Problem:

Given a rooted DAG and several updates, find the total time to execute all the updates. Each update is of the form <node, number of levels, entry time, processing time>.

- node is the node in the graph on which some processing is going to be performed.
- the number of levels from the node is also going to be involved in the processing. Default value 1 means only the node is involved, value 2 means the node and its neighbors are involved, value 3 means the node, its neighbors and their neighbors are involved, and so on.
- entry time is the time after which this processing can be started (it is simply an integer).
- processing time is the amount of time required for the processing after all the required nodes are available (integer).

A condition is that no node can be processed by two updates at the same time. Thus, if two updates are <n1, 1, 5, 10> and <n1, 1, 9, 2>, the two updates may conflict.

Your task is:

- create random DAGs (or use existing DAGs if available online). Make sure it is well-connected. Number of edges could be four times the number of nodes.
- create random updates
- think of, implement, and optimize an algorithm to solve the problem.
- run it on graphs with a million nodes and a million updates.

You can send me your queries and updates whenever you make progress or get stuck. Finally, I will need a link to the following:

- code + instructions on running the code
- a report mentioning the problem statement, your approach, and a summary of results
- a googlesheet containing the following information: #nodes, #edges, #updates, total time, time taken by your program to find the total time.

In the experiments,

- keep graph + updates fixed and vary k from 1 to 5 where k is the number of levels; then
- keep graph + k fixed and increase updates from 1 million to 5 million
- keep updates + k fixed and increase the graph size from 1 million to 5 million nodes.