Miscellaneous Problems in R

Here are some miscellaneous problems in R.

Lists

After all that work with subsetting, let's do something a bit more enjoyable.

- Write a function `nesting_depth(L)` that takes as input a list L and returns the nesting depth of L. (For example, `nesting_depth(list(1, list(2, 3), list(4, 5)))` would return 2.)

Let's call an n-domino a list with two integers, where both entries are integers from 0 to n inclusive. For example, `list(4,5)` is an n-domino (for any $n \geq 4$).

- Write a function to return a list of every unique n-domino, given n. Treat `list(a, b)` as being equivalent to `list(b, a)`. (*Hint:* If you have a list L, you can append an item to it by directly assigning something to its `(length(L)+1)`th position.)

    - It's very slow to continually append single items to lists over and over again, because you're copying over the structure with every iteration. If you can precalculate the nmber of list entries you'll need, you can *initialize* a list with `vector("list", list_size)`. (If possible, modify your code to do this and quantify the improvements in runtime using the `timeit` package.)

A valid *circle* of n-dominoes is given by a list of n-dominoes, with the following properties:

- Given two consecutive dominoes `list(n, m)` and `list(p, q)`, where the latter domino is located immediately after the former domino in the circle of n-dominoes, we require that `m == p`.

- The 1st entry of the 1st n-domino is equal to the last entry of the last n-domino.

For example, `list(list(1, 2), list(2, 3), list(3, 1))` is a valid circle of n-dominoes.

- Write a function `is_circle(L)` that returns a logical value corresponding to whether or not L is a valid circle of n-dominoes.

Data frames

Take a look at the built-in variable `letters`.

- Write a function that uses `grep()` to count the number of times each letter appears in the column names of an input dataframe. It should return a numeric vector with appropriate names and of length 26 where the ith entry is the determined frequency of letter i.

- Write a function that uses `gsub()` to modify the column names of an input dataframe, (1) changing every space (" ") into a dot (".") and (2) appending "_mod" to the end of each name.

- Write a function that *removes* the last 4 characters of every column name of an input dataframe. (If the name is 4 or fewer characters long, turn it into an empty string.) You may find `substr()` and `nchar()` helpful.

- Write a function that prints all of the row names of an input data frame joined together by an underscore ("_") between each name. You may find `paste()` useful. (*Hint:* If you're struggling to use `do.call()`, read the documentation closely! You don't have to use `do.call()`, though.)

- Given a data frame of purely numeric data, write a function that returns the entries of the data frame ordered in a "spiral" fashion starting at the top left and proceeding counterclockwise and inward.

  - For example, the function appplied to `data.frame(matrix(1:9, nrow=3))` would return `c(1, 2, 3, 6, 9, 8, 7, 4, 5)`.

  - Add a `"clockwise"` parameter to your function, defaulting to `FALSE`, which if set to `TRUE` returns the entries corresponding to a clockwise traversal of the spiral.

- Think back to the exercise yesterday about the divisibility properties of Fibonacci numbers. Let F_i denote the ith Fibonacci number, starting with $F_1 = F_2 = 1$.

  - Make a data frame where the nth column is a logical vector with `TRUE` in position m if F_m divides F_n and `FALSE` otherwise. (The data frame can be as large as you want.)

  - Make another data frame in the same way, except look at whether or not m divides n (instead of F_m and F_n).

  - Explain yesterday's computational results using the patterns that you notice today.

Miscellaneous

- Write a function that prints out the longest "run" (sequence of consecutive identical values) in an input vector. (If there's more than one, print out the one that occurs first.)

  - Rewrite your function, incorporating the usage of `rle()`.

- Find a counterexample to the following statement: By changing at most a single digit of any positive integer, we can obtain a prime number. (Memoization may be useful to speed up computation.)

The following continuation to the study of n-dominoes is **challenging** and purely optional.

Suppose that you have a single copy of every unique n-domino for some value of n.

- Write a function `make_circle(n)` that tries to construct a valid circle of n-dominoes from a *single copy* of every unique n-domino.

    - In the process of doing so, keep track of your various approaches.

    - Are there values of n for which no approach seems to work?

    - If so, can you make an argument about why you can't make a valid circle of n-dominoes for those values of n (using a single copy of every n-domino)? It may be instructive to look at the intermediate steps of your algorithm and how it fails.

    - Give a proof of your heuristic results.