Intro Ex5 – Recursion

Submission deadline: Monday, 25.11.2013, 20:55

Objectives: understand recursive code, write recursive code.
Task: This exercise contains three tasks that you are required to accomplish:

Part A: Convert recursive code to non-recursive code.

The following mystery code contains one important function: **`mystery_computation`**. As you can see, this method is implemented by calling a recursive function named **`mystery_recursion`**. The names and parameters of both functions, as well as the lack of comments and documentations, do not follow the course readability guidelines, but for a good reason: as a first step, you have to figure out what this code does!
The mission is to understand code without running it; therefore, it is given as image.

```
def mystery_computation(number):
    return mystery_recursion(number, number-1)

def mystery_recursion(a, b):
    if (b <= 0):
        return 0
    else:
        c = mystery_recursion(a, b-1)
        if (a % b == 0):
            c += b
        return c
```

Your mission is to implement, inside a new file named **`NonRecursiveMystery.py`**, the function **`mystery_computation(number)`** that uses a non-recursive approach to carry out the same computation as **`mystery_computation(number)`** (so it has to return the same result for every n). This function may not call any function (be them other functions or itself). Do not forget to add thorough documentation, concisely but precisely explaining what the function does and the meaning of its return value!

Part B: Get to the Zero

James Bond must pass the corridor leading to his enemy's lair undetected. To help him, Q provides him with a special gadget detecting the timing of when each part of the corridor is checked for intruders. The resulting map looks like this, where if a part of the corridor contains e.g. the number 4, then it is possible to move, undetected, from this part of the corridor only to a part exactly 4 steps away from it.

3	6	4	1	3	4	2	5	3	0

The mission starts with James Bond on the leftmost (first) part of the corridor, and his goal is to reach rightmost (last) part, occupied by a zero, denoting the villain. At each step, James Bond may move a distance indicated by the integer in the part of the corridor he is currently at. Bond may move either left or right along the corridor, but may not move past either end and may not change direction mid-way through a step. For example, the only legal first move is for him to run as fast as he can three squares to the right (to the corridor part marked with a 1), because there is no room to move three squares to the left.
For example, the above puzzle may be solved by making the following series of moves:

3	6	4	1	3	4	2	5	3	0
3	6	4	**1**	3	4	2	5	3	0
3	6	**4**	1	3	4	2	5	3	0
3	6	4	1	3	4	**2**	5	3	0
3	6	4	1	3	4	2	5	**3**	0
3	6	4	1	3	**4**	2	5	3	0
3	6	4	1	3	4	2	5	3	0

Even though the above puzzle is solvable — and indeed has more than one solution — some puzzles of this form may be unsolvable. Consider, for example, the following puzzle:

3	1	2	3	0

Here, one can bounce between the two 3's, but cannot reach any other square. (Don't worry, though. James Bond always will find a way to reach the villain, and will return.)
In a new file named **`GetToTheZero.py`**, write the function: `is_solvable(start, board)` that takes a starting position of the agent, and a puzzle board (list). The function should return true if it is possible to solve the puzzle from the configuration specified in the parameters, and should return false if it is impossible. (Well, for normal humans. Nothing is impossible for Bond.)
You may assume that the list is full with positive integers, except for the last entry, the goal square, which is always zero. The values of the elements in the list must be the same after calling your function as they are beforehand (which means that if you choose to change them during processing, you need to change them back!). Your implementation should be recursive, but efficient enough to be able to handle large puzzles.

Part C: DNA Alignment

Deoxyribonucleic acid (DNA) is a nucleic acid that contains the genetic instructions used in the development and functioning of all known living organisms and some viruses.
A DNA strand is a sequence of four bases in various orders. The four bases found in DNA are adenine (abbreviated A), cytosine (C), guanine (G) and thymine (T). Thus DNA strands can be coded as long strings consisting of four letters (A,C,G,T).
One of the most basic tasks in computational biology is to measure the similarity of 2 given DNA sequences. This can be done by finding the best alignment between the two sequences.
Given two DNA sequences, x and y, an alignment x', y' is achieved by adding dashes at arbitrary places in one or both sequences, making them the same length (this means we introduce 'holes' in the DNA sequence, actually admitting that there is something missing but we cannot determine what). The introduction of the dashes is such that the constraint that no two dashes are in the same position. If $x'[i]$ denotes the i'th element of x' and $y'[i]$ is the same for y', an alignment is scored by the equations:

$$\sum_{i=1}^{\text{length of } x'} \text{score}(x', y', i)$$

$$\text{score}(x', y', i) = \begin{cases} +1, & \text{if match} \\ -1, & \text{if mismatch} \\ -2, & \text{if gap} \end{cases}$$

$\text{match: } x'[i] == y'[i]$
$\text{mismatch: } x'[i] \neq y'[j] \text{ and } (x'[i] \neq \text{ "}-\text{" and } y'[i] \neq \text{ "}-\text{"})$
$\text{gap: } x'[i] \neq y'[i] \text{ and } (x'[i] == \text{ "}-\text{" xor } y'[i] == \text{ "}-\text{"})$

Hence, for the given two DNA strands ACA and TACG, a possible (not optimal) alignment is:

```
-ACA-
TAC-G
```

and its score is -4. Another possible alignment is:

```
-ACA
TACG
```

with a score of -1.

In a new file named **`AlignDNA.py`**, write two functions:

1. The function get_alignment_score that takes two aligned DNA strands (you may assume that they have the same length) and three optional scores value: match with default value of 1, mismatch with default value of -1 and gap with default value of -2. The function returns the score of the alignment. The implementation of this function doesn't have to be recursive. You may assume the strands contain only valid DNA and gap ('A','C','G','T','-').
2. The function get_best_alignment_score that takes two unaligned DNA strands (strings) and three optional scores value (the same as before). The function calculates and returns a tuple of the highest score for the alignment of the two strands and the two aligned strands. The implementation of this function must be recursive.
   You may assume the strands contain only valid DNA ('A','C','G','T').

You don't need to care for efficiency in the implementation of this function. It will usually be slow, long sequences might take hours (and possibly crash).

<u>Bonus</u>

• A bonus of 5 points will be granted for superb efficiency in the implementation.

Submission and Further Guidelines:

Cheating

- Don't.
- Write your own code.
- Don't copy from the internet. We also know how to find these solutions.

Creating a tar file

- By now, you should have created/edited the following files:
    1. NonRecursiveMystery.py
    2. GetToTheZero.py
    3. AlignDNA.py
    4. README (as explained in the course guidelines)
- Create a TAR file named ex5.tar containing only the above files by invoking the shell command:
  tar cvf ex5.tar NonRecursiveMystery.py GetToTheZero.py AlignDNA.py README
  The TAR file should contain only these files!

Submitting the tar file

- You should submit the file ex5.tar via the link on the course home page, under the ex5 link.
- Note that submission requires you to be registered as a student and logged in.
- **Few minutes** (much more than other exercises) after you upload the file you should receive an email to your university mailbox. That email contains the test results of your submission and the pdf file that was passed to the grader,
- Read the submission file again and make sure that all your files appear and fully readable.

Running the tests yourself

- Running testers will be very slow (around 5 minutes).
- There are two ways you can run these testers and see the results without submitting your exercise and getting the pdf:
    1. The first – place your tar file in an empty directory, go to that directory using the shell and type: `~intro2cs/bin/testers/ex5 ex5.tar`
    2. The second – download a file - ex5testing.tar.bz2. Extract the contents (using tar –xjv ex5testing.tar.bz2) of the file and follow the instructions on the file named TESTING.

School Solution

- Can be found in:
    1. ~intro2cs/bin/ex5/mystery
    2. ~intro2cs/bin/ex5/GetToTheZero
    3. ~intro2cs/bin/ex5/AlignDNA

Good luck!