CSC373 – Problem Set 5

Remember to write your **full name(s)** and **student number(s)** prominently on your submission. To avoid suspicions of plagiarism: at the beginning of your submission, **clearly state any resources (people, print, electronic) outside of your group, the course notes, and the course staff, that you consulted**.

Remember that you are required to submit your problem sets as both LaTeX `.tex` source files and `.pdf` files. There is a 10% penalty on the assignment for failing to submit both the `.tex` and `.pdf`.

Due Oct 31, 2020, 22:00; required files: maximum_likelihood.py

Answer each question completely, always justifying your claims and reasoning. Your solution will be graded not only on correctness, but also on clarity. Answers that are technically correct that are hard to understand will not receive full marks. Mark values for each question are contained in the [square brackets].

You may work in groups of up to THREE to complete these questions.

Please see the course information sheet for the late submission policy.

[15 points]

You are playing the new video game "Super Luigi", and are experiencing difficulties passing through Level "Superstar road". You want to estimate how likely it is that you can complete the level. To this end, you've decided to model the level as a directed graph, where each vertex corresponds to a safe location you can reach in the level, and each edge $(a \to b)$ represents a jump you can make from a safe location a to a safe location b. (Note that the edges are directed, and an $(a \to b)$ edge need not imply a $(b \to a)$ edge). After an excessive amount of experimentation, you have come up with some estimates, which allow you to assign each edge $e = (a \to b)$ a number $p(e) \in [0, 1]$, which can be thought of as the probability that you can successfully move from location a to location b.

Your goal is to design and implement an algorithm that finds a directed path from the starting safe location s to the final safe location t that *maximizes* the probability of a successful traversal, together with the total probability of the path.

Assume that different jumps are independent, and hence the outcome of one jump does not affect the probabilities of the outcome of another jump.

For example, given a directed graph $G = (V, E)$, with

$$V = \{1, 2, 3, 4\}, \quad E = \{(1 \to 2), (1 \to 3), (1 \to 4), (2 \to 4), (3 \to 4)\},$$
$$p((1 \to 2)) = 0.2, p((1 \to 3)) = 0.5, p((1 \to 4)) = 0.1, p((2 \to 4)) = 0.5, p((3 \to 4)) = 0.3,$$
$$s = 1, t = 4,$$

the (s, t) path that maximizes the probability of a successful traversal is $(1 \to 3 \to 4)$ achieving a probability of $0.5 \times 0.3 = 0.15$.

(15 points) Implement the function `maximum_likelihood` which has the following input parameters:

- `num_vertices`: an integer $n \geqslant 0$ specifying the number of vertices in the graph. (You can assume that the vertices are numbered $1, \ldots, n$)
- `edges`: a list of triples, where each triple (a, b, p) represents a directed edge $(a \to b)$ with probability $p = p((a \to b))$

- `s`: an integer s representing the starting vertex
- `t`: an integer t representing the target vertex

Your algorithm should output:

- a list of integers representing the path that maximizes the probability of a successful traversal.
- a float number representing the weight of such path.

For the above example, a call to the `maximum_likelihood` looks as follows:

```
maximum_likelihood(
 4,
 [[1,2,0.2], [1,3,0.5], [1,4,0.1], [2,4,0.5], [3,4,0.3]],
 1, 4)
```

The output should the path `[1,3,4]` and the weight `0.15`.
Requirements:

- Your code must be written in Python 3, and the filename must be `maximum_likelihood.py`.
- We will grade only the `maximum_likelihood` function; please do not change its signature in the starter code. You may include as many helper functions as you wish
- You are not allowed to use the built-in Python dictionary or set or any other built-in data-structure (other than a list or array).
- To get full points, your algorithm must have an asymptotic worst-case time complexity of $O(m \log n)$, where m is the number of edges in the graph.
- For each test-case that your code is tested on, your code must run within 10x the time taken by our solution. Otherwise, your code will be considered to have timed out.

Please include comments in your code to explain what your algorithm is doing.