A Searching Engine on Subject Description Form

YUNFEI LIU*, JIAMING HAN*, ZHUCHEN WANG*, and LONGLING GENG*, The Hong Kong Polytechnic University, China

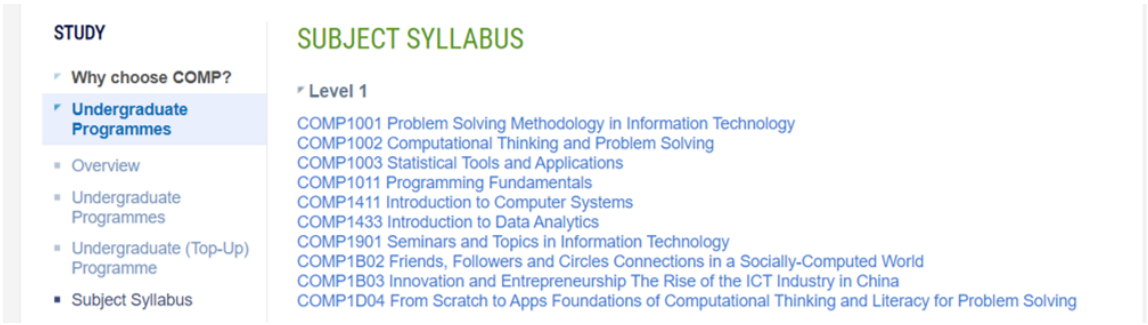

Fig. 1. Subjects offered by PolyU COMP

CCS Concepts: • **Theory of computation** → *Sorting and searching*.

Additional Key Words and Phrases: algorithm, searching, data structure, education

ACM Reference Format:
Yunfei LIU, Jiaming HAN, Zhuchen WANG, and longling GENG. 2021. A Searching Engine on Subject Description Form. 1, 1 (May 2021), 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

*These authors contributed equally to this research.

Authors' address: Yunfei LIU, yunfei.liu@connect.polyu.hk; Jiaming HAN, jiaming.han@connect.polyu.hk; Zhuchen WANG, zhuchen.wang@connect.polyu.hk; longling GENG, long-ling.geng@connect.polyu.hk, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong, China.

1 INTRODUCTION

Nowadays, many messages, webpages are created every day and we can't read them one by one. To get data quickly and effectively, searching is of vital importance. Meanwhile, how to search for the needed information in the shortest time, and how to find the correct webpages in numerous web pages on Internet, is a challenging question.

The searching algorithm is developed to save people's time of finding what they want, and it has a long history. Since the first search engine *WebCrawler* appeared in 1994, humans have entered the information era. In this project, we revisited this part of history and figured out how the search engine is working. We developed our own data structures and algorithm to accelerate the searching process. Although our method is not mature enough, it is a good start for us to improve our programming skills and learn some experience.

2 ALGORITHM

We referenced the algorithm of *ElasticSearch*, which is powerful, open-source, and easy to understand [4]. However, even though the searching method is easy for experienced programmers, it is too complicated for us. Also, the data amount is small in this project, and complicating algorithms may lead to opposite effects [3, 5]. Therefore, we learned the algorithm thoroughly and understand its essence. Finally, we simplified the algorithm, making it easier to program with just a slight increase in the searching time [9].

The main idea of this algorithm is to use a dictionary to search for the data. Firstly, users' input will be cleaned, which means that the special characters, numbers, punctuations will be removed, and uppercase letters will be transformed to lowercase letters. By doing these, users' input will be transformed into snippets that only contain lowercase alphabet, and it will be very helpful for further matching.

Then, the word snippets will be translated to word ID with a black-red tree dictionary [7]. The black-red tree is an advanced self-balanced binary search tree, and the time complexity is $O(logn)$, which can shorten the searching time significantly.

3 DATA STRUCTURE

To implement the idea of object-oriented programming, as well as make the best use of C++ features, we defined two classes to represent subjects and words separately.

The first class named **Course**, each instance represents one subject, contains essential information related to the subject including subject ID, subject code, subject pre-requisite, subject level, subject credit, and subject content. To implement the red-black tree, we use C++ STL container 'map', which is a balanced hash table [2]. It can search objects quickly. We use the map to store how many times the word appeared in this subject description form content. Meanwhile, in our opinion, the words that appeared in the title are much more important than the words that appeared in the content. Therefore, we assigned more

impact index on words that appeared in the title, and use two map objects to store the title and content. Also, we think that data security is of vital importance in nowadays society, so we use private objects to store the data, which is safer and easier for future maintenance [11, 14].

```
class Course
{
public:
    Course() {this->Id=-1;};
    void SetID(int id) {this->Id=id;};
    void SetCode(std::string code) {this->Code=code;};
    void SetContent(std::map<std::string, int> contentDict) {this->ContentDict=contentDict;};
    int GetID() {return this->Id;};
    std::string GetCode() {return this->Code;};
    std::map<std::string, int> GetContent() {return this->ContentDict;};
private:
    int Id;
    std::string Code;
    std::map<std::string, int> ContentDict;
};
```

Fig. 2. A part of **Course** class, showing public and private objects

The second class names **Dictionary**, each instance represent one word that has appeared at least once in the subject description form contents. The class includes word name, word ID, the subject id where the word appeared in title or content. We use word ID to shorten the search time and make it easier to be searched in the whole content. Each Dictionary object contains one word, and the total amount of words is around 5000. Therefore, we defined an array of Dictionary objects to store all the words, and by accessing this array, the program can read all the information it needs. Also, we make good use of C++ STL containers, we choose to use a set to store the subject ID that the word has appeared, which can save time when searching [12].

```
class Dictionary {
    public:
        Dictionary(){this->wordID=-1;};
        string word;
        int wordID;
        set<int> appearTitle;
        set<int> appearDoc;
};
```

Fig. 3. The C++ code of **Dictionary** class

4 PROGRAM DESIGN

4.1 Overview

For the data preparation part, we use *Python* because of its flexibility and cross-platform feature. Since it is a C++ project, we only use Python downloading and transforming data, because C++ does not have libraries that can process *PDF* files. Considering that the program may be implemented on the web later, and we want to leave some immigration ability to make it easier to run on different platforms, we use *JSON* to save and process the data, which is a widely used data type between frontend and backend [10].

4.2 Ideas

To make the program easier to maintain, we try to use object-oriented programming skills. We divided the whole program into different functions, and each function can finish one small task, and they can be used as objects. By doing this, adding more functions becomes easy and convenient [6].

4.3 Problem-Solving

We have divided the program into the follow four parts:

- Data Pre-Process
- User Input Process
- Relative Index Calculation
- Result Display

To solve the problem of searching, we should first establish a database which stores all data. Then, we should check the users' input to avoid bugs caused by invalid input. After that, we should calculate the words relative index and finally display the searching result.

The relative index calculation formula is displayed as follows:

$$R.I. = \sum_{i} \left[N_{i(title)} \times 100 + N_{i(content)} \times 5 \right] \tag{1}$$

While $N_{i(title)}$ represents the times that keywords shown in subject title and $N_{i(content)}$ represents the times that keywords shown in subject content.

4.4 Designed Functions

(1) **Input cleaning**

Ask user to type the keywords, and then remove the unnecessary characters such as ",","'","@&*", etc. Transform all uppercase letters to lowercase.

string* InputCleaning(): should return a pointer pointing to a string array, which needs to contain processed words user typed

(2) **Relevance Calculation**
This function should integrate with the third function, and it should calculate the relevant score of each document.
A algorithm which can represent the relevance of documents, you have this information: word input by users, how many times the word appear in the title of each document and how many times the word appear in the content of each document.

(3) **Dictionary Lookup**
This function receives parameters including the return value of the previous function *InputCleaning()* and dictionaries, and should search in the dictionary with the method of ElasticSearch.
int *Search(string *inputs): should return a pointer pointing to an integer array. the array contains the scores of each subject description form

(4) **Result Display**
After calculation, the search documents should be displayed in order, and the program should allow users interact with files, such as choose to display the content, or choose to download the pdf file.
void DisplayResult(float *scores): display the search result and offer some interactions.

5 DATA PROCESS

5.1 Data Collection

As required in the project description, the data we need is on the PolyU website https://www.comp.polyu.edu.hk/en-us/study/undergraduate-programmes/subject. The content of the website includes all course numbers, names, and description forms. To analyze them, we need to download them first then process them.

To download the files, the implementation of C++ of downloading files from a webpage is very complicated. Therefore, we use Python as an alternative method. Third-party libraries including *urllib* and *bs4* are used for processing the webpage and search for pdf files. The Python script is attached in the **DataCollectionPythonScript** folder [13].

5.2 Data Transform

Of course, C++ cannot process PDF files directly because they are not plain text. To get rid of this problem, we need to transform the PDF files to plain text. Python is used again because of its convenience. The third-party library *pdfplumber* is used to transform the subject description forms to TXT files [1].

5.3 Data Cleaning

The subject description forms in TXT format are messy and cannot be used by C++ programs directly, so we need to clean the useless characters, and extract features from

the TXT files. A Python script *getfeature.py* attached in the **DataCollectionPythonScript** folder solves this problem. The script transforms the *txt* files to *JSON* format with the features including subject code, title, level, credit, pre-requisite, and content. The content is also transformed into JSON format to record each word and its appeared times. By cleaning data, the C++ program can process it faster and more conveniently [8].

6 DISCUSSION

We have finish the required tasks successfully, and we add more functions to make the program more powerful, such as searching content and download the subject description forms. However, there are more things we can do. For example, we can move the program in a docker so that it can run everywhere. Also, we can program an webpage similar to Google for general public to use. Moreover, we can use SQL instead of JSON to allow larger data amount and frequently addition or deletion. The algorithm can be improved to adjust high concurrency, too.

7 CHALLENGES

The first challenge we encountered is that the C++ do not have libraries which can process JSON files. Luckily, we finally found a third-party header which is called *nlohmann json*. This library can process json files and transform it to C++ STL containers such as *set* and *map* [15].

The second challenge we encountered is class design. We should abstract the data and fit them with a framework. After several attempts, we finally create two classes for them.

The third challenge is the docking between different programmers. As the functions are written by different programmers, the arguments and return value expected are different. After communications, we reached the agreement.

8 CONCLUSION

In this program, we gain experiences about how to develop a program from the beginning to the end. We meet many problems and we have to find solution to it by ourselves. In this process, we learn a lot knowledge that we cannot learn in class. Also, during the presentation, we know how to show our achievements to public. All in all, developing this project is a valuable experience.

9 APPENDIX

9.1 Work Distribution

Name	Work
Yunfei LIU	Data Collection, Data Transform, Data Structure, Demo Make, Bug Fix, Program Combine, Report, Presentation Slides
Jiaming HAN	Algorithm Design, Relative Index Calculation
Zhuchen WANG	Result Display, Demo Make
Longling GENG	User Input Process, Demo Make

9.2 Useful links

(1) GitHub Repository (Public on May 12) https://github.com/liu-yunfei/COMP1011GroupProject
(2) Subject Description Form Download Page https://19040822.xyz/

REFERENCES

[1] Jiaze Chen, Liangcai Gao, and Zhi Tang. 2016. Information extraction from resume documents in pdf format. *Electronic Imaging* 2016, 17 (2016), 1–8.
[2] Dibyendu Das, Madhavi Valluri, Michael Wong, and Chris Cambly. 2008. Speeding up STL Set/Map Usage in C++ Applications. In *SPEC International Performance Evaluation Workshop*. Springer, 314–321.
[3] Manda Sai Divya and Shiv Kumar Goyal. 2013. ElasticSearch: An advanced and quick search technique to handle voluminous data. *Compusoft* 2, 6 (2013), 171.
[4] Clinton Gormley and Zachary Tong. 2015. *Elasticsearch: the definitive guide: a distributed real-time search and analytics engine.* " O'Reilly Media, Inc.".
[5] Robin J Hogan. 2014. Fast reverse-mode automatic differentiation using expression templates in C++. *ACM Transactions on Mathematical Software (TOMS)* 40, 4 (2014), 1–16.
[6] Robert Lafore. 1997. *Object-oriented programming in C++*. Pearson Education.
[7] Jianwei Li, Yubin Xu, and Hong Guo. 2004. Memory organization in a real-time database based on red-black tree structure. In *Fifth World Congress on Intelligent Control and Automation (IEEE Cat. No. 04EX788)*, Vol. 5. IEEE, 3971–3974.
[8] Mark Lutz. 2001. *Programming python.* " O'Reilly Media, Inc.".
[9] Yiannis N Moschovakis. 2001. What is an algorithm? In *Mathematics unlimited—2001 and beyond*. Springer, 919–936.
[10] Nurzhan Nurseitov, Michael Paulson, Randall Reynolds, and Clemente Izurieta. 2009. Comparison of JSON and XML data interchange formats: a case study. *Caine* 9 (2009), 157–162.
[11] Steve Oualline. 2003. *Practical C++ programming.* " O'Reilly Media, Inc.".
[12] Stephen Prata. 2002. *C++ primer plus.* Sams Publishing.
[13] Leonard Richardson. 2007. Beautiful soup documentation. *Dosegljivo: https://www. crummy. com/software/BeautifulSoup/bs4/doc/.[Dostopano: 7. 7. 2018]* (2007).
[14] Clifford A Shaffer. 1997. *A practical introduction to data structures and algorithm analysis.* Prentice Hall Upper Saddle River, NJ.
[15] MT Skinner. 1992. *The C++ primer: a gentle introduction to C++.* Silicon Press.