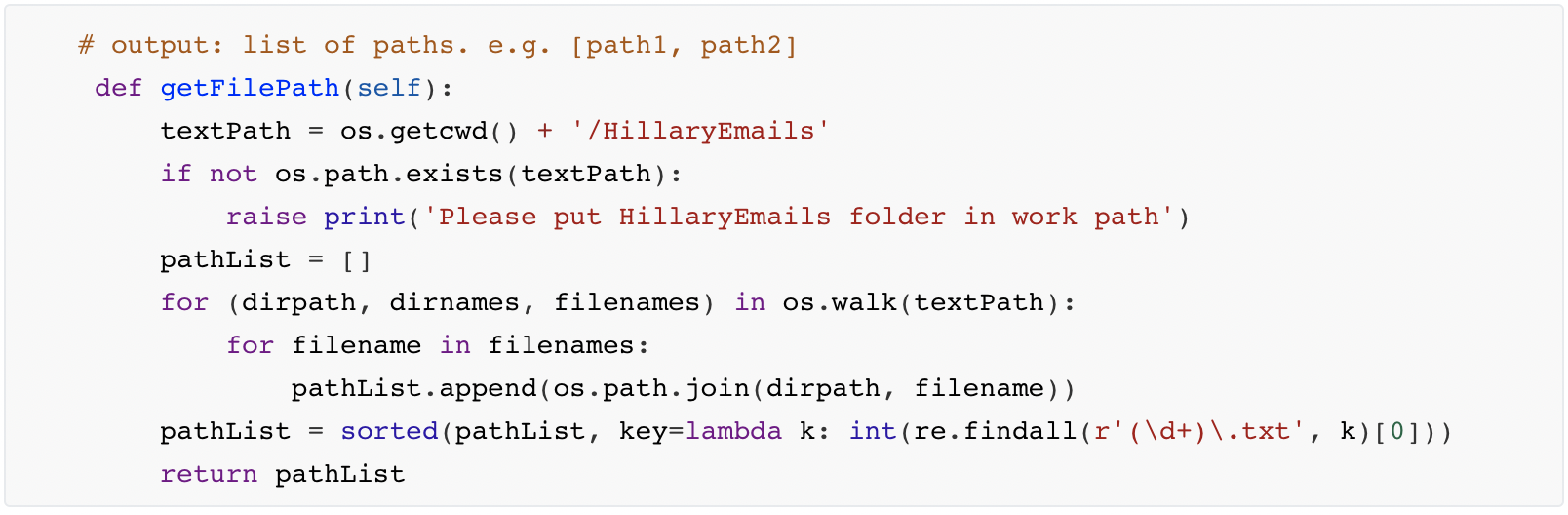
**Test & Optimization**

1. **Optimization Step**

We currently store the full document path in the posting list. Each document path is stored multiple times, which we believe would affects the memory and time requirements for creating and querying indexes. Thus, in the optimization process, we try to store all the file paths in the list to improve the efficiency. Below are our optimization steps:

1. We defined a function named getFilePath() which will return a list of paths (eg. [path1, path2]) shown in below:



1. In sortTokens part, we use the getFilePath() to get the pathList, then we use the index of the pathList to obtain the content of the .txt file in corresponding path after which we used the tokenization() function to get the tokenlist which is followed by stemmer() function for stemming. Finally, we push those newTokenList to the allTokenList. Then we sort the allTokenList and return the sortedToken.



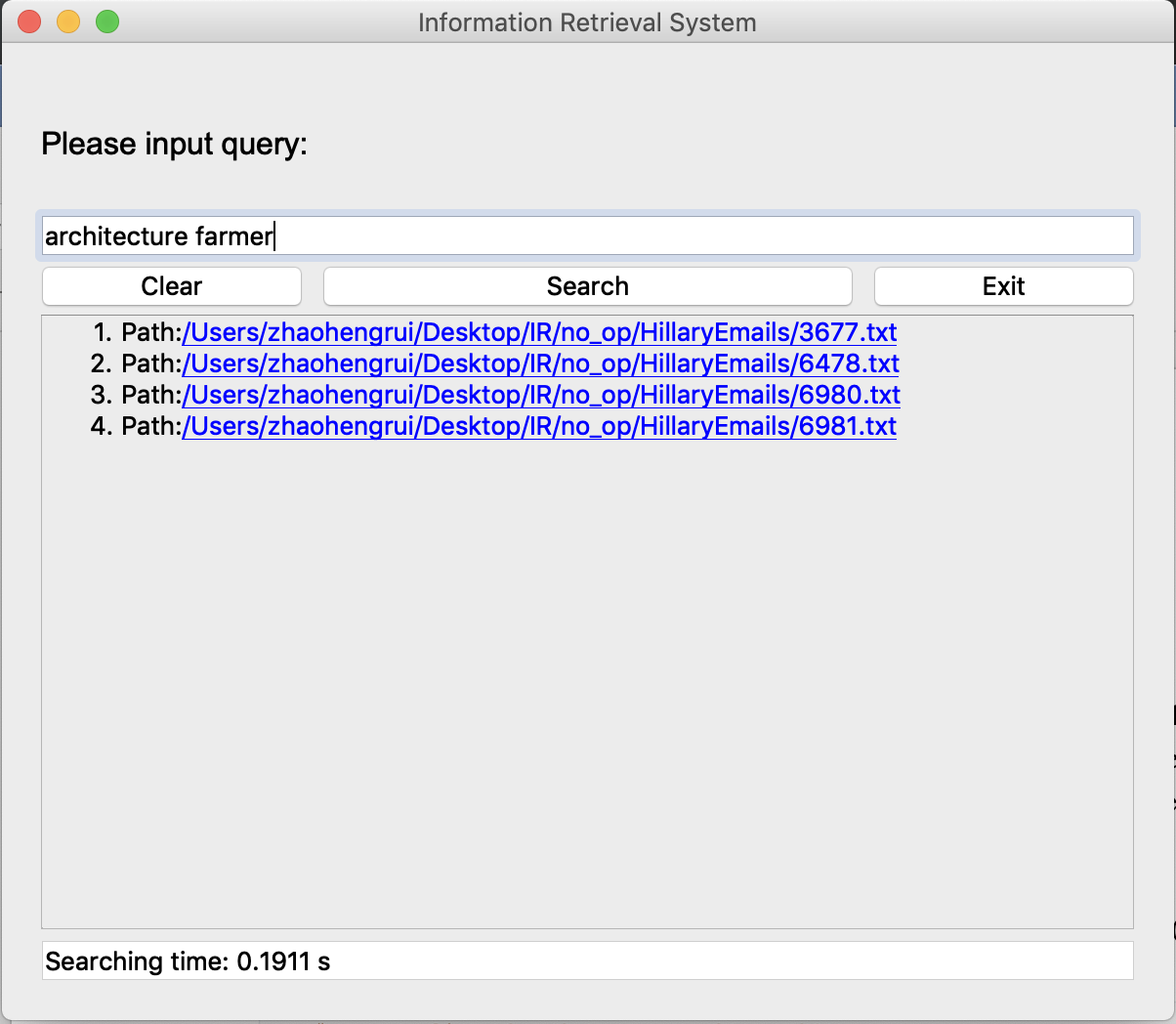
1. In getResult() function, We traverse all the items in result and output all the results as pathList[index]. To enhance our user experience and enrich our functionality, we used a string containing HTML tags to present the final result.

（1）Use ordered list tag <ol><li>to give the answer sort id.

（2）Use the <a> tag to make the hyperlink to each result item to make it clickable. When the user clicks on the result item, the .txt file of the corresponding path will be automatically opened.



An example of the result:



1. **Test**

In this section, we will start with a small document set of 5 files to do a set of demo functionality test, after which we do black-box test to the whole document to check the accuracy of the query results which includes multiple test cases, such as single word query, multi-word query and OR mixed query. After that, we will test the time and size requirement of Information Retrieval System before and after optimization to evaluate our optimization. It mainly consists of two key steps of testing, creating indexes and querying search results.

1. Testing Environment

Device: Macbook Pro (13-inch, 2017, Four Thunderbolt 3 Ports)

OS: macOS Mojave 10.14.3

IDE: PyCharm

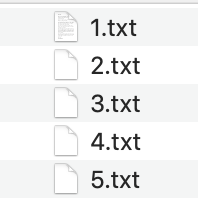
RAM: 8GB

Time test tool: Profile

Memory test tool: Memory\_profiler

1. Demo-test(Small document set of 5 files)

We have retained 5 documents for demo testing.



Case 1 “father”:



Case 2 “mother”

No Result.

Case 3 “wednesday”



Case 4 “father wednesday”

Expected Answer: “no result”

No Result

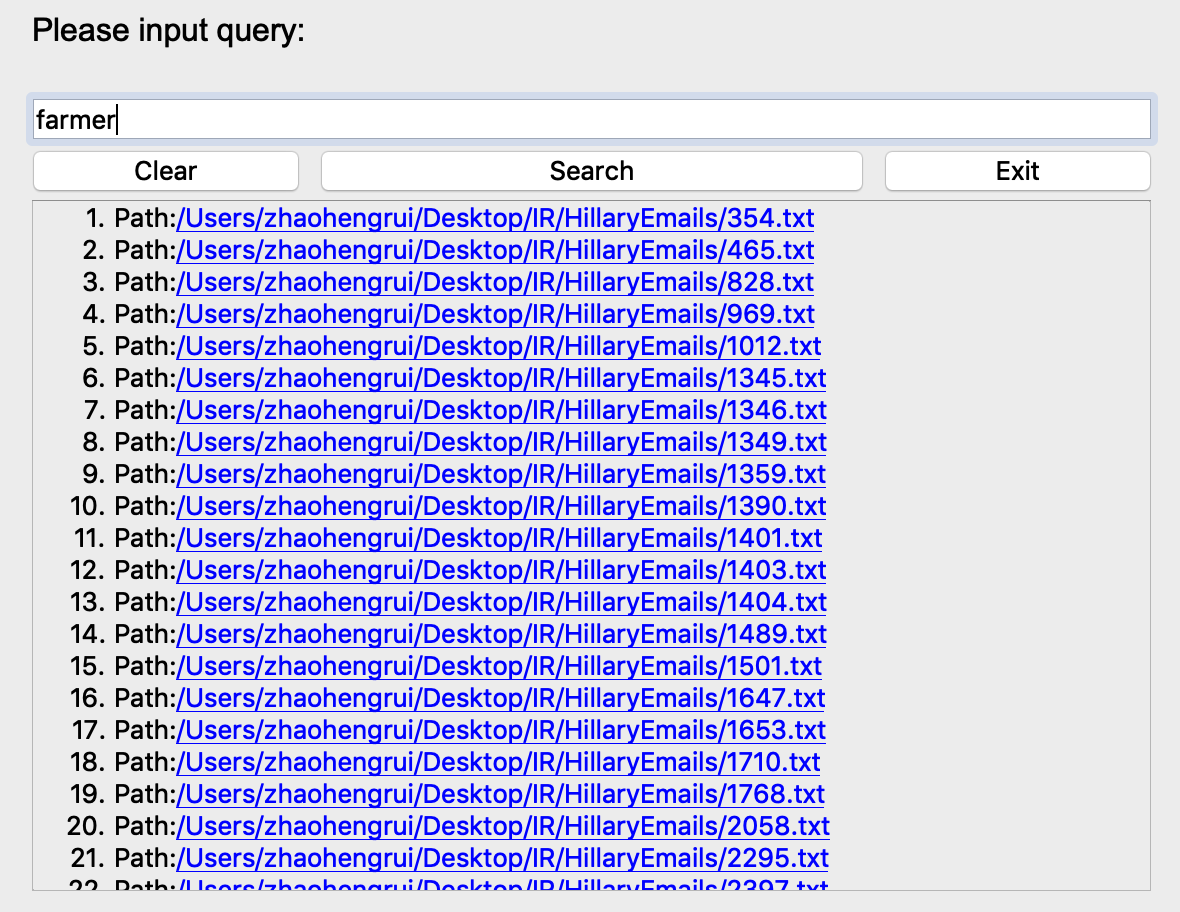
Case 5 “father OR wednesday”

Expected Answer: “1.txt, 2.txt, 3.txt, 4.txt, 5.txt”



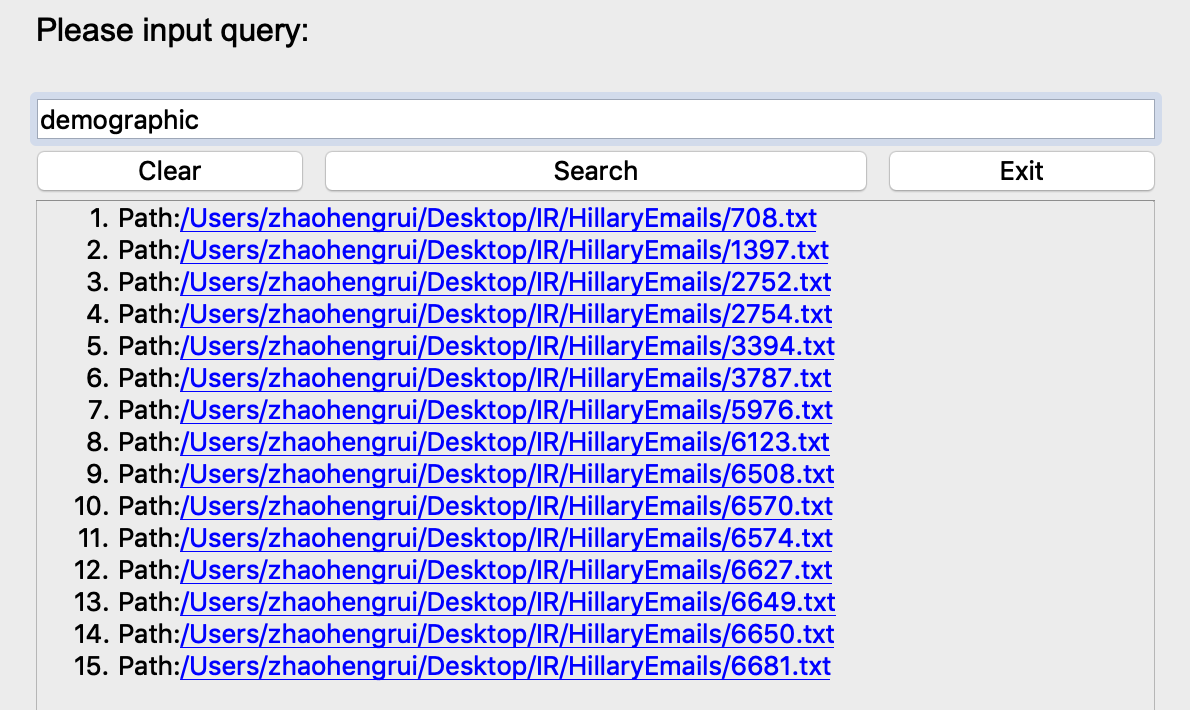
1. Black-box testing for whole document set
2. Single-word query

Test Case 1: ‘farmer’:



The result has 62 items. After check, the result is accurate.

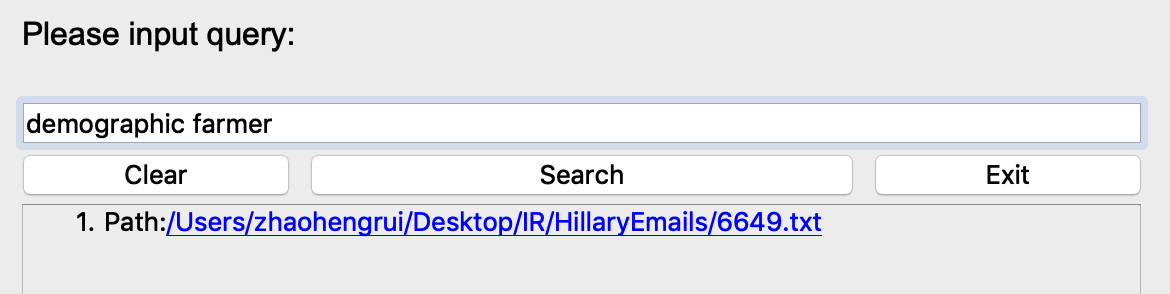
Test Case 2: ‘demographic’



The result has 15 items. After check, the result is accurate.

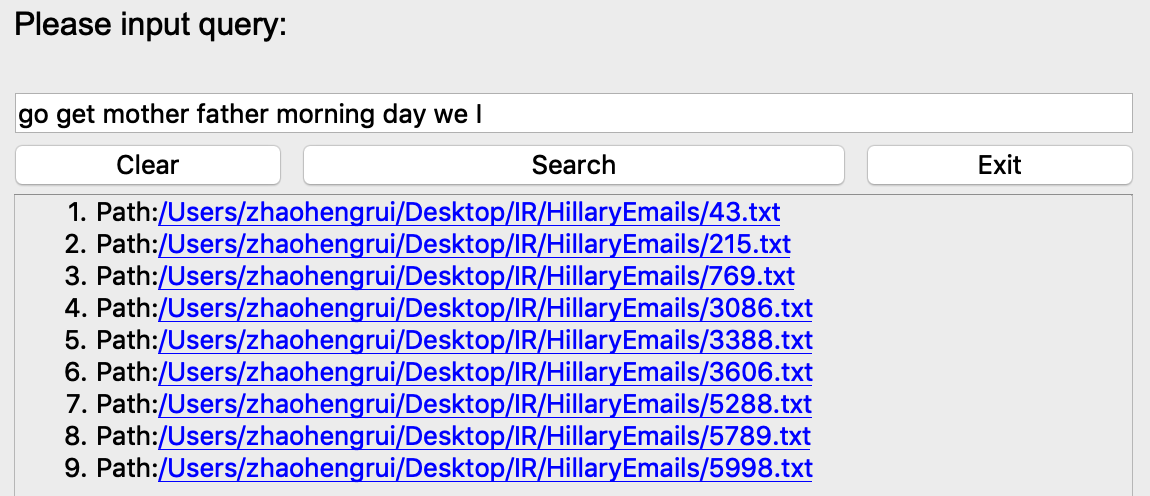
1. Multi-words query

Test Case 1: “farmer demographic” (Shorter Query)



Test Case 2: “go get mother father morning day we I” (Longer query)

The result has 1 item. After check, the result is accurate.



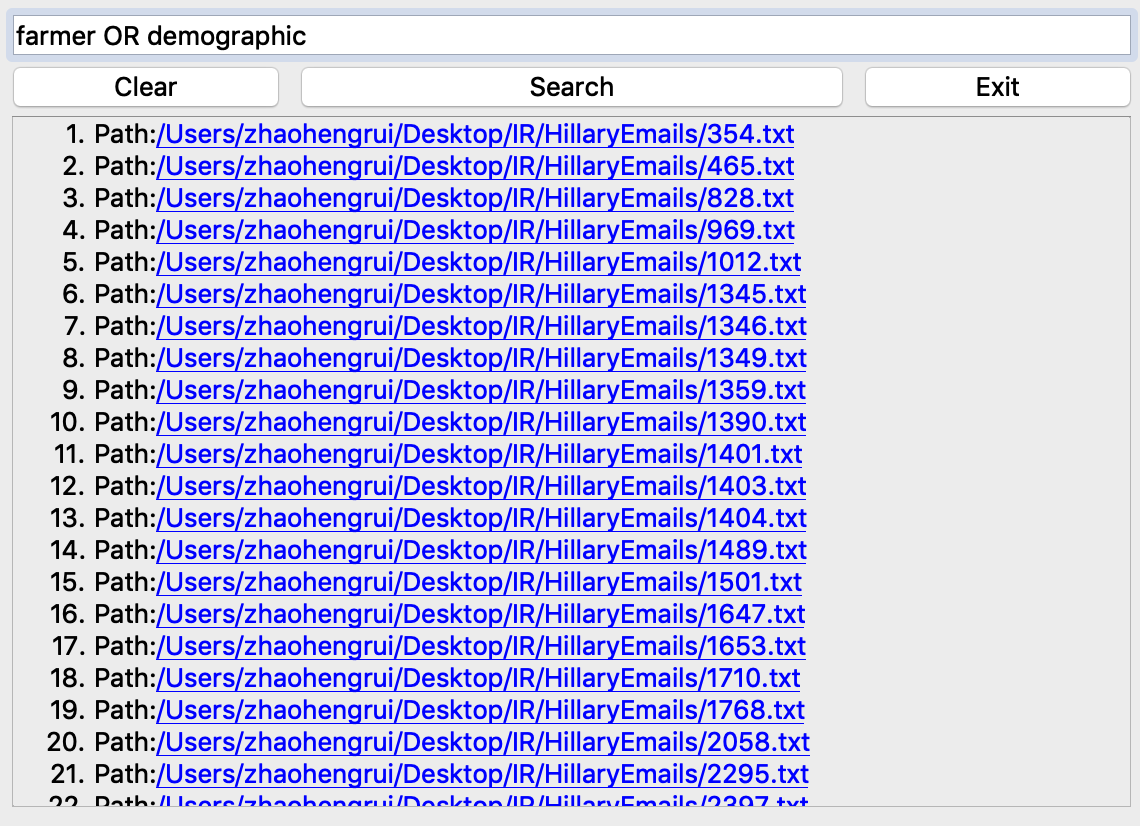
The result has 9 items. After check, the result is accurate.

Test Case 3:” ﻿alphabet dangerous”

No Result.

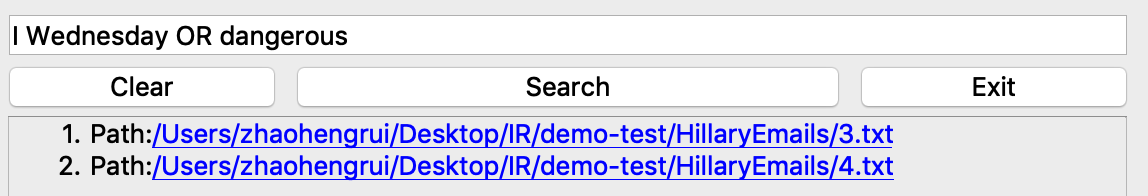
1. ‘OR’ Mixed query

Test Case 1: “farmer OR demographic”



The result has 76 items. After check, the result is accurate.

Test case 2: “I Wednesday OR dangerous”



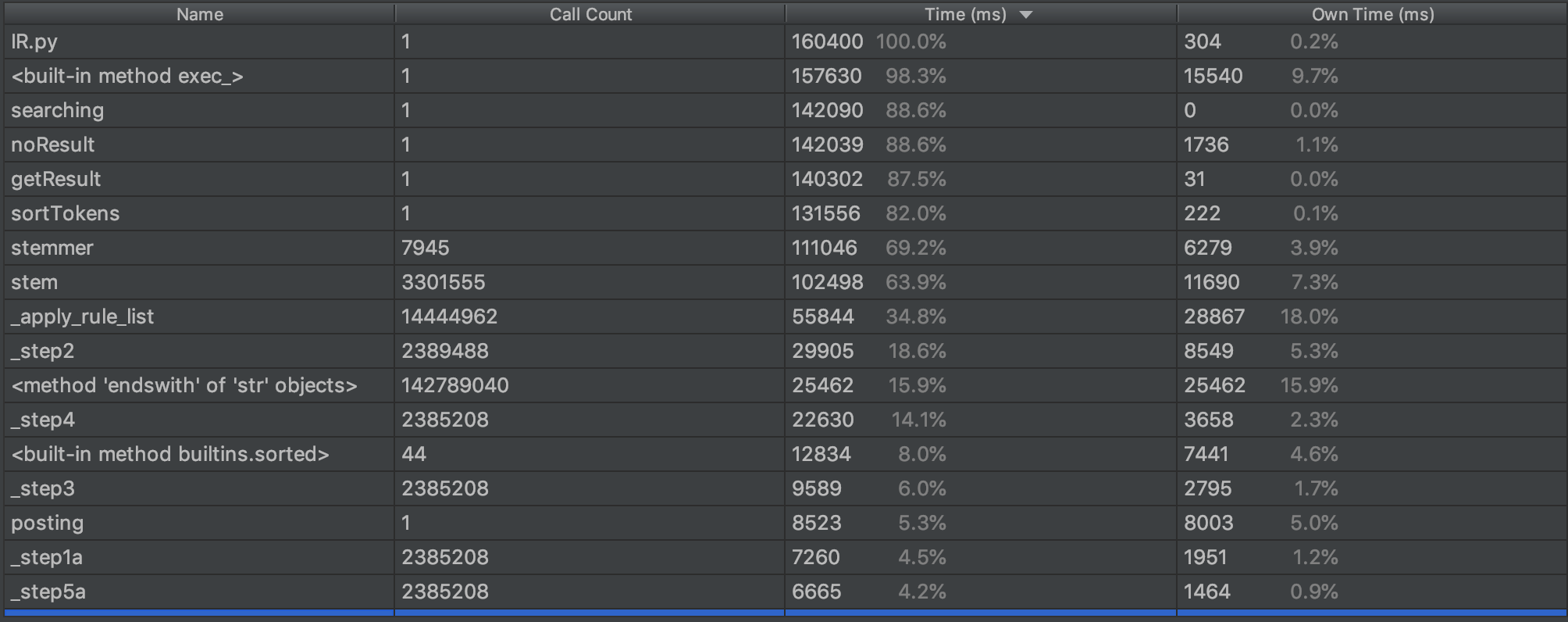
The result has 2 items. After check, the result is accurate.

1. Time&Size requirement Testing

In this test session, we try to ensure that the environment is consistent and close other unrelated processes. Our first test scenario is the first time to run the program and search "architecture" keyword, click on the search to get the search results, then close the GUI interface to complete the test. Another test scenario is to search “architecture farmer” in both before and after optimization system after creating indexes and calculate the average of 5 search times.

1. Before Optimization

Time profile result:



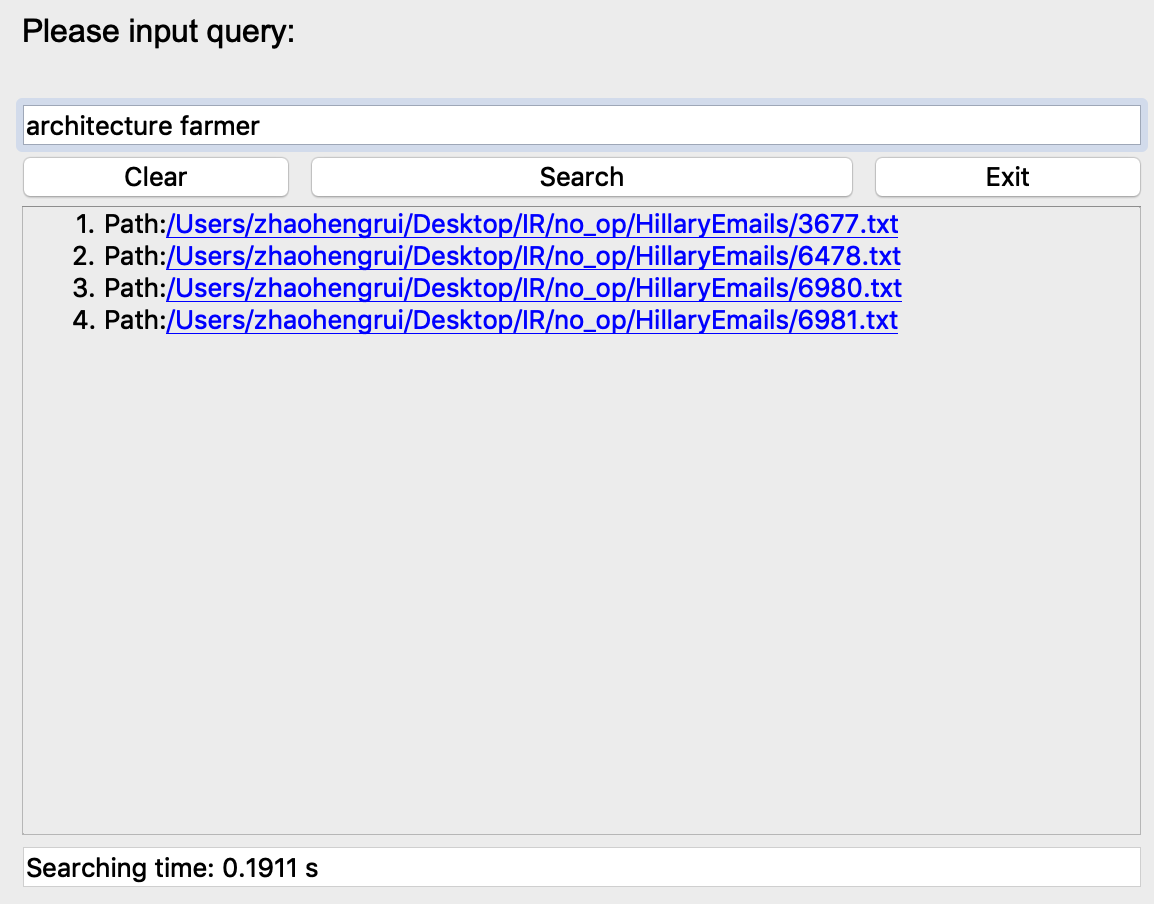
It takes 160400ms for the whole process.

Call graph：

图片包含 黑色, 室内, 墙壁

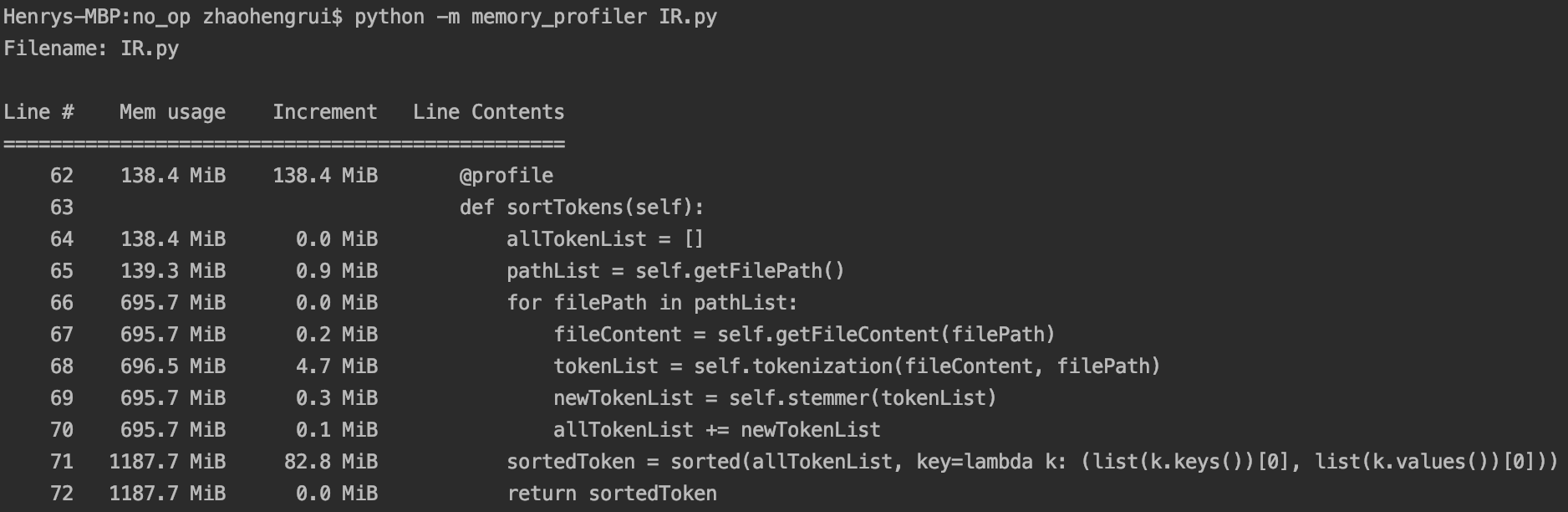
描述已自动生成

Searching time（query case=”farmer architecture”）



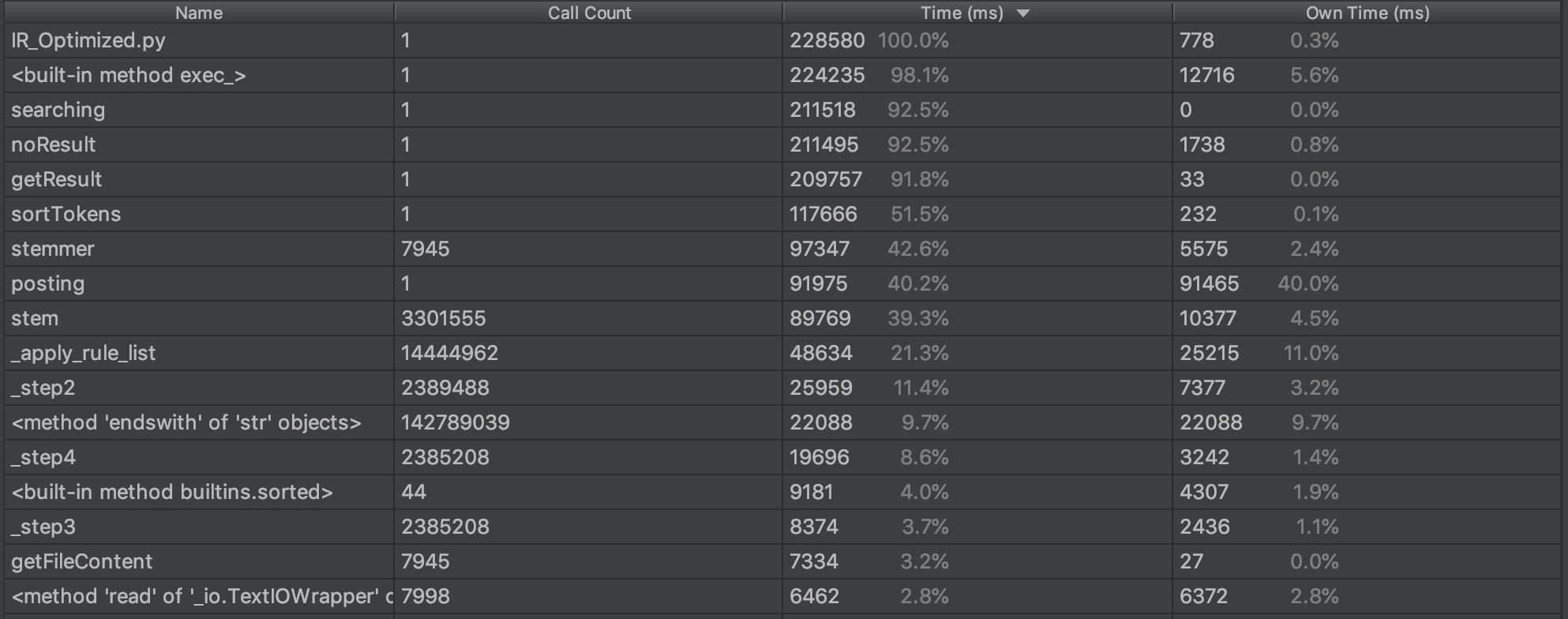
Searching time is 0.1911s.

Size Consumption Result(Partial):



1. After Optimization:

Time profile result:



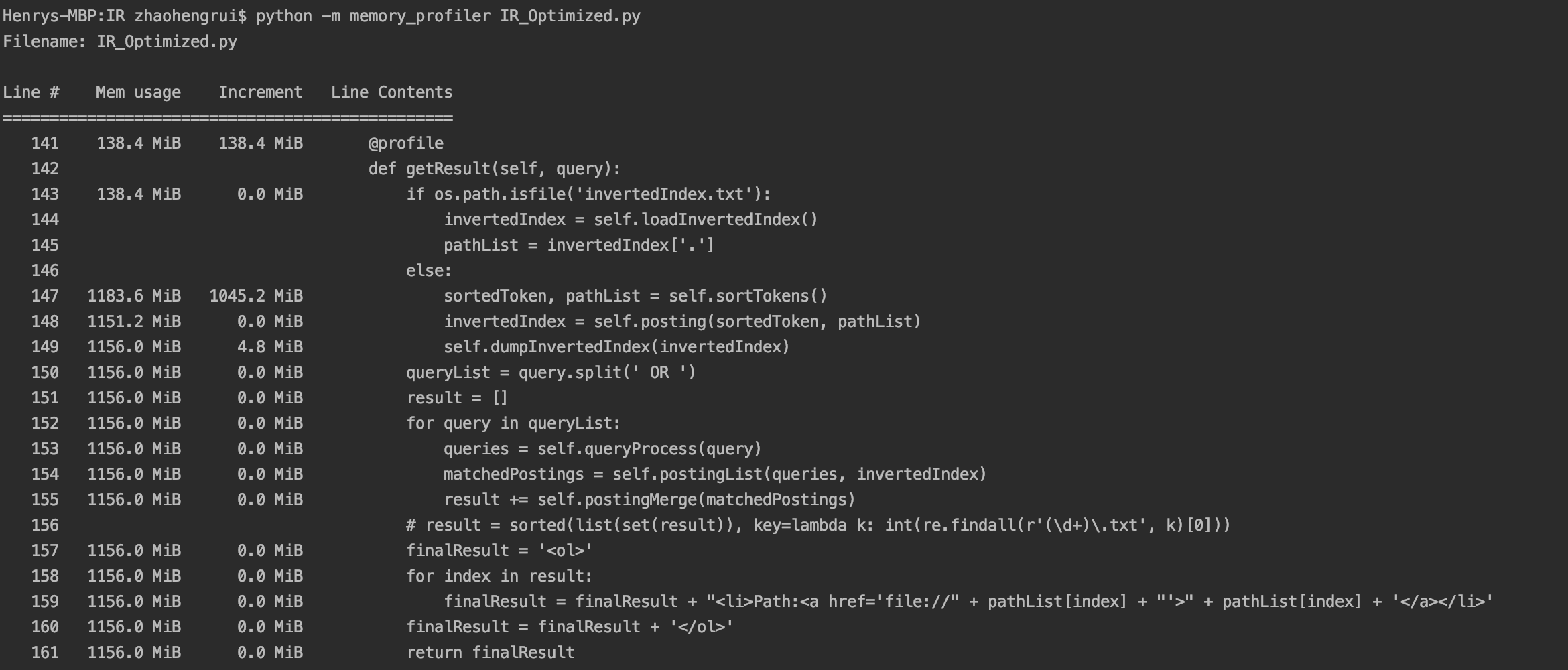
It takes 228580ms for the whole process.

Call graph:

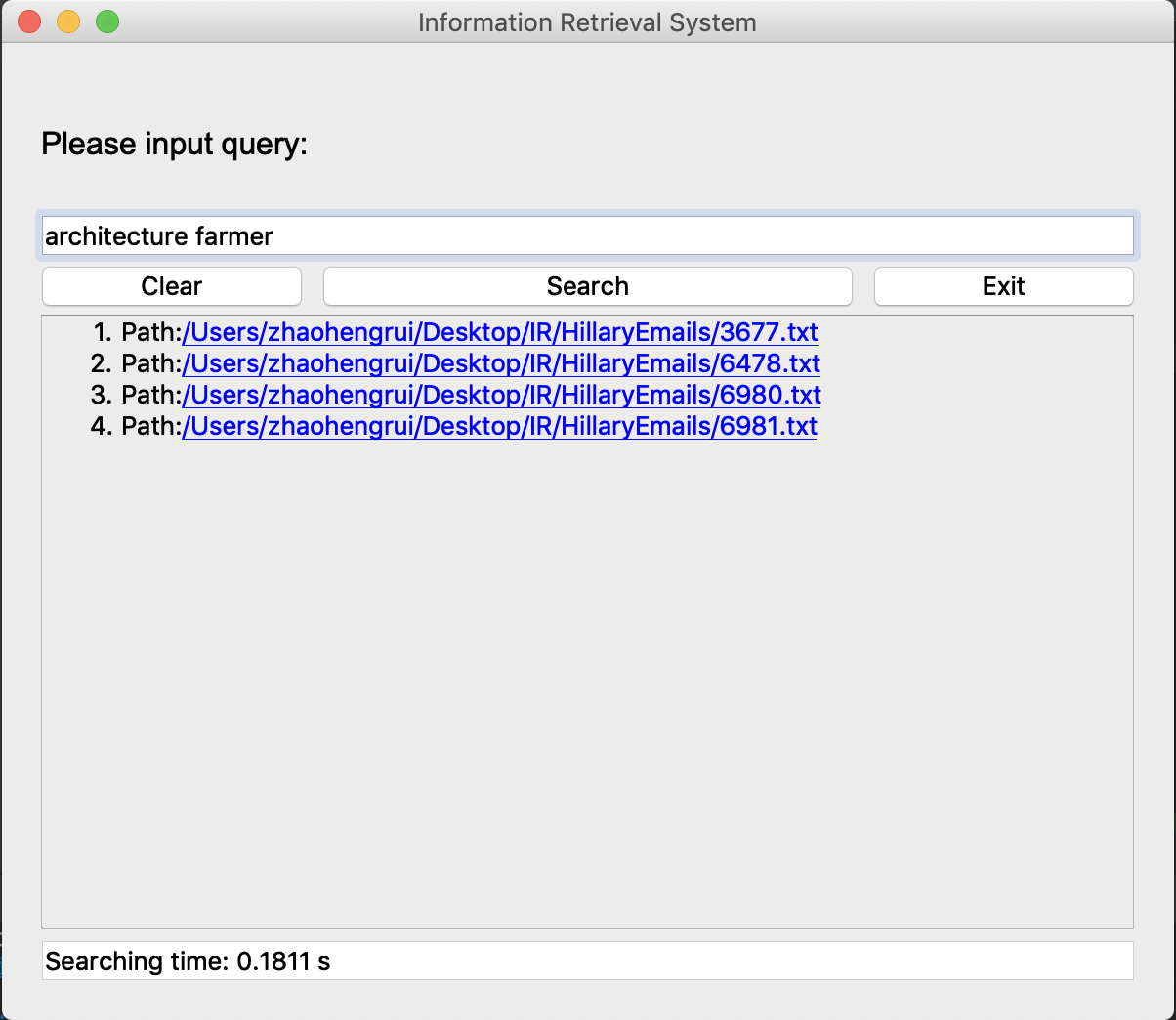
图片包含 黑色, 室内

描述已自动生成

Size Consumption Result(Partial):



Searching time（query case=”farmer architecture”）



Searching time is 0.1811s, which is slightly faster than unoptimized one.

1. Optimization Analysis

The results are very unexpected, the test results show that the system before the optimization has less memory consumption and only slightly longer running time. And for searching time, after measuring the search time of the same query and processing the average value, there is no large difference in search time before and after optimization. In theory, we put all the file paths into a list to reduce the call times of the path so as to reduce the size requirement and time consumption. We analyze the possible reasons as follows：

1. We originally stored the token and path in the form of {token，path}, and later changed to token:pathList(index), according to the characteristics of the python language, we read pathList(index) every time. It will consume more time. The process of creating a pathList will add many operations to it which reduce the performance for the first-time system runs.
2. In the posting() function, we introduce pathList after optimization and execute invertedIndex['.'] = pathList. The increase of this operation directly causes the posting time to rise from 8003ms to 91465ms.
3. Everything in Python is an object, and each object needs to maintain a reference count, adding extra work. We have added the huge list object of pathList, which has reduced the running performance to some extent.