COSC 251 – Programming Languages
Project 2
Spring 2014

Objective: Use Python to solve a bevy of problems.

Your Task: The SMCM Programming Team competes each fall in a programming competition hosted by colleges in our region (talk to Lindsay if you're interested in joining the team). 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 4 of these problems. You will be required to answer all four questions and each question is worth 25 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.

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 $0 \rightarrow 1 \rightarrow 2$, $0 \rightarrow 2$), 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 $0\rightarrow2\rightarrow3\rightarrow2\rightarrow3\rightarrow2$ is different from $0\rightarrow2\rightarrow3\rightarrow2$.

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.

Q2: Consider the sequence of all words formed entirely of lower-case letters and having the following properties:

- A word *x* appears before a word *y* if *x* is shorter than *y*.
- Any two words of the same length appear in alphabetical order.
- The sequence contains exactly the words whose letters appear in strictly increasing order (for example ‘a’, ‘ab’, ‘abc’, but not ‘ba’ or ‘bb’).

To each word in this sequence, associate a positive integer index, starting with 1:

a → 1
b → 2
…
z → 26
ab → 27
ac → 28
…
az → 51
…
vwxyz → 83681

Your program is to read a series of lower-case words from one to five letters long, separated by whitespace. For each word read, if the word is invalid print the number 0, and otherwise print its index in the sequence.

Example:

```
Input:
z a

cat
vwxyz
```

```
Output:
26
1
0
83681
```

Method signature: Problem2(s)
No user input allowed.

Q3: If you've ever taken a chemistry course, you've seen this sort of thing denoting a chemical reaction:

$$2Na + 2H_2O = 2NaOH + H_2$$

The symbols 'H', 'O', and 'Na' stand for various kinds of elements; each term (separated by '+' and '=') represents a molecule; the subscripted integer numerals after each element (defaulting to 1) represent the number of that element in the molecule (though elements can occur multiple times – as in acetic acid, 'CH_3COOH'); and the integer numeric coefficients in front of a molecule (again defaulting to 1) represent the number of participating molecules of the attached type. You are to write a program that checks such equations for balance. For example, your program will accept

$$6H_2O + 6CO_2 = 6O_2 + C_6H_{12}O_6$$

but indicate that

$$O_2 + H_2 = H_2O$$

is erroneous.

The input to your program will consist of equations of the form shown above, separated by whitespace, except that the equations themselves contain no whitespace and subscripted numerals are not written with subscripts. Each chemical element is denoted by a single upper-case letter, an upper-case letter followed by one letter, or an upper-case letter followed by two letters (Uuo for instance).

For each equation, produce a line of output that echoes the equation followed either by the phrase "balances" or "does not balance" in the format shown in the example below.

Example:

Input:	Output:
6H2O+6CO2=6O2+C6H12O6 2Na+2H2O=2NaOH+H2 C6H12O6=3C2H2+3O2	6H2O+6CO2=6O2+C6H12O6 balances 2Na+2H2O=2NaOH+H2 balances C6H12O6=3C2H2+3O2 does not balance

Method signature: Problem3(s)
No user input allowed.

Q4: You are a lone human, without guns or even a crowbar, who is attempting to run across a field of zombies and not get eaten. The field is represented as a square 2D grid, where each grid cell contains a number representing the number of zombies on that square. You start in the lower-left hand corner, and need to reach the upper-right hand corner.

For each square you need to cross on the path that has a non-zero number of zombies, your chance of survival will be reduced by 5% per zombie. So if you cross two squares with one zombie on them or one square with two zombies on it, your overall chances of survival will be 90%. The starting and ending squares will never have zombies on them.

Movement can only consist of moving up or right, with no diagonal moves allowed. Thus each possible path will have exactly 2(k-1) moves, where k is the width and height of the grid.

The input will consist of a number, k, which is the vertical and horizontal size of the grid where 0 < k <= 15. This will be followed by a newline character and k lines of k numbers between 0 and 2 (inclusive), separated by spaces.

Output should print a single number between 0 and 100, representing the probability of survival for the best possible path.

Example:

Input1: 3 0 0 0 0 1 0 0 2 0	Output1: 100
Input2: 4 1 0 2 0 0 2 1 0 0 1 0 0 0 1 0 1	Output2: 95

Method signature: Problem4()
User input required.

Deliverables: your Python source. All four 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 pairs for this project, though remember that you are not allowed to work with the same partner from the C++ project. If you choose to work in a pair, one member of the team is required to email me with who they are working with by 5pm, 2/28.

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

Credit: 2011-2012 Programming Team, Lindsay Jamieson and the 2010 UC-Berkeley Programming Competition. Some problems are from a UVa repository of problems and one of the problems is from a Russian high school competition.

DUE: March 12th, 11:59pm via Blackboard, pair information due 5pm 2/28 via email.