COSC 251 – Programming Languages
Project 2
Spring 2016

Objective: Use Python to solve a bevy of problems.

Your Task: The SMCM Programming Team has traditionally competed each fall in a programming competition hosted by colleges in our region. As part of their preparation, they solve a wide variety of problems all of which could be solved via Python without too many issues. For this project, you will provide solutions to 3 of these problems. You will be required to answer all three questions and each question is worth 33 1/3 points.

For all questions, input may be provided to your function through the parameter list, or through user input handled by your function. Pay attention to each description for information on which questions are which. Also, all output should be handled by your function, do not return any data. Some problems also have a "code golf" aspect to them, you must get your solutions under a particular character count, or you will be penalized.

Q1: This problem involves determining the number of routes available to an emergency vehicle operating in a city of one-way streets. Given the intersections connected by one-way streets in a city, you are to write a program that determines the number of routes between each pair of intersections. A route is a sequence of one-way streets connecting two intersections

Intersections are identified by non-negative integers. A one-way street is specified by a pair of intersections: *(j, k)* indicates a street going from intersection *j* to intersection *k*. We can model two-way streets by specifying two one-way streets: *(j, k)* and *(k, j)* indicate that there is a two-way street between intersections *j* and *k*. We will input such pairs as integers separated by whitespace, dispensing with the comma and parentheses.

Consider a city of four intersections connected by the four one-way streets (0, 1), (0, 2), (1, 2), and (2,3). There is one route from intersection 0 to 1, two routes from 0 to 2 (the routes are 0 → 1 → 2, 0 → 2), two routes from 0 to 3, one route from 1 to 2, one route from 1 to 3, one route from 2 to 3, and no other routes.

It is possible for an infinite number of different routes to exist. For example if the intersections above are augmented by the street (3, 2), there is still only one route from 0 to 1, but there are infinitely many different routes from 0 to 2. This is because the street from 2 to 3 and back to 2 can be repeated yielding a different sequence of streets and hence a different route. Thus the route 0→2→3→2→3→2 is different from 0→2→3→2.

The input is a sequence of city specifications. Each specification begins with the number of one-way streets to be input in the city followed by that many one-way streets given as pairs of intersections. In all cities, intersections are numbered sequentially from 0 to the "largest" intersection. All integers in the input are separated by whitespace. The user inputs this specification. Note that while the user will only be entering one-way pairs, he or she can still create two-way streets as noted above. There will never be a one-way street from an intersection to itself. No city will have more than 30 intersections. You can assume that the user knows what he or she is doing and will enter input in the proper format. The end of the input is indicated by the user entering -1 in the input.

For each city specification, print a square matrix of the number of different routes from intersection *j* to intersection *k* is printed. That is, the entry at row *j*, column *k* is the number of different routes from intersection *j* to intersection *k*. Print each matrix in the format shown in the examples, preceded by the string "matrix for city n" where n is the number of the city starting at 0, going to the number of cities noted in the sequence.

Print -1 to denote an infinite number of different paths between two intersections. The amount of whitespace used to separate entries in a row is irrelevant; you need not worry about justifying and aligning the output of the matrices.

NOTE: There may be more than one city per input string! You also cannot assume that end-of-line indicates the end of input either!

Example:

```
Input:
7 0 1 0 2 0 4 2 4 2 3 3 1 4 3
5
0 2
0 1 1 5 2 5 2 1
9
0 1 0 2 0 3
0 4 1 4 2 1
2 0
3 0
3 1 -1
```

```
matrix for city 0
0 4 1 3 2
0 0 0 0 0
0 2 0 2 1
0 1 0 0 0
0 1 0 1 0
matrix for city 1
0 2 1 0 0 3
0 0 0 0 0 1
0 1 0 0 0 2
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
matrix for city 2
-1 -1 -1 -1 -1
0 0 0 0 1
-1 -1 -1 -1 -1
-1 -1 -1 -1 -1
0 0 0 0 0
```

Method signature: Problem1( )
User input required. No code golf requirement.

Q2: During the October Revolution of 1917, the Bolsheviks used a simple transposition cipher to send messages. The message was broken into 25-character blocks. The end of the message would have additional characters added until the message length was a multiple of 25. Each block was then handled independently.

The message was written in a 5x5 grid. Spaces between words were ignored.
Message: `THE COMMISSAR SAYS HELLO`
Written in grid, with padding characters added:

```
THECO
MMISS
ARSAY
SHELL
OABCD
```

The encryption key is a permutation of the integers 1-5, which determines the order in which the *columns* are read to produce the ciphertext:

Key: `54123`
```
OSYLD CSALC TMASO HMRHA EISEB
```

You are to take in a string with a Bolshevik-encrypted message. Each string is written with the key (a permutation of the integers from 1-5), followed by the 25-character message in 5-character groups. The message will be entirely alphabetic (no digits or punctuation), all upper-case. Our sample message would be presented as

```
54123 OSYLD CSALC TMASO HMRHA EISEB
```

Your output is the decrypted message, also in 5-letter groups, each group separated by a single space with a newline (with no trailing space) at the end of each message. You do not need to strip out the padding.

```
THECO MMISS ARSAY SHELL OABCD
```

Example:

Input:	Output:
54123 OSYLD CSALC TMASO HMRHA EISEB	THECO MMISS ARSAY SHELL OABCD
41532 IEVEA AATST SAENA GRNMA IGITA	AIGIS AGREA TINVE STMEN TAAAA

Method signature: Problem2(s)
No user input allowed. Par: 166 characters, including whitespace.

Q3: Write an interpreter for the brainf*ck language (http://www.muppetlabs.com/~breadbox/bf/). Input will be passed to the method via a string parameter of ASCII characters and may be up to 30,000 characters in length. Input and output should be executed as specified by the brainf*ck language. You do not need to accommodate for nested loops.

Input: ++++++++[>++++[>++>+++>+++>+<<<<-]>+>+>->>+[<]<-]>>.>---.+++++++..+++.>>.<-.<.+++.------.--------.>>+.>++.
Output: Hello World!

Method signature: Problem3(s)
No user input allowed. Par: 352 characters, including whitespace.

Deliverables: your Python source. All three sets of code should be stored in a single file named Proj2.py, following the above method signatures.

Expectations: The code should be clean, concise, well-commented and correct. If you use an outside source, be sure to document that source. Significant use of outside sources will result in a deduction. Grading rubric will be provided a week ahead of the due date. A driver with the input from the examples will be provided shortly. You are allowed to work in teams of up to three for this project. If you choose to work in a team, one member of the team is required to email me with who they are working with by 5pm, 2/29.

Learning Targets: Python development experience, classic problem solving, a bit of code optimization, and a ton of reading comprehension.

Credit: Collegiate programming competitions.

DUE: March 8th, 11:59pm via Blackboard, team information due 5pm 2/29 via email.