Euler Problem 33 takes us back to the world of fractions from our primary school days. Many kids hate and struggle learning about fractions but once you master them, a new world of numbers opens up. Unfortunately, the proliferation of digital calculators has negated the use of fractions in favour of decimal expressions. [Fractions](https://lucidmanager.org/euler-problem-26/) are an aesthetic way to express numbers, without having to resort to ugly random sequences of decimals. This is why I prefer to use 22/7 as an [approximation of Pi](https://lucidmanager.org/approximations-of-pi/) over the ugly infinite series of decimals.

This Numberphile video below explains fractions and Farey sequences. A Farey sequence contains all fractions between 0 and 1 with a maximum denominator. More formally, a [Farey sequence](https://en.wikipedia.org/wiki/Farey_sequence) of order *n* is the sequence of completely reduced fractions between 0 and 1 which, when in lowest terms, have denominators less than or equal to n, arranged in order of increasing size. For example, the Farey Sequence with order 3 is:

F_3 = \Big\{ \frac{0}{1},\frac{1}{3},\frac{1}{2},\frac{2}{3},\frac{1}{1}\Big\}

These sequences can be visualised in fractal-esque Ford Circles, but before we get to this, first solve Euler problem 33.

**Euler Problem 33 Definition**

The fraction 49/98 is a curious fraction, as an inexperienced mathematician in attempting to simplify it may incorrectly believe that 49/98 = 4/8, which is correct, is obtained by cancelling the 9s. We shall consider fractions like 30/50 = 3/5, to be trivial examples.

There are exactly four nontrivial examples of this type of fraction, less than one in value, and containing two digits in the numerator and denominator. If the product of these four fractions is given in its lowest common terms, [find the value of the denominator](https://projecteuler.net/problem=33).

**Proposed Solution in R**

To solve this problem, we create a pseudo-Farey sequence by generating all different fractions with two decimals in the numerator and denominator. The loop generates all combinations of demoninators and numerators, excluding the trivial ones (multiples of 10 and 11). This solution converts the numbers to strings, strips any common duplicates, and tests the condition. The code concatenates vectors, which is not good practice. However, the loop is so short it does not matter much.

You can view the code below or download it from my [GitHub page](https://github.com/pprevos/r.prevos.net/blob/master/ProjectEuler/problem033.R).

num <- vector()

den <- vector()

for (a in 11:99) {

for (b in (a + 1):99) {

trivial <- (a %% 10 == 0 | b && 10 == 0 | a %% 11 == 0 | b %% 11 == 0)

if (!trivial) {

i <- as.numeric(unlist(strsplit(as.character(a), "")))

j <- as.numeric(unlist(strsplit(as.character(b), "")))

digs <- c(i, j)

dup <- digs[duplicated(digs)]

digs <- digs[which(digs != dup)]

if (length(digs) == 2 & a/b == digs[1]/digs[2]) {

num <- c(num, a)

den <- c(den, b)

}

}

}

}

answer <- prod(den) / prod(num)

print(answer)

**Farey Sequences and Ford Circles**

Next step is to generalise Euler problem 33 and write a function to generate Farey Sequences and visualise them using Ford Circles.

The farey function generates a data table with the numerators (p) and denominators (q) of a Farey sequence. The function builds a list of all possible fractions for the solution space, excluding those with one as a Greatest Common Dominator, as defined by the gcc function.

farey <- function(n) {

fseq <- list()

fseq[[1]] <- c(0, 1)

i <- 2

gcd <- function(a, b) { # Euclid's method

if (a == 0) return(b)

if (b == 0) return(a)

gcd(b, a%%b)

}

for (q in 2:n) {

for (p in 1:(q - 1)){

if (gcd(p, q) == 1) {

fseq[[i]] <- c(p, q)

i <- i + 1

}

}

}

fseq[[i]] <- c(1, 1)

fseq <- as.data.frame(do.call(rbind, fseq))

names(fseq) <- c("p", "q")

fseq <- fseq[order(fseq$p / fseq$q), ]

return(fseq)

}

Standard ggplot2 cannot draw circles where the radius of the circles is related to the coordinate system. I tried to use the [ggforce](https://cran.r-project.org/web/packages/ggforce/vignettes/Visual_Guide.html) package to plot circles in ggplot2, but for some reason, I was not able to install this package on Ubuntu. As an alternative, I used a circle function sourced from [StackOverflow](https://stackoverflow.com/questions/6862742/draw-a-circle-with-ggplot2). This function is called in a for-loop to build the circles on top of the empty canvas.

Farey Sequence and Ford Circles (n = 20).

library(tidyverse)

lm\_palette <- c("#008da1", "#005395", "#262e43", "#3b2758", "#865596", "#f26230")

ford\_circles <- farey(20) %>%

mutate(x = p / q,

y = 1 / (2\* q^2),

r = y,

c = lm\_palette[(q - 1)%%6 + 1])

g\_circle <- function(r, x, y, color = NA, fill = "black", ...) {

x <- x + r \* cos(seq(0, pi, length.out = 100))

ymax <- y + r \* sin(seq(0, pi, length.out = 100))

ymin <- y + r \* sin(seq(0, -pi, length.out = 100))

annotate("ribbon", x = x, ymin = ymin, ymax = ymax,

color = color, fill = fill, ...)

}

p <- ggplot(ford\_circles, aes(x, y))

for (i in 1:nrow(ford\_circles)) {

p <- p + g\_circle(ford\_circles$r[i], ford\_circles$x[i], ford\_circles$y[i],

fill = ford\_circles$c[i])

}

p + xlim(c(0, 1)) + coord\_fixed() + theme\_void()