CS130A Programming Assignment 2
Due 2017-03-15 by 11:59PM

Union Find, and Your own Composite Data Structure

In programming assignment 1, you made a discrete event simulation. We are going to revisit the simulation with a different setup. You are going to simulate more network shenanigans, this time with multiple attackers and sysadmins. The network no longer has a trivial topology, and the network routing tables are built upon a minimum spanning tree generated using kruskal's algorithm. Compromised machines will be added to a sysadmin fix queue.
For the sysadmin fix queue, you will implement a composite data structure that enables constant time enqueue, constant time dequeue, and constant time membership check (re-broken machines don't get a re-entry).
You can use standard library data structures as the basis for your implementation of union-find, kruskal's algorithm, the event priority queue (though it must be heap based), and the efficient queue used by the sysadmins.

Sample Input

template
./program_name number_of_attackers number_of_sysadmins number_of_nodes random_seed
example
./program2 20 20 1000 1234

The Network

You are going to construct a graph using the following algorithm:

1. Seed the random number generator with the random_seed
2. For i = 0 to number_of_nodes-1
    a. For j = 0 to i-1
        i. cost(i,j) = uniform random integer (-120, 100)
3. If all edges for a node are non-positive, select the greatest one and reset it to a uniform random integer (1,100)
4. Cull all non-positive edges

This results in a somewhat sparse graph, though still $O(n^2)$ on average. All remaining edges have positive costs. The final graph can be stored however is convenient for your code.

Stopping Condition

The stopping conditions for this simulation is that 2,000 attacks have occurred.

Events

There are a minimum number of event types that must be implemented by the simulation. Remember, events are objects that are stored in the event queue. All events have a scheduled time > simulated_now.

attack(time, source, target)

Schedule an attack event for time from attacker to random target node. Each attacker will schedule their next attack at a uniform random integer between (100, 1000) in the future. Compromised machines and all adjacent edges are removed from the network and added to the (global) sysadmin fix queue.
As soon as the attack is completed, the network will check if it was partitioned. If it was, a rebuild_spanning_tree will be schedule 20 in the future (this may allow additional attacks to take place in the interim). If a rebuild is already scheduled, do not schedule a new rebuild.

fix(time)

Schedule a node to be fixed. It can be compromised again. Each sysadmin starts fixing computers every random (1000, 2000) in the future after the first node is compromised. Rebuild_spanning_tree will be scheduled 20 in the future. If a rebuild is already scheduled, do not schedule a new rebuild.

rebuild_spanning_tree(time)

The network is missing zero or more nodes (fixes could have happened as well since this was scheduled), the network will restore its spanning tree property on surviving nodes using kruskal's algorithm, but using any edges already in the spanning tree. Print out a statement indicating the current spanning tree cost and the list of missing nodes. Also compute the actual MST for the current graph (exclude all compromised nodes and their edges, but do not pick up the current graph as the starting point). Print out the optimal MST value at this point.
NOTE: it is possible that no spanning tree is possible.

Agent Types

These are agents that respond or produce events (or both)

Attacker

Each attacker randomly compromises a computer with a random iteration interval.

Sysadmin

Sysadmins fix computers.

Your Program Outputs

Every time an event is processed, print to standard out the event type, time, and any fields using constructor syntax (i.e. “Attack(1500, 1, 2)” is fine).

Build

You must be able to build and run your application with make and make run (provide some default values for make run). Your program must also be able to be run from the command line as above.

Purpose

1. You should have a good implementation of Kruskal’s algorithm to generate MSTs on (potentially) incomplete graphs.
2. You should have an implementation of Kruskal’s that starts with a forest as a starting point that might not be along the optimal path. Note that Kruskal’s algorithm makes this somewhat convenient.
3. You should make an effort to make your implementation efficient (the queue maintained by sysadmins has an explicit requirement).
4. You should have a greater understanding of how to design and implement a discrete event simulation.