Death Run

Flavor Text

I want to play a game.

Jigsaw
Saw (2004)

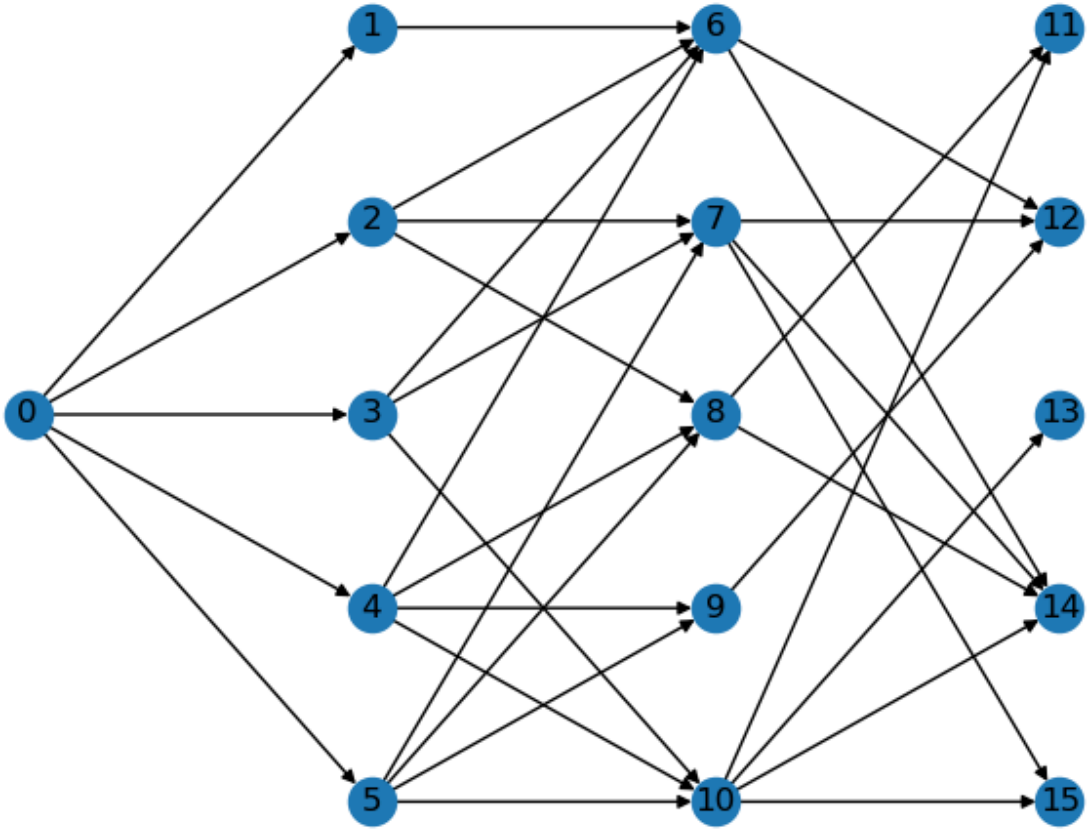
A criminal organization named “Figsaw" has taken hold of Pittsburgh, mass-kidnapping students of Carnegie Mellon University and forcing them to participate in an utterly depraved obstacle course. This obstacle course does not cause any harm to its players, instead accomplishing something worse---it wastes their time.

As students attempt to navigate this obstacle course, eager to return to their schoolwork, members of Figsaw watch from above, pushing buttons which trigger several traps along the obstacle course. Each trap increases the amount of time required to traverse a certain path in the course.

You are assigned to either be a criminal of Figsaw or a student at Carnegie Mellon University. As a criminal, your aim is to cause the most direct time wastage of students in the obstacle course. As a student, your aim is to reach any one of the obstacle course's exits as fast as possible.

Problem Statement

There are ten players per game. All players are initially presented with the same directed, acyclic, weighted graph $G = (V, E)$ representing the obstacle course. The graph is a modified layered graph, in which all vertices in the first layer are connected to a supersource vertex. Edges between layers exist with 1/2 probability, except that we guarantee that all vertices in a layer have an in-degree of one and all vertices in the non-last layer have an out-degree of one. Edge weights, which represent how long it takes to traverse through that edge, are chosen uniformly at random from the range [1, 10]. An example of such a graph with three layers and a height of five is:



where vertex 0 is the supersource. In this game, all graphs will have fifteen layers, each with a height of eight.

Two players are designated as criminals and eight players are designated as students. Students wish to move from the supersource vertex to any vertex in the final layer, in the shortest path length they can achieve. Criminals have the opportunity to increase edge weights along the graph, aiming to increase students' path length. The game is turn-based, with the first turn belonging to the criminals.

Each criminal begins with a budget of 100. In the criminals' turn, each criminal may choose any edge $(u, v) \in E$ and increase its weight by a value of their choice, x. x must be greater than or equal to 0, and less than or equal to the criminal's current budget. If a criminal does not provide a valid edge, or they do not provide a valid x, they do nothing in that turn. If x is valid, then it is added to the weight of (u, v) and deducted from that criminal's budget. A criminal's score is computed as follows. Say that in a given turn, the criminal chooses to increase the weight of edge (u, v) by x. Then after this turn, say that n students choose to use the edge (u, v). nx is then added to the criminal's score. Criminals wish to maximize their score.

All students begin on the same vertex, the supersource. In the students' turn, each student must choose a vertex that they wish to move to in this turn. Say that the student was on vertex u, and they choose to move to vertex v. Then, the edge (u, v) must exist; if not, the student will be forced to move to a random valid vertex in that turn. The game ends when all students reach a vertex in the final layer (i.e. a vertex with an out-degree of 0). A student's score is computed as follows: say that in a given turn, a student moves from vertex u to vertex v. Then, the current weight of edge (u, v) is added to that student's score. Students wish to minimize their score.

You are to implement two classes, one for play as a criminal (inherits the provided class `BaseCriminal`) and one for play as a student (inherits the provided class `BaseStudent`).

The criminal class must implement two functions, `__init__`, which is called once at the beginning of the game, and `strategy` which is called every time it is the criminals' turn.

`__init__` takes as input:

- `edge_list`: A list of tuples representing the edge list of the graph. Tuples are of the form `(u, v, w)` indicating that an edge exists from vertex `u` to vertex `v` with weight `w`.
- `begin`: The vertex which all students begin on.
- `ends`: A list of vertices that are valid exits for students.

`__init__` does not provide anything as output.

`strategy` takes as input:

- `edge_updates`: A dictionary where the key is a tuple `(u, v)` and the value is an integer `w`. Indicates that in the previous criminals' turn, the edge `(u, v)` had a weight increase of `w`. Note that this only contains information about the previous criminals' turn, and not of turns before that.
- `vertex_count`: A dictionary where the key is an integer `u` and the value is an integer `n`. Indicates that there are currently `n` students on vertex `u`.
- `budget`: An integer representing your current budget.

`strategy` must provide as output a tuple `(u, v, w)`, indicating that you wish to increase the weight of edge `(u, v)` by `w`.

The student class must implement two functions, `__init__`, which is called once at the beginning of the game, and `strategy`, which is called every time it is the students' turn.

`__init__` takes as input:

- `edge_list`: A list of tuples representing the edge list of the graph. Tuples are of the form `(u, v, w)` indicating that an edge exists from vertex `u` to vertex `v` with weight `w`.
- `begin`: The vertex which all students begin on.
- `ends`: A list of vertices that are valid exits for students.

`__init__` does not provide anything as output.

`strategy` takes as input:

- `edge_updates`: A dictionary where the key is a tuple `(u, v)` and the value is an integer `w`. Indicates that in the previous criminals' turn, the edge `(u, v)` had a weight increase of `w`. Note that this only contains information about the previous criminals' turn, and not of turns before that.
- `vertex_count`: A dictionary where the key is an integer `u` and the value is an integer `n`. Indicates that there are currently `n` students on vertex `u`.
- `current_vertex`: An integer representing the vertex that you are currently on.

`strategy` must provide as output an integer representing the vertex that you wish to move to.

Unlike other problems, you will have access to a non-standard library: NetworkX.