CS2040 Tutorial 10

Week 12, starting 31 Oct 2022

Q1 Mario 3D

Suppose you are playing Super Mario. Mario wants to reach his princess to save her from the enemies. Mario and the princess are in an **R*****C** grid, together with **N** enemies who each take up 1 cell in the grid. Mario can move {up, down, left, right} but never diagonally. The princess and her enemies do NOT move.

2 examples:

In both examples, the game is a 5*4 grid

Gray cells represent those occupied by enemies, $\mathbf{N_A}$ == 4 and $\mathbf{N_B}$ == 3

In grid A, the princess is doomed as Mario cannot reach her

In grid B, there are several ways for Mario to reach the Princess, the shortest having distance of 4

Grid A

(0,0) Mario	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)
(3,0)	(3,1)	(3,2)	(3,3)
(4,0)	(4,1)	(4,2) Princess	(4,3)

Grid B

(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1) Mario	(1,2)	(1,3)
(2,0)	(2,0)	(2,2)	(2,3)
(3,0)	(3,1)	(3,2)	(3,3)
(4,0)	(4,1)	(4,2) Princess	(4,3)

Given a grid with the positions of the **N**+2 game characters, how can you:

(a) detect whether Mario can save the princess

(b) find the min number of cells Mario needs to move to be in the same square as the princess

Assuming Mario can reach the princess, how do you find each of the following, independent of one another, given that each cell also has:

(c) a positive **calorie** cost that Mario will incur on moving into the cell - to find the **min total number of calories** Mario will burn upon reaching the princess

(d) a **difficulty** level (could be -ve, 0, +ve) to move into the cell - to find the **min total difficulty** Mario needs to go through to be in the same square in the princess

(e) a **difficulty** level (could be -ve, 0, +ve) to move into the cell - to find the **most difficult cell** that is along the "easiest" path from Mario to the princess, "easiest" being determined solely by the most difficult cell in that path

Find an efficient algorithm for each of **(a)** - **(e)** and state its time complexity

Q2 SSSP In a DAG

In each of **V** passes, Dijkstra's algorithm confirms the distance estimate of the reachable unconfirmed vertex with the minimum distance from source. This works because the confirmed vertices will never be updated again when there are **NO negative-weighted edges**.

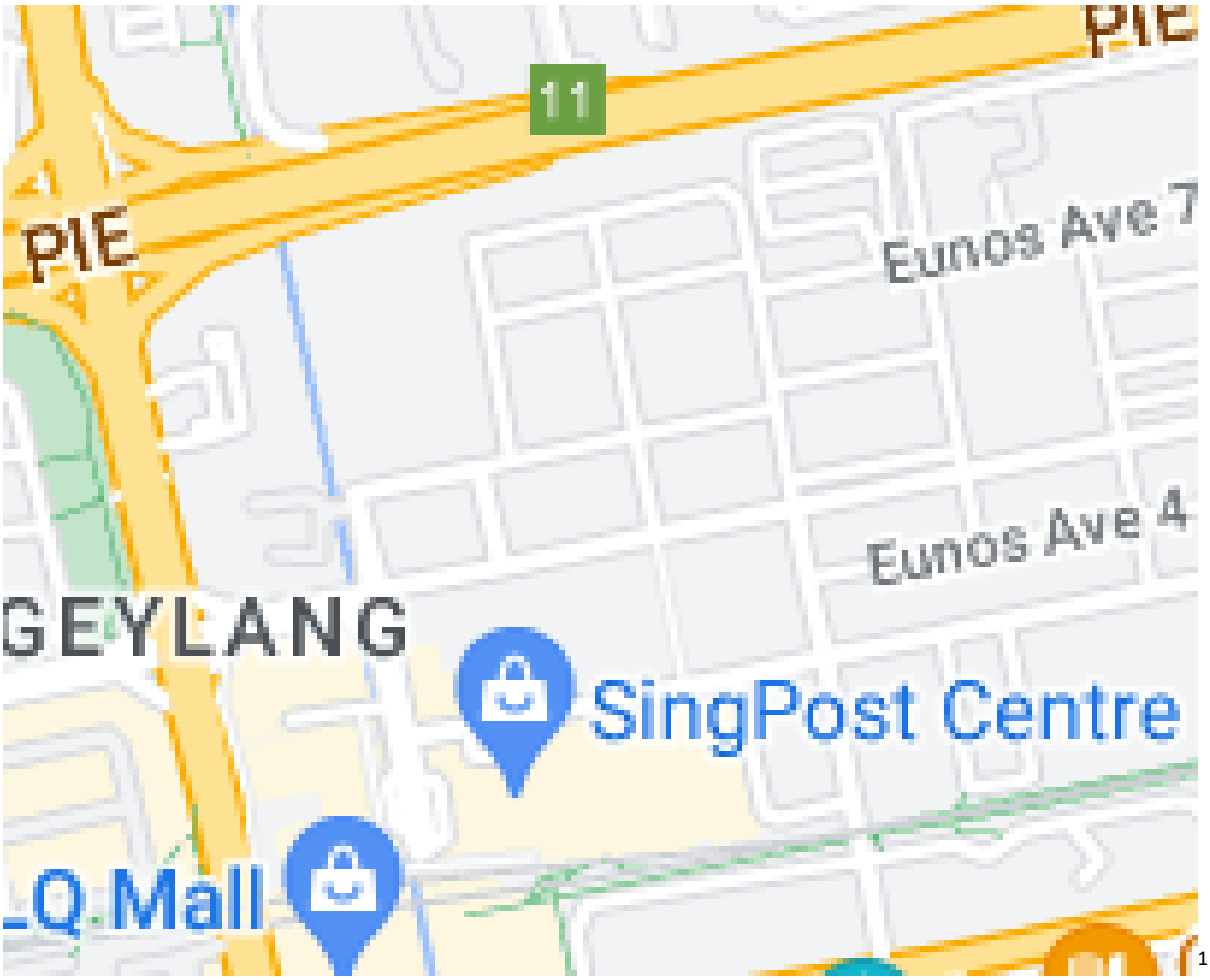
Now if we have a weighted **directed acyclic** graph, can we modify Dijkstra's algorithm to run in O(**V** + **E**) time, and at the same time allow negative-weighted edges (but no negative-cycles) to be present? The output of your algorithm should be **another** weighted **directed** graph containing the shortest-path spanning-tree.

Hints: How do you

- remove the O(log **V**) cost per vertex and per edge
- pick vertices to confirm, so that all confirmed vertices will **NEVER** be revisited

?

Question 3 (Online Discussion) – Noosepaper



[1]

You are a software developer at Lion Noosepaper & Co, and you want to find the shortest time it takes for a delivery truck to exit your *base* and reach a *road segment*. For simplicity, you may assume that every *road segment* is situated in between 2 *road junctions*.

[1] Image cut from Google Maps, 2022

A *road segment* can be on a 1-way road, or a 2-way road. As delivery takes place early in the morning, the roads are clear, therefore the estimated travel time on a *road segment*, regardless of valid direction, is (given and) fixed. For simplicity, there are only 2 kinds of *road junctions*:

Cross Junction

T Junction

Junction orientations may be rotated.

Some *road junctions* do not allow right turns, some allow u-turns (in Singapore, u-turns at junctions are not legal unless otherwise indicated =( ). Each *road junction* also has a (given) different estimated time taken to turn left, go straight ahead, turn right, and/or u-turn if legal (turn left time may be different from go straight ahead time).

Given:

- the position of the **N** *road junctions* and **M** *road segments*
- the nature of each of the N road junctions – cross / T? Can turn right? Can u-turn?
- the nature of each of the **M** *road segments* – 1-way / 2-way
- the relationships between the *road junctions* and *road segments*
- the turn left / straight ahead / turn right / u-turn times, if exists, to clear each of the **N** *road junctions*
- the travel time taken to clear each of the **M** *road segments*
- the position of a base (one of the **M** *road segments*) where your truck starts from

you are to **efficiently find** the shortest time it takes to hit a desired road segment

How do you model the graph, what algorithm should you use, and what time complexity is required?