CSC 120 (R Section)— Lab Exercise 7

This is a non-credit exercise, which you do not hand in.
You may work on your own or together with another student, as you please.

The purpose of this lab is to (a) provide more general practice in R programming, (b) give you an opportunity to try out the debugger in R / RStudio, (c) demonstrate how information can be represented in different ways, which have different advantages. You can also try finishing the function we were writing in class at the end of the last lecture.

Debugging a function. As demonstrated in the last lecture, you can ask R to stop when it starts a function by using `debug(f)`, where `f` is the name of the function. When this function is called, R will stop, and display a message. You can then go through the function one step at a time, by typing "n" (or just Enter/Return) for each step. You can also examine variables or see the values of expressions computed from them by just typing the variable name or expression (but you will need to put the name in parentheses if it is the same as a debugger command, such as "n"). You can exit the deubugger and the function you're running with "Q".

RStudio displays the position in the function you are debugging in a separate pane of the window. It also displays the variables in the current function.

You can set more than one function to be debugged. To undo debugging, use `undebug(f)`. Note that redefining a function will also have the effect of undoing debugging.

You should try using the debugger when doing the rest of this lab, when you need to find out why something isn't working. If you happen to get your functions to work first time, you can still try out the debugger just to see how it lets you step through a function.

Converting data representations. In the first assignment, you used a list of vectors to represent links between websites. Another way to represent the same information is by a matrix of 0s and 1s. If there are n websites, we can use an n-by-n matrix, in which a 1 in row i and column j means that website i links to website j (with 0 when there is no link).

You should write a function called `matrix_from_list` to convert from the list representation used in Assignment 1 to this matrix representation. You should also write the inverse function, `list_from_matrix` to go the other way. You can test these using the data sets from Assignment 1 (which are still on the web page).

Using the matrix representation. One advantage of the matrix representation is that it is very easy to go from a representation of which other sites a site has links **to** to a representation of which other sites a sites a site has links **from** — we just need to transpose the matrix (flip around rows and columns), which is done with the R function `t`. See what you get when you apply `matrix_from_list`, `t`, and `list_from_matrix` in succession to the small data set from Assignment 1.

Another use of the matrix representation is in seeing what sites link to other sites in two or more steps — eg, whether one can get from site i to site j by following two links in succession. This can be found out using matrix multiplication, done with `%*%` in R. Try multiplying the result of `matrix_from_list` applied to the small data by itself to see what you get. Can you write a function that produces a matrix of 0s and 1s representing whether you can get from one web site to another in k or fewer links? What happens to the result of such a function as you increase k?

You might find the `pmin` or `pmax` function to be useful for this. But it's a bit tricky! You can try reading `help(pmin)` and playing around with it in the R console to see what it does. And of course, you may want to use the debugger when you try to use it in your function.

The partial function from lecture. In last lecture, we started writing a function to create an n by n numerical matrix with a "peak" of 1 in the centre, 0 on the edges, and intermediate values in square rings moving outwards from the centre peak.

Here's as far as we got (this is also at `http://www.cs.utoronto.ca/~radford/csc120/partial-peak.r`):

```
peak <- function (n) {
  if (n %% 2 != 1) stop("n must be odd")
  M <- matrix(0,nrow=n,ncol=n)
  values <- seq(1,0,length=((n+1)/2))
  for (ring in 0:((n-3)/2)) {
      value <- values[ring+1]
      # We need to replace the statement below by
      # statements that store 'value' in the ring
      # of elements identified by 'ring'.
      cat("ring",ring,"value",value,"\n")
  }
  M
}
```

You can try finishing this function in this lab. You can use the debugger to help fix problems. Also, recall the strategy we were using of testing the partially-written function as soon as it does some of what is needed.

Once you have it working, you could try using `persp` or `contour` to view the peak.