DSA Report

[TOC]

Data Structures and Comparison

AA Tree

AA tree is a data structure modified by red black tree.
It's implementation detail is in
http://akira.ruc.dk/~keld/teaching/algoritmedesign_f03/Artikler/03/Andersson93.pdf

How to use AA Tree in the project First, we set up 5 AA trees, which is:

	fromSet	*toSet*	*idSet*	*wordSet*	*LengthSet*
typename of each element types in the pair	*fromPair* <**From**, **ids**>	*toPair* <**To**, **ids**>	*idPair* <**id**,**mail**>	*wordPair* <**word**, **ids**>	*lengthPair* <**len**, **ids**>

(We compare the elements by the first type of the pair)

From	**To**	**ids**	**mail**	**word**	**len**
string	string	AA<int>	AA<obj>	string	int

(obj is a class containing all information of a mail)

The AAtrees above in program-liked style can be view as:

Second, we need to perform the task. To perform add :

1. Use $from(to)Pair$ to contain the From(To) and ID label of the mail, then add it to $from(to)Set$
2. Use *id* to contain the id of mail and *mailContent* to contain the imformation of the mail, then paired into *idPair* and add into *idSet*.
3. For every word in content, we find if it is in *wordSet* first. If it was found, we just add the id into *wordElement*. Otherwise, we need to add *wordPair* into *wordSet* after adding the id into *wordPair*.
4. If we find the length in *lengthSet*, then add id in $lengthIDSet$. Otherwise, the add length in *lengthSet*, then add id in $lengthIDSet$.

To perform `remove` : 1. Use id to find the content of the mail, which is in *wordSet*. Then we have from, to, content , id label and length of the mail. 2. Use $from(to)Pair$ to contain the From(To) and ID label of the mail, then delete it in $from(to)Set$ 3. Use *id* to contain the id of mail and *wordElement* to contain the words of the mail, then paired into *idPair* and delete it in *idSet*. 4. For every word in content, we find if it is in *wordSet* first. If it was not found or the size of *wordPair* is 1, we just delete the id in *wordPair*. Otherwise, we need to delete *wordPair* in *wordSet*. 5. If we find the length in *lengthSet* and the size of it >= 1, then delete id in $lengthIDSet$. Otherwise, delete length in $lengthIDSet$.

To perform `query` : 1. The expression is composed by words. The from, to, word parameter is just $fromSet$, *toSet* and *wordSet*, which is mentioned in the first part. Take union, intersection and difference of them, then we can get the new set, which is the ids of the mail.

To perform `longest` : Find the largest element in *lengthSet*.

Time Complexity Let number of mails in the dataset be M , the word(unique) of all mails to be **W**, the word of one mail to be M_w, the number of mails contain the keyword to be W_m. The time complexity of `add` and `remove` is $O(logM + W_m(logW + logM_w))$ The time complexity of `query` is $O(M)$ The time complexity of `logest` is $O(logM)$

Trie

Implementation https://en.wikipedia.org/wiki/Trie#References

How to Use We can use a trie, say T, to represent a mapping from a keyword to a set of of id. To achieve this, we store them in an container under each node of the trie. Note that it cannot support the function `longest`. To perform `add` and `remove`, we scan through all keywords in the mail. For each keword K: - If we can find K in T, then - insert the id into the container under the node (for `insert`) - remove the id from the container under the node (for `remove`) - If K is not in T, then by insert K into T, - we will create a new node (and the container under it) in T. Then we insert the id into the container. (for `insert`) - throw an error since it should be in T (for `remove`)

To perform `query`, we can use T and the `<Expression>` to find the set S_q of all id that may meet the requirment of `query`. Since each keyword K in the `<expression>` can be map into a set S_K, which contans mail id. Then by take union, intersection and difference of them and a set of S of all id , we can get S_q. Then by checking the information of those mails, we can filter out the mails that does not meet the requirments.

Time Complexity Let the time complexity to find the information (from ,to. . .) of mail from an id be $O(M)$, and from a file path be $O(F)$. Let the length of a keyword K be L_K.

Time complexity of the container under a node: (N is the size of the container) | | vector | blanced binary tree | |-|-|-| | insert: $I \subseteq$ | $O(1)$ | $O(\log N)$ | | remove: $R \subseteq$ | $O(N)$ | $O(\log N)$ | | iterate all entry: $V \subseteq$ | $O(N)$ | $O(N)$ | Time complexity of using trie to perfrom the three operations: | | add | remove | query | ——— | ——— | ——— | ——- | | general | $O(F+L_K+I)$ | $O(M+\sum_{K\in mail}(L_K+R_K))$ | $O(\sum_{K\in expr}L_K+V_K))$ | | worst | $O(F+\max\{L_K\}+I)$ | $O(M+\max\{\sum(L_K+R_K)\})$ | $O(\max\{\sum(L_K+V_K)\})$

Our Implementation

Structure

per mail: - `MailForSearch`: storing id, length, date, from, to, and most importantly, a hashed map `contents` - `contents`: A hashed map whose values are keywords from the mail's content and subject and key are their hashed value

global: - `mails`: Red-Black tree mapping from mail id to `MailForSearch` instance for each mail - `mail_by_from`: Hashed map from "from"(string) to mail ids(int array) that are from that person - `mail_by_to`: Hashed map from "to"(string) to mail ids(int array) that are to that person - `mail_by_date`: Hashed map from "date"(int) to mail ids(int array) that are to of that day - `mail_by_length` : Red-Black tree that map from "length"(int) to mail ids(int array) of that length

Operation Time Analyses

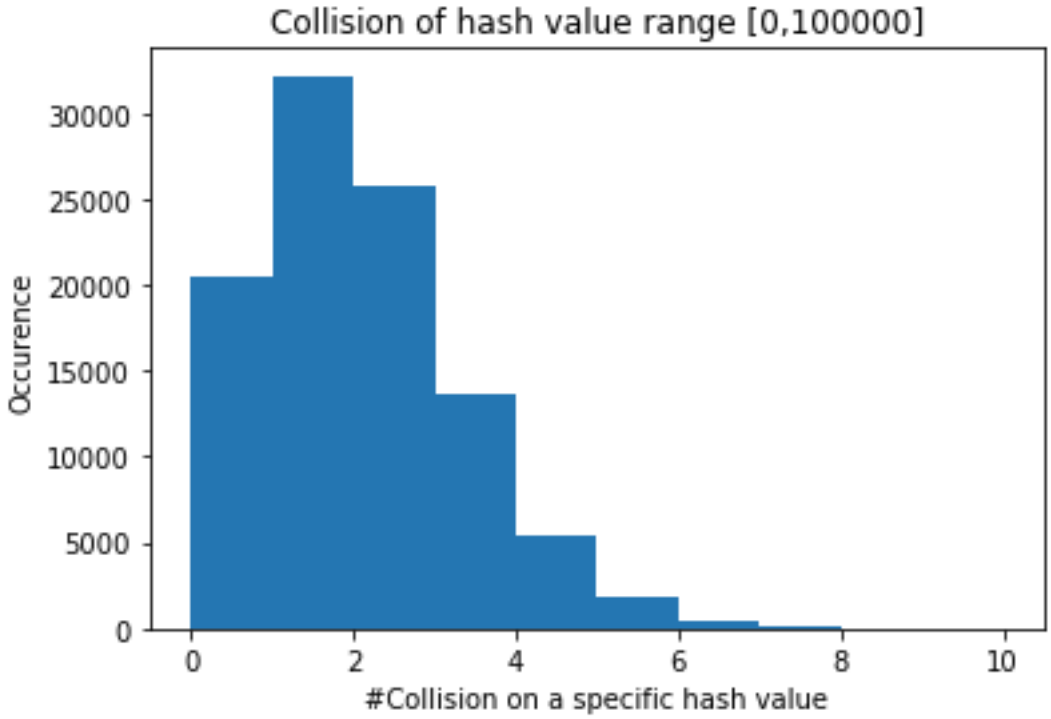
- add : Create a `MailForSearch` instance, and its content are split into keywords and added to `contents`; time: $O(K)$ where K is #keyword of the mail
- remove : Remove an entry from all global data structures; time: $O(\log N)$ for Red-Black tree and $O(1)$ for Hashed map
- longest : $O(1)$ for extracting largest entry from an binary tree

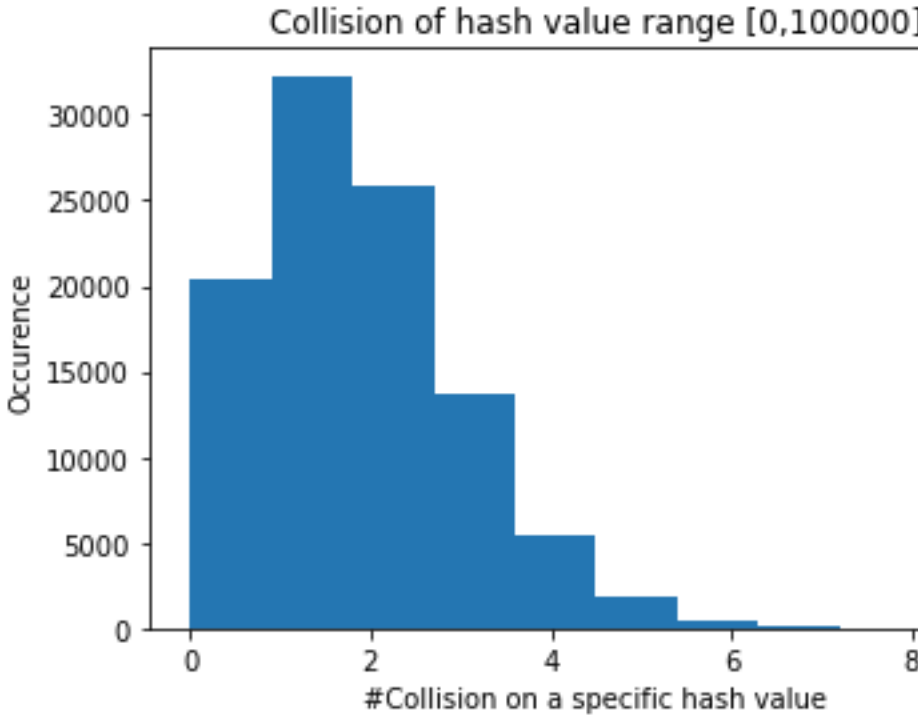
- query
  - initialize a list L containing all mails
  - query string has -d<fromdate>~<todate>: Lookup `mail_by_date` to get a mail id list within the date range, then intersect with L
  - query string has -f"from" or -t"to": Lookup `mail_by_from` or `mail_by_to` to get a mail id list, then intersect with L
  - expression: for each mail in L, map each keyword in expression to a boolean value indicating whether the mail has the keyword, then evaluate the mapped expression; . time: Lookup `mails` by ids in L: $O(|L| \log N) = O(N \log N)$ where N is #mails. In pracice, if "from" or "to" is specified, $|L|$ seldom exceeds $N/2$

Our Effort on Efficiency

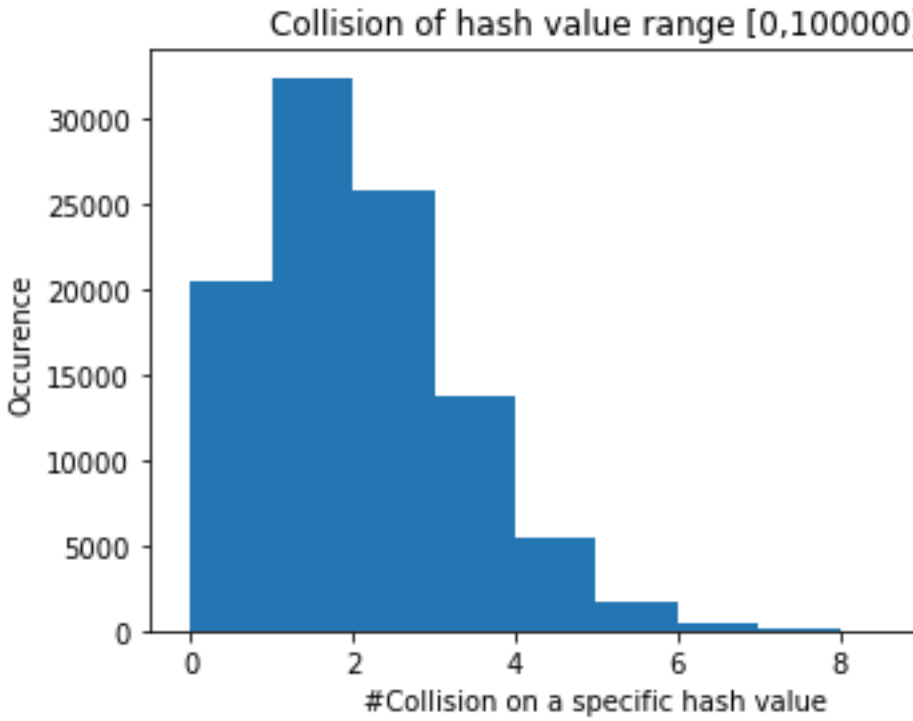
- Unbuffered and Asynchronous I/O Unbuffered I/O saves the time copying from LIBC buffer to main program Asynchronous I/O means that in `add()` our program can update `content` of a new mail (inserting keywords into hashed map) while not blocked by reading the mail file if the file is large
- `mail_by_from` and `mail_by_to` : very useful, about 1/2 runtime since a person will have a limited occurence in sender/receiver of all mails
- `mail_by_date`: only brings efficiency if complete range (from~to) rather than half-open range (from~ or ~to) is specified
- query keyword cache We've attempted to cache queries (mail id, keyword, result=0,1) into a small Red-Black tree to avoid two large lookup: lookup in large Red-Black tree `mails` and lookup in hashed map `content`). But that don't make much effieciency improvement on judge system since if the input query is randomized, the hit rate is too small, while the overhead to lookup the cache exists. If we store the keyword hash instead of keyword in cache, the efficiency may improve more, but in the cost of losing accuracy due to collision. In real world, we expect it to work because, say, a student taking DSA course in NTU may frequently searching his mails for TA's announcement by typing "DSA" as keyword, and that will hit the cache.
- trying different string hash functions
  - murmurhash2 This is default in g++
  - murmurhash3
  - cityhash
  - siphash
  - gperf perfect hash, only applicable to static keys but not in our case

Testing collision using all keywords in mail set: std::hash : (collision rate = 0.47317)





MurmurHash2: (collision rate = 0.47275)



MurmurHash3: (collision rate = 0.47231)

Runtime on our own input (10000): std::hash : 3.81s MurmurHash2: 3.85s MurmurHash3: 4.16s

There are also other hash fucnctions:

Hash function	collision rate	how many minutes to finish
MurmurHash3	6.?%	4m15s
Jenkins One..	6.1%	6m54s
Bob, 1st in link	6.16%	5m34s
SuperFastHash	10%	4m58s
bernstein	20%	14s only finish 1/20
one_at_a_time	6.16%	7m5s
crc	6.16%	7m56s

Bonus Features

We provide an GUI to make users search mails in a more convenient way. We also use python to perform the functions of sending and retrieving emails through gmail server (by SMTP and IMAP protocols). In addition, we use linux named pipe to handle the communication bwtween the to python script and the C++ GUI. (see GUI part below).

Send / Recieve Email

The user need to turn on **Access to less secure apps** and **IMAP** of his/her google account to enable these two features.

Main Functions: ++Send Mail++:

```
def send_msg(account, password, dir_path):
```

1. Login gmail with `account` and `password`
2. Receive user input on GUI and contstruct message by `construct_msg()`
3. Send the mail and let smtp.gmail.com to forward mails
4. Save a mail file under `dir_path` formatted by the spec of this homework.

++Load mails from gmail++:

```
def read_gmail(account, password, dir_path, startID):
```

1. Login gmail with `account` and `password`
2. Receive user input on GUI to determine how many mails to be load, said N.
3. Load N mails from `account` to local client and formate them by the spec of this homework
4. Save the loaded mail file under `dir_path`

Restriction The format of email that we are able to send is restricted. It can only contains following items:

- From: the account of user's gmail
- To: a valid <mail address>
- Subject: <strings>
- Content: <strings>, cannot contain any attachment file

Name Pipe for Communication Our shell script creates four name pipes to forward the signals for communication of python script and c++ GUI. - Python -> `/tmp/py_out_pipe` -> `/tmp/qt_in_pipe` -> C++ GUI - C++ GUI -> `/tmp/qt_out_pipe` -> `/tmp/py_in_pipe` -> Python ### GUI

We use Qt creator to devolop a GUI which can actually send email(ex: gmail) and achieve the functions the homework required. We use C++ languange base on OOP to write code.

Features ++Login Interface++ When the user run this program, the first thing they need to do is login to the gmail. The accountInputBox and passwordInputBox objects will get the words user entered. When the user click login button, the program will communicate to the python program above and check if it is login successfully. If the user log in successfully, they will go into main menu page. Otherwise, a message box will jump out and tell you "login failed". When th user click exit button, the program will just closed.

++Main Menu Interface++ If the user login successfully, they will go into this page. When the user clicked reload button, the program will call the python function and load gmail into database. When the user clicked longest button, they will see the mail with longest content. If there isn't any mail in the data base, the ID label and length label will show "-1". After the user login, the only way to close this program gracefully is to click the exit button, and the button is only appears here. Namely, they must back to the main page if they want to stop the program. If the user click other buttons, they will go into the interfaces below.

++Send Email Interface++ When the user click send email button, they will go into this page. The user must to type something in the label of "To, Subject and Content" so that they can send email. When the user click the send button, the program will do: (1) Check if they do type something in the label of "To, Subject and Content", or else they will see a message box jump out and says "Must fill in to, subject and content". (2) Call the python code above. If it fail to send the email, they will see the message box contains "cannot send email".

++Add Email to Database Interface++ When the user click add email button, they will go into this page. The difference between add email and send email is that: Send email really send gmail to the people you want to sent, and they will see a mail when they open their gmail page. Add email just add an email which you load from reload function in the menu page to the datbase. When the user click

the add button, they will see how many number of mails are in the database if they add successfully. Otherwise, they will see “-1” in the mail num.

++Search Email with keyword Interface++ When the user click query button, they will go into this page. When the user clicked search button, the program will check if the user fill in the expression field. If not, they will see “type message” in the message box. If they did not find anything,they will just show nothing in the result id field.

Reference

- https://github.com/gcc-mirror/gcc/blob/master/libstdc++-v3/libsupc++/hash_bytes.cc
- https://sites.google.com/site/murmurhash/
- http://code.google.com/p/smhasher/
- https://github.com/google/cityhash
- https://stackoverflow.com/questions/7666509/hash-function-for-string/45641002#45641002