CS2040 Lab 11
SSSP

One Day Assignment 9 – Lost Map

- Shortest path from u to v is given in the adjacency matrix
- Also know that the original graph is a tree
- Pick the smallest edge (u, v) where u != v, and which does not form a cycle. Edge must be part of the original graph
  - Proof on next slide
- Basically an MST

One Day Assignment 9 – Lost Map

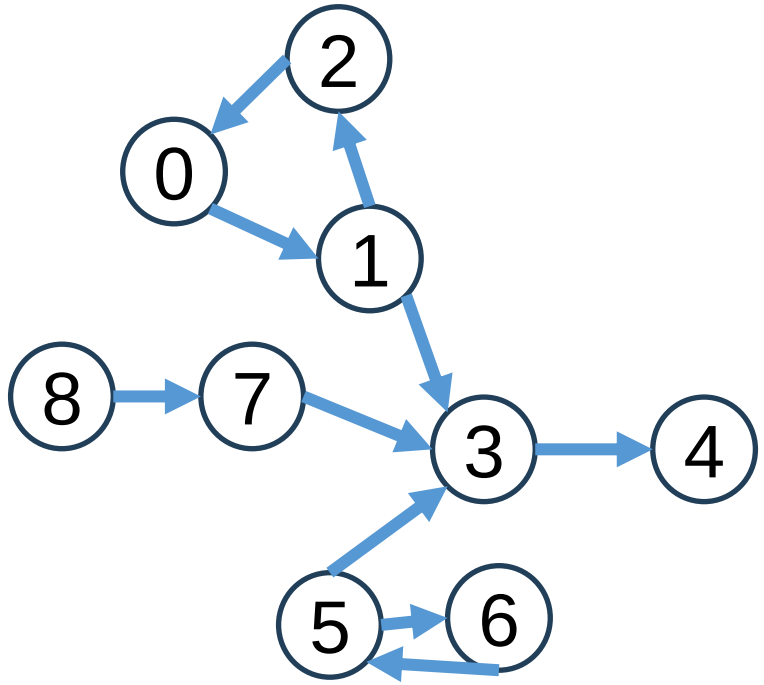
- In the proof below, (u ~> v) (curly arrow) represents a path (ie. may be more than one edge), while (u -> v) represents a direct edge
- Proof by contradiction: Suppose a different path from u ~> k ~> v for some vertex k exists, such that cost(u ~> k ~> v) <= cost(u -> v), where u -> v is the edge selected by the MST algorithm
- Cost of paths (u ~> k) and (k ~> v) must both be less than cost(u -> v), since no edge between two distinct vertices has cost 0 (ie. neither (u ~> k) nor (k ~> v) are cost 0)
- But u -> v is the smallest edge, therefore a contradiction occurs
- Therefore u -> v must be part of the original graph

Take Home Assignment 4a – Millionaire Madness

- Finding the minimum ladder length required to reach the lower right corner: standard minimax problem
- Can be solved by finding MST
- However, edges in this graph are bidirectional (different edge weights in different directions)
  - Kruskal's may not work as a result
- Perform Prim's from the starting point to get a directed MST, and perform BFS/DFS to find the largest edge along the path
  - Alternatively, return the weight of the largest edge found so far when the ending point is added to the MST when running Prim's

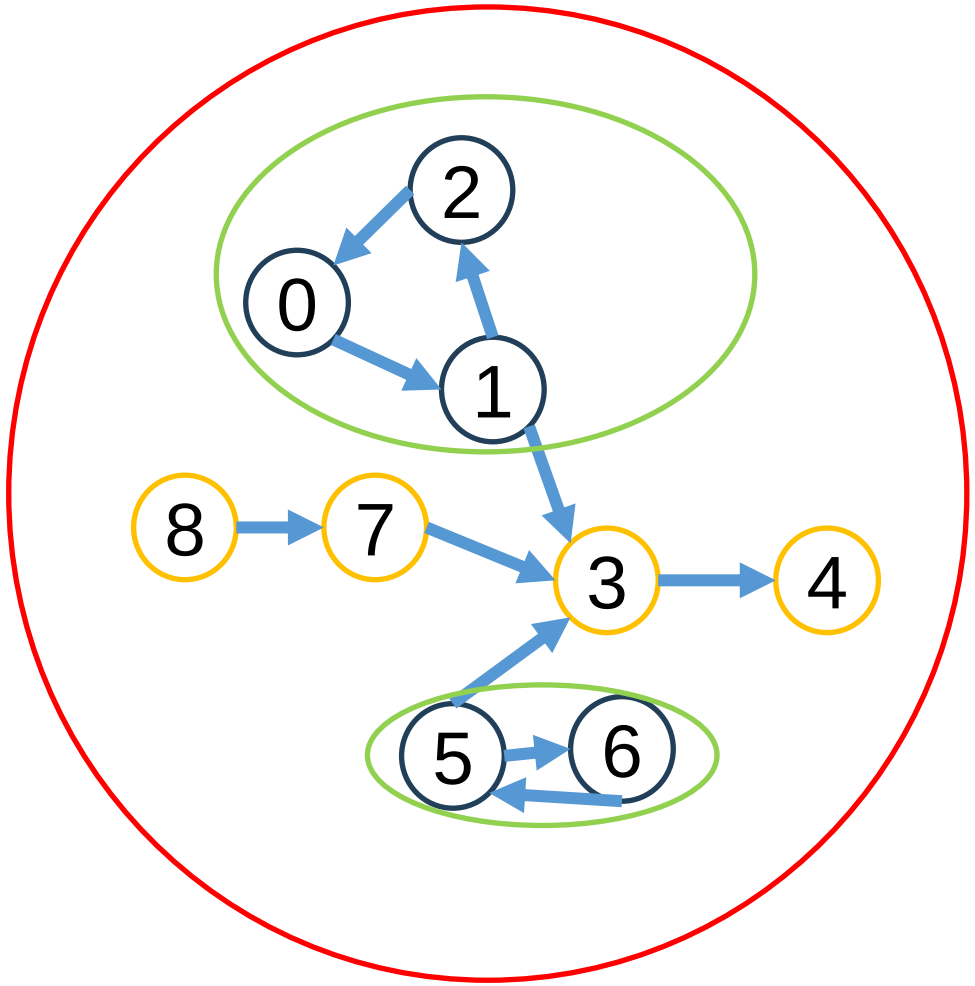
Take Home Assignment 4b – Bots

- Glossary
  - Network -> Graph
  - Bot -> Vertex
  - Signal(u, v) -> HasDirectedEdge(u, v)
  - Message(u, v) -> IsReachable(u, v)
  - Botnet -> SCC of size > 1
  - Solobot -> SCC of size == 1



Take Home Assignment 4b – Bots

- Glossary
  - Network -> Graph
  - Bot -> Vertex
  - Signal(u, v) -> HasDirectedEdge(u, v)
  - Message(u, v) -> IsReachable(u, v)
  - Botnet -> SCC of size > 1
  - Solobot -> SCC of size == 1
- SCCs **CAN** be of size 1
  - Common misconception that SCC must have at least 2 nodes

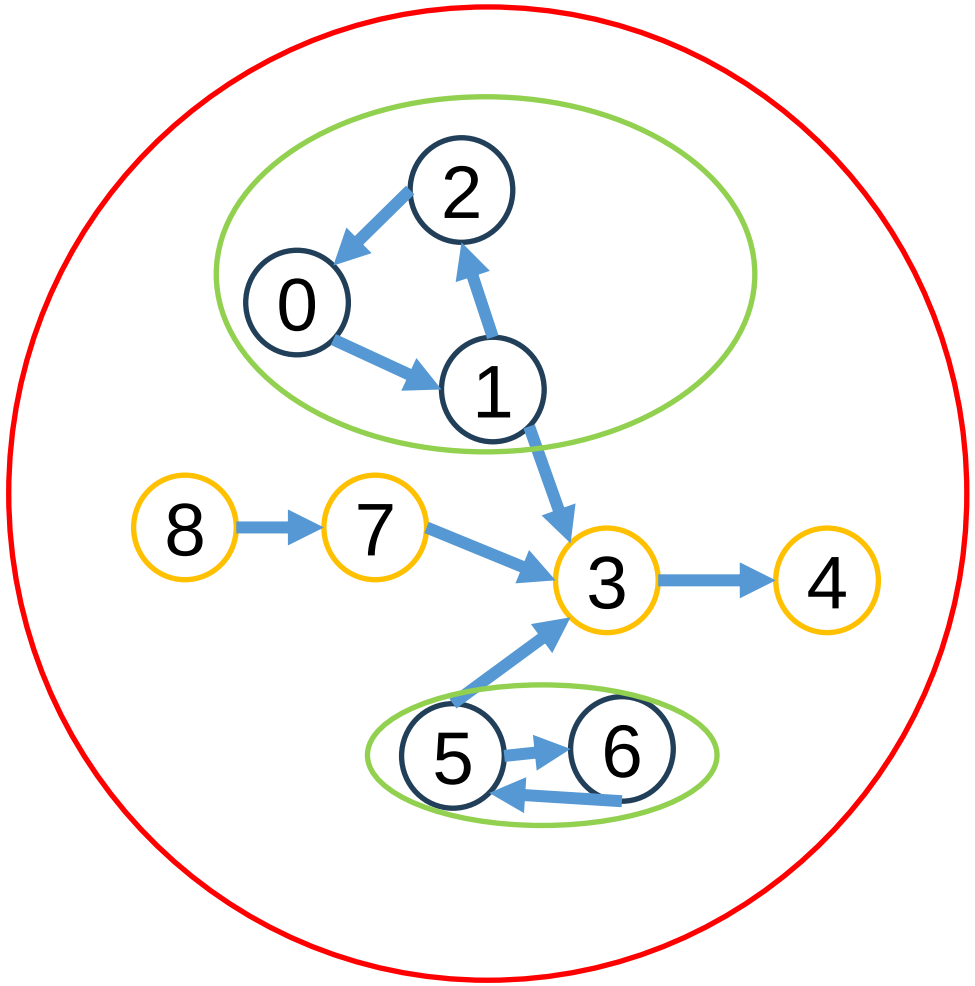


Take Home Assignment 4b – Bots

- How to minimize number of bots you have to issue program to?

- No point issuing program to:
  - 7, 3, 4
  - More than one bot in {5, 6}
  - More than one bot in {0, 1, 2}

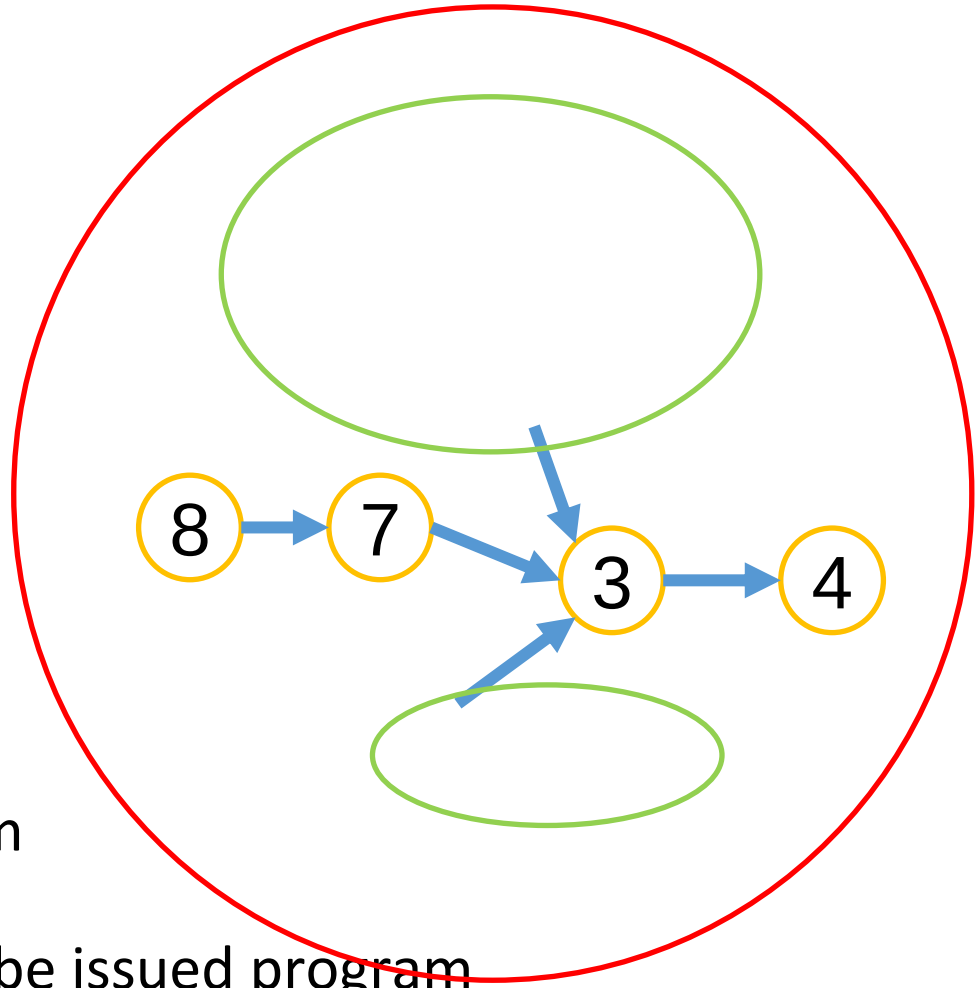
- Only issue program to:
  - 1 bot in {5, 6}
  - 1 bot in {0, 1, 2}
  - solobot 8



Take Home Assignment 4b – Bots

- Consider SCC kernel graph
  - Treat one SCC as one super-vertex
  - No cycles, Kernel graph is a DAG (original graph can be cyclic)

- Find SCCs that are sources in the SCC kernel graph
  - Source ←→ in-degree 0
  - Sink ←→ out-degree 0 e.g. solobot 4
  - Source solobot has to be issued program
  - Each source SCC (in kernel graph) needs exactly 1 bot (doesn't matter which) to be issued program



Take Home Assignment 4b – Bots

1. Construct the directed, unweighted graph
2. Find the SCCs using Kosaraju's algorithm in **O(V + E)** time
   - Also label the SCCs, and get the size of each SCC
3. Check which SCCs are "source SCC"
4. Among the "source SCC", count the number of singleton and non-singleton SCCs

Take Home Assignment 4b – Bots

- Kosaraju's algorithm
  - Why do we need to transpose graph **G** → **G^T**?
  - Runs of 2 types of DFS for different purposes
  - First type on **G^T** to find source SCCs in **G^T**, i.e. sink SCCs in **G**
    - Run **post-order** DFS – "visit" neighbours first, then myself
    - Accumulate **post-order visited** nodes, then reverse
  - Second type to label SCCs in **G**
    - Run the usual **pre-order** DFS – "visit" myself first, then neighbours
- Common mistake:
  - "Run first type of DFS on G but don't reverse the post-order"
  - "Run second type of DFS on G following that post-order visitation sequence"
  - You **may not** be following a sequence from sink SCC to source SCC

Lab 11 – SSSP

- By this point, several different SSSP algorithms have been covered, each suited for a different kind of graph
- As such, when attempting any problem involving SSSP, it might be best to consider the graph types first
  - 1. Is the graph a tree? (use DFS/BFS)
  - 2. Is the graph unweighted? (use BFS)
  - 3. Is the graph a DAG? (use one-pass Bellman Ford)
  - 4. Does the graph have only nonnegative edges? (use Original/Modified Dijkstra's)
  - 5. Does the graph have no negative cycles? (use Modified Dijkstra's)
  - 6. If all else fails, use standard Bellman Ford

Lab 11 – SSSP

Disclaimer: you can use Floyd Warshall to solve SSSP problems too, though Floyd Warshall will typically run slower as it solves a different problem (APSP)

One Day Assignment 10 – Human Cannonball Run

- Given a starting point, an end point, and a number of cannon (note: plural of cannon is cannon), determine the shortest time required to reach the end point from the starting point
- Running speed is 5 m/s
- A cannon launches you a distance of exactly 50m (this cannot be changed), and takes 2 seconds to do so

One Day Assignment 10 – Human Cannonball Run

- The following may be useful in solving the problem
- Pythagorean Theorem: Given a right-angled triangle, $a^2 + b^2 = c^2$
  - c can also be calculated using Java's Math.hypot(a, b) method