Project 1

Project

The goal of the project is to write a function, named `readInGraph()`, that reads in a graph stored in a file with a simple text format. First off, a graph is a set of vertices V and a set of edges E. Each edge is a set of two vertices. Mathematically speaking, a graph is a couple (V, E), where V is a finite set and E is a finite set of couples of vertices. Graphs can be represented in various ways. In this project, a graph is represented as a list of vertices followed by a list of vertices that are direct neighbors. As an example, this is the content of a file `graph_small.data`.

```
{1, {2, 3, 4, 5, 8}}
{2, {1}}
{3, {1, 6, 7}}
{4, {1}}
{5, {1}}
{6, {3, 9, 10}}
{7, {3}}
{8, {1}}
{9, {6}}
{10, {6}}
```

The file describes a graph with ten vertices numbered from 1 through 10. Each line in a file describes connections that can be done from a given vertex to other vertices. As an example, the first line in the file says that from the first vertex, there are edges to vertices $2, 3, 4, 5, 8$. The same goes for all other lines in the file.

The task in this project is to write the function that reads in the file in the described above format and stores it in a list with the following format. Each line of the file is to be stored in a list with two slots. The first slot, named `node` stores the vertex from which the connections are made. The second slot, named `connections`, stores vertices to which connections are made. As an example, the first line of the file is to be stored in the following list `list( node = 1, connections = c( 2, 3, 4, 5, 8))`.

There are three graphs in the archive ZIP. We will use these graphs in the following examples. We start with the smallest graph, we read it in and then print it.

```
### Reading in a small graph
gSmall <- readInGraph( path = "./graph_small.dat")
gSmall
```

```
[[1]]
[[1]]$node
[1] 1

[[1]]$connections
[1] 2 3 4 5 8


[[2]]
[[2]]$node
[1] 2

[[2]]$connections
[1] 1


[[3]]
[[3]]$node
[1] 3

[[3]]$connections
```

```
[1] 1 6 7


[[4]]
[[4]]$node
[1] 4

[[4]]$connections
[1] 1


[[5]]
[[5]]$node
[1] 5

[[5]]$connections
[1] 1


[[6]]
[[6]]$node
[1] 6

[[6]]$connections
[1]  3  9 10


[[7]]
[[7]]$node
[1] 7

[[7]]$connections
[1] 3


[[8]]
[[8]]$node
[1] 8

[[8]]$connections
[1] 1


[[9]]
[[9]]$node
[1] 9

[[9]]$connections
[1] 6


[[10]]
[[10]]$node
[1] 10

[[10]]$connections
[1] 6
```

Using the defined function, we can write some simple examples of using it. We start by making a data frame containing information on a number of direct neighbors for each vertex in the graph. Note that we use the larger graph.

```
### Reading in a medium sized graph
gMedium <- readInGraph( path = "./graph_medium.dat")

### Showing first four vertices
gMedium[1:4]
```

```
### Creating data frame with direct neighbors numbers
info <- data.frame( vertex = 1:length(gMedium),
                    "neighbors" = unlist( lapply( X = gMedium,
                                                  FUN = function( s){
                                                    length(s$connections)
                                                  }
                                                  )))

### Listing head of the data frame
head( info)
```

```
[[1]]
[[1]]$node
[1] 1

[[1]]$connections
 [1]  2  3  4  5  7  9 10 11 13 15 17 21 26 31 38 46 47 63 71 76 92


[[2]]
[[2]]$node
[1] 2

[[2]]$connections
[1]  1 58


[[3]]
[[3]]$node
[1] 3

[[3]]$connections
[1] 1


[[4]]
[[4]]$node
[1] 4

[[4]]$connections
[1]  1  8 22 44

  vertex neighbors
1      1        21
2      2         2
3      3         1
4      4         4
5      5         4
6      6         2
```

Using the data frame, we create a visualization. On the X-axis, we map vertices, while on the Y-axis, we map a number of direct neighbors.

```
png( file = "./fig1.png")
plot( info, type = "h", lwd = 4, col = "orange", axes = FALSE)
axis( side = 1, at = c( 1, seq(10, 100, 10)))
axis( side = 2, at = sort( unique( info$neighbors))
dev.off()
```

The resulting figure is presented below.

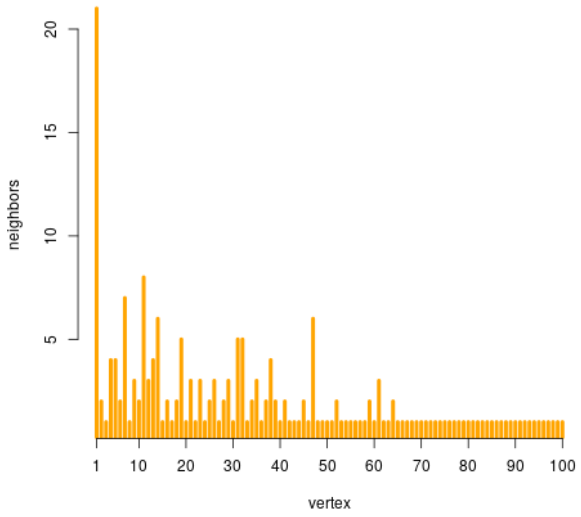

Figure 1: Visualization of the number of direct neighbors

Technical conditions

The project should be solved in a single file. Within the file, there should be a solution and an example of use. If one of these parts is missing, the solution will not be accepted. The use of additional packages is not allowed unless explicitly stated in the project's description. The R file with a solution must not contain any non-ASCII characters. The optimal coding is UTF-8.

Date: 2020-05-11 Mon 00:00
Author: Michał Ramsza
Created: 2020-05-11 Mon 20:44
Validate