Project 2018

Graph Theory

Due: last commit on or before April 8th

This document contains the instructions for Project 2018 for Graph Theory. Please be advised that all students are bound by the Quality Assurance Framework [1] at GMIT which includes the Code of Student Conduct and the Policy on Plagiarism.

Context

The following project concerns a real-world problem – one that you could be asked to solve in industry. You are not expected to know how to do it off the top of your head. Rather, it is expected that you will research and investigate possible ways to tackle the problem, and then come up with your own solution based on those. A quick search for solutions online will convince you that many people have written solutions to the problem already, in many different programming languages, and many of those are not experienced software developers. Note that a series of videos will be provided to students on the course page to help them with the project.

Problem statement

You must write a program in the Go programming language [2] that can build a non-deterministic finite automaton (NFA) from a regular expression, and can use the NFA to check if the regular expression matches any given string of text. You must write the program from scratch and cannot use the `regexp` package from the Go standard library nor any other external library.

A regular expression is a string containing a series of characters, some of which may have a special meaning. For example, the three characters ".", "|", and "*" have the special meanings "concatenate", "or", and "Kleene star" respectively. So, 0.1 means a 0 followed by a 1, 0|1 means a 0 or a 1, and 1^* means any number of 1's. These special characters must be used in your submission.

Other special characters you might consider allowing as input are brackets "()" which can be used for grouping, "+" which means "at least one of", and "?" which means "zero or one of". You might also decide to remove the concatenation character, so that 1.0 becomes 10, with the concatenation implicit.

You may initially restrict the non-special characters your program works with to 0 and 1, if you wish. However, you should at least attempt to expand these to all of the digits, and the characters a to z, and A to Z.

You are expected to be able to break this project into a number of smaller tasks that are easier to solve, and to plug these together after they have been completed. You might do that for this project as follows:

1. Parse the regular expression from infix to postfix notation.
2. Build a series of small NFA's for parts of the regular expression.
3. Use the smaller NFA's to create the overall NFA.
4. Implement the matching algorithm using the NFA.

Overall your program might have the following layout.

```
type nfa struct {
  ...
}
func regexcompile(r string) nfa {
  ...
  return n
}
func (n nfa) regexmatch(n nfa, r sting) bool {
  ...
  return ismatch
}
func main() {
  n := regexcompile("01*0")
  t := n.regexmatch("01110")
  f := n.regexmatch("1000001")
}
```

Minimum Viable Project

The minimum standard for this project is a GitHub repository containing a single Go program that can execute regular expressions against strings over

the alphabet $0, 1$ and with the special characters ., |, and $*$. The README should clearly document how to compile, run and test your program. It should also explain how your program works, and how you wrote it. A better project will be well organised and contain detailed explanations. The architecture of the system will be well conceived, and examples of running the program will be provided.

Submissions

GitHub must be used to manage the development of the software. Your GitHub repository will form the main submission of the project. You must submit the URL of your GitHub repository using the link on the course Moodle page before the deadline. You can do this at any time, as the last commit before the deadline will be used as your submission for this project.

Any submission that does not have a full and incremental git history with informative commit messages over the course of the project timeline will be accorded a proportionate mark. It is expected that your repository will have at least tens of commits, with each commit relating to a reasonably small unit of work. In the last week of term, or at any other time, you may be asked by the lecturer to explain the contents of your git repository. While it is encouraged that students will engage in peer learning, any unreferenced documentation and software that is contained in your submission must have been written by you. You can show this by having a long incremental commit history and by being able to explain your code.

Marking scheme

This project will be worth 50% of your mark for this module. The following marking scheme will be used to mark the project out of 100%. Students should note, however, that in certain circumstances the examiner's overall impression of the project may influence marks in each individual component.

25%	**Research**	Investigation of problem and possible solutions.
25%	**Development**	Clear architecture and well-written code.
25%	**Consistency**	Good planning and pragmatic attitude to work.
25%	**Documentation**	Detailed descriptions and explanations.

Advice for students

- Your git log history should be extensive. A reasonable unit of work for a single commit is a small function, or a handful of comments, or a small change that fixes a bug. If you are well organised you will find it easier to determine the size of a reasonable commit, and it will show in your git history.

- Using information, code and data from outside sources is sometimes acceptable – so long as it is licensed to permit this, you clearly reference the source, and the overall project is substantially your own work. Using a source that does not meet these three conditions could jeopardise your mark.

- You must be able to explain your project during it, and after it. Bear this in mind when you are writing your README. If you had trouble understanding something in the first place, you will likely have trouble explaining it a couple of weeks later. Write a short explanation of it in your README, so that you can jog your memory later.

- Everyone is susceptible to procrastination and disorganisation. You are expected to be aware of this and take reasonable measures to avoid them. The best way to do this is to draw up an initial straight-forward project plan and keep it updated. You can show the examiner that you have done this in several ways. The easiest is to summarise the project plan in your README. Another way is to use a to-do list like GitHub Issues.

- Students have problems with projects from time to time. Some of these are unavoidable, such as external factors relating to family issues or illness. In such cases allowances can sometimes be made. Other problems are preventable, such as missing the submission deadline because you are having internet connectivity issues five minutes before it. Students should be able to show that up until an issue arose they had completed a reasonable and proportionate amount of work, and took reasonable steps to avoid preventable issues.

- Go easy on yourself – this is one project in one module. It will not define you or your life. A higher overall course mark should not be determined by a single project, but rather your performance in all your work in all your modules. Here, you are just trying to demonstrate to yourself, to the examiners, and to prospective future employers, that

you can take a reasonably straight-forward problem and solve it within a few weeks.

References

[1] GMIT. Quality assurance framework.
`https://www.gmit.ie/general/quality-assurance-framework`.

[2] Google. The go programming language.
`https://golang.org/`.