CSC263 – Problem Set 3

Remember to write your **full name** and **student number** 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 Monday, March 30, 2020, 22:00; required files: ps3.pdf, ps3.tex and cities.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.

1. **Number of Shortest Paths [total: 10]** Let $G = (V, E)$ be a unweighted directed connected graph. Design an algorithm that determines for two given nodes $s, t \in V$ the number of shortest paths from s to t in $O(|V| + |E|)$.

   (a) Write your algorithm in pseudocode and describe its functionality in concise (but precise) English.

   (b) Prove the correctness and runtime of your algorithm in Part 1a.

2. **MST [total: 8]** Let $G = (V, E)$ be a weighted undirected connected graph that contains a cycle, and let e be a maximum-weight edge among all edges in the cycle. Prove that there exists a minimum spanning tree of G which does NOT include e.

3. **Catch the Human [total: 10]** In a rectangular field with `m` by `n` squares there is a human **H** and two robots **R1** and **R2** (see Figure 1). Each of the three starts off at some initial square. At the beginning of the game, **H, R1, R2**, all know each other's positions.

   Each turn, the **H** makes the first move, followed by **R1** and **R2**. A move can be either exactly one square up, down, left, or right (or can choose to stay at their current location). Grey squares are inaccessible as they mark obstacles. If **R1** or **R2** can move to the same square which **H** is on, they catch **H**.

   In order to escape, at the beginning of the game, before the first turn, **H must declare** a single square **Y** (e.g. $(3, 1)$) at the edge of the field (the yellow zone) that the human is going to reach for. If **H** reaches the square **Y**, it is able to escape at its next turn if it is not caught by **R1** or **R2** before that. Otherwise, **H** is unable to escape.

   Note that the game can end only at the end of a full turn. So in an example scenario where: 1) **Y** is empty before a turn, 2) **H** makes its next move and reaches **Y**, 3) **R1** makes its next move and reaches **Y**; **H** gets caught and is unable to escape.

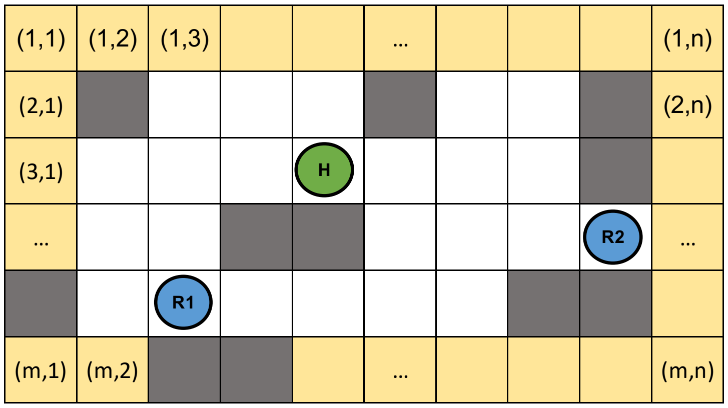

Figure 1: Rectangular field for *Catch the Human* game.

   Develop an efficient algorithm with the following input and output:

   Input

- values of `m` and `n`
- coordinates of the initial squares for **H** as (x_H, y_H), **R1** as (x_{R1}, y_{R1}), and **R2** as $= (x_{R2}, y_{R2})$ with $x \in \{1, n\}$ and $y \in \{1, m\}$
- a list of coordinates of inaccessible (grey) squares

Output

- Output **True** if **H** can declare a yellow square **Y** such that **H** is able to escape. If there is no such **Y**, output **False**.

Assumptions

- Inaccessible squares are placed in a way that it is possible to move from any free square to any other free square in the field

Describe and design your algorithm by answering the following questions:

(a) Describe how to construct a graph to solve this problem. More precisely, describe the detailed procedure of creating the adjacency list of the graph from the given input. Clearly indicate which data structures you use. Analyse the runtime of your graph construction procedure.

(b) Given the graph that is constructed in Part 3a, how do you efficiently determine whether **H** can escape? Describe your algorithm in concise (but precise) English, and analyse its runtime.

Programming Question

The best way to learn a data structure or an algorithm is to code it up. In each problem set, we will have a programming exercise for which you will be asked to write some code and submit it. You may also be asked to include a write-up about your code in the PDF/TEXfile that you submit. Make sure to **maintain your academic integrity** carefully, and protect your own work. The code you submit will be checked for plagiarism. It is much better to take the hit on a lower mark than risking much worse consequences by committing an academic offence.

4. **Around the World Trip** [**total: 12**] You are planning an around-the-world trip with your two best friends for the summer. There is a total of n cities that the three of you want to visit. As you are traveling around the world, you are worried about time zones and airport access. Therefore, some cities can only be visited after visiting another city first, which is in a nearby timezone or has an airport, which are expressed as a list of pairs (cityX,cityY) (cityX can only be visited after visiting cityY).

Given the total number of cities and a list of dependency pairs, is it possible for you all to visit all cities?

Your task is to write the function `can_visit_all_cities`, which determines whether visiting the n cities is possible or not given the dependencies. **(8 points)**

Requirements

- Your code must be written in Python 3, and the filename must be `cities.py`.
- We will grade only the `can_visit_all_cities` function; please do not change its signature in the starter code. include as many helper functions as you wish.
- You are **not** allowed to use the built-in Python dictionary or set.
- To get full marks, your algorithm must have average-case runtime $\mathcal{O}(m+n)$, where m is the number of dependencies given.

Input/Output Specification Please see the python starter code for input / output specifications.

Write-up (4 points): in your `ps3.pdf/ps3.tex` files, include the following: an explanation of how your code works, justification of correctness, and justification of desired $\mathcal{O}(m+n)$ average-case runtime.