Lab 7

COMP9021, Session 1, 2015

1 ☞ Obtaining a sum from a subsequence of digits

Write a program 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:

```
$ python question_1.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.
$ python question_1.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
$ python question_1.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.
$ python question_1.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 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:

```
$ python question_2.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.
$ python question_2.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.
$ python question_2.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.
$ python question_2.py
Please input the first string: ab
Please input the second string: cd
Please input the third string: adcb
No solution
$ python question_2.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.
$ python question_2.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.
$ python question_2.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 The n-queens puzzle

This is a well known puzzle: place n chess queens on an $n \times n$ chessboard so that no queen is attacked by any other queen (that is, no two queens are on the same row, or on the same column, or on the same diagonal). There are numerous solutions to this puzzle that illustrate all kinds of programming techniques. You will find lots of material, lots of solutions on the web. You can of course start with the wikipedia page: http://en.wikipedia.org/wiki/Eight_queens_puzzle. You should try and solve this puzzle in any way you like.

One set of technique consists in generating permutations of $[0, 1, ..., n-1]$, a permutation $[k_0, k_1, ..., k_{n-1}]$ requesting to place the queen of the first row in the $(k_0 + 1)$st column, the queen of the second row in the $(k_1 + 1)$nd column, etc. For instance, with $n = 8$ (the standard chessboard size), the permutation `[4, 6, 1, 5, 2, 0, 3, 7]` gives rise to the solution:

```
0 0 0 0 1 0 0 0
0 0 0 0 0 0 1 0
0 1 0 0 0 0 0 0
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 1
```

The program `cryptarithm_v1.py`, written by Raymond Hettinger as part of ActiveState Code Recipes, demonstrates the use of the `permutations()` function from the `itertools` modules. With `cryptarithm_v2.py`, we have a variant with an implementation of Heap's algorithm to generate permutations and a technique to 'skip' some of them. We could do the same here. For instance, starting with `[0,1,2,3,4,5,6,7]`, we find out that the queen on the penultimate row is attacked by the queen on the last row, and skip all permutations of `[0,1,2,3,4,5,6,7]` that end in `[6,7]`. If you have acquired a good understanding of the description of Heap's algorithm given in `Permutations.pdf` and of `cryptarithm_v2.py`, then try and solve the n-queens puzzle generating permutations and skipping some using Heap's algorithm; this is the solution I will provide. Doing so will bring your understanding of recursion to new levels, but it is not an easy problem, only attempt it if you want to challenge yourself...

Here is a possible interaction. It is interesting to print out the number of permutations being tested.

```
$ python
...
>>> from question_3 import *
>>> puzzle = QueenPuzzle(8)
>>> puzzle.print_nb_of_tested_permutations()
3544
>>> puzzle.print_nb_of_solutions()
92
>>> puzzle.print_solution(0)
 0 0 0 0 1 0 0 0
 0 0 0 0 0 0 1 0
 0 1 0 0 0 0 0 0
 0 0 0 0 0 1 0 0
 0 0 1 0 0 0 0 0
 1 0 0 0 0 0 0 0
 0 0 0 1 0 0 0 0
 0 0 0 0 0 0 0 1
>>> puzzle.print_solution(45)
 0 0 0 0 0 1 0 0
 1 0 0 0 0 0 0 0
 0 0 0 0 1 0 0 0
 0 1 0 0 0 0 0 0
 0 0 0 0 0 0 0 1
 0 0 1 0 0 0 0 0
 0 0 0 0 0 0 1 0
 0 0 0 1 0 0 0 0
>>> puzzle = QueenPuzzle(11)
>>> puzzle.print_nb_of_tested_permutations()
382112
>>> puzzle.print_nb_of_solutions()
2680
>>> puzzle.print_solution(1346)
 0 0 0 0 0 1 0 0 0 0 0
 0 0 0 0 0 0 0 1 0 0 0
 0 0 1 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 1
 0 0 0 0 0 0 1 0 0 0 0
 0 1 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 1 0
 1 0 0 0 0 0 0 0 0 0 0
 0 0 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 0 0
 0 0 0 0 1 0 0 0 0 0 0
```