Lab 6

COMP9021, Session 1, 2018

1 ☞ Obtaining a sum from a subsequence of digits

Write a program `sum_of_digits.py` that prompts the user for two numbers, say `available_digits` and `desired_sum`, and outputs the number of ways of selecting digits from `available_digits` that sum up to `desired_sum`. For instance, if `available_digits` is `12234` and `sum` is `5` then there are 4 solutions:

- one solution is obtained by selecting 1 and both occurrences of 2 ($1 + 2 + 2 = 5$);
- one solution is obtained by selecting 1 and 4 ($1 + 4 = 5$);
- one solution is obtained by selecting the first occurrence of 2 and 3 ($2 + 3 = 5$);
- one solution is obtained by selecting the second occurrence of 2 and 3 ($2 + 3 = 5$).

Here is a possible interaction:

```
$ python3 sum_of_digits.py
Input a number that we will use as available digits: 12234
Input a number that represents the desired sum: 5
There are 4 solutions.
$ python3 sum_of_digits.py
Input a number that we will use as available digits: 11111
Input a number that represents the desired sum: 5
There is a unique solution.
$ python3 sum_of_digits.py
Input a number that we will use as available digits: 11111
Input a number that represents the desired sum: 6
There is no solution.
$ python3 sum_of_digits.py
Input a number that we will use as available digits: 1234321
Input a number that represents the desired sum: 5
There are 10 solutions.
```

2 ☞ Merging two strings into a third one

Say that two strings s_1 and s_2 can be merged into a third string s_3 if s_3 is obtained from s_1 by inserting arbitrarily in s_1 the characters in s_2, respecting their order. For instance, the two strings *ab* and *cd* can be merged into *abcd*, or *cabd*, or *cdab*, or *acbd*, or *acdb*, ..., but not into *adbc* nor into *cbda*. Write a program `merging_strings.py` that prompts the user for 3 strings and displays the output as follows:

- If no string can be obtained from the other two by merging, then the program outputs that there is no solution.
- Otherwise, the program outputs which of the strings can be obtained from the other two by merging.

Here is a possible interaction:

```
$ python3 merging_strings.py
Please input the first string: ab
Please input the second string: cd
Please input the third string: abcd
The third string can be obtained by merging the other two.
$ python3 merging_strings.py
Please input the first string: ab
Please input the second string: cdab
Please input the third string: cd
The second string can be obtained by merging the other two.
$ python3 merging_strings.py
Please input the first string: abcd
Please input the second string: cd
Please input the third string: ab
The first string can be obtained by merging the other two.
$ python3 merging_strings.py
Please input the first string: ab
Please input the second string: cd
Please input the third string: adcb
No solution
$ python3 merging_strings.py
Please input the first string: aaaaa
Please input the second string: a
Please input the third string: aaaa
The first string can be obtained by merging the other two.
$ python3 merging_strings.py
Please input the first string: aaab
Please input the second string: abcab
Please input the third string: aaabcaabb
The third string can be obtained by merging the other two.
$ python3 merging_strings.py
Please input the first string: ??got
Please input the second string: ?it?go#t##
Please input the third string: it###
The second string can be obtained by merging the other two.
```

3 ☞ Unit fractions

Let N and D be two strictly positive integers with $N < D$. The fraction N/D can be written as a sum of unit fractions, that is, there exists integers $k, d_1, \ldots, d_k \geq 1$ with $d_1 < d_2 < \ldots < d_k$ such that

$$\frac{N}{D} = \frac{1}{d_1} + \frac{1}{d_2} + \cdots + \frac{1}{d_k}.$$

There are actually infinitely many such representations. Indeed, since

$$1 = \frac{1}{2} + \frac{1}{3} + \frac{1}{6}$$

if $\frac{N}{D} = \frac{1}{d_1} + \frac{1}{d_2} + \cdots + \frac{1}{d_k}$ then also

$$\frac{N}{D} = \frac{1}{d_1} + \frac{1}{d_2} + \cdots + \frac{1}{d_{k-1}} + \frac{1}{2d_k} + \frac{1}{3d_k} + \frac{1}{6d_k}.$$

One particular representation is obtained by a method proposed by Fibonacci, in the form of a greedy algorithm. Suppose that N/D cannot be simplified, that is, N and D have no other common factor but 1. If $N = 1$ then we are done, so suppose otherwise. Let d_1 be the smallest integer such that $\frac{N}{D}$ can be written as $\frac{1}{d_1} + f_1$, with f_1 necessarily strictly positive by assumption. Looking for the smallest d_1 is what makes the algorithm greedy. Of course, d_1 is equal to $D \div N + 1$. By the choice of d_1, $\frac{1}{d_1 - 1} > \frac{N}{D}$, hence $D > N(d_1 - 1)$, hence $N > Nd_1 - D$. Since f_1 is equal to $\frac{N}{D} - \frac{1}{d_1} = \frac{Nd_1 - D}{Dd_1}$, it follows that $\frac{N}{D}$ can be written as $\frac{1}{d_1} + \frac{N_1}{D_1}$ with $N_1 < N$. If $N_1 > 1$ then the same argument allows one to greedily find $d_2 > d_1$ such that for some strictly positive integers N_2 and D_2, $\frac{N}{D}$ can be written as $\frac{1}{d_1} + \frac{1}{d_2} + \frac{N_2}{D_2}$ with $N_2 < N_1$, and if $N_2 > 1$ then the same argument allows one to greedily find $d_3 > d_2$ such that for some strictly positive integers N_3 and D_3, $\frac{N}{D}$ can be written as $\frac{1}{d_1} + \frac{1}{d_2} + \frac{1}{d_3} + \frac{N_3}{D_3}$ with $N_3 < N_2 \ldots$ After a finite number of steps, we are done.

The number of summands in the sum of unit fractions given by Fibonacci's method is not always minimal: it is sometimes possible to decompose $\frac{N}{D}$ as sum of unit fractions with fewer summands. For instance, Fibonacci's method yields

$$\frac{4}{17} = \frac{1}{5} + \frac{1}{29} + \frac{1}{1233} + \frac{1}{3039345}$$

whereas $\frac{4}{17}$ can be written as a sum of 3 unit fractions, actually in 4 possible ways:

$$\frac{4}{17} = \frac{1}{5} + \frac{1}{30} + \frac{1}{510}$$
$$\frac{4}{17} = \frac{1}{5} + \frac{1}{34} + \frac{1}{170}$$
$$\frac{4}{17} = \frac{1}{6} + \frac{1}{15} + \frac{1}{510}$$
$$\frac{4}{17} = \frac{1}{6} + \frac{1}{17} + \frac{1}{102}$$

Write a program `unit_fractions.py` that implements two functions, `fibonacci_decomposition()` and `shortest_length_decompositions()`, that both take two strictly positive integers N and D as arguments, and writes N/D as, respectively:

- a sum of unit fractions following Fibonacci method, plus an integer in case $N \geq D$ (in a unique way);
- a sum of unit fractions with a minimal number of summands, plus an integer in case $N \geq D$ (in possibly many ways).

Here is a possible interaction:

```
>>> from unit_fractions import *
>>> fibonacci_decomposition(1, 521)
1/521 = 1/521
>>> fibonacci_decomposition(521, 521)
521/521 = 1
>>> fibonacci_decomposition(521, 1050)
521/1050 = 1/3 + 1/7 + 1/50
>>> fibonacci_decomposition(1050, 521)
1050/521 = 2 + 1/66 + 1/4913 + 1/33787684 + 1/2854018941421956
>>> fibonacci_decomposition(6, 7)
6/7 = 1/2 + 1/3 + 1/42
>>> shortest_length_decompositions(6, 7)
6/7 = 1/2 + 1/3 + 1/42
>>> fibonacci_decomposition(8, 11)
8/11 = 1/2 + 1/5 + 1/37 + 1/4070
>>> shortest_length_decompositions(8, 11)
8/11 = 1/2 + 1/5 + 1/37 + 1/4070
8/11 = 1/2 + 1/5 + 1/38 + 1/1045
8/11 = 1/2 + 1/5 + 1/40 + 1/440
8/11 = 1/2 + 1/5 + 1/44 + 1/220
8/11 = 1/2 + 1/5 + 1/45 + 1/198
8/11 = 1/2 + 1/5 + 1/55 + 1/110
8/11 = 1/2 + 1/5 + 1/70 + 1/77
8/11 = 1/2 + 1/6 + 1/17 + 1/561
8/11 = 1/2 + 1/6 + 1/18 + 1/198
8/11 = 1/2 + 1/6 + 1/21 + 1/77
8/11 = 1/2 + 1/6 + 1/22 + 1/66
8/11 = 1/2 + 1/7 + 1/12 + 1/924
8/11 = 1/2 + 1/7 + 1/14 + 1/77
8/11 = 1/2 + 1/8 + 1/10 + 1/440
8/11 = 1/2 + 1/8 + 1/11 + 1/88
8/11 = 1/3 + 1/4 + 1/7 + 1/924
>>> fibonacci_decomposition(4, 17)
4/17 = 1/5 + 1/29 + 1/1233 + 1/3039345
>>> shortest_length_decompositions(4, 17)
4/17 = 1/5 + 1/30 + 1/510
4/17 = 1/5 + 1/34 + 1/170
4/17 = 1/6 + 1/15 + 1/510
4/17 = 1/6 + 1/17 + 1/102
```

4 The Gale Shapley algorithm (optional, advanced)

Read the AMS Feature column on the stable marriage problem and the Gale Shapley algorithm. Write a program gale_shapley.py that

- lets the user input the number n of couples,
- either lets the user input names or uses the default names M_1, ..., M_n for men and W_1, ..., W_n for women,
- either lets the user define preferences or randomly generates preferences.

If the preferences have been randomly generated then they are output. Finally, the Gale Shapley algorithm is applied and the matches are displayed.

Here is a possible interaction (the matches are given with women in first position, and lexicographically ordered):

```
>>> run_algorithm()
Enter a strictly positive number for the number of couples: 4

Enter 4 names for the men, all on one line and separated by spaces,
  or just press Enter for the default "names" M_1, ..., M_4:

Enter 4 names for the women, all on one line and separated by spaces,
  or just press Enter for the default "names" W_1, ..., W_4:

Press Enter to get a default preference for all men or women.
Otherwise, input one or more nonspace characters before Enter
  to be prompted and enter the preferences of your choice:

Preferences for M_1: W_3 W_1 W_4 W_2
Preferences for M_2: W_2 W_1 W_4 W_3
Preferences for M_3: W_2 W_4 W_1 W_3
Preferences for M_4: W_3 W_1 W_2 W_4

Preferences for W_1: M_2 M_3 M_4 M_1
Preferences for W_2: M_4 M_2 M_3 M_1
Preferences for W_3: M_1 M_4 M_3 M_2
Preferences for W_4: M_4 M_1 M_2 M_3

The matches are:
W_1 -- M_4
W_2 -- M_2
W_3 -- M_1
W_4 -- M_3
```

```
>>> run_algorithm()
Enter a strictly positive number for the number of couples: 4

Enter 4 names for the men, all on one line and separated by spaces,
  or just press Enter for the default "names" M_1, ..., M_4:

Enter 4 names for the women, all on one line and separated by spaces,
  or just press Enter for the default "names" W_1, ..., W_4:

Press Enter to get a default preference for all men or women.
Otherwise, input one or more nonspace characters before Enter
  to be prompted and enter the preferences of your choice: add

List preferences for M_1, in decreasing order: W_1 W_2 W_3 W_4
List preferences for M_2, in decreasing order: W_1 W_4 W_3 W_2
List preferences for M_3, in decreasing order: W_2 W_1 W_3 W_4
List preferences for M_4, in decreasing order: W_4 W_2 W_3 W_1

List preferences for W_1, in decreasing order: M_4 M_3 M_1 M_2
List preferences for W_2, in decreasing order: M_2 M_4 M_1 M_3
List preferences for W_3, in decreasing order: M_4 M_1 M_2 M_3
List preferences for W_4, in decreasing order: M_3 M_2 M_1 M_4

The matches are:
W_1 -- M_3
W_2 -- M_4
W_3 -- M_1
W_4 -- M_2
```

5 Sydney temperatures (optional)

Write a program `sydney_temperatures.py` that extracts from the file `IDCJCM0037_066062.csv`, stored in the working directory, the mean min and mean max temperatures for the 12 months of the year for the years 1859 to 2016, and plots them thanks to the `matplotlib.pyplot` module, which it is convenient to import as `plt`.

- The picture is 5 inches wide and 3.5 inches high—check out `plt.figure()`, passing as argument the system's resolution (in dots per inch) for best results. It has as title, with a font size of 10 points, `Mean min and max temperatures in Sydney`—check out `plt.title()`. The grid should be displayed—check out `plt.grid()`.
- Denoting by min_temp the largest integer smaller than or equal to the smallest mean min temperature and by max_temp the smallest integer greater than or equal to the largest mean max temperature, the plotting area is a rectangle whose lower left corner has coordinates $(0.5, \text{min\_temp} - 1)$ and whose upper right corner has coordinates $(12.5, \text{max\_temp} + 1)$—check out `plt.axis()`.
- The mean min and mean max temperatures for the i^{th} month, $i \in \{1, \ldots, 12\}$, are displayed at points of x-coordinate i, while the averages of the mean min and mean max temperatures for December and January are displayed both at points of x-coordinate 0.5 and at points of x-coordinate 12.5, on blue curves for the mean min temperatures, on red curves for the mean max temperatures—check out `plt.plot()`.
- The area between the curves for the mean min and mean max temperatures is coloured in grey with, to create an appearance of transparency, a value of 0.5 on the alpha channel—check out `plt.fill_between()`.
- The x-axis labels at coordinates 1 to 12 the names of the months of the year, conveniently retrieved from the `month_name` list of the `calendar` module, with a font size of 8 points, and displayed slanted—check out `plt.xticks()` and `autofmt_xdate()`.
- The y-axis labels the temperatures between min_temp and max_temp, in steps of 0.5, with a font size of 4 points—check out `plt.yticks()`.

To display the figure, check out `plt.show()`. Here is the expected output:

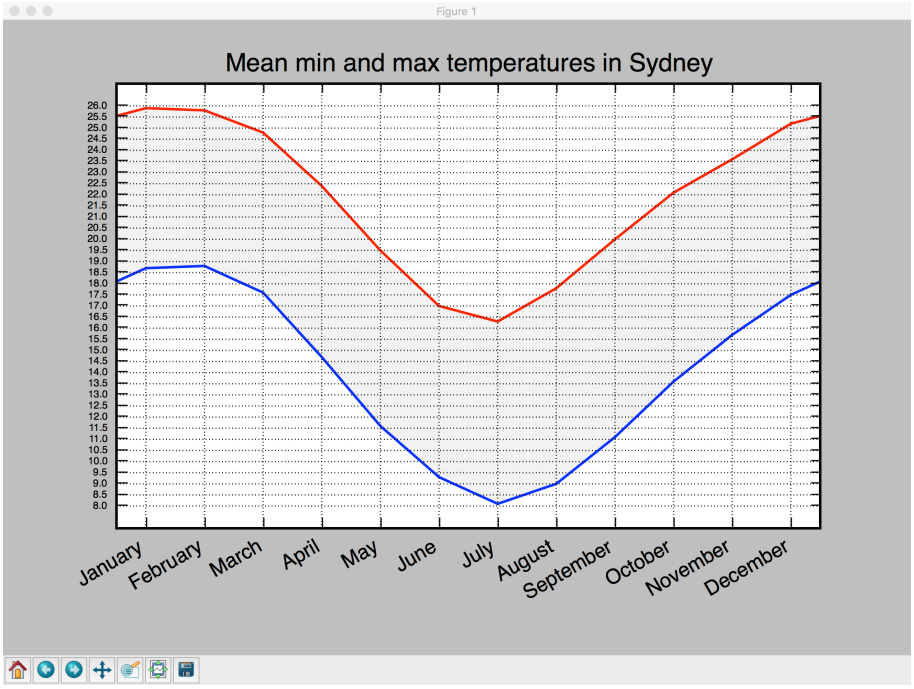
Figure 1
Mean min and max temperatures in Sydney
26.0
25.5
25.0
24.5
24.0
23.5
23.0
22.5
22.0
21.5
21.0
20.5
20.0
19.5
19.0
18.5
18.0
17.5
17.0
16.5
16.0
15.5
15.0
14.5
14.0
13.5
13.0
12.5
12.0
11.5
11.0
10.5
10.0
9.5
9.0
8.5
8.0
January
February
March
April
May
June
July
August
September
October
November
December