Graded Assignment 4

Due date: May 21, 2018 at 20:00

This is an individual assignment. You may discuss it with others, but your formulations, your code, and all the required material must be written on your own. In any case, you must acknowledge the sources used and clearly mention any help received from colleagues.

Exercise 1. *(50% grade)*

Consider a connected graph G. An *articulation point*(or cut vertex) in G is any vertex v such that, removing v (and its adjacent edges) from G results in a *disconnected graph.*

Write a Python function called *articulation_points(G)* that takes a connected graph G in its adjacency-list representation, and returns an array containing the articulation points in G.

Submit your solution within a Python program that reads a graph from the standard input, and then outputs all its articulation points, one per line, on the standard output. The input graph G is formatted as follows: the first line contains two numbers n and m that is the number of vertices in G and the number of edges in G, then each of the following m lines consists of two numbers u and v, separated by a space, and represents an undirected edge (u, v) in G. The complexity of your algorithm should be $O(n + m)$.

Exercise 2. *(50% grade)*

You're trapped inside a maze represented as a $n \times m$ grid where every cell can be empty or blocked. You start in the cell marked by s and would like to reach the cell marked by e, furthermore you would like to do it in as few steps as possible. In one step, if you're located in cell (x, y), you can go to any of the 4 following cells: $(x+1, y)$, $(x-1, y)$, $(x, y+1)$, $(x, y-1)$ as long as the cell you go to is a valid one and is not blocked.

The grid is given in the input as follows. The first line contains two integers $n, m \leq 1000$ which represent the number of rows and number of columns of the grid. Each of the next n lines contains m characters where an `o` represents a free cell, an `x` represents a blocked cell, `s` represents the starting position and `e` represents the ending position. There will be exactly one `s` and one `e` in the grid.

Output on a single line *no* if you can't reach the ending cell, or a single integer - the minimum number of steps necessary in order to reach the ending cell from the start cell. The complexity of your algorithm should be $O(n \cdot m)$.

Examples:

Input	Output	Input	Output

`3 3`
`osx`
`xoo`
`eox`
	`3`	`3 4`
		`ooso`
		`ooxx`
		`oxeo`
			`no`