**Program Report**

**I. Task Description**

Given two files “invoice.txt” and “suppliernames.txt”. The first file (“invoice.txt”) contains the words of an invoice (see invoice.pdf) generated by the OCR engine. Each line has the text of the word and information about the location of the word in the document. For example, the line:

{'pos\_id': 1, 'cspan\_id': 1, 'rspan\_id': 0, 'right': 87.06, 'word': 'INVOICE', 'line\_id': 0, 'top': 4.0, 'height': 1.52, 'width': 10.51, 'left': 76.55, 'page\_id': 1, 'word\_id': 1}

contains the word 'INVOICE', the page it came from (page\_id = 1), the line number (line\_id = 6), the position of the word in that line (pos\_id = 0).

The second file, “suppliernames.txt”, contains a list of supplier names. Part of the file is listed below:

22636206,Z WAIOURU

22636196,Vivace

22635843,5601689-0001

22635842,WD DAVENPORT & CO LIMITED

The requirement is to find the supplier name of that invoice by matching the given list of supplier names to the invoice.

**II. Program Details (Version 1)**

The program is written with Python (3.4). The main steps of the program are listed below:

1. Information extraction

2. Invoice reconstruction

3. Name search

Now I explain the details for each step.

***1. Information extraction***

The main goal of this step is to extract out the useful information from the txt files.

For the names of suppliers, the “ID” and “SupplierName” are always separated by “,”. In addition, none of the supplier names contains “,”. Therefore, it is reasonable to use “,” as a token which identifies the border between “ID” and “SupplierName” in the file.

It is also noteworthy that the formats of capitalization is very irregular (the names are not capitalized according to the same rule). In order to remove the effect of mismatching due to the differences of capitalization, I changed all the names into lower letters.

Furthermore, to make my solution scalable to hundreds of thousands of supplier names, I also sorted the list of the supplier names. This would facilitate the implementation of “binary search” in Step 3.

It is trickier to deal with the information contained in the file “invoice.txt”, since different punctuations are sometimes contained in the field ‘word’, for example “ashland,” (at line index 7, position 0). Therefore, I use the token “\’” (single quotation mark in Python) to identify the information needed. In this program, I extracted the information including “page\_id”, “line\_id”, “pos\_id” and “word” (content of word obtained by OCR).

The information extracted from “invoice.txt” is written into the file “word\_list.xlsx”:

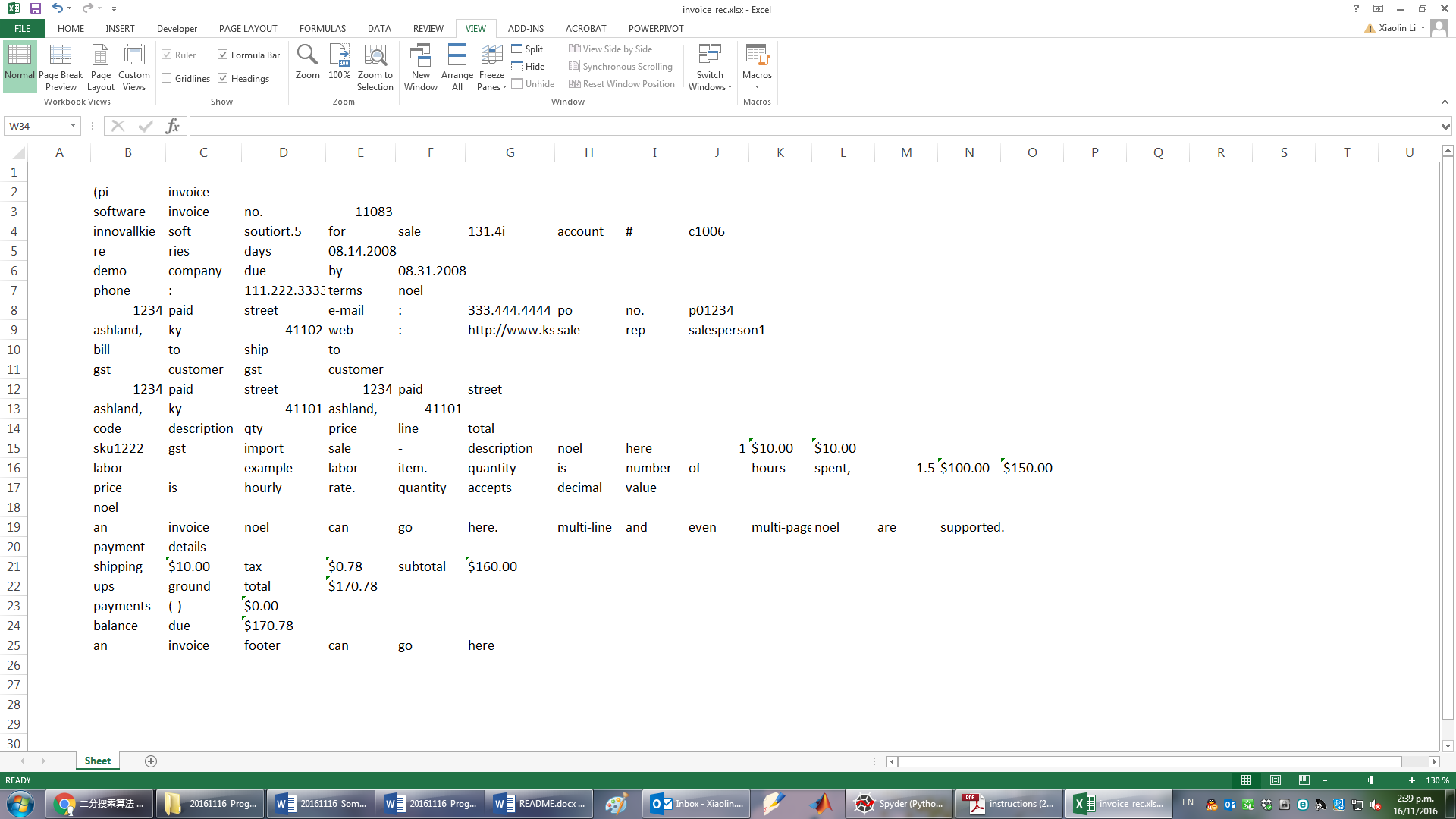
- First column: page number

- Second column: line number

- Third column: position of word

***2. Invoice reconstruction***

This step is to reconstruct the invoice according to the information of page number, line number and position index. The output is written into the file “invoice\_rec.xlsx” (as shown in the picture below).



**Fig. 1. Reconstructed Invoice from the output of OCR (“invoice\_rec.xlsx”)**

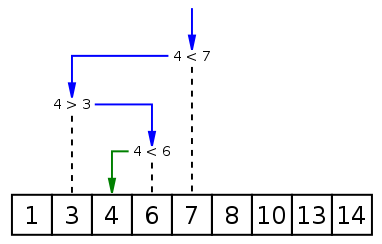
***3. Name search***

Though the requirement is to find out all the supplier names included in the invoice, it seemed to me that further information is needed to “exactly” identify the “supplier name” (e.g. location, always on the upper left corner). My concern is that the customer names (“Test Customer”) might also be listed in “suppliernames.txt” in future applications.

With current limited information, in my program I could just searched all the possible supplier names (listed in “suppliernames.txt”) appeared in “invoice.txt”. The difficulty in this step is that the name “demo company” has been divided by OCR as two words. My solution is to compare each word in “invoice\_rec.xlsx” with the “first word” of supplier names. When the first word matches, then the program continues to compare the following words, and if the sequential words match the rest of the words in the supplier name, then a name of the supplier is found. And the names are written into the file “Sup\_name\_found.xlsx”.

The function “binary\_search” returns the row index of the supplier if the input word matches the “first word” of supplier names. For example, in this program, word “demo” would correspond to the first word of “demo company”, while “total” corresponds to the first word of “total harbour city guards ltd”.

In the implementation, to make the program scalable, a binary search algorithm (comparing strings alphabetically in this case) is used. The time complexity is, where the number of suppliers in the list is.

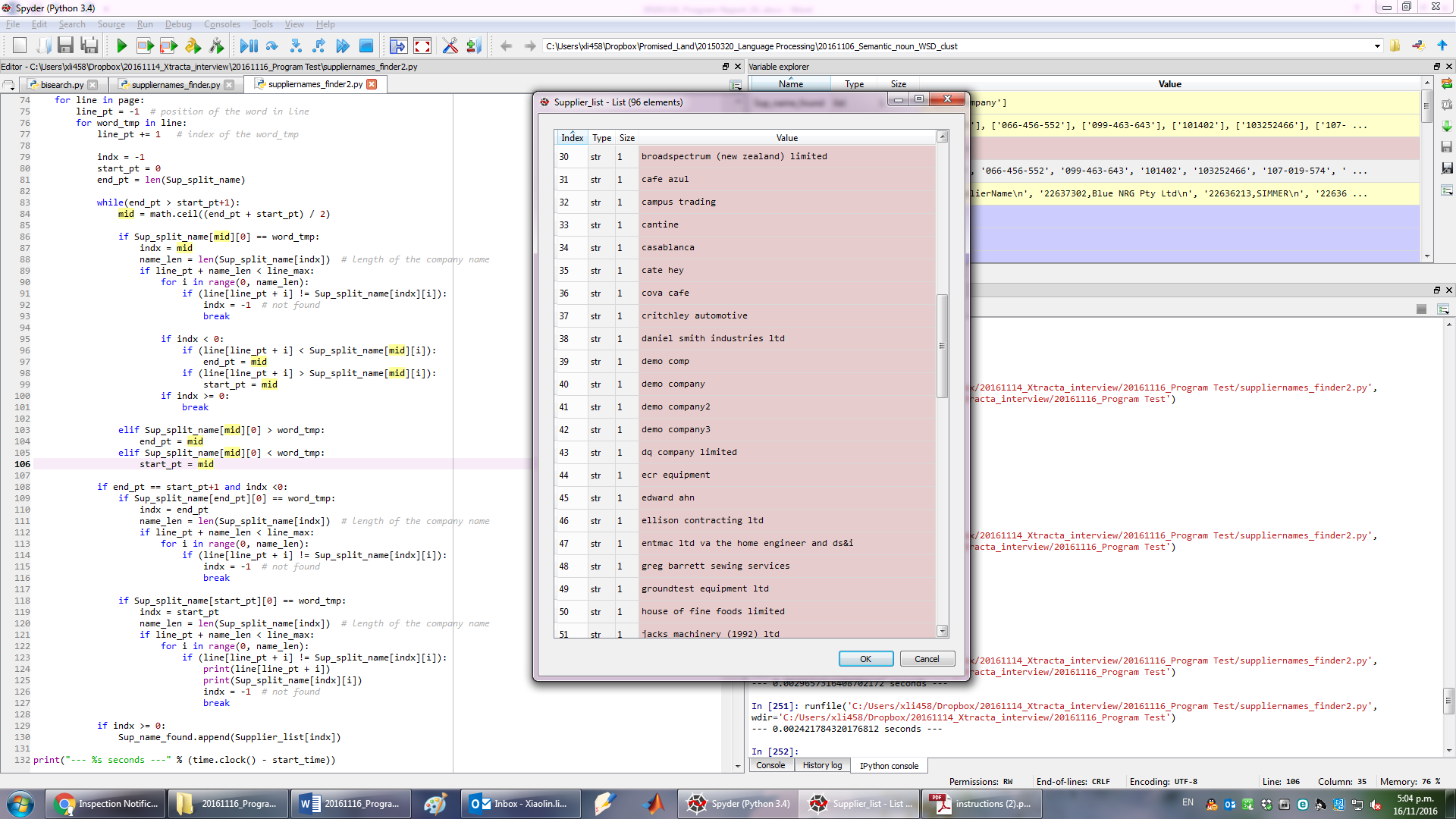


**Fig. 2. Binary Search Algorithm**

**III. Further Considerations**

***Implementation Version 2***

In the implementation of “suppliernames\_finder.py”, I think of one issue which though does not exist in this data set. When there are many supplier names, and there could be many different companies having the same first word, e.g. “demo company”, “demo comp”, “demo company2”, “demo company3” etc. In that case, either the function “binary\_search” should return the list of all row indices if a word matches “demo”. But if the function “binary\_search” only returns one possible index (as I did in “suppliernames\_finder.py”), then it would be problematic in this case.



**Fig 3. Supplier names started with “demo” (modified data set)**

A improved implementation to solve the above problem should be when the word “word\_tmp” matches the first word of a supplier’s name, then it is better to see directly whether the rest of consequential words match or not. If the entire name of the supplier is matched, then the loop could be terminated safely. But this is would break the structure of the “binary\_search” function, and therefore in “suppliernames\_finder2.py”. Since in “suppliernames\_finder2.py”, I did not write another separate function, this implementation has more rows of codes.

***Errors from OCR***

It seemed that the OCR engine would make many mistakes, but the issue is that whether it would make mistakes at the crucial part of the invoice. For example, “l1083” is misinterpreted as “11083”. In addition, the logo image (upper left corner of the invoice) could not be correctly recognized.





**Fig 4. Mistakes from OCR**

I think the final overall performance of the system has to be tested, which means one may need to manually review a large number of the scanned documents, and compare the final error rate.