Project 1

Technical requirements

- *Deadline:* 29 Jan 20024
- *Format:* A single R file encoded as UTF-8.
- *Content:* The script must contain (a) your name and student's ID number in the first line of a script as a comment, (b) a definition of the function, and (c) two or more examples of the function's applications (you can use the following ones).
- *Packages:* Only standard core packages coming with a typical R installation are allowed.
- *Instructor :* All files must be sent via email from your school account to the instructor's school account.

Project description

Recently a new version of the Ubuntu Linux was released together with a new wallpaper. The whole wallpaper is given below.

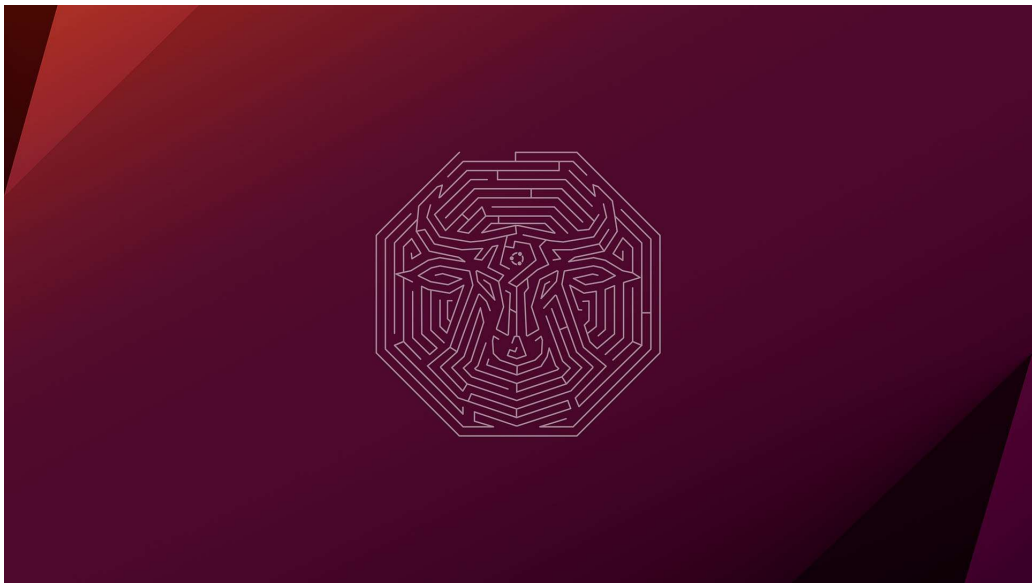

Figure 1: The new wallpaper coming with Ubuntu 23.10

The most exciting part of the wallpaper is the maze in the center. Right there in the very middle of the labyrinth is the Ubuntu Linux logo. The task is to decide if it is possible to go from the outside of the labyrinth to the logo. To this end, we need to write a function. More precisely, the goal of the project is to write a simple function `pathQ()` that takes three arguments:

- `map` that is a logical matrix representing a labirynth;
- `startPoint` that is a list with two keys `x` and `y` with assigned numerical vectors representing an entrance to the labirynth;
- `endRegion` that is a list with two keys `x` and `y` with assigned numerical vectors representing the search for region.

So first, I used the following simple script, which is not a part of the project, to cut out the labyrinth.

```
### Cleaning

### Libraries
library(magick)
```

```
### Read the image
d0 <- image_read(path = "./wallpaper.png")

### Get only the relevan part
dims <- rev(dim(as.raster(d0)))
d1 <- image_crop(image = d0, geometry = "800x800+560+140")

### Quantizing
d2 <- as.raster(image_quantize(image_normalize(d1), max = 2))

### Changing to TRUE / FALSE matrix
d3 <- array(
  ifelse(as.vector(d2) == unique(as.vector(d2))[1], TRUE, FALSE),
  dim(d2)
)

### Saving
saveRDS(object = d3, file = "./maze.RDS")
```

The resulting file with a logical matrix representing only the maze is an RDS file that you can read in with the function `readRDS()`. The following simple snippet reads in the file and displays it.

```
### Reading in the maze
d0 <- readRDS(file = "./maze.RDS")

### Plotting
plot(as.raster(d0))
```

The above snippet creates the following image.

Figure 2: The maze read in from the RDS file

The maze is rotated but this is not a problem. The logo is inside the rectangle $[387, 413] \times [322, 348]$ and the starting position is $(1, 1)$. First, I define the appropriate lists and then just call the function `pathQ()`.

```
### End region
logoPosition <- list(x = 387:413, y = 322:348)

### Starting point
startPoint <- list(x = 1, y = 1)

### Testing if there is a path leading from the starting point to the
### end region
pathQ(d0, startPoint = startPoint, endRegion = logoPosition)
```

```
[1] TRUE
```

As you can see, the function returns `TRUE`. It means that there is a path from the outside of the labyrinth to the logo. Notice that the ending point is not a point but rather a region. It makes sense for a logo.

The function in this example uses a straightforward flooding algorithm. In particular, it does not search for a path. The animation showing how this algorithm works is below.

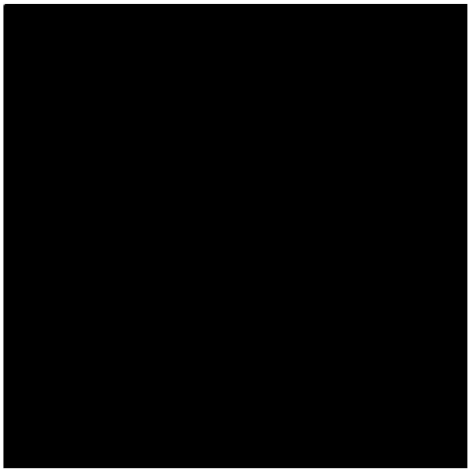

Figure 3: Flooding algorithm

You can use any algorithm you like. However, note the size of the search space here. The algorithm needs to be efficient enough to check the condition quickly. The animation above is a slowed version of what happens during the working of the algorithm I used.

Let me finish the project's description with another example. Here, I define a different ending point and show, using the `pathQ()` function, that there is no path from the outside of the maze to the endpoint.

```
### End region
endPosition <- list(x = 220:230, y = 325:335)

### Starting point
startPoint <- list(x = 1, y = 1)

### Testing if there is a path leading from the starting point to the
```

```
### end region
pathQ(d0, startPoint = startPoint, endRegion = endPosition)
```

```
[1] FALSE
```

The function returns `FALSE` corroborating the initial hypothesis. Have fun!

Data: 2023-04-26 Wed 00:00
Autor: Michał Ramsza
Created: 2023-10-25 Wed 18:09
Validate