CSC373 – Problem Set 2

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 Nov 14, 2022, 22:00; required files: ps2.pdf, ps2.tex, likeliest_evolution.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.

1. [**15 points**]

   You are planning a new office for the tech giant SpaceNook. You know that one important aspect to keeping your employees happy and functioning well is quick access to free snacks. You will determine amongst all the seating locations for employees in the new office, what is the longest time it would take for an employee to reach their nearest snack station.

   You have access to a possible map for the new office for SpaceNook. All the employee locations in the new office have been numbered from $[n] = \{1, \ldots, n\}$.

   You are given a list P of pathways in the office (including hallways, stairs, elevators), with $|P| = m$. Each such pathway connects two end points that are among the office locations numbered $1, \ldots, n$, and is specified as a pair of such locations. For each pathway, you are also provided the time in minutes required to travel along the pathway in either direction.

   Finally, you are given two lists. The first list $E \subseteq [n]$ contains the locations of all the employees in the office. The second list $S \subseteq [n]$ represents the list of snack stations in the office.

   Your goal is to determine, the shortest distance for each employee location, to its *closest* snack station. Follow the steps below to design an algorithm for this problem that has a worst-case time complexity of $O(m \log n)$.

   (a) **(1 point)** Describe how will you model the problem, and the data-structure(s) that you will use to store the input information.

   (b) **(2 points)** Briefly describe in up to 3 lines how you can solve the problem in $O(|E| m \log n)$ worst-case time, where $|E|$ denotes the number of employee locations in the office.

   (c) **(2 points)** Briefly describe in up to 3 lines how you can solve the problem in $O(|S| m \log n)$ worst-case time, where $|S|$ denotes the number of snack stations in the office.

   (d) **(5 points)** Propose an algorithm that solves the problem in $O(m \log n)$ worst-case time, and justify its correctness.

   (e) **(3 points)** Write the pseudo-code for the final algorithm described in part 1d

(f) **(2 points)** Describe a short modification to the above algorithm so that it also finds the closest snack station for each employee location in addition to the shortest distance in $O(m \log n)$ worst-case time.

2. **[15 points]**

You work for a national biotechnology lab, and are studying the evolution of the SARS-CoV-2 virus (also referred to as Covid-19). You are trying to determine the most likely way the current variant evolved from the original variant.

You have a database of genomic sequences of lots of variants of the virus. You have access to a software that predicts the probability of a variant Var1 evolving into another variant Var2. However, this software only works accurately for a small set of the pairs of variants that have similar genomes. These probabilities are given to you as tuples in a list, e.g. [(Var1, Var2, 0.95), (Var2, Var1, 0.95), (Var1, Var3, 0.90), (Var2, Var4, 0.85). (Var3, Var4, 0.90)]. The tuple (Var1, Var2, 0.95) indicates that there is a 0.95 probability of Var1 evolving into Var2.

For the purpose of this problem, we will assume that all the probabilities given in the above list are accurate, and that if a pair is not in the list, that evolution cannot happen directly (but can happen via intermediate variants). In the above example, Var1 cannot directly evolve into Var4, but can indirectly evolve into Var4 (via either Var2 or Var3).

Your goal is to design and implement an algorithm to find a sequence of possible evolutions starting from a given starting variant, e.g. Var1, to the final variant, e.g. Var4, that maximizes the probability of evolution, together with the probability of this sequence of evolutions.

Assume that different jumps are independent, and hence the outcome of one evolution does not affect the probabilities of other evolutions.

In the above example, the sequence of evolutions VAR1 → VAR3 → VAR4 has a probability of 0.90*0.90 = 0.81. This is larger than the probability of 0.95*0.85 = 0.8075 for the sequence of evolutions VAR1 → VAR2 → VAR4.

(12 points) Implement the function `likeliest_evolution` which has the following input parameters:

- `num_variants`: an integer $n \geqslant 0$ specifying the number of variants being considered. (For simplicity, you can assume that the variants are numbered $1, \ldots, n$ rather than using strings)
- `poss_evolutions`: a list of triples, where each triple (i, j, p) represents a possible evolution from variant i to variant j, along with the probability p of transition.
- `s`: an integer s representing the initial variant
- `t`: an integer t representing the final variant

Your algorithm should output:

- a list of integers representing the sequence of variant evolution that maximizes the probability of evolution of the final variant, starting from the initial variant.
- a float representing the total probability of evolution along the above sequence.

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

(3 points) Clearly and concisely explain your approach for the above algorithm.

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

```
likeliest_evolution(
 4,
 [[1,2,0.95], [2,1,0.95], [1,3,0.90], [2,4,0.85], [3,4,0.90]],
 1, 4)
```

The output should the sequence of evolution `[1,3,4]` and the probability of this sequence as `0.81`.

Requirements:

- Your code must be written in Python 3, and the filename must be `likeliest_evolution.py`.
- We will grade only the `likeliest_evolution` 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 $O(|T| \log n)$, where T is the size of the list `poss_evolutions`.
- 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.

3. **[15 points]**

You are working for an electric self-driving taxi startup in Toronto. In order to determine the minimum battery capacity your taxis need to have, you want to compute the maximum time taken by a taxi to travel between any source location and any destination location in the city.

We're going to build the first basic model to estimate this time. For simplicity, we will assume that the city can be described as an $n \times n$ grid, aligned with the cardinal directions (North, South, East, West), where $(0, 0)$ is the South-West corner of the city. You are given an $n \times n$ array M such that $M[i][j]$ corresponds to the (i, j) square in the grid for $0 \leqslant i, j < n$. Every entry $M[i][j]$ is either 1 or 0, where a 1 indicates that the (i, j) square in the city is a 'road', and a taxi can be present in this square at any time. For simplicity, we will assume that on a square representing a road, the car can potentially be facing any of the four directions (North, South, East, West). $M[i][j]$ being 0 indicates an obstacle at the square (i, j), and a taxi can never be present in such a square.

For this basic model, we will assume that the taxi can execute two kinds of instructions,

(a) Move straight ahead by k squares (for some k).

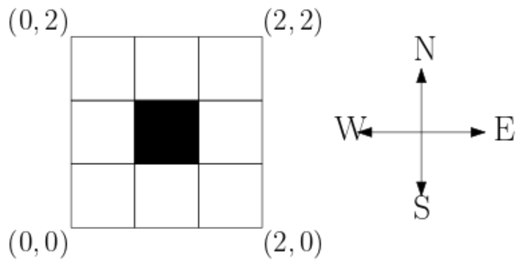
The time taken to move straight ahead by k squares is $(a * k + b)$ minutes where a, b are fixed positive constants that are provided as part of the input. This models the fact that starting and stopping the car takes some extra time (specified by b), in addition to the time taken to move per square (specified by a). A taxi can execute such an instruction only if the k locations straight ahead are roads and not obstacles.

(b) Turn right or left within the same square by 90 degrees.

The time taken to take a single 90 degree turn is c minutes (a non-negative constant specified as part of the input). Note that the car can take a U-turn by taking 2 consecutive turns.

Your goal is to determine a starting square along with a starting orientation (N, S, E, W), and a finishing square along with a finishing orientation, such that the shortest time to go from the starting configuration (location and facing direction) to the finishing configuration is the largest possible over all possible valid choices of starting and finishing configurations. Note that the taxi can never be on a square representing an obstacle, and can never leave the grid.

For example, consider the 3×3 grid, where black indicates an obstacle, and a white square indicates a road.



For $a = b = c = 1$, the maximum possible time taken by the taxi would be 11 minutes. One possible starting and ending configuration that requires 11 minutes is to go from the starting square $(1, 2)$, where at the beginning the taxi is facing North, to go to the final location $(1, 0)$ with the taxi facing North at the final location. For this, the fastest sequence of steps for the taxi would be as follows:

(a) Turn right (takes 1 minute, the taxi is now at $(1, 2)$ and faces East)
(b) Go ahead 1 steps (takes 2 minutes, the taxi is now at $(2, 2)$ and faces East)
(c) Turn right (takes 1 minute, the taxi is now at $(2, 2)$ and faces South)
(d) Go ahead 2 steps (takes 3 minutes, the taxi is now at $(2, 0)$ and faces South)
(e) Turn right (takes 1 minute, the taxi is now at $(2, 0)$ and faces West)
(f) Go ahead 1 steps (takes 2 minutes, the taxi is now at $(1, 0)$ and faces West)
(g) Turn right (takes 1 minute, the taxi is now at $(1, 0)$ and faces North)

Design an algorithm which takes the following inputs as parameters:

- `grid_size`: an integer $n \geqslant 0$ specifying the size of the grid.
- `a`: a number $a \geqslant 0$
- `b`: a number $b \geqslant 0$
- `c`: a number $c \geqslant 0$
- `M`: A 2-dimensional $n \times n$ array M specifying the layout of the city.

Your algorithm should return the maximum possible time taken by the taxi to go from a starting configuration to an ending configuration using the fastest possible sequence of steps, where the maximum is taken over all possible feasible configurations.

We will model this as a graph problem. If necessary, you are allowed to invoke algorithms covered in class without describing how they work, but you must clearly describe the inputs provided to the algorithms. e.g. If invoking a graph algorithm, clearly describe the graph: what are the vertices, edges (undirected/directed), edge weights (if any) for the graph? Also describe any other inputs to the algorithm, e.g. starting vertex for Dijkstra's algorithm.

You are also free to design your own algorithm from scratch.

(a) (7 points) Design an $O(n^6)$ time algorithm for this problem. Explain clearly why your graph models the problem correctly, and why your algorithm returns the correct answer.

(b) (8 points) Consider the special case of $b = 0$. (This means that we ignore the time taken by the taxi to start/stop, and hence "Move straight by k steps" is the same as "Move straight by 1 step" executed k times in sequence).

Design an $O(n^4 \log n)$ time algorithm for the problem in this special case. Explain clearly why your graph models the problem correctly, and why your algorithm returns the correct answer.