Zhaoning Yu (Group Five)

Professor Ray Renner

Group member: Zhaoning Yu, Hang Ruan, Shaochuan Cheng, Bing Li

CSCI 6221 Advanced Software Paradigms

03/24/2019

Group Project Report: Division of labor

Web crawler part was done by Hang Ruan. After that, Shaochuan Cheng performed word segmentation on the collected data. Then, Bing Li showed the processed data in charts. Finally, Zhaoning Yu helped to modify the code, integrated the code of each part then summarized the conclusions, and completed the report. The whole project used Go to implement.

Part 1. web Crawler

In this part, because there are already many mature crawler frameworks on the web, our group decided to use the framework to complete the crawler part. Colly is a lightning fast and elegant scraping framework. It is easy to use it to build a complex asynchronous website crawlers.

Firstly, we set up our crawler and created the data structure hash table to store our results. We created a new struct which will will represent an article, and contains all the fields we are going to be collecting with our crawler.

With this done, we began writing our main function. To create a new crawler we created a NewCollector, which itself returns a Collector instance. The NewCollector function takes a list of functions which are used to initialize our crawler. In our case we are only calling one function within our NewCollector function, which is limiting our crawler to pages found on required page like “stackoverflow.com”.

After that, we placed some limits on our crawler. As Golang is a very performant and many websites are running on relatively slow servers we probably want to limit the speed of our crawler. Here, we were setting up a limiter which matches everything contains keywords like "stackoverflow" in the URL. By setting the parallelism to 1 and setting a delay of a second, we ensured that we only crawl one URL a second.

To collect data from our target site, we created a clone of our Colly collector. We also created a slice of our 'Article' struct to store the results we will be collecting. We also added a callback to our crawler which will fire every time we make a new request, this callback just prints the URL which are crawler will be visiting.

We then added another "OnHTML" callback which is fired every time the HTML is returned to us. This was attached to our original Colly collector instance and not the clone of the Collector. Here we passed in CSS Selector, which pulls out all of the href's on the page. We can also use some logic contained within the Colly framework which allows us to resolve to URL in question. If URL contains 'python', we submit it to our cloned to Collector, while if 'python' is absent from the URL we simply visit the page in question. This cloning of our collector allowed us to define different OnHTML parsers for each clone of original crawler.

Then, We added an 'OnHTML' callback to our 'detailCollector' clone. Again we used a CSS Selector to pull out the content of each post contained on the page. From this we can extract the text contained within the post's "H3" tag. We finally, picked out all of the 'div' containing the id 'questions', we then iterate over all the elements pulling out our code snippets. We then add our collected data to our slice of Articles.

All there is left to do, is start of the crawler by calling our original collector's 'Visit' function with our start URL. The crawler should finish within around 20 seconds. Colly makes it

very easy to write powerful crawlers with relatively little code. It does however take a little while to get used the callback style of the programming.

Word Segmentation

In the word segmentation part, we firstly filtered the data crawled by the crawler and removed the symbols and expressions. Then we iterated over the data and compared the data to the target keywords. After that, we used a variable to record the number of times a keyword appears. Finally, we sorted the keywords in the order of the number of occurrences.

Data Representation

In the data representation section, we used the package plot to graph the data we have gotten. Types implementing the Plotter interface can draw to the data area of a plot using the primitives made available by the package plot. We have drawn two graphs showing the number of occurrences of the target keyword in a question and the number of the question containing the target keyword.

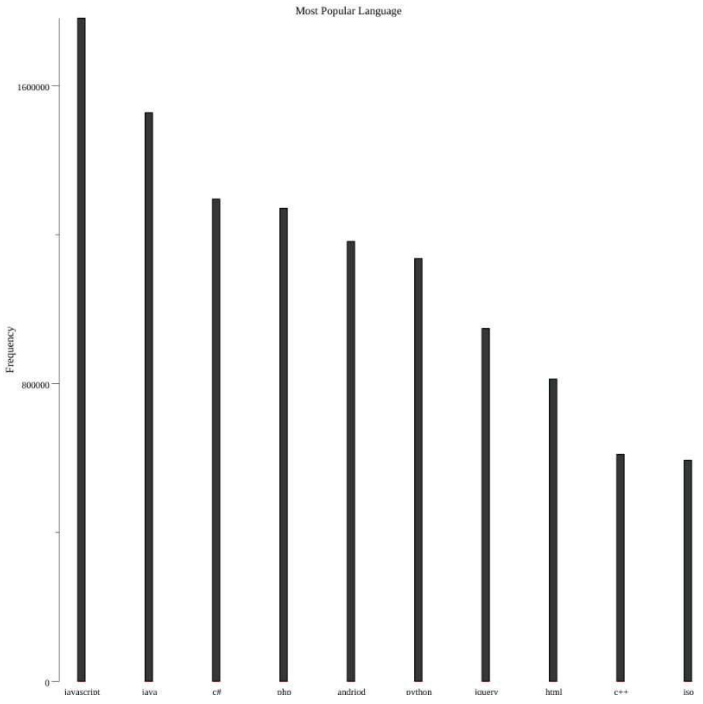

Fig. 1. The most popular language

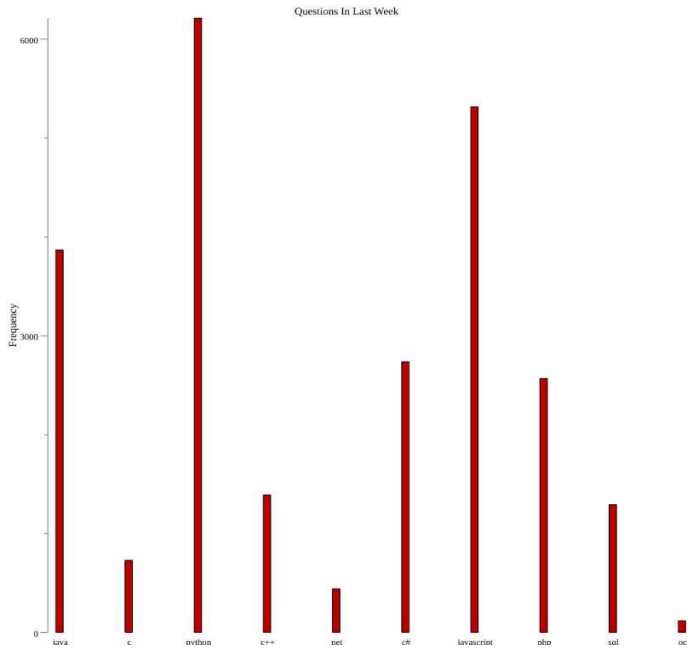

Fig. 2. Questions in the last week of top 10's languages in TIOBE Index for March 2019

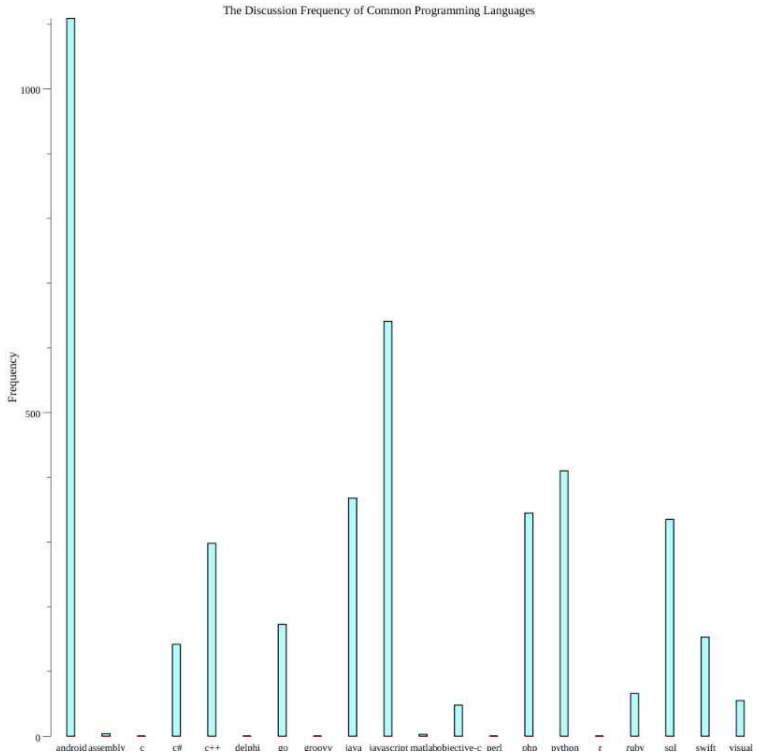

Fig. 3. The discussion frequency of common programming languages

Result and Conclusion

It can be seen from the charts that the top ten most frequently discussed languages are JavaScript, Java, C#, PHP, Android, Python, jQuery, html, C++, IOS. Comparing the data we obtained with the TIOBE Index for March 2019, we can see that most popular languages have a high discussion rate in the forum. Only objective-c doesn't have much attention, and last week there were only 117 questions about it.

From this project, we learned how to use Go to build a web crawler and how to use Go to do data processing and analyze the data. Go is a really light language to program and learning it benefits us a lot.

Works Cited

"Colly documentation." Web. <http://go-colly.org/docs/>.

"The go programming language." Web. <https://golang.org/doc/>.

"package plot." Web. <https://godoc.org/gonum.org/v1/plot>.