Advent of Code 2021 - Day 4

Matthias Zepper

12/04/2021

Solutions to the Day 4 tasks of the Advent of code

Part 1

Bingo is played on a set of boards each consisting of a 5x5 grid of numbers. Numbers are chosen at random, and the chosen number is marked on all boards on which it appears. (Numbers may not appear on all boards.) If all numbers in any row or any column of a board are marked, that board wins. (Diagonals don't count.)

The score of the winning board can now be calculated. Start by finding the sum of all unmarked numbers on that board. To guarantee victory against the giant squid, figure out which board will win first. What will your final score be if you choose that board?

Reading in the input data

This time, reading the input data is tricky, because we have a plain text file with data in two different shapes.

```
bingo_draws <- as.numeric(read.delim("day_4_input.txt", nrows=1, sep=",", header=FALSE))
print(head(bingo_draws))
```

```
## [1] 83 69 34 46 30 23
```

```
bingo_boards <-
  read.delim(
    "day_4_input.txt",
    skip = 1,
    blank.lines.skip = TRUE,
    allowEscapes = TRUE,
    header = FALSE
  )
bingo_boards <-
  unlist(lapply(bingo_boards, function(x) {
    temp <- unlist(strsplit(x, split = " "))
    temp <- temp[temp != ""]
    return(as.numeric(temp))
  }))

bingo_boards_matrices <-
  as.list(seq(1, length(bingo_boards), by = 25))
bingo_boards_matrices <- lapply(bingo_boards_matrices, function(n) {
  t(matrix(bingo_boards[c(n:(n + 24))], nrow = 5, ncol = 5))
})

print(bingo_boards_matrices[[1]])
```

```
##      [,1] [,2] [,3] [,4] [,5]
```

```
## [1,]   68   73   98   51   49
## [2,]   82   56   87   64    8
## [3,]   46    7   21   38   30
## [4,]   66    5   86   97   74
## [5,]   60   63   76   55   39
```

Determine if a board is completed for a given series

Now that we have the data available in a proper shape, we will write a function `check_bingo_board` that accepts such a matrix and a series of value from `bingo_draws` and checks, if any of the rows or columns are completed.

A subfunction `evaluate_series` will evaluate the intersection of the draws and the rows respectively columns.

```
evaluate_series <- function(series,draws){all(is.element(series,draws))}
```

```
check_bingo_board <- function(bingoboard,draws){
  rows <- apply(bingoboard,1,evaluate_series,draws=draws)
  cols <- apply(bingoboard,2,evaluate_series,draws=draws)
  any(c(cols,rows))
}
```

Now apply this function to each of the boards within a while-loop:

```
bingo_boards_eval <- logical(length(bingo_boards_matrices))
i <- 0

while (!any(bingo_boards_eval)) {
  i <- i + 1
  draws <- bingo_draws[c(1:i)]
  print(paste("Evaluating ", paste(draws, collapse = ",")))
  bingo_boards_eval <-
    sapply(bingo_boards_matrices, check_bingo_board, draws = draws)
  if (any(bingo_boards_eval)) {
    print(paste("Board number", which(bingo_boards_eval), "for the win!"))
  }
}
```

```
## [1] "Evaluating  83"
## [1] "Evaluating  83,69"
## [1] "Evaluating  83,69,34"
## [1] "Evaluating  83,69,34,46"
## [1] "Evaluating  83,69,34,46,30"
## [1] "Evaluating  83,69,34,46,30,23"
## [1] "Evaluating  83,69,34,46,30,23,19"
## [1] "Evaluating  83,69,34,46,30,23,19,75"
## [1] "Evaluating  83,69,34,46,30,23,19,75,22"
## [1] "Evaluating  83,69,34,46,30,23,19,75,22,37"
## [1] "Evaluating  83,69,34,46,30,23,19,75,22,37,89"
## [1] "Evaluating  83,69,34,46,30,23,19,75,22,37,89,78"
## [1] "Evaluating  83,69,34,46,30,23,19,75,22,37,89,78,32"
## [1] "Evaluating  83,69,34,46,30,23,19,75,22,37,89,78,32,39"
## [1] "Evaluating  83,69,34,46,30,23,19,75,22,37,89,78,32,39,11"
## [1] "Evaluating  83,69,34,46,30,23,19,75,22,37,89,78,32,39,11,44"
## [1] "Board number 69 for the win!"
```

BINGO!, this is our winning board:

```
bingo_boards_matrices[[which(bingo_boards_eval)]]
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   98   74   36   23    6
## [2,]   65   17   78   95   96
## [3,]   68   63   47   16   18
## [4,]   87   30   53   51   57
## [5,]   69   11   44   75   89
```

After execution of the loop, `bingo_boards_eval` and `i` are still available, so it is fairly easy to figure out which values are still missing:

```
setdiff(bingo_boards_matrices[[which(bingo_boards_eval)]], bingo_draws[c(1:i)])
```

```
##  [1] 98 65 68 87 74 17 63 36 47 53 95 16 51  6 96 18 57
```

Use `intersect` to get the winning values:

```
intersect(bingo_boards_matrices[[which(bingo_boards_eval)]],bingo_draws[c(1:i)])
```

```
## [1] 69 30 11 78 44 23 75 89
```

and manually apply the `evaluate_series` function to get the successful row (if any):

```
apply(bingo_boards_matrices[[which(bingo_boards_eval)]], 1, evaluate_series, draws =
        bingo_draws[c(1:i)])
```

```
## [1] FALSE FALSE FALSE FALSE  TRUE
```

or evaluate the columns by changing the MARGIN parameter of `apply`

```
apply(bingo_boards_matrices[[which(bingo_boards_eval)]], 2, evaluate_series, draws =
        bingo_draws[c(1:i)])
```

```
## [1] FALSE FALSE FALSE FALSE FALSE
```

The final solution is the sum of all unchecked numbers multiplied by the number that was just called when the board won, so just a combination of above snippets:

```
sum(setdiff(bingo_boards_matrices[[which(bingo_boards_eval)]], bingo_draws[c(1:i)])) *
  bingo_draws[i]
```

```
## [1] 41668
```

Part 2

On the other hand, it might be wise to try a different strategy: let the giant squid win. You aren't sure how many bingo boards a giant squid could play at once, so rather than waste time counting its arms, the safe thing to do is to figure out which board will win last and choose that one. That way, no matter which boards it picks, it will win for sure.

To do so, it is sufficient to tweak to the loop condition and evaluation a bit. To not clutter the output, I will only start printing when less than 5 boards are still left:

```
bingo_boards_eval <- logical(length(bingo_boards_matrices))
i <- 1

while (!all(bingo_boards_eval)) {
  draws <- bingo_draws[c(1:i)]
  bingo_boards_eval <-
    sapply(bingo_boards_matrices, check_bingo_board, draws = draws)
```

```
  if (!all(bingo_boards_eval) & sum(!bingo_boards_eval) <= 3) {
    remaining_boards <- which(!bingo_boards_eval)
    print(paste0("Evaluating ", bingo_draws[i], " (", i, "th)"))
    print(paste(
      "Board(s)",
      paste(remaining_boards, collapse = ","),
      "has/have not won yet!"
    ))
  }
  i <- i + 1
}
```

```
## [1] "Evaluating 59 (81th)"
## [1] "Board(s) 6,9,15 has/have not won yet!"
## [1] "Evaluating 73 (82th)"
## [1] "Board(s) 6,15 has/have not won yet!"
## [1] "Evaluating 33 (83th)"
## [1] "Board(s) 15 has/have not won yet!"
```

Now this board must be played to the bitter end:

```
board_rows <- logical(5)
board_cols <- logical(5)
i <- 0
while (!(any(board_cols) | any(board_rows))) {
  i <- i + 1
  board_rows <-
    apply(bingo_boards_matrices[[remaining_boards]], 1, evaluate_series, draws =
            bingo_draws[c(1:i)])
  board_cols <-
    apply(bingo_boards_matrices[[remaining_boards]], 2, evaluate_series, draws =
            bingo_draws[c(1:i)])
  score <-
    bingo_draws[i] * sum(setdiff(bingo_boards_matrices[[remaining_boards]], bingo_draws[c(1:i)]))
}
print(
  paste(
    "Final board score is",
    score,
    "; last evaluated round is",
    i,
    "and last drawn number is",
    bingo_draws[i]
  )
)
```

```
## [1] "Final board score is 10478 ; last evaluated round is 84 and last drawn number is 31"
```