CS 3430: SciComp with Py
Coding Exam 1

Vladimir Kulyukin
Department of Computer Science
Utah State University

February 14, 2017

1 Introduction

- This exam has three problems. You can code your solutions either in Py2 or Py3. The files `p2_exam_1.py` and `py3_exam_1.py` in the zip archive contain the stubs of all the functions you should define for this exam. Write your name and A-number at the beginning of either file before you submit it via Canvas. The zip archive also contains the file `students.txt` for Problem 3.
- This exam is open books and open notes. If you use any online matrials, please mention them in the comments at the beginning of the file you submit.
- You may not collaborate with anyone on this exam orally, digitally, or in writing.
- If you cannot code on your RPi (e.g., you have to be on campus for an earlier class and your RPi setup is at home), you can code everything up on your laptop or an available desktop.
- Your solutions are due at Canvas by 10:30am sharp on Feb. 14, 2017. You are always better off submitting an incomplete solution for partial credit than a late complete solution for zero credit. To put it differently, do not email me your solutions after the Canvas submission closes. You will not receive any credit for it. If you think this is unfair, think of a coding phone interview where an interviewer gives you a specific amount of time to complete a problem. What do you think the interviewer's reaction will be if, when the time is up, you will ask him or her for more time to work on the problem? Obviously, the time constraints do not apply to students with learning disabilities who may need additional time on exams.
- Happy Hacking!

2 Problem 1 (5 pts)

A railroad switch consits of a left track, a spur, and a right track, as shown in Figure 1. From the left track, only its rightmost car can be sent either to the spur or to the right track. From the spur a car can be sent only to the right track. When a car is sent to the right track, it becomes the leftmost car there. The spur functions in the first-in-last-out manner: a car on the spur cannot be sent to the right track unless all the cars that came after it have been dispatched to the right track. In other words, only the top car on the spur can be dispatched to the right track. No car can be dispatched from the right stack either to the left stack or the spur.

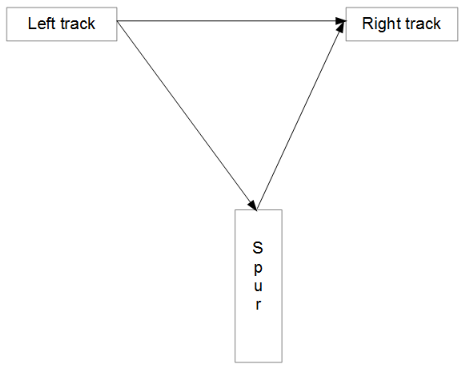

Figure 1: An railroad switch

In the initial state, all cars numbered 1 through n are on the left track with 1 being the leftmost car and n being the rightmost car, as shown in Figure 2.

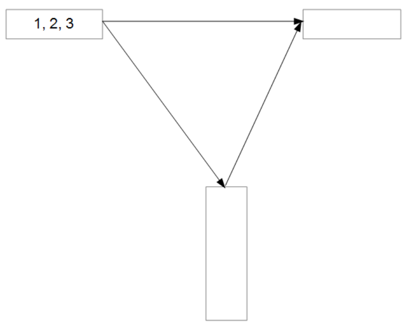

Figure 2: Initial state of the switch

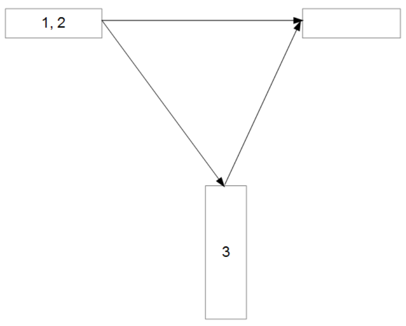

Figure 3: Car 3 dispatched to the spur

In the state in Figure 2, car 3 can be sent either to the spur, which results in the state of the switch depicted in Figure 3 or to the right track, which results in the state depicted in Figure 4.

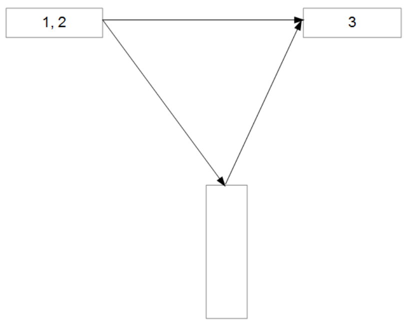

Figure 4: Car 3 dispatched to the right track

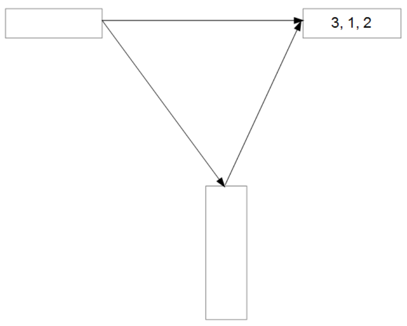

Figure 5: A final state of the swtich with a complete arrangement of cars on the right track

A final state of the switch occurs when the left track and the spur are both empty and all the cars that were initially on the left track are now on the right track, as shown in Figure 5. Implement the function `arrange_cars(n)` that takes the number of cars n on the left track arranged in ascending order from 1 up to n and outputs all possible arrangements of cars on the right track as a list of n-tuples. Here are a few test runs.

```
>>> arrange_cars(1)
[(1,)]
>>> arrange_cars(2)
[(1, 2), (2, 1)]
>>> arrange_cars(3)
[(1, 2, 3), (2, 1, 3), (1, 3, 2), (3, 1, 2), (3, 2, 1)]
```

Use list comprehension to write the function `find_trains_starting_with_car(car, n)` that takes the output of `arrange_cars(n)` and returns all arrangements that start with *car*. A few test runs.

```
>>> find_trains_starting_with(1, 2)
[(1, 2)]
>>> find_trains_starting_with(3, 3)
[(3, 1, 2), (3, 2, 1)]
```

Problem 2 (3 points)

A square bottom line relief (SBLR) tank is a 2D $h \times w$ matrix whose bottom line consists of w stacks of black square blocks. Positive integers h and w do not have to be equal. Figure 6 shows three 3×3 SBLR tanks.

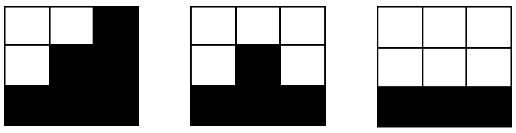

Figure 6: Three SBLR tanks

The three tanks can be presented as a tuple of tuples with 0's standing for empty (i.e., non-block) cells and 1's for block cells. For example, the three tanks in Figure 6 can be represented as follows:

```
left_tank = ((0, 0, 1),
             (0, 1, 1),
             (1, 1, 1))

mid_tank = ((0, 0, 0),
            (0, 1, 0),
            (1, 1, 1))

right_tank = ((0, 0, 0),
              (0, 0, 0),
              (1, 1, 1))
```

When the tanks are filled with water to the brim, water can go only into the non-block cells. The amount of water in a tank is the number of non-block cells filled with water. Figure 7 shows three tanks filled to the brim and the corresponding water amounts. The blue cells depict cells with water.

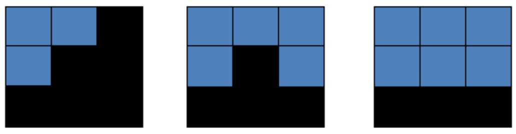

Figure 7: Three SBLR tanks filled with water to the brim. Water amounts are: 3 (left); 5 (middle), 6 (right)

In a tank of height h a water line can be placed at any row from 0 down to $h - 1$. Figure 8 shows three water lines placed at all possible rows of 3×3 tanks.

Given a water line and a tank, the amount of water in that tank is the number of empty cells at and below the line. Figure 9 shows three tanks with different lines and the corresponding water amounts.

Write the function `water_amount(tank, h, w, wl)` that computes the amount of water in an $h \times w$ tank *tank* represented as a 2D tuple given water line *wl*. A few test runs.

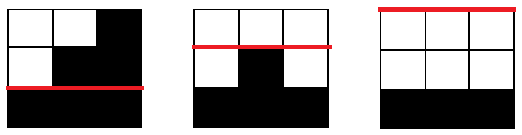
Figure 8: Three water lines in empty tanks: at row 2 (left), at row 1 (middle), at row 0 (right)

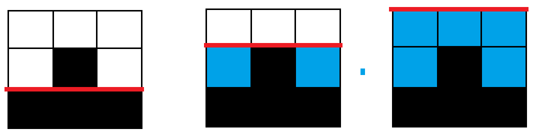
Figure 9: Three water lines and amounts of water at and below the lines: 0 (left), 2 (middle), 5 (right)

```
>>> tank1 = ((0, 0, 0),
             (0, 1, 0),
             (1, 1, 1))

>>> water_amount(tank1, 3, 3, 2)
0
>>> water_amount(tank1, 3, 3, 1)
2
>>> water_amount(tank1, 3, 3, 0)
5
```

Problem 3 (2 points)

A students record line is a tab-separated string that contains four key-value pairs specifying the students initials, the students GPA, the first four digits of the students A-number, and the student's date of birth. Five student records are shown below.

```
INITS=J.P. GPA=3.5 ANUM=1927 BD=11/01/2000
GPA=4.0 INITS=N.K. ANUM=2732 BD=12/01/1997
ANUM=9803 GPA=3.25 INITS=M.D. BD=10/03/2002
BD=05/01/1999 ANUM=1248 GPA=2.87 INITS=D.W.
GPA=3.80 INITS=L.M. BD=03/17/2000 ANUM=6752
```

A student's initials are prefixed with `INITS=`. A student's GPA is prefixed with `GPA=`. The first four digits of a student's A-Number are prefixed with `ANUM=`. A student's birthday is prefixed with `BD`. The key-value pairs can occur in any order. Implement the function `build_record(line)` that takes a string `line` and constructs a list of 2-tuples corresponding to the four key-value pairs. The first elements of tuples are keys, i.e., `INITS`, `ANUM`, `GPA`, and `BD`. The second elements are key values. The value of `GPA` is a float. The rest are strings. Note that `build_student_record` does not require the use of regexps. Here is a test run.

```
>>> srec = 'INITS=J.P.      GPA=3.5     ANUM=1927      BD=11/01/2000'
>>> build_student_record(srec)
[('INITS', 'J.P.'), ('GPA', 3.5), ('ANUM', '1927'), ('BD', '11/01/2000')]
```

Write the function `sort_student_records(fp)` that takes a file path *fp* to a file with one student record per line, builds a list of tuple records for all lines, and sortes the tuple records by GPA in descending order. The zip archive for the exam contains the file `students.txt` that contains five student records given above. Here is what a call to `sort_student_records('students.txt')` should return.

```
>>> sort_student_records('students.txt')
[[('GPA', 4.0), ('INITS', 'N.K.'), ('ANUM', '2732'), ('BD', '12/01/1997')],
[('GPA', 3.8), ('INITS', 'L.M.'), ('BD', '03/17/2000'), ('ANUM', '6752')],
[('INITS', 'J.P.'), ('GPA', 3.5), ('ANUM', '1927'), ('BD', '11/01/2000')],
[('ANUM', '9803'), ('GPA', 3.25), ('INITS', 'M.D.'), ('BD', '10/03/2002')],
[('BD', '05/01/1999'), ('ANUM', '1248'), ('GPA', 2.87), ('INITS', 'D.W.')]]
```

What To Submit

Submit `py2_exam_1.py` or `py3_exam_1.py` via Canvas by 10:30am on Feb. 14, 2017.