

"Object Oriented Programming" Final Report

Class: D1

Name: Wang HongBo

Student ID: 18098532-I011-0018

Project name: OOP_Project2.cpp

Start and end date of completion: 2020/06/01 - 2020/06/06

Part A

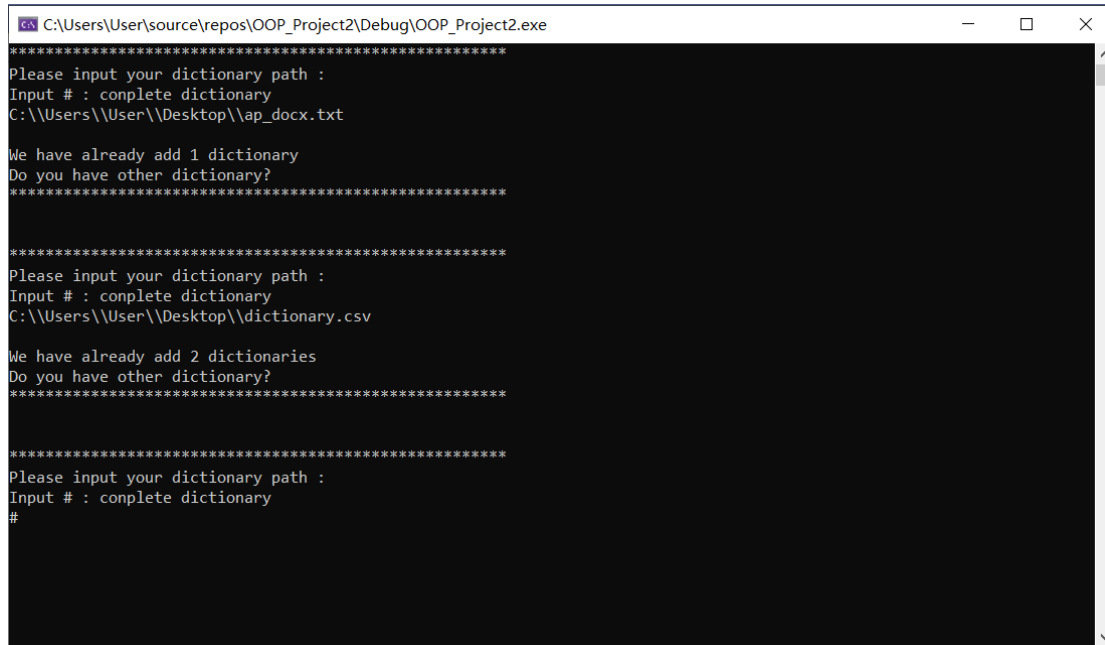
• features of system:

In order to solve the user's puzzle word query problem, we performed two steps to complete it

- **fill_completions ()**: Get the absolute path entered by the user, and return a dictionary that has been built
 - (1) **IoTxt (fd)**: input the fd path entered by the user into the program
 - (2) **txtLine ()**: Count the number of lines in a txt file and create a vector array
 - (3) **ioOut ()**: Method to input txt into the program
 - (4) **stringArrarCopy ()**: output the array deal_with.h header file
 - (5) **deal_csv()**: deal the .csv file
 - (6) **getStringArrayCopy()**: Extract the vector array from the DealWith header file
 - (7) **insert()**: Sum up the vector array into the main vector, waiting for the merged words
- **deal_vector ()**: All the words that are processed and stored in the vector array, use map to merge the words, and add the number of them, and finally get a vector array with the number of words from most to few
 - (1) **sort ()**: Sort vector<pair<int,string>>
 - (2) **reverse ()**: Invert the vector array to arrange words from more to less
 - (3) **substr ()**: Separate the form of (word, quantity) in the vector array, key->word, value->quantity
 - (4) **map ()**: Combine the same key, add the number of values
- **find_completions ()**: We first compare the length of the word entered by the user with the word in vector<string>, if the length is the same, proceed to the next step, compare whether each known letter is equal, and if all are equal, insert it into a new vector<string> array

• user manual of system:

When "Please input your dictionary path :" is displayed, enter your first path, then press Enter, enter repeatedly, add all the files, and the process of building the dictionary has been completed , Complete the construction dictionary input "#", exit the construction dictionary



```
C:\Users\User\source\repos\OOP_Project2\Debug\OOP_Project2.exe
*****
Please input your dictionary path :
Input # : complete dictionary
C:\Users\User\Desktop\ap_docx.txt

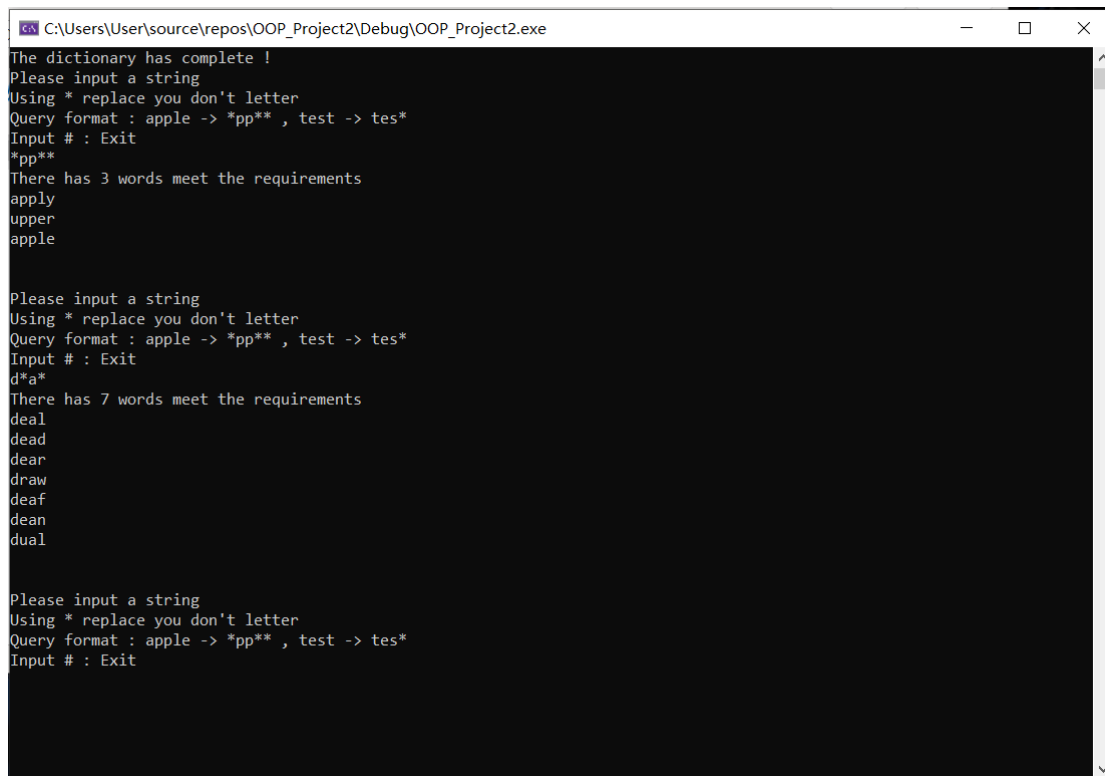
We have already add 1 dictionary
Do you have other dictionary?
*****

Please input your dictionary path :
Input # : complete dictionary
C:\Users\User\Desktop\dictionary.csv

We have already add 2 dictionaries
Do you have other dictionary?
*****

Please input your dictionary path :
Input # : complete dictionary
#
```

When the user enters, replace the unknown letters with *, such as: apple->*p*l*, test->*s*, press Enter after searching, Complete the search and enter "#" to exit the query word



```
C:\Users\User\source\repos\OOP_Project2\Debug\OOP_Project2.exe
The dictionary has complete !
Please input a string
Using * replace you don't letter
Query format : apple -> *pp** , test -> tes*
Input # : Exit
*pp**
There has 3 words meet the requirements
apply
upper
apple

Please input a string
Using * replace you don't letter
Query format : apple -> *pp** , test -> tes*
Input # : Exit
d*a*
There has 7 words meet the requirements
deal
dead
dear
draw
deaf
dean
dual

Please input a string
Using * replace you don't letter
Query format : apple -> *pp** , test -> tes*
Input # : Exit
```

- **source codes of system:**

Please see the attached file OOP_Project.cpp

Time complexity is $O(\log N)$

Average running time:

(1) Dictionary preparation time: 3472ms

(2) Find word time: 16ms

Procedural highlights the author believes:

- (1) In fill_completions, the combine_PartA () algorithm: Using map(), the search and insertion speed of map is very fast ($\log N$), which makes the merger very simple and fast
- (2) In find_completions (), First compare the number of words, if the number is the same, then make a more accurate comparison, this process makes the speed of find much faster

Part B

(1)What is the most efficient way you can think of to find predictive text?

First, we will sort and sort, and then use the dichotomy to find the first word that matches the first letter, and finally start the search in the first word of the first letter, until the first letter changes, propose a prediction text that meets the requirements, and finally Output result

(2)What is the most efficient way you can think of to find crosswords?

First determine the length of the word, under the condition of equal length, perform a one-to-one matching, if all match, then save it to the array, and finally output the array

(3)How expensive are they?

The time complexity of the first one is $\log N$, and the time complexity of the second one is n

(4) Finally, explain your modified OOP design for new requirements?
Separate txt file and csv file, and finally store in a vector array, then use map for deduplication operation, and then put it in vector to sort according to the frequency of the word, and finally get a word array arranged from most to few