Advent of Code 2018

Maria Paula Caldas

2018-12-02

Contents

Session info

I will use the following packages:

```
library(tidyverse)
```

My setup at the time:

```
## Session info -------------------------------------------------------------

##  setting  value
##  version  R version 3.5.0 (2018-04-23)
##  system   x86_64, darwin15.6.0
##  ui       X11
##  language (EN)
##  collate  en_US.UTF-8
##  tz       Europe/Paris
##  date     2018-12-02

## Packages -----------------------------------------------------------------

##  package    * version date       source
##  assertthat   0.2.0   2017-04-11 CRAN (R 3.5.0)
##  backports    1.1.2   2017-12-13 CRAN (R 3.5.0)
##  base       * 3.5.0   2018-04-24 local
##  bindr        0.1.1   2018-03-13 CRAN (R 3.5.0)
##  bindrcpp   * 0.2.2   2018-03-29 CRAN (R 3.5.0)
##  bookdown     0.7     2018-02-18 CRAN (R 3.5.0)
##  broom        0.5.0   2018-07-17 CRAN (R 3.5.0)
##  cellranger   1.1.0   2016-07-27 CRAN (R 3.5.0)
##  cli          1.0.0   2017-11-05 CRAN (R 3.5.0)
##  colorspace   1.3-2   2016-12-14 CRAN (R 3.5.0)
##  compiler     3.5.0   2018-04-24 local
##  crayon       1.3.4   2017-09-16 CRAN (R 3.5.0)
##  datasets   * 3.5.0   2018-04-24 local
##  devtools     1.13.5  2018-02-18 CRAN (R 3.5.0)
##  digest       0.6.16  2018-08-22 cran (@0.6.16)
##  dplyr      * 0.7.6   2018-06-29 CRAN (R 3.5.1)
##  evaluate     0.10.1  2017-06-24 CRAN (R 3.5.0)
##  forcats    * 0.3.0   2018-02-19 CRAN (R 3.5.0)
##  ggplot2    * 3.0.0   2018-07-03 CRAN (R 3.5.0)
##  glue         1.3.0   2018-07-17 cran (@1.3.0)
##  graphics   * 3.5.0   2018-04-24 local
##  grDevices  * 3.5.0   2018-04-24 local
##  grid         3.5.0   2018-04-24 local
##  gtable       0.2.0   2016-02-26 CRAN (R 3.5.0)
##  haven        1.1.2   2018-06-27 CRAN (R 3.5.0)
##  hms           0.4.2   2018-03-10 CRAN (R 3.5.0)
##  htmltools     0.3.6   2017-04-28 CRAN (R 3.5.0)
##  httr          1.3.1   2017-08-20 CRAN (R 3.5.0)
##  jsonlite      1.5     2017-06-01 CRAN (R 3.5.0)
##  knitr         1.20    2018-02-20 CRAN (R 3.5.0)
##  lattice       0.20-35 2017-03-25 CRAN (R 3.5.0)
##  lazyeval      0.2.1   2017-10-29 CRAN (R 3.5.0)
##  lubridate     1.7.4   2018-04-11 CRAN (R 3.5.0)
##  magrittr      1.5     2014-11-22 CRAN (R 3.5.0)
##  memoise       1.1.0   2017-04-21 CRAN (R 3.5.0)
##  methods     * 3.5.0   2018-04-24 local
##  modelr        0.1.2   2018-05-11 CRAN (R 3.5.0)
##  munsell       0.4.3   2016-02-13 CRAN (R 3.5.0)
##  nlme          3.1-137 2018-04-07 CRAN (R 3.5.0)
##  pillar        1.2.2   2018-04-26 CRAN (R 3.5.0)
##  pkgconfig     2.0.1   2017-03-21 CRAN (R 3.5.0)
##  plyr          1.8.4   2016-06-08 CRAN (R 3.5.0)
##  purrr       * 0.2.5   2018-05-29 CRAN (R 3.5.0)
##  R6            2.2.2   2017-06-17 CRAN (R 3.5.0)
##  Rcpp          0.12.16 2018-03-13 CRAN (R 3.5.0)
##  readr       * 1.1.1   2017-05-16 CRAN (R 3.5.0)
##  readxl        1.1.0   2018-04-20 CRAN (R 3.5.0)
##  rlang         0.2.2   2018-08-16 CRAN (R 3.5.0)
##  rmarkdown     1.9     2018-03-01 CRAN (R 3.5.0)
##  rprojroot     1.3-2   2018-01-03 CRAN (R 3.5.0)
##  rstudioapi    0.7     2017-09-07 CRAN (R 3.5.0)
##  rvest         0.3.2   2016-06-17 CRAN (R 3.5.0)
##  scales        0.5.0   2017-08-24 CRAN (R 3.5.0)
##  stats       * 3.5.0   2018-04-24 local
##  stringi       1.1.7   2018-03-12 CRAN (R 3.5.0)
##  stringr     * 1.3.1   2018-05-10 CRAN (R 3.5.0)
##  tibble      * 1.4.2   2018-01-22 CRAN (R 3.5.0)
##  tidyr       * 0.8.1   2018-05-18 CRAN (R 3.5.0)
##  tidyselect    0.2.4   2018-02-26 CRAN (R 3.5.0)
##  tidyverse   * 1.2.1   2017-11-14 CRAN (R 3.5.0)
##  tools         3.5.0   2018-04-24 local
##  utf8          1.1.3   2018-01-03 CRAN (R 3.5.0)
##  utils       * 3.5.0   2018-04-24 local
##  withr         2.1.2   2018-03-15 CRAN (R 3.5.0)
##  xfun          0.1     2018-01-22 CRAN (R 3.5.0)
##  xml2          1.2.0   2018-01-24 CRAN (R 3.5.0)
##  yaml          2.2.0   2018-07-25 cran (@2.2.0)
```

Chapter 1

Chronal Calibration

Import puzzle imput for the day:

```r
puzzle_input <- as.numeric(readLines("data-raw/day1.txt", warn = FALSE))
```

1.1 Puzzle 1

> Starting with a frequency of zero, what is the resulting frequency after all of the changes in frequency have been applied?

Easy enough:

```r
sum(puzzle_input)
```

```
## [1] 472
```

1.2 Puzzle 2

> What is the first frequency your device reaches twice?

Let's create a cute little tibble.

```r
(tib <- tibble(
  input = puzzle_input,
  cumsum = cumsum(input),
  index = seq(1:length(input))
  ))
```

```
## # A tibble: 1,000 x 3
##    input cumsum index
##    <dbl>  <dbl> <int>
##  1   -16    -16     1
##  2    12     -4     2
##  3   -18    -22     3
##  4    -1    -23     4
##  5     5    -18     5
##  6    -8    -26     6
##  7     9    -17     7
## 8   -15    -32     8
## 9    12    -20     9
## 10     6    -14    10
## # ... with 990 more rows
```

First, let see how many frequencies have been reached more than once (i.e. have duplicates).

```
count(tib, cumsum, sort = TRUE)
```

```
## # A tibble: 1,000 x 2
##    cumsum     n
##     <dbl> <int>
##  1   -111     1
##  2   -107     1
##  3   -103     1
##  4   -100     1
##  5    -98     1
##  6    -97     1
##  7    -96     1
##  8    -95     1
##  9    -94     1
## 10    -91     1
## # ... with 990 more rows
```

Apparently none... Maybe I should do it twice?

```
tib2 <- tibble(
  input = rep(puzzle_input, 2),
  cumsum = cumsum(input),
  index = seq(1:length(input))
  )

tib2 %>%
  count(cumsum) %>%
  count(n)
```

```
## # A tibble: 1 x 2
##       n    nn
##   <int> <int>
## 1     1  2000
```

So no, need to do it more than twice.

OK, let's just keep making the vector bigger until **at least** one frequency is repeated. Also, let's just go back to base R.

```
growing_vector <- puzzle_input

while ( !any(duplicated(cumsum(growing_vector))) )
  growing_vector <- c(growing_vector, growing_vector)
```

The new vector is 256 times the size of the original input vector.

Now, let's get the frequencies:

```
cumsum_big_vector <- cumsum(growing_vector)
```

And the indices of those that are repeated:

```
indx <- duplicated(cumsum_big_vector)
```

Which allows me to get the **first** frequency that is repeated twice:

```
cumsum_big_vector[indx][1]
```

```
## [1] 66932
```

1.2.1 Wrong attempts

At first, I thought the correct answer was:

```
growing_vector[indx][1]
```

```
## [1] 18
```

… which is in fact the change in frequency that leads to the first frequency that appears twice!

Chapter 2

Inventory Management System

Import puzzle imput for the day:

```
puzzle_input <- readLines("data-raw/day2.txt", warn = FALSE)
```

2.1 Puzzle 1

> What is the checksum for your list of box IDs?

```
any_rep <- function(id, rep = c(2, 3)) {
  count_per_letter <- map_int(letters, ~ str_count(id, .x))
  any(count_per_letter == rep)

}

tibble(
  input = puzzle_input,
  any_twice = map_lgl(input, any_rep, rep = 2),
  any_thrice = map_lgl(input, any_rep, rep = 3)
  ) %>%
  summarise(n_twice = sum(any_twice), n_thrice = sum(any_thrice)) %>%
  mutate(cumcheck = n_twice * n_thrice)
```

```
## # A tibble: 1 x 3
##   n_twice n_thrice cumcheck
##     <int>    <int>    <int>
## 1     247       31     7657
```

2.2 Puzzle 2

> What letters are common between the two correct box IDs? (In the example above, this is found by removing the differing character from either ID, producing `fgij`.)

This one took me a while, but I learned a lot:

- Never forget to vectorise functions, especially those that are going to go through a `dplyr::mutate()`

- `purrr::cross_df()` is awesome, although not the right tool for this type of problem (I end up with twice the amount of combinations that I need)

```r
are_almost_same <- function(vector1, vector2) {

  are_almost_same_ <- function(string1, string2) {

    chars1 <- str_split(string1, "")[[1]]
    chars2 <- str_split(string2, "")[[1]]

    sum(chars1 == chars2) == 25
  }

  map2_lgl(vector1, vector2, are_almost_same_)
}

get_common_letters_ <- function(string1, string2) {

  chars1 <- str_split(string1, "")[[1]]
  chars2 <- str_split(string2, "")[[1]]

  paste0(chars1[chars1 == chars2], collapse = "")

}

puzzle_input %>%
  list(x = ., y = .) %>%
  cross_df(.filter = `==`) %>%
  mutate(are_almost_same = are_almost_same(x, y)) %>%
  filter(are_almost_same) %>%
  slice(1) %>% # because of the cross_df()
  {get_common_letters_(.$x, .$y)}
```

```
## [1] "ivjhcadokeltwgsfsmqwrbnuy"
```

Chapter 3

Methods

We describe our methods in this chapter.

Chapter 4

Applications

Some *significant* applications are demonstrated in this chapter.

4.1 Example one

4.2 Example two

Chapter 5

Final Words

We have finished a nice book.