Single Crawler Description

We'd like you to write a simple web crawler in a programming language of your choice. Feel free to either choose one you're very familiar with or, if you'd like to learn some Go, you can also make this your first Go program! The crawler should be limited to one domain - so when crawling tomblomfield.com it would crawl all pages within the domain, but not follow external links, for example to the Facebook and Twitter accounts. Given a URL, it should output a site map, showing which static assets each page depends on, and the links between pages.
Ideally, write it as you would a production piece of code. Bonus points for tests and making it as fast as possible!
This exercise is not meant to test whether you can program at all, but instead you should think of it as a software design test. This means that we care less about a fancy UI or rendering the resulting sitemap nicely and more about how your program is structured, the trade-offs you've made, what behavior the program exhibits etc..

Dependencies

During my implementation I tried not to spend time “reinventing the wheel” so I have used a few packages written by other people. In practice/production, it probably isn’t best to use packages that are from outside of the standard go libraries/ the company. Install these via the standard `go get <package>`

- github.com/asaskevich/govalidator → used to validate that a URL is valid
- github.com/golang/glog → used for logging
- golang.org/x/net/html → parsing html documents

Documentation

You will find 3 or 4 UML documents describing the overall design of the program in this google drive. The one you are most interested in is probably Single Domain Crawler UML, which is an overview of the entire program in a simple, concise way. The other documents were mainly for me, where I considered all the possible states the program can get into - and made sure that there were well defined contingencies to exit those states when desired.

Inside of the code, every function header is well commented so that if needed, we could use a documentation generator to produce a document library for this code. You can also find lots of comments inside of the functions themselves - I did this mainly because I wanted it to be easy for you guys to look through this implementation. In general, for production code, many of the comments inside of the code are not necessary.

Robustness/Testing

Included in this package are unit tests for all functions. These tests were designed to break the package as best as I could possible, to prove that the program is resilient and robust enough for production. These tests can be found in the directory “petitcrawlertest”. The tests in

file ‘petitcrawler_commandline_test.go’ are entire program tests, that will run the crawler. The tests in ‘petitcrawler_internal_test.go’ are unit tests on each function.

To ensure that the program never reaches an unexpected state, you can view the diagrams in the folder, where I have outlined all possible states and contingencies. The unit tests are meant to prove the robustness, by testing each function for all possible good/bad/malicious inputs. **When you run the test suite, make sure to specify a URL to the command line also**. Run all of the tests with the usual command in the petitcrawlertest directory:

- Go test -url http://exmple.com

Scalability

We could transform this program into one that runs over multiple single- domains. Think about if instead of discarding external links, we wanted to create a separate crawler and sitemap for those. Or what if we wanted to crawl an extremely large site, and we’d like to distribute the work among multiple computers? There would be some things we would need to change if we were intending to make this program scalable, listed are a few.

- Write Sitemap information to a database - and we’d want it to write this information periodically
- Distribute the work of the crawler to multiple systems
- Take steps to ensure safe and effective communication across systems
- Introduce monitoring for watching crawlers/processes

Using the Program

You are going to find that this program is very restrictive of user input, for example, an input url always needs to have a scheme, domain, and be a valid url to be accepted. This project was kind of open-ended, so I have chosen to be very restrictive and precise. This doesn’t have to be the case for all programs I write.

Example main using the package:

```
Package main
Import ( “petitcrawler” )
Func main() {
        Crawler, err := petitcrawler.New()
         if err != nil {
                glog.Fatal("Failed to create a crawler: %s", err)
         }

          err =Mycrawler.Run()
          if err !=nil {
                glog.Fatal("Failed to run crawler: %s", err)
```

```
        }
}
```

Example command line using the package:
./test -url <URL> -maxtime 60 -log_dir=”./” -numworkers=100 -filename MySiteMap.txt

Although, the program runs fine, with defaults:
./test -url <URL>

And to display the help:
./test
./test -help