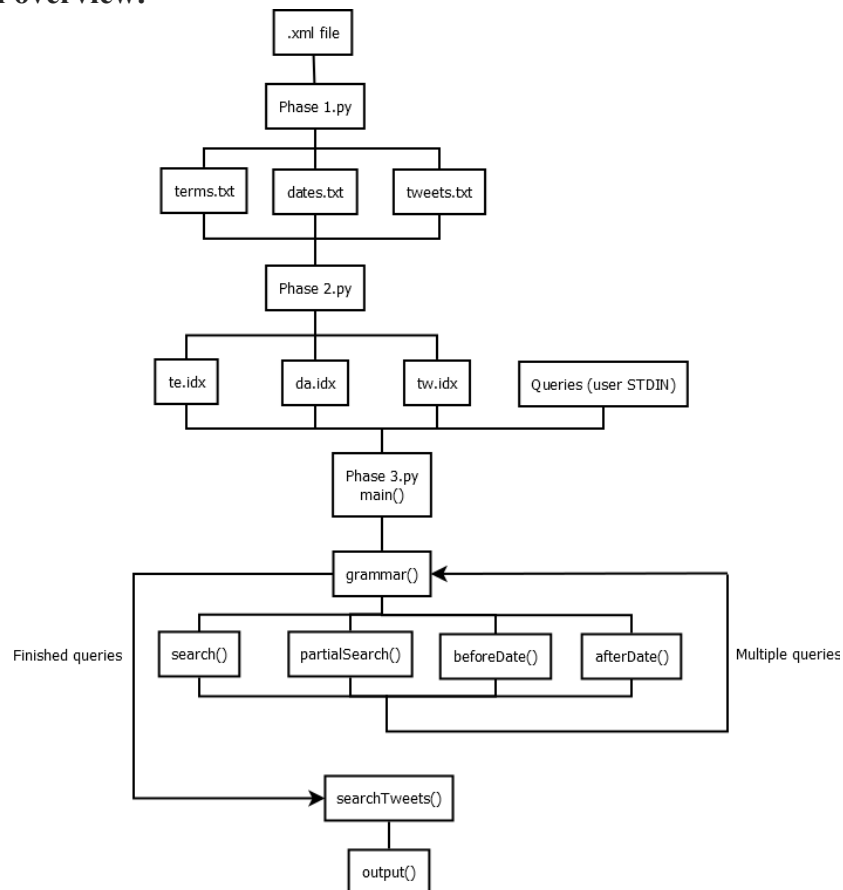
Members:

Mingwei Li

Shuyang Li

Xuanyi Wu

(a) A general overview:



We divide the whole program into 3 parts, one part for each phase, and write each part in Python 3.

phase1.py:

1. Read the files;
2. handle each term by using regular expression;
3. ignore special characters and terms with length less than or equal to 2;
4. save the qualified data to terms.txt, dates.txt, tweets.txt respectively.

phase2.py:

1. Use Linux sort command 'sort -u...' to sort the three txt files generated in phase1 without duplicate key-data pairs; 2.
2. use 'perl break.pl ...' to get the ideal format for db_load;
3. use 'db_load -c duplicates=1...' command to get the .idx files (allow duplicate keys).
4. use 'db_dump' to check the correctness of db_load

phase3.py:

provide a query interface and return the results

User Guide:

Format 1, user enter ____:____

This query returns exact match results

The previous blank can be filled in with: text or name or location or date; the second blank is the keyword(format year/month/date for the search of date). The program searches terms in text, name, location, date respectively.

Format 2: user enter ____:____%

This query returns partial match results

The previous blank can be filled in with: text or name or location; the second blank is the keyword. The program searches terms in text, name, location, respectively.

Format 3: user enter _____

The blank can be filled in with any keyword, and the program returns exact match results by searching terms in text, name and location

Format 4: user enter _____%

The blank can be filled in with any keyword, and the program returns partial match results by searching terms in text, name and location

Format 5: user enter _____date>_____

It is a range search, the blank can be filled in with any date with format year/month/date, and returns the tweets posted after that date.

Format 6: user enter _____date<_____

It is a range search, similar with format 5 but returns the tweets posted before that date.

Format 7: user enter _____ (query 1) (query 2) ... (query n)

Multiple search, user can enter any combination of the previous 6 queries separated by blank. It returns the tweets satisfy all queries.

(b) A description of your algorithm for evaluating queries

1. load the three .idx files to database
2. use split(' ') to split the queries (in grammar function)
3. for each query, decide it is what kind of search (in grammar function)
4. exact search : (call search(termPrefix, term) function)
 - for Format 1: termPrefix is '-t' or '-n' or '-l' or empty for date
 - for Format 3: termPrefix is '-t' and '-n' and '-l' and union all results
 - 4.1. encode the termPrefix+term into the key
 - 4.2 use curs.get(key, db.DB_SET) to get the first qualified record
 - 4.3 recursively use curs.get(key, db.DB_NEXT_DUP) to get all qualified records
 - 4.4 if the id is not in the result query, append it into the result query
- 5.partial search(wild cards): (call partialSearch(termPrefix, term) function)
 - for Format 2: termPrefix is '-t' or '-n' or '-l'
 - for Format 4: termPrefix is '-t' and '-n' and '-l' and union all results
 - 5.1. encode the termPrefix+term into the key
 - 5.2 use curs.get(key, db.DB_RANGE) to get the first candidate record
 - 5.3 recursively use curs.get(key, db.DB_NEXT_DUP) to get all candidate records
 - 5.4 for each candidate, use result[0].startswith(key) to check whether it is a prefix
 - 5.5 if it is and the id is not in the result query, append it into the result query
- 6.range search:
 - 6.1 encode the date; search in da.idx
 - 6.2 use daCurs.get(date, db.DB_SET_RANGE) get the first candidate record
 - 6.3 for Format 5 (call afterDate(date) function)
 - when the date != the date we entered, use daCurs.get(date, db.DB_NEXT)
 - recursively to get all records after that date (we can do that since the file ins sorted)
 - 6.4 for Format 5 (call beforeDate(date) function)
 - when the date != the date we entered, use daCurs.get(date, db.DB_PREV)
 - recursively to get all records before that date (we can do that since the the file ins sorted)
 - 6.5 append the ids to the result list
- 7.multiple search:

since for each query we get a result list, we intersect all the result lists to get the result for multiple search (i.e. every time we get a new result list, we do
`tids=list(set(tids).intersection(newTids))`)

8. get the full record (call `searchTweets(tids)` function)
 - 8.1 traverse list `tids` (a list with all qualified `tids`); search the `ids` in `tw.idx`
 - 8.2 append the matching records in the result list
9. make it readable (call `output(line)` function)
 use regular expression again to get the readable format and print them.

An analysis of the efficiency of your algorithm:

The time efficiency of search algorithm depends on two parts: the efficiency of B+ tree and the efficiency of hash. We first search each query in B+ tree (`te.idx` and `da.idx`) which costs $O(\log_f N)$ (f is fan-out, N is the number of leaf pages). Then we use the results from last step to find intersection of them (final result `tids`), and use each final result to search in the hash (`tw.idx`) which costs $O(1)$. Therefore, the time efficiency is:

$$C * O(\log_f N) + R * O(1) = O(\log_f N)$$

where C is the number of queries, R is the number of result.

The space efficiency also depends on the space efficiency of B+ trees and the hash. We assume in `te.idx`, `da.idx`, and `tw.idx`, the largest number of records is n . Therefore, the space efficiency is:

$$3 * O(n)$$

(c) Testing Strategies

On phase 1, we run our code with `1k.txt` and use `diff` to compare our result with the correct result offered in spec. Then we analyze the difference and revise our code.

On phase 2, we use `db_dump -p` to export our result in readable file. Then we use `diff` to compare the index file with the sorted and file which is processed by `break.pl` to make sure every record is in the index file.

On phase 3, we test our code by input some critical queries, such as finding records before a non-existing date. We also imported the `.xml` file into Excel as a table, and compare the query result with the excel filter result which can improve our testing efficiency.

(d) Group Work Break-down Strategy

For this project, we didn't have too much code to write, so we emphasized on discussing how to implement the project and test several simple cases to make sure this method does work.

We discussed several ways for phase 1 breaking the `.xml` file down to 3 `.txt` files. Finally, we choose to use regex which is the easiest and fastest way, and Shuyang Li individually spent some time to figure out how to write correct regex.

We discussed about phase 2 together and searched some ways to run Unix command in python and implement it in a short time.

For phase 3, we discussed how to analyze the grammar of query. Mingwei Li wrote some parts of the code, and then Shuyang Li and Xuanyi Wu finished the rest part.

We tested the code separately and reported the bug we found to each other via email and IM software.

Work time:

Mingwei: 12 hours

Shuyang Li: 12 hours

Xuanyi Wu: 11 hours