Assignment 5 – R Functions, Efficiency, Debugging

STA141B Spring 2023

Professor Duncan Temple Lang

Due: 7pm Thursday, June 15th 2023

NO EXTENSIONS - at all! I have to submit grades immediately.

Submit via Canvas - Submit PDF report and R code in .R files for functions and scripts.

Background

Consider the function `utils::URLdecode()`, i.e., in R's own `utils` package. It converts/decodes percent-encoded strings such as `%24abc%5D%2B` to `$abc]+` where, e.g., the `$` was represented by `%24`. This is the opposite of what we have to do to escape/encode characters in a URL for an HTTP request, e.g., map each space to `%20`.

The `utils::URLdecode()` function is designed to work with URLs which are limited to 2048 characters. These are short and the functions works fine with these. However, it can also be useful when decoding content in the body of a POST request in an HTTP operation. See calcareers.ca.gov, specifically https://www.calcareers.ca.gov/CalHRPublic/Search/AdvancedJobSearch.aspx The body of that request is URL-encoded, starting with

```
ctl00%24ToolkitScriptManager1=ctl00%24cphMai
```

The full string has 591,977 characters. When we pass this in a call to `utils::URLdecode()`, it takes about 16 minutes on my machine! That's a problem. We need a faster version for these larger inputs.

Tasks

You are to to create two different implementations of `URLdecode`, one adapting the existing version and the other implementing a different approach that uses vectorization. We will evaluate the performance of these three functions, including the original, for various strings with different input sizes, i.e., length of the character string.

Below are several helpful steps along the way:

1. https://www.w3schools.com/tags/ref_urlencode.ASP contains a table mapping the UTF-8 percent-encoding to a character. Read that table into a named-character vector (or data.frame with two columns.)
2. Create a collection of sample input strings of different sizes containing regular characters and percent-encoded content.
3. Estimate the run-time performance of `utils::URLdecode()` as a function of the size of the input string. While it takes a character vector with more zero or more strings, we will focus on a single input string.
4. Modify a copy of the original `utils::URLdecode()` function to use preallocation of the `out` variable and insert (rather than concatenate) the values in each iteration into the appropriate elements of `out`. Ensure that `out` has the correct length when you return the result.
5. Verify that your function gets the correct answer for a variety of inputs. Describe your strategy for devising tests and confirming the results are correct.
6. Implement a version of the function that uses a vectorized approach on a single input string (`URL`) and not a while loop. The table from the first task should be of some use. **As appropriate, break this function into a collection of smaller helper functions performing specific subtasks.**
7. Describe any bugs you had in your preallocation and vectorized functions and how you found and fixed the problem.
8. Show the run-time curves for all three functions - `utils::URLdecode()`, the preallocation version and the vectorized version.
   - Does one out perform the others?
9. Time the functions on the 591K percent-encoded string from the CalCareers.ca.gov HTTP request available here.
   - How good was your estimate of the run-time for `utils::URLdecode()`?
   - Which function was fastest?

Submit

- the R code for your functions in one or more R files
- the R script to test the functions
- the PDF of your report containing the plots