Computer lab 3

Måns Magnusson

August 25, 2015

Instructions

- This lab should be conducted by students **two by two**.
- The lab constists of writing a package that is version controlled on github.com with a published release.
- Both student should **contribute equally much** to the package.
- In the lab some functions can be marked with an *. These parts is only mandatory for students taking the advanced course or students working together in groups of three.
- The deadline for the lab can be found on the **webpage**
- The lab should be turned in as a url to the repository containing the package on github using **LISAM**. This should also include name, github user names and liuid of the students behind the project.

Contents

1 To create a package in R **3**
1.1 Write the R code . . . 3
1.1.1 `euclidean()` . . . 3
1.1.2 `* dijkstra()` . . . 3
1.2 Create the R package . . . 4
1.2.1 Initialize the package . . . 4
1.2.2 Document the package using `roxygen2` . . . 4
1.2.3 Include the dataset `wiki_graph` in the package . . . 5
1.2.4 Write unit tests for your package . . . 5
1.2.5 Finalize your package . . . 5
1.3 Seminar and examination . . . 5
1.3.1 Examination . . . 5

Chapter 1

To create a package in R

In this lab we will create our first R package in R. To be able to get everything to work you need to have the following software installed:

1. R
2. R-Studio (not nessecary but makes it a lot easier)
3. Git

This lab will be a walkthrough on how to create a package. This is not the only way to do this but one way that work for most.

1.1 Write the R code

In this first R package we will implement two famous algorithms, the euclidian algorithm to find the greates common divisor of two integers and Dijkstra's shortest path algorithm in a graph. For both these algorithms you will have pseudocode for the algoritm, so the job is to implement these algorithms in R. Store each function in their own R file with the name of the function.

1.1.1 `euclidean()`

The first algoritm to implement is the Euclidian algorithm to find the greatest common divisor of two numbers. The description of the algorithm with pseudocode can be found **here**. Assert that the arguments are numeric scalars or integers.

Below is an example of the `euclidan()` function.

```
euclidian(123612, 13892347912)

[1] 4

euclidian(100, 1000)

[1] 100
```

1.1.2 `* dijkstra()`

The next algorithm to implement is one of the most famous algorithms in computer science, Dijkstras algorithm. The algorithm takes a graph and an initial node and calculates the shortest path from the initial node to every other node in the graph. A decription with pseudocode can be found at the wikipedia page **here**. If you're not very familiar with graphs, vertices and edges, see **this** wikipedia page for a fast introduction.

The function should be named `dijkstra()` and have the argument `graph` and `init_node`. The graph should be a `data.frame` with three variables (`v1`, `v2` and `w`) that contains the edges of the graph (from `v1` to `v2`) with the weight of the edge (`w`). The dijkstra function should return the shortest path to every other node from the starting node as a vector.

Assert that the graph argument have the above structure and that `init_node` is a numeric scalar that exist in the graph.

Below is code to create the first graph at the wikipedia page (this is not the most memory efficient way to express the edges but it makes it easier to implement the function) and the results of the function `dijkstra()`.

```
wiki_graph <-
data.frame(v1=c(1,1,1,2,2,2,3,3,3,3,4,4,4,5,5,6,6,6),
v2=c(2,3,6,1,3,4,1,2,4,6,2,3,5,4,6,1,3,5),
w=c(7,9,14,7,10,15,9,10,11,2,15,11,6,6,9,14,2,9))

dijkstra(wiki_graph, 1)

[1]  0  7  9 20 20 11

dijkstra(wiki_graph, 3)

[1]  9 10  0 11 11  2
```

1.2 Create the R package

1.2.1 Initialize the package

To create a package can be done in many different ways. This is one suggestion on how to do it.

1. Create a new repository at GitHub (username is needed) and invite your collaborators to this repository. Initialize the repo with a README.md file (otherwise R-Studio will have a problem of cloning the repo).
2. Open the .gitignore file and add `*.Rproj` .This will make git ignore the R project file - we do not want this on github.
3. Create a project in R-Studio based on this github repository. See chapter "Git and Github" in [2] for details.
4. Create a package skeleton using the function `package.skeleton(name=''yourpackagename'')`. You can choose the name of the package freely. Remove the read-and-delete-me file that was created.
5. Fill out the `DESCRIPTION` file with what you find suitable. See chapter "Package metadata" in [2] for details.
6. In your R-Studio session configure the (package) build tools by Build -¿ Configure build tools.
   (a) Choose the package directory (ie the directory the package skeleton created)
   (b) Click that ROxygen should be used (fill in all subalternatives)
7. Put your R files in the folder R in the package folder.
8. In the Build tab in R-Studio, click "Build & Reload". You have created your own package. Clear the global environment and try that your functions is now in your searchpath.
9. Commit the new package and push it to github.

1.2.2 Document the package using `roxygen2`

The next step is to document the functions and the data using ROxygen. ROxygen makes it easy to include documentation in direct connection to the functions, making it much easier to both document and read the documentation when you inspect the code. See chapter "Object documentation" in [2].

1. Document each function. The documentation should include ...

(a) Arguments
(b) Description of the algorithm
(c) What the function returns
(d) A reference to the wikipedia page of each algorithm.

2. Document the package

Commit the documentation to github.

1.2.3 Include the dataset `wiki_graph` in the package

The next step is to include the dataset wiki_graph created above as dataset in the package. See "Data" in [2] for details on how to do this.

Document the dataset using ROxygen

1. The variables in the data.frame
2. A reference to the wikipedia page

1.2.4 Write unit tests for your package

The last step is to write uinit tests for your package. See chapter "Testing" in [2] or [1].

The tests should be designed in a way that it is possible to introduce a bug in the code and you will find out that we have introduced that bug.

Run "test package" under the Build tab in R-Studio to check that your functions passes all tests. Commit and push your tests to github.

1.2.5 Finalize your package

Now everything should be working in your package. As a final step we should check that everything works with your package. Do the following steps:

1. Check that your package is working by pressing the "check" button in R-Studio. Correct any warnings or errors, see "Checking" in [2] for details.
2. Push your final package to github and test that it is possible to install your package using the following code in R.

```
devtools:install_github("[yourusername/repo]", subdir="your subdirectory")
```

3. Create a release of your package (ex. v. 1.0) on github. Now you're done!

1.3 Seminar and examination

During the seminar you will bring your own computer and demonstrate your package and what you found difficult in the project.

We will present as many packages as possible during the seminar and you should

1. Show that the package can be built using R Studio and that all unit tests is passing.
2. Present the unit tests you've written.
3. We will try to introduce a bug in the code and check that this bug is found by the unit tests (and by git).

1.3.1 Examination

Turn in a the adress to your github repo with the package using LISAM.

Bibliography

[1] Hadley Wickham. testthat: Get started with testing. *R Journal*, 2011:1.

[2] Hadley Wickham. *R packages*. " O'Reilly Media, Inc.", 2015.