Practice 5

COMP9021, Term 3, 2019

1 Longest sequence of consecutive letters

Write a program longest_sequence.py that prompts the user for a string w of lowercase letters and outputs the longest sequence of consecutive letters that occur in w, but with possibly other letters in between, starting as close as possible to the beginning of w.

Here is a possible interaction:

```
$ python3 longest_sequence.py
Please input a string of lowercase letters: a
The solution is: a
$ python3 longest_sequence.py
Please input a string of lowercase letters: abcefgh
The solution is: efgh
$ python3 longest_sequence.py
Please input a string of lowercase letters: abcefg
The solution is: abc
$ python3 longest_sequence.py
Please input a string of lowercase letters: ablccmdnneofffpg
The solution is: abcdefg
$ python3 longest_sequence.py
Please input a string of lowercase letters: abcdiivjwkaalbmmbz
The solution is: ijklm
$ python3 longest_sequence.py
Please input a string of lowercase letters: abcpqrstuvwxbcbcdddeffghijklrst
The solution is: abcdefghijkl
```

2 Mediants

Let two distinct reduced positive fractions $F_1 = \frac{p_1}{q_1}$ and $F_2 = \frac{p_2}{q_2}$ be given, with the denominator set to 1 in case the fraction is 0. The mediant of F_1 and F_2 is defined as $\frac{p_1+p_2}{q_1+q_2}$; it is also in reduced form, and sits between F_1 and F_2. Let a reduced fraction $F = \frac{p}{q}$ in $(0, 1)$ be given. It can be shown that starting with $\frac{0}{1}$ and $\frac{1}{1}$, one can compute a finite number of mediants and eventually generate F. More precisely, there exists $n \in \mathbf{N}$ and a sequence of pairs of fractions $(F_1^i, F_2^i)_{i \leq n}$ such that:

- $F_1^0 = \frac{0}{1}$ and $F_2^0 = \frac{1}{1}$;

- for all $i < n$, either F_1^{i+1} or F_2^{i+1} is the mediant of F_1^i and F_2^i, with F_2^{i+1} equal to F_2^i or F_1^{i+1} equal to F_1^i, respectively, depending on whether that mediant is strictly smaller or strictly greater than F, respectively.
- F is the mediant of F_1^n and F_2^n.

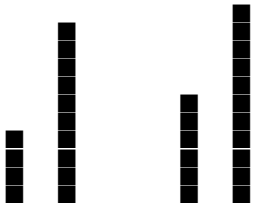
Write a Python program `mediants.py` that defines a function `mediants_to()` that given as arguments two strictly positive integers p and q with $p < q$ and $\gcd(p, q) = 1$, computes the sequence $(F_1^i, F_2^i)_{i \leq n}$ previously defined. The function returns `None` but displays that sequence, one pair per line, with the mediant of the pair in-between, and for all pairs except the last one, indicating with the `*` character whether $\frac{p}{q}$ is between the first member of the pair and the mediant, or between the mediant and the second member of the pair. The numerators and denominators of all fractions are aligned and displayed in a field of width equal to the maximum of the number of digits in p and the number of digits in q. Five spaces, or two spaces, the `*` character and two spaces, precede and follow the display of all mediants. Using the `doctest` module to test `mediants_to()`, the following behaviour would then be observed:

```
>>> mediants_to(1, 2)
0/1     1/2     1/1
>>> mediants_to(41, 152)
  0/1     *     1/2          1/1
  0/1     *     1/3          1/2
  0/1           1/4     *    1/3
  1/4     *     2/7          1/3
  1/4     *     3/11         2/7
  1/4           4/15    *    3/11
  4/15          7/26    *    3/11
  7/26    *    10/37         3/11
  7/26    *    17/63        10/37
  7/26         24/89    *   17/63
 24/89        41/152       17/63
>>> mediants_to(71, 83)
 0/1       1/2    *   1/1
 1/2       2/3    *   1/1
 2/3       3/4    *   1/1
 3/4       4/5    *   1/1
 4/5       5/6    *   1/1
 5/6    *  6/7        1/1
 5/6      11/13   *   6/7
11/13     17/20   *   6/7
17/20     23/27   *   6/7
23/27     29/34   *   6/7
29/34     35/41   *   6/7
35/41     41/48   *   6/7
41/48     47/55   *   6/7
47/55     53/62   *   6/7
53/62     59/69   *   6/7
```

```
59/69      65/76  *   6/7
65/76      71/83      6/7
```

3 The game of nim

Let a natural number k and k natural numbers $n_1, \ldots, n_k$ be given. Conceive of the latter as the number of coins in k piles of coins, aligned from left to right. For instance, with $k = 5$ and $n_1 = 4$, $n_2 = 10$, $n_3 = 0$, $n_4 = 6$ and $n_5 = 11$, the piles can be depicted as:



Two players take turns and take coins (possibly only one coin, possibly all coins, but necessarily at least one coin) from the top of one of the nonempty piles. The play ends when there are no coins left, the player's whose turn it is to play being the loser. In particular:

- if there is no pile or all piles are empty to start with, then the second player is the winner;
- if there is one and only one nonempty pile, then the first player just has to take all coins to immediately win.

One of the players has a winning strategy, that is, a way to play that guarantees him to win whichever way the other player plays. Moreover, the first player has a winning strategy iff the nim sum of the game, that is, $n_1 \oplus \ldots \oplus n_k$, is not equal to 0. With the example just considered:

n_1	4	0100
n_2	10	1010
n_3	0	0000
n_4	6	0110
n_5	11	1011
$\bigoplus_{i=1}^{5} n_i$	3	0011

so the first player has a winning strategy.

Let us verify those claims. When there are no coins, the nim sum is equal to 0. Suppose that there is at least one coin in some pile, and let s and s' be the nim sums before and after the player P whose turn it is to play has taken some coins from some nonempty pile, respectively. If $s = 0$ then s' is necessarily different to 0. If s is not equal to 0 then $s' = 0$ provided that P played as follows. Let p be the position of the leftmost 1 in the binary representation of s. Then at least one of n_1, …, n_k has a 1 in its binary representation at position p. Let $j \in \{1, \ldots, k\}$ be such that n_j is such a number. Then $n_j \oplus s$ is smaller than n_j, and removing $n_j - (n_j \oplus s)$ coins from the j-th column,

that is, leaving $n_j \oplus s$ coins in the j-th column, indeed makes s' equal to 0 (since $s \oplus n_j$ "erases" n_j, and so $s' = (s \oplus n_j) \oplus (s \oplus n_j) = 0$). The winning strategy can then consist in making such a move, and it can be fixed by letting j be minimal. With the example considered above, $j = 2$, and since $n_2 \oplus \bigoplus_{i=1}^{5} n_i = 10 \oplus 3 = 9$, the chosen winning strategy lets the player remove one coin from the second pile.

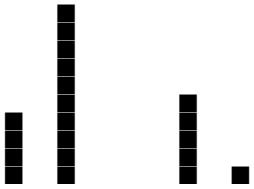
Write a Python program `nim.py` that prompts the user to enter natural numbers meant to represent the number of coins in piles—as many piles as numbers being entered—, outputs which player has a winning strategy, and simulates the playing of the game. The player having a winning strategy should play according to the specific winning strategy previously described. The other player should play randomly, by:

- randomly selecting a pile amongst those still having coins, using the `choice()` function from the `random` module with as argument, the list of indexes of piles still having coins, the first pile having an index of 0 (with the example above, that list would be [0, 1, 3, 4]);
- randomly selecting from that pile the number of coins to leave in the pile using the `randrange()` function from the `random` module, providing the number of coins in the pile as argument (with the example above, if the second column, of index 1, was chosen then `randrange(10)` would determine how many coins to leave in the pile).

To be able to replicate a given simulation, the program prompts the user to optionally input a seed; if an integer is input then it is given as argument to the `seed()` function from the `random` module. A coin is displayed as the Unicode character `'\u2b1b'` (a black square). Two consecutive columns are separated by 2 spaces. The following behaviour might be observed and should be observed for the following three executions of the program, respectively:

```
$ python3 nim.py
Describe the piles: 1 2 3

Second player will play smart and win!
Input seed if desired:

Game to be played:
■  ■  ■
■  ■

First player making random move:
■  ■  ■

Second player making smart move:
■     ■

First player making random move:
■

Second player making smart move:
$ python3 nim.py
Describe the piles: 4 10 0 5 1
```

```
First player will play smart and win!
Input seed if desired: 0

Game to be played:
```



```
First player making smart move:
```

```
Second player making random move:
```

```
First player making smart move:
```

```
Second player making random move:
```

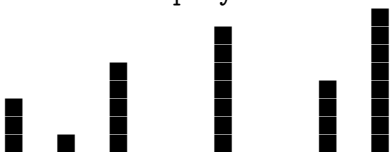
```
First player making smart move:
```



```
Second player making random move:
```

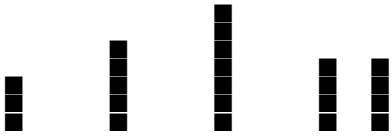


```
First player making smart move:
$ python3 nim.py
Describe the piles: 3 1 5 0 7 0 4 8

First player will play smart and win!
Input seed if desired: 1

Game to be played:
```

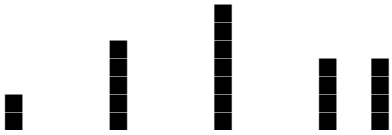


```
First player making smart move:
```

Second player making random move:



First player making smart move:



Second player making random move:

First player making smart move:

Second player making random move:

First player making smart move:

Second player making random move:

First player making smart move:

Second player making random move:

First player making smart move:

Second player making random move:

First player making smart move:

Second player making random move:

■

```
First player making smart move:
```

4 Sierpinski triangle (optional)

Write a program `sierpinski_triangle.py` that generates Latex code, a `.tex` file, that can be processed with pdflatex to create a `.pdf` file that depicts Sierpinski triangle, obtained from Pascal triangle by drawing a black square when the corresponding number is odd. A simple method is to use a particular case of Luca's theorem, which states that the number of ways of choosing k objects out of n is odd iff all digits in the binary representation of k are digits in the binary representation of n. For instance:

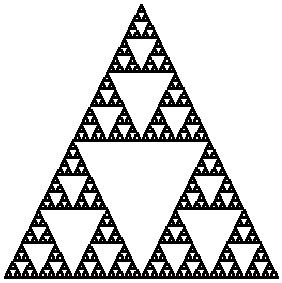
- $\binom{5}{3} = 10$, which corresponds to a white square as 10 is even; indeed, 5 is 101 in binary, 3 is 11 in binary, and there is at least one bit set to 1 in 11 (namely, the leftmost one), which is not set to 1 in 101;
- $\binom{6}{2} = 15$, which corresponds to a black square as 15 is odd; indeed, 6 is 110 in binary, 2 is 10 in binary, and all bits (actually, the only bit) set to 1 in 10 are set to 1 in 110.

So your program has to generate a file named `Sierpinski_triangle.tex`, similar to the one provided; examine the contents of this file to see which text needs to be output.

The file `Sierpinski_triangle.pdf` is also provided, but if you want to generate it yourself from `Sierpinski_triangle.tex`, you need to have Tex installed on your computer (install it if that is not the case, see `http://www.tug.org/texlive/`), and then execute

```
pdflatex Sierpinski_triangle.tex
```

from the command line, or open `Sierpinski_triangle.tex` in the Latex editor that comes with your distribution of Tex, and it will just be a matter of clicking a button…



5 A calendar program (optional, advanced)

Write a program `calendar.py` that provides a variant on the Unix `cal` utility (in particular because it lets the weeks start on Monday, not Sunday), following this kind of interaction:

```
$ python3 calendar.py
I will display a calendar, either for a year or for a month in a year.
The earliest year should be 1753.
For the month, input at least the first three letters of the month's name.
Input year, or year and month, or month and year: 3194 Sept
    September 3194
 Mo Tu We Th Fr Sa Su
           1  2  3  4
  5  6  7  8  9 10 11
 12 13 14 15 16 17 18
 19 20 21 22 23 24 25
 26 27 28 29 30
$ python3 calendar.py
I will display a calendar, either for a year or for a month in a year.
The earliest year should be 1753.
For the month, input at least the first three letters of the month's name.
Input year, or year and month, or month and year: dEcEm 3194
    December 3194
 Mo Tu We Th Fr Sa Su
           1  2  3  4
  5  6  7  8  9 10 11
 12 13 14 15 16 17 18
 19 20 21 22 23 24 25
 26 27 28 29 30 31
```

```
$ python3 calendar.py
I will display a calendar, either for a year or for a month in a year.
The earliest year should be 1753.
For the month, input at least the first three letters of the month's name.
Input year, or year and month, or month and year: 3194
                                3194

       January                February                March
Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su
                   1  2      1  2  3  4  5  6      1  2  3  4  5  6
 3  4  5  6  7  8  9   7  8  9 10 11 12 13   7  8  9 10 11 12 13
10 11 12 13 14 15 16  14 15 16 17 18 19 20  14 15 16 17 18 19 20
17 18 19 20 21 22 23  21 22 23 24 25 26 27  21 22 23 24 25 26 27
24 25 26 27 28 29 30  28                    28 29 30 31
31

        April                  May                   June
Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su
                1  2  3                  1         1  2  3  4  5
 4  5  6  7  8  9 10   2  3  4  5  6  7  8   6  7  8  9 10 11 12
11 12 13 14 15 16 17   9 10 11 12 13 14 15  13 14 15 16 17 18 19
18 19 20 21 22 23 24  16 17 18 19 20 21 22  20 21 22 23 24 25 26
25 26 27 28 29 30     23 24 25 26 27 28 29  27 28 29 30
                      30 31

         July                 August              September
Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su
                1  2  3   1  2  3  4  5  6  7            1  2  3  4
 4  5  6  7  8  9 10   8  9 10 11 12 13 14   5  6  7  8  9 10 11
11 12 13 14 15 16 17  15 16 17 18 19 20 21  12 13 14 15 16 17 18
18 19 20 21 22 23 24  22 23 24 25 26 27 28  19 20 21 22 23 24 25
25 26 27 28 29 30 31  29 30 31              26 27 28 29 30
       October               November              December
Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su
                   1  2      1  2  3  4  5  6            1  2  3  4
 3  4  5  6  7  8  9   7  8  9 10 11 12 13   5  6  7  8  9 10 11
10 11 12 13 14 15 16  14 15 16 17 18 19 20  12 13 14 15 16 17 18
17 18 19 20 21 22 23  21 22 23 24 25 26 27  19 20 21 22 23 24 25
24 25 26 27 28 29 30  28 29 30              26 27 28 29 30 31
31
```

In doing this exercise, you will have to find out (or just remember…) how leap years are determined, and what is so special about the year 1753…