Data Structure and Algorithm, Spring 2021
Final Project - Email Searcher 2.0

Competition & Report Due: 23:59:59, Tuesday, June 22, 2021

TA E-mail: dsa_ta@csie.ntu.edu.tw

Rules and Instructions

- This document describes the theme of final project. A more detailed spec will be released later.
- Any form of cheating, lying, or plagiarism will not be tolerated. Students can get zero scores and/or fail the class and/or be kicked out of school and/or receive other punishments for those kinds of misconducts.
- In this Final Project, you are required to team up and participate a programming competition on *DSA Judge* (https://dsa-2021.csie.org). You are also required to upload a report about your implementation and analysis to *Gradescope* (https://www.gradescope.com).
- By default, you are asked to work as a team of size three. A one-person or two-people team is allowed only if you are willing to be as good as a three-people team. It is expected that all team members share balanced work loads. Any form of unfairness in a three-people team, such as the intention to cover the other member's work, is considered a violation of the honesty policy and will cause both members to receive zero or negative score.
- In the programming competition, each team will have 5 quotas per day to submit your programs in the C language to the judge system. The score of your submission will be evaluated and shown right after your submission is judged. We will score your programming part depending on your rankings on the scoreboard (see Grading section below).
- The PDF file for the report part should be submitted to Gradescope before the deadline. Unless granted by the instructor in advance, no late submissions will be allowed.
- If you have questions about the Final Project, please go to the course forum and discuss (**strongly preferred**, which will provide everyone a better learning experience). If you really need an email answer, please follow the rules outlined below to get a fast response:
  - The subject should contain `"[Final Project]"`. For example, `"[Final Project] Failed to compile the test environment in my environment"`. Adding the

tag allows the TAs to track the status of each email and to provide faster responses to you.

  - If you want to provide your code segments to us as part of your question, please upload it to Gist or similar platforms and provide the link. Screenshots or code segments directly included in the email are discouraged and may not be reviewed.

- You are allowed to use the functionalities provided by the C Standard Libraries. But we ban several system calls that can cause unfairness, such as `fork`. Those calls will result in `Security Error`.

- Discussions are encouraged, but you should write the final solutions alone and understand them fully. In order to maximize the level of fairness in this class, lending and borrowing solutions to other team are both regarded as dishonest behaviors and will be punished according to the honesty policy. Books, notes, and Internet resources can be consulted, but not copied from. That is, **you cannot copy the code that you find online.** In order to maximize the level of fairness in this class, lending and borrowing solutions to other team are both regarded as dishonest behaviors and will be punished according to the honesty policy.

- If you found some security vulnerabilities (e.g. ways to bypass the timing mechanism) in the final project. Please report the vulnerabilities to the TAs (and only to the TAs) by email immediately. This helps us fix the issue as soon as possible. We surely will not punish you and will reward your honesty in some way.

Introduction

The main theme of this project is an upgraded version of email searcher (the theme of last year). The upgraded email searcher helps retrieve precious mails and perform analysis on mails in a distributed way, which is done with the help of many workers. These workers fetches mails and queries from the server, and reply the answers to get rewards within a limited time. In this final project, you need to develop a efficient worker to earn the points as much as you can.

Problem Description

In this project, you will be given a set of emails (will be made public) and many queries regarding to the contents of the mails, each has its own reward. Your job is to develop a strategy which helps you earn the most points, either by solving as many queries as possible, or by cleverly choosing the queries to solve. You may answer queries in any order you like. The three types of queries you may encounter are:

- Expression-Match(Expression, Constraints)
  In this type of query, you will be given an expression and other optional constraints. You need to answer all the mail IDs that match the expression and other constraints, sorted in increasing order.

- Find-Similar(Message-ID, Threshold)
  In this type of query, you will be given an message ID and a threshold. You need to answer all the other mail IDs that shares similarity greater than the threshold with such mail and sorted in increasing order of mail IDs.

- Group-Analyse(Message-IDs)
  In this type of query, you will be given a set of mail IDs. Assume a mail represents a transitive relationship between the sender and recipient, and a group is defined by users that can directly or indirectly have relationship with each other. You need to answer the number of groups and the size of the largest group among the mails' senders and recipients.

Each query will have its own reward R and penalty P. If you answer a query with the exact right answers, you earn the reward R. However, if your answers are wrong, then you will be charged with a penalty of P. The total points of your submission will be the sum of all the rewards and penalties.

Competition

A competition will be hold on the good old DSA Judge. Your team will have 5 quotas per day to submit programs for the above problem. In each submission, your program will be granted 5 seconds to process and answer queries right after getting all the mails and queries by calling the initialization function in header file. The score of your submission will be the sum of all the rewards and penalties (yes, your score can be negative for low accuracy) your program accumulated within the time limit. A rank list with all team's score will be made public.

The performance of your program is measured only by accuracy (whether your program returns the right answers) and efficiency (the number of queries the program can answer within the time limit). We somehow need to put a limit on memory usage for security and fairness. Each submission will be allow to use at most 10 gigabytes, and exceeding the limit will result in `Memory Limit Error` or `Runtime Error` on the judge system.

After the competition begins, you may see that the time limit on judge system is little longer than the 5 seconds above. This is because we want to ensure your program has the entire time limit duration to execute, excluding the time spent on data reading and writing, which is provided and unified by the functions in the header file. Each calls to answering function will trigger a time limit check, if your program has run more than 5 seconds after the initialization function returns, the answering function will terminate the execution of your function by invoking the `exit` system call. Noted that calling the initialization function after the first 15 ms will result in `Security Error`, in order to prohibit you from stealing the time before the timer is actually initialized.

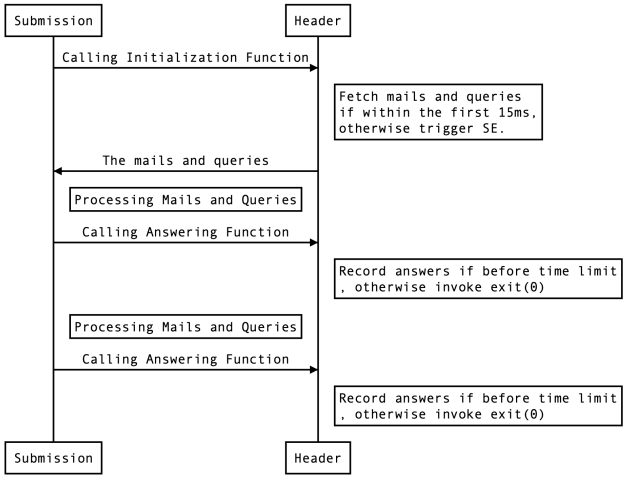

Figure 1: Example Execution Flow

Report

In addition to participating the programming competition, we also require each team to hand in a report to briefly explain your work. The report will account for 60 percent of your final project grade (see next Section). Please write the report in human-readable English (preferred) or Chinese, lest the grading TAs should not be able to fully understand your work.

You are suggested to use the latex report template to formulate your report, but please feel free to use other software to produce the report. The report you submit to the Gradescope should contain below information and sections in **no more than 3 A4 pages**:

- Information
  - Team ID (eg. Team No.0)
  - Student IDs of all members (eg. B08902071, B08902143)
- Sections
  1. **Data Structures & Algorithms**
     Explain what data structures and algorithms you have tried and why you choose to use them. For example: We use _____ to _____, so that we can _____ in O(___).
  2. **Cost Estimations of Queries**
     Provide and compare the cost estimation of your implementation on each types of queries. Using a table to show difference is encouraged. For example: Our implementation solves _____ in O(___), which is faster than solving _____ in O(___).
  3. **Scheduling Strategy**
     Describe how you design the strategy to answer queries. Does your strategy work as well as you imagined? If so/not, why? For example: We decide the priorities depending on _____, which is more relevant to _____ than _____, since _____.
  4. **Additional Notes (*optional)**
     Supply any other efforts/findings you have tried/discovered that (might) improves your implementation. For example: We also tried _____, and the result is _____.

Grading

By default, final project accounts for 10 percent of your raw score of this course. We will map the final score you got in final project to that 10 percent by a monotonic (increasing) function. The total score of final project is designed to be 1000, and split into the competition part and report part.

$$\text{Score}_{\text{Final}} = 1000 \cdot (0.4 \cdot \text{Score}_{\text{Competition}} + 0.6 \cdot \text{Score}_{\text{Report}})$$

The score within each part will be percentages, graded with different criteria. The Competition Score will be assigned depending on your ranking and points of your best submission. You may found the baselines released by the TAs on the scoreboard, eg: Baseline (30%), Baseline (70%) or TA Submission (Jason). Those baselines with percentage attached means if you surpass such baseline, you will at least get such percentage of Competition Score. The baselines without percentages are usually submitted by TAs just for fun or to show how good they can solve such problem. However, you may receive secret gifts from some TAs by surpassing their submissions.

The Report Score will be qualitatively graded with letters: A[0.95], B[0.8], C[0.65], D[0.5], F[0.3], with +/- denoting +0.05/-0.05. The qualitative description of each letter is listed below:

A. Explain the implementation detailed and concisely. Provide profound and accurate analysis on performance and costs. Being able to design effective and explainable strategies based on the analysis.

B. Some implementation details are missing. Small mistakes in assumptions and analysis. Slightly flawed explanation of strategy.

C. Mediocre explanation with flawed analysis and unconvincing strategy explanation.

D. Vague explanation of implementation, seriously wrong analysis on performance or costs. Unable explain the strategy design.

F. Some part of the report is basically not understandable, missing, seriously incorrect, or found committing plagiarism.

Data Structure and Algorithm, Spring 2021
Final Project - Email Searcher 2.0 Spec

Red Correction Date: 05/18/2021 17:40

Competition & Report Due: 23:59:59, Tuesday, June 22, 2021

TA E-mail: dsa_ta@csie.ntu.edu.tw

Instructions

- This document describes the spec of final project. Please refer to the initial announcement for other rules.

Header File

In this project, you can only access the data and reply the answer through functions provided in a given header file, `"api.h"`. **You must include this header file as the first header file in your submission.** Before introducing the provided functions, we first introduce the data types predefined in `"api.h"` for mails and queries.

Predefined Types

Mail Type

The type `mail` is defined as follows.

```
typedef struct mail {
    int id;
    char from[32], to[32], subject[256], content[100000];
} mail;
```

Each mail will have an integer within the range of `[0, n_mails - 1]`, where `n_mails` is the number of mails to be discussed later, as its unique mail ID. Each of the other entry will store a printable string that ends with `'\0'`. The set of mails is guaranteed to be a subset of the released mail data of last year's final project located at

https://www.csie.ntu.edu.tw/~htlin/course/dsa20spring/project/.

Query Type

The type `query` is defined as follows.

```
typedef struct query {
    int id;
    double reward;

    enum query_type { expression_match, find_similar, group_analyse } type;
    union query_data {
        struct { char expression[2048]; } expression_match_data;
        struct { int mid; double threshold; } find_similar_data;
        struct { int len, mids[512]; } group_analyse_data;
    } data;
} query;
```

The `type` entry stores the type of query. Then, the corresponding data is stored in the entry at `data.*_data`, where `*` is the string that matches the type. For instance, if the type is `expression_match`, the corresponding data can be access through the `data.expression_match_data` entry. Each query will also have its reward in the `reward` entry, and its unique query `id`. **For simplicity, we will take the penalty of returning a wrong answer to the query to be -0.5 times the reward.**

Helper Functions

Initialization Function

We provide a function called `api.init`, which helps receive the mails and queries. **Noted that this function can only be called within the first 30 milliseconds of your program execution.** The timer of judging your program begins when `api.init` returns. The function prototype is defined as follows.

```
void api.init(int *n_mails, int *n_queries, mail **mails, query **queries);
```

You are supposed to pass two integers by their addresses (i.e. call by reference in C) as the first two arguments. The function will then save the number of mails (and queries) in the corresponding integer before returning. You are also supposed to pass two pointers by their addresses (i.e. call by reference in C) as the last two arguments. The function will then allocate two arrays, one for storing mails and the other for storing queries, and save the address of the array in the corresponding pointer.

Answering Function

To reply to queries, we provide a function called `api.answer`. The function prototype is defined as follows.

```
void api.answer(int query_id, int answer[], int answer_length);
```

For each query, you need to store the answers in an array and reply with this function so that the answer will be judged. **Note that for each query, only the first attempt of answering will be judged, and subsequent attempts will be ignored.** If your program reaches the time limit within the answer function, the program can terminate without completing the answer function.

Example Usage

```
#include "api.h" /*"api.h" must be include as the first header file.*/

int n_mails, n_queries;
mail *mails;
query *queries;

int main(){
    api.init(&n_mails, &n_queries, &mails, &queries); /* initialization */
    for(int i = 0; i < n_queries; i++){
        /* guessing no-match for all expression-match queries */
        if(queries[i].type == expression_match)
            api.answer(i, NULL, 0);
    }
    return 0;
}
```

Query

There will be three types of queries, Expression-Match, Find-Similar and Group-Analyse, each with its own requirements and rewards. We start with some definitions first.

Definitions

Tokens

Tokens, the basic components in expressions, are defined as successive alpha-numeric characters, or `/([A-Za-z0-9]+)/g` in regular expression. That is, all non-alpha-numeric characters can be viewed as delimiters of the tokens. For example, `"dsa2021"` and `"mails"` are tokens while `"A+"` is not (only `"A"` within `"A+"` is a token). Also, the `"ppl"` within `"apple"` is not a token. Here is another example of tokens in a mail's content. Furthermore, tokens are **case-insensitive**, which means `"Apple"` and `"apple"` represent the same token. When speaking of the token set of a mail, it refers to the set formed by the tokens within the mail's subject and content.

Context Similarity

The context similarity of two mails is defined as $\frac{|A \cap B|}{|A \cup B|}$, where A, B denotes the token sets of the two mails.

Expressions

Expressions are recursively defined in the order of precedence as follows, let `[token]` and `[expr]` denotes arbitrary token and expression.

- `[token]` - a single-token expression returns true on a mail iff the mail contains the token in its subject or content
- `([expr])` - an expression wrapped by parentheses evaluates to true iff the internal `[expr]` evaluates to true
- `![expr]` - an expression negated by `!` evaluates to true iff the `[expr]` evaluates to false
- `[expr]&[expr]` - two expressions with `&` evaluates to true iff both `[expr]`'s evaluate to true
- `[expr]|[expr]` - two expressions with `!` evaluates to true iff at least one of the `[expr]`'s evaluates to true

Given that `[token]` and all expression operators do not contain white spaces, `[expr]` will not contain any white spaces as well.

Query Tasks

Expression Match

In the EXPRESSION-MATCH query, an `expression` will be given. To answer such query, you need to find all the mails on which the expression returns true. Then, return the matching mail IDs in **increasing order** through `api.answer`. As the type `query` suggests, the length of expression is restricted to 2048 characters, including the ending null byte. We also guarantee that all the expressions are legal, and all the tokens exist in certain mails.

Find Similar

In the FIND-SIMILAR query, a mail ID `mid` and a threshold `threshold` is given. You need to find all the other mails, excluding `mid`, that are of context similarity higher than the `threshold` with respect to `mid`. Then, return those mail IDs in **increasing order** through `api.answer`. We guarantee that the mail ID will be valid and the threshold is within $[0, 1]$. We also guarantee that the threshold would be chosen such that you can directly compare your similarity to the threshold without worrying about the floating point precision.

Group Analyse

In the GROUP-ANALYSE query, you will be given a subset of mails by their IDs and need to analyse on the equivalence relation among the senders and recipients in the mail set. The size of the set is stored in `len` and the mail IDs are stored from `mids[0]` to `mids[len-1]`. You will only need to consider all senders and recipients in the mail set for each query. The relation between the senders and recipients is defined as follows.

- If A sends a mail to B in the mail set, then A is related to B and B is related to A.
- If A is related to B and C is related to B, then A is related to C and C is related to A.

With the relation definition, a group is defined as a set such that every member is related to every other member within the set. In this task, you need to calculate the number of groups n_g, and the size of the largest group l_g. Then, return an array of length 2 through `api.answer`, where the first value of the array stores n_g and second stores l_g. We guarantee that each mail ID will be valid and unique. In addition, the mail set will not be empty.

Task Rewards

As you might have discovered, the three types of queries come with different difficulties. Therefore, different types of queries are associated with different rewards. Basically, the reward R of

a certain task is roughly normal distributed, $R \sim \mathcal{N}(\mu, \sigma^2)$, but each query type is associated with a different (μ, σ^2).

Local Testing Environment

Besides the DSA Judge, a local test environment will be released for the ease of testing your code without wasting quotas. The environment works for different operation systems, but the behavior (and of course, the "clock") may differ across different systems. ~~However, the testing environment only supports Unix-based systems. For those who uses the Windows Operating System, we will also release a Repl.it Template for you to fork and test your solution.~~

The score shown in test environment is only for your reference. We will only take the score on the DSA Judge scoreboard as the grading criteria. So please remember to submit your best solutions to the DSA Judge. Also, it is common that the score fluctuates in this kind of performance competitions. So we decide to increase the submission quota from 5 to 10 each day.

It is worth knowing that the input format and hash function used will be different between the local testing environment and the DSA Judge environment. Therefore, please follow the guideline and use the api provided to access the data, instead of manipulating the data by yourself in any manner. Failure to follow the access api could mean violating the course policy and may result in serious penalties.

Baselines

The baselines will be release later (at last 2 weeks a head of due date).