Breadth-First Search (BFS)

Lecture Question

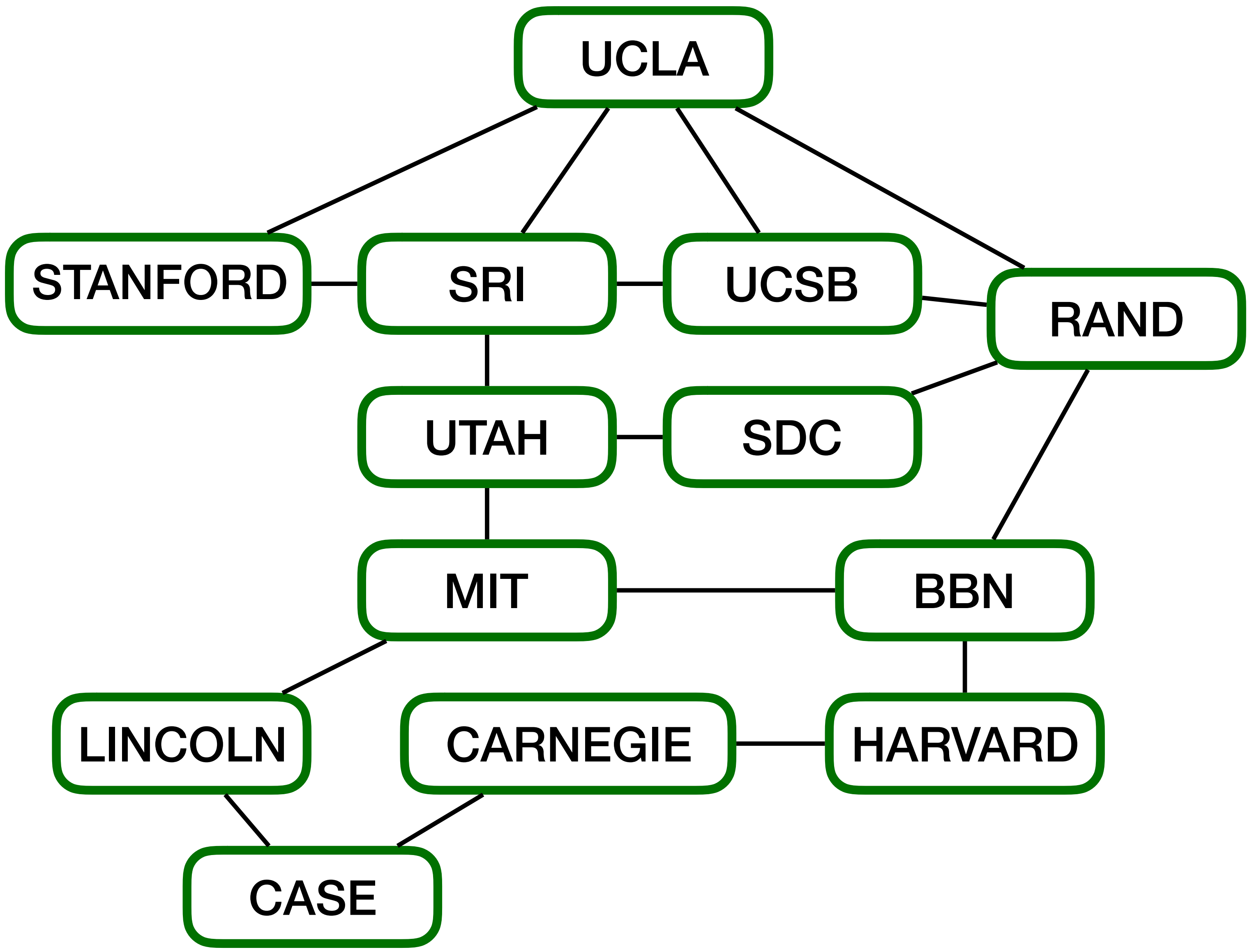
Task: Determine if two nodes connected

- In the Graph class
    - Write a method named areConnected that takes two node indices (Ints) and determines if the two nodes are connected in the graph
    - Return true if they are connected, false if they are not

* This question will be open until midnight

Connected Component

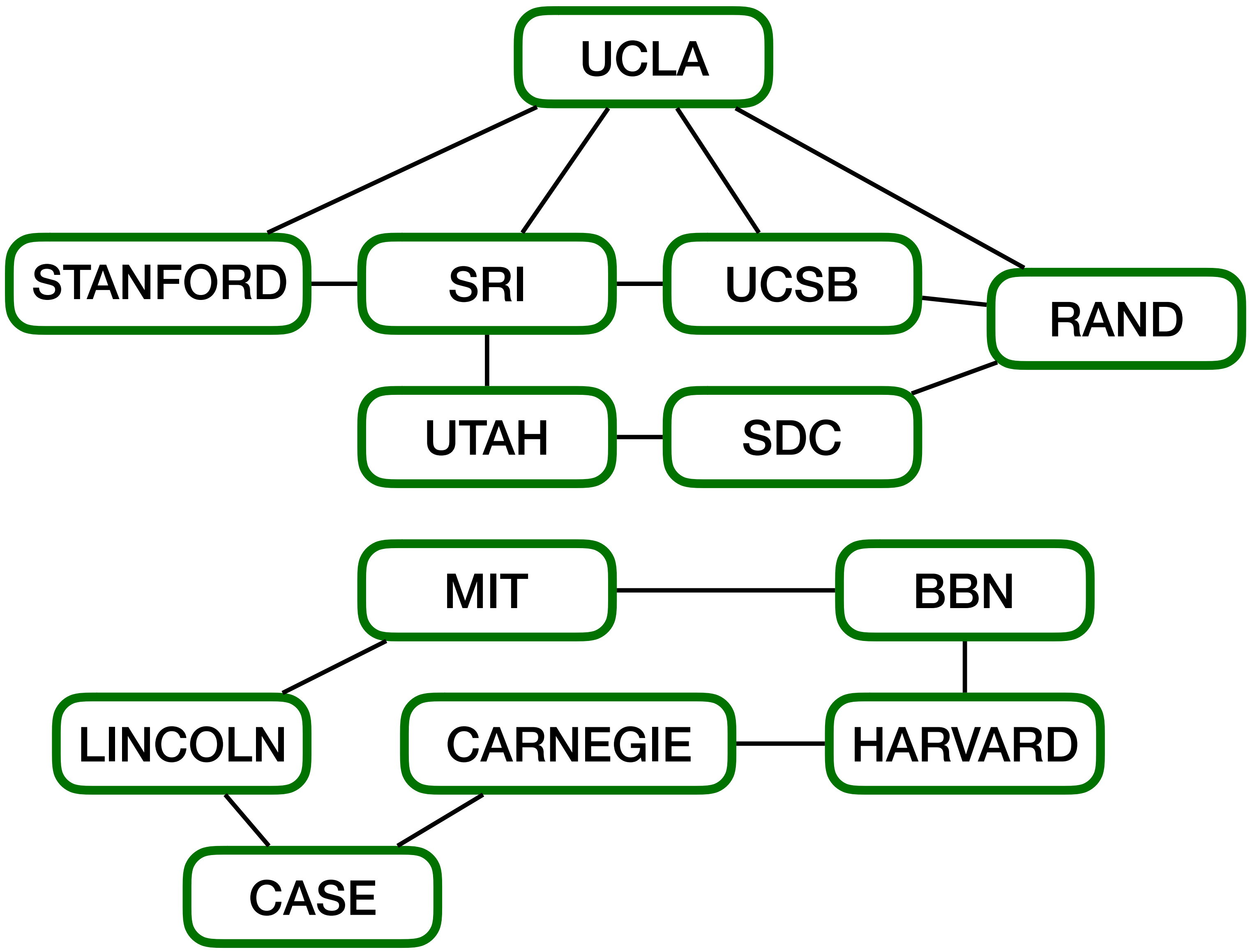
- This graph is connected
  - There exists a path between any 2 nodes in the graph



UCLA	STANFORD, SRI, UCSB, RAND
STANFORD	UCLA, SRI
SRI	UCLA, UCSB, UTAH, STANFORD
UCSB	UCLA, SRI, RAND
RAND	UCLA, UCSB, SDC, BBN
UTAH	SRI, SDC, MIT
SDC	UTAH, RAND
MIT	UTAH, BBN, LINCOLN
BBN	RAND, MIT, HAVARD
LINCOLN	MIT, CASE
CARNEGIE	CASE, HARVARD
HARVARD	BBN, CARNEGIE
CASE	LINCOLN, CARNEGIE

Connected Component

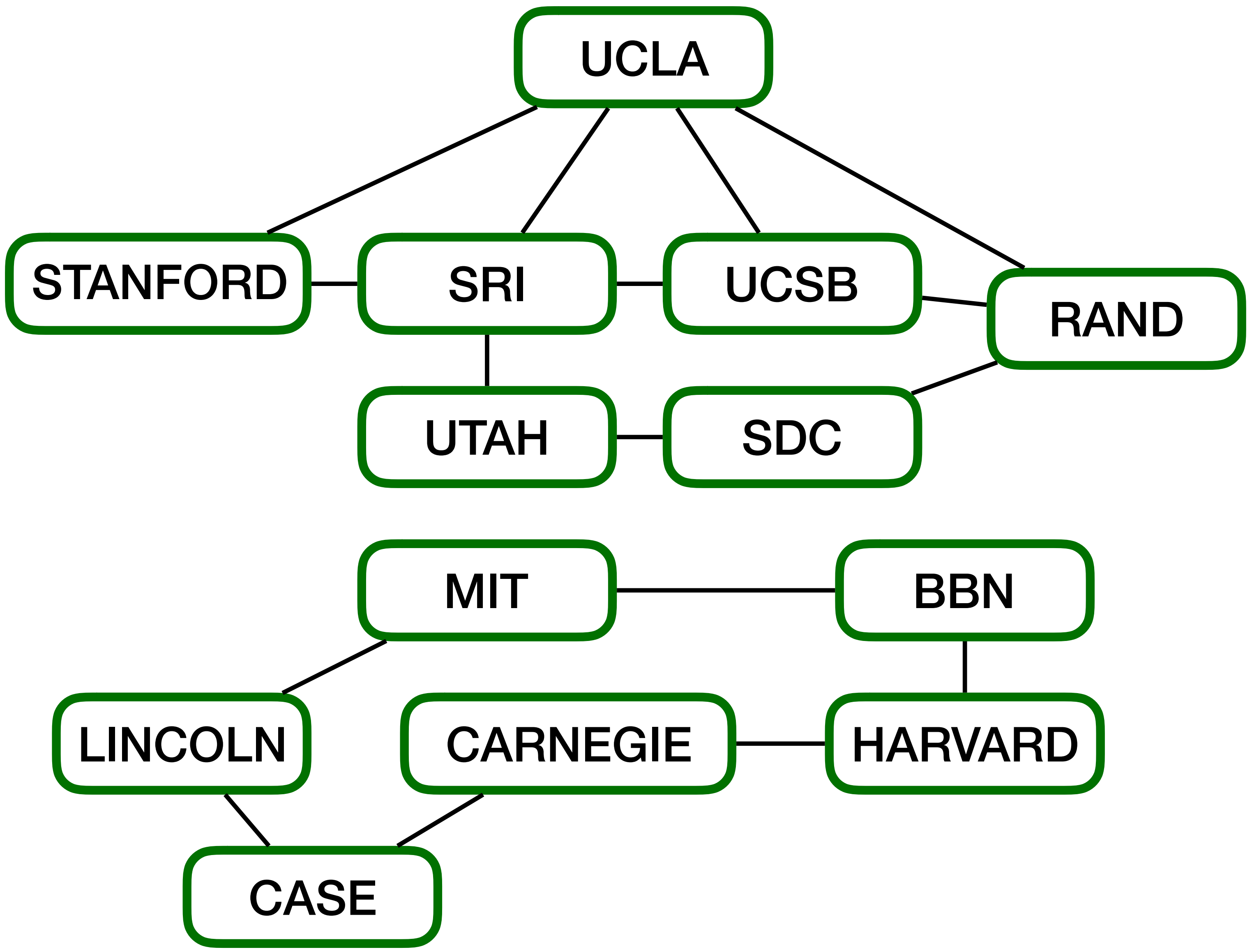
- What if a few connections a broken?
    - How can we tell if two nodes are connected?



UCLA	STANFORD, SRI, UCSB, RAND
STANFORD	UCLA, SRI
SRI	UCLA, UCSB, UTAH, STANFORD
UCSB	UCLA, SRI, RAND
RAND	UCLA, UCSB, SDC
UTAH	SRI, SDC
SDC	UTAH, RAND
MIT	BBN, LINCOLN
BBN	MIT, HAVARD
LINCOLN	MIT, CASE
CARNEGIE	CASE, HARVARD
HARVARD	BBN, CARNEGIE
CASE	LINCOLN, CARNEGIE

Connected Component

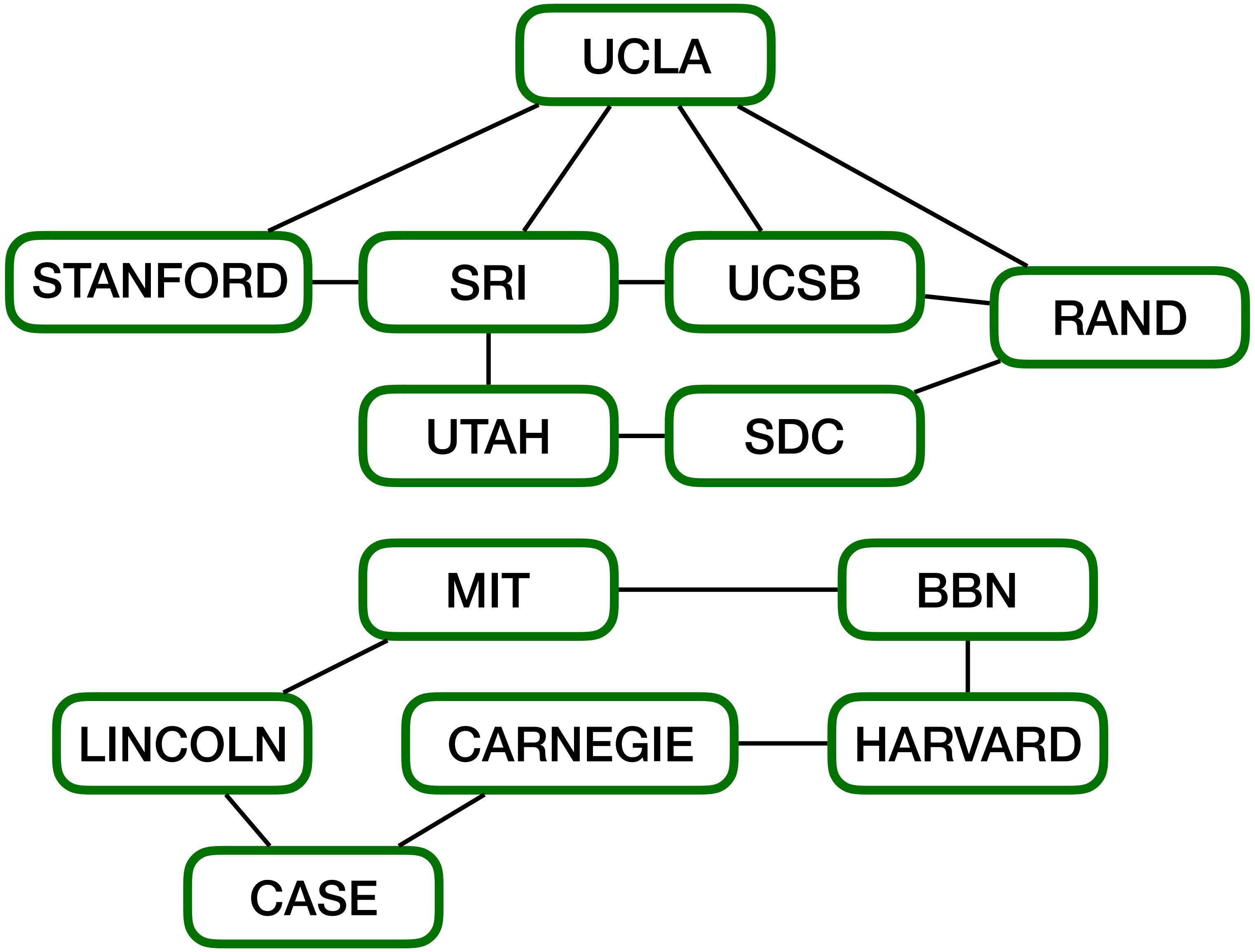
- What if a few connections a broken?
    - How can we tell if two nodes are connected?

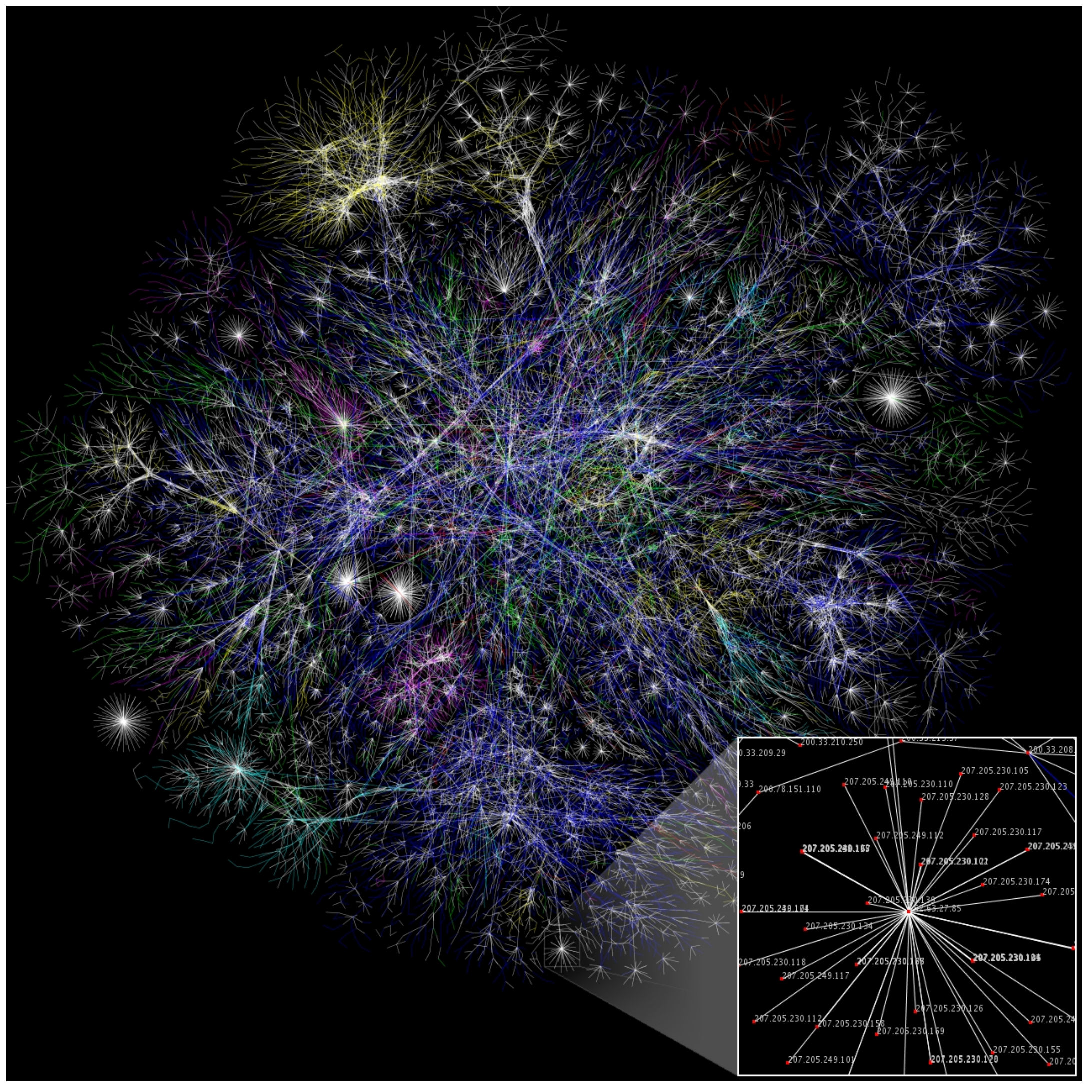


UCLA	STANFORD, SRI, UCSB, RAND
STANFORD	UCLA, SRI
SRI	UCLA, UCSB, UTAH, STANFORD
UCSB	UCLA, SRI, RAND
RAND	UCLA, UCSB, SDC
UTAH	SRI, SDC
SDC	UTAH, RAND
MIT	BBN, LINCOLN
BBN	MIT, HAVARD
LINCOLN	MIT, CASE
CARNEGIE	CASE, HARVARD
HARVARD	BBN, CARNEGIE
CASE	LINCOLN, CARNEGIE

Connected Component

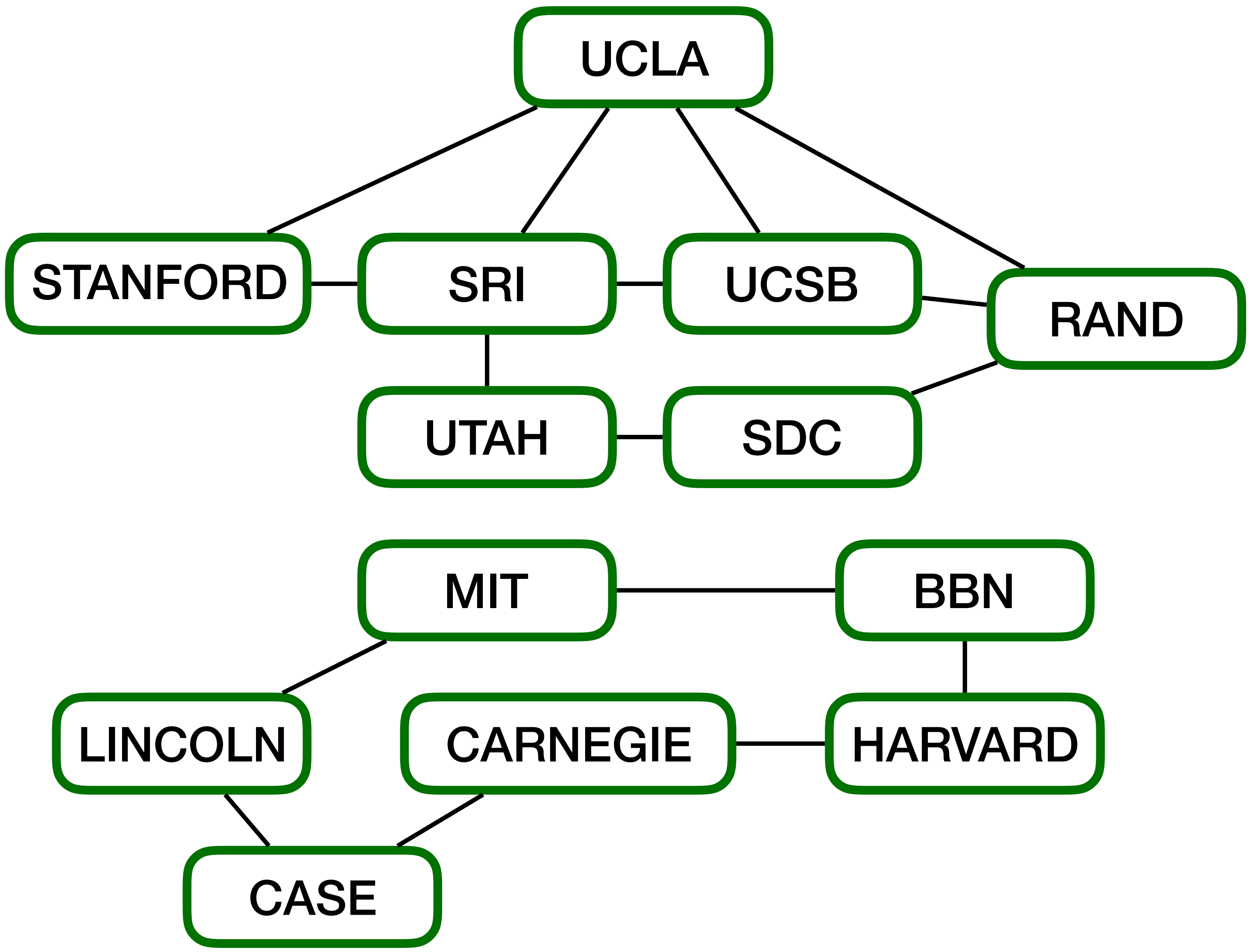
- We could verify manually for this graph
- But the Internet has gotten a *little* bigger over time
- Need to code an algorithm to solve this for us





BFS

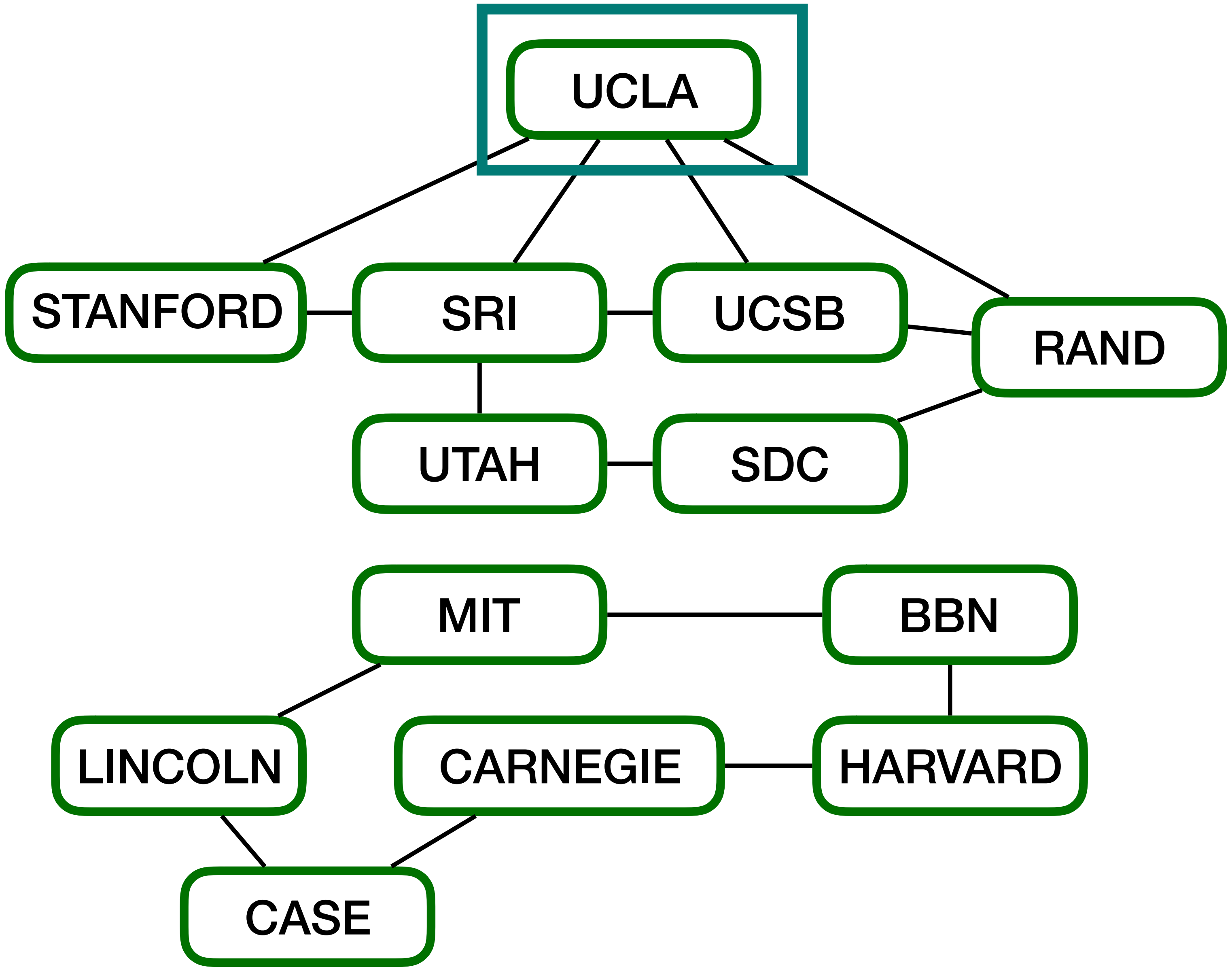
- The Algorithm: Breath-First Search (BFS)
  - Choose a starting node
  - Continuously explore connected nodes



UCLA	STANFORD, SRI, UCSB, RAND
STANFORD	UCLA, SRI
SRI	UCLA, UCSB, UTAH, STANFORD
UCSB	UCLA, SRI, RAND
RAND	UCLA, UCSB, SDC
UTAH	SRI, SDC
SDC	UTAH, RAND
MIT	BBN, LINCOLN
BBN	MIT, HAVARD
LINCOLN	MIT, CASE
CARNEGIE	CASE, HARVARD
HARVARD	BBN, CARNEGIE
CASE	LINCOLN, CARNEGIE

BFS

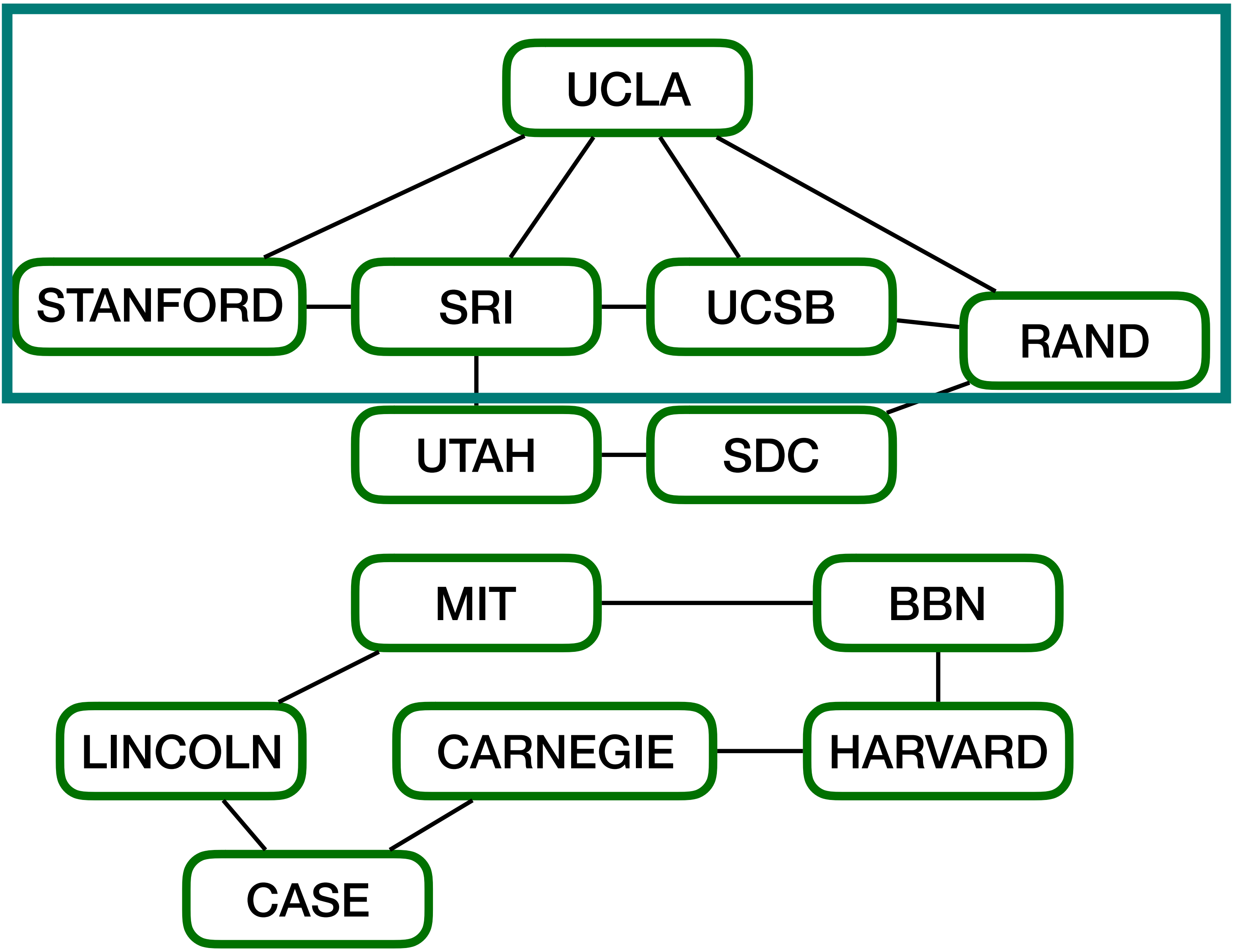
- Choose a starting node



UCLA	STANFORD, SRI, UCSB, RAND
STANFORD	UCLA, SRI
SRI	UCLA, UCSB, UTAH, STANFORD
UCSB	UCLA, SRI, RAND
RAND	UCLA, UCSB, SDC
UTAH	SRI, SDC
SDC	UTAH, RAND
MIT	BBN, LINCOLN
BBN	MIT, HAVARD
LINCOLN	MIT, CASE
CARNEGIE	CASE, HARVARD
HARVARD	BBN, CARNEGIE
CASE	LINCOLN, CARNEGIE

BFS

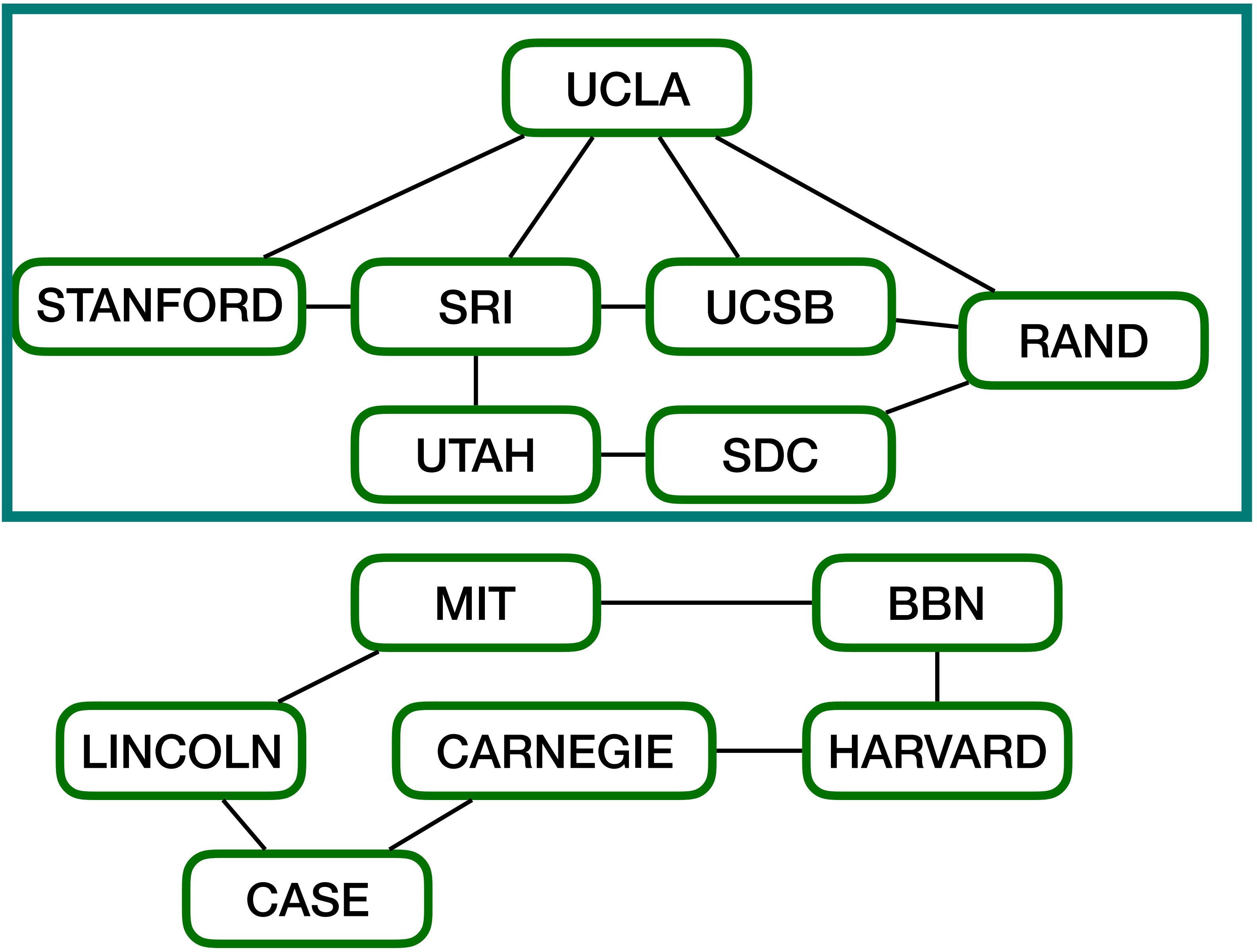
- Explore all nodes connected to the striating node



UCLA	STANFORD, SRI, UCSB, RAND
STANFORD	UCLA, SRI
SRI	UCLA, UCSB, UTAH, STANFORD
UCSB	UCLA, SRI, RAND
RAND	UCLA, UCSB, SDC
UTAH	SRI, SDC
SDC	UTAH, RAND
MIT	BBN, LINCOLN
BBN	MIT, HAVARD
LINCOLN	MIT, CASE
CARNEGIE	CASE, HARVARD
HARVARD	BBN, CARNEGIE
CASE	LINCOLN, CARNEGIE

BFS

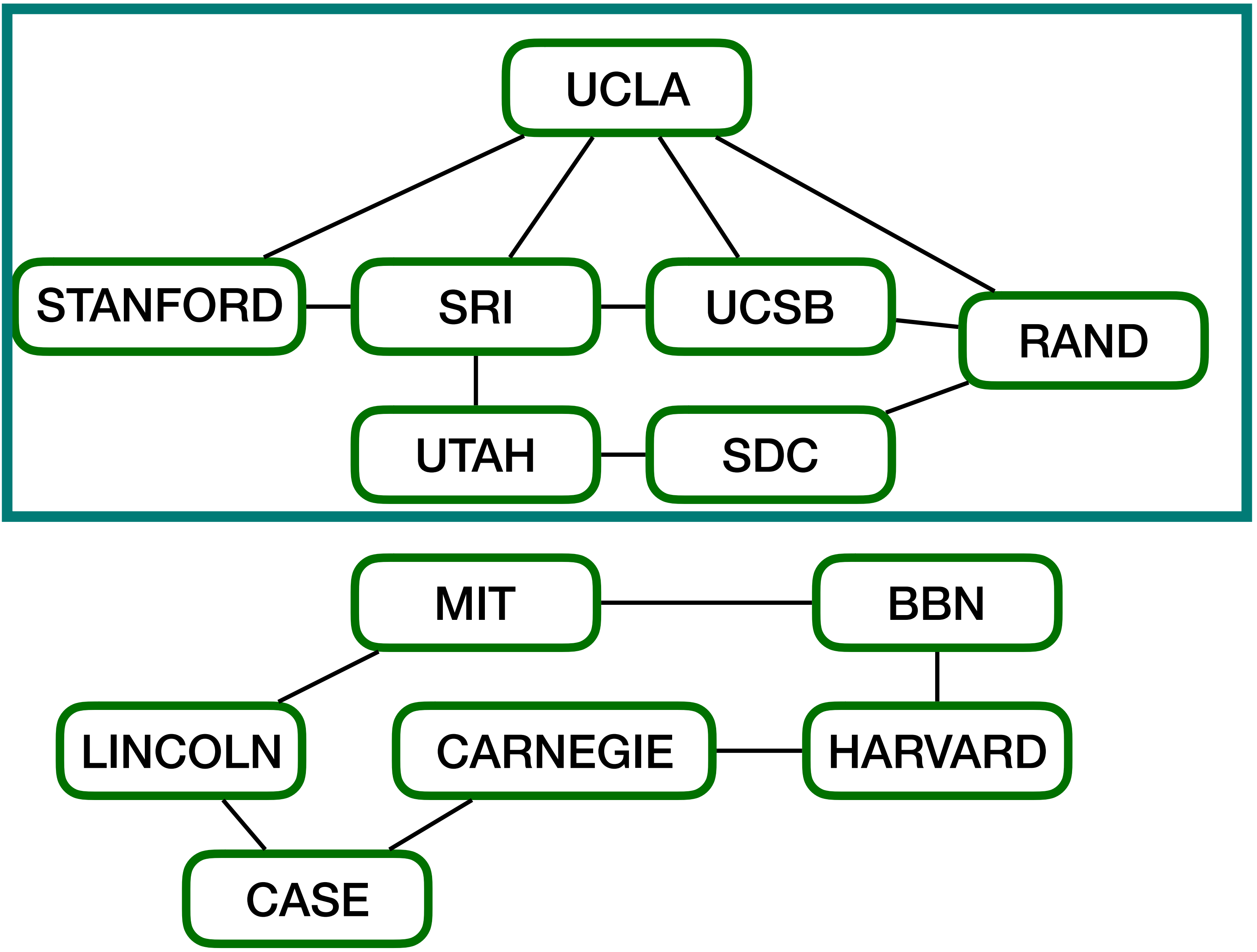
- Repeatedly explore nodes that were visited in the last round



UCLA	STANFORD, SRI, UCSB, RAND
STANFORD	UCLA, SRI
SRI	UCLA, UCSB, UTAH, STANFORD
UCSB	UCLA, SRI, RAND
RAND	UCLA, UCSB, SDC
UTAH	SRI, SDC
SDC	UTAH, RAND
MIT	BBN, LINCOLN
BBN	MIT, HAVARD
LINCOLN	MIT, CASE
CARNEGIE	CASE, HARVARD
HARVARD	BBN, CARNEGIE
CASE	LINCOLN, CARNEGIE

BFS

- Repeat until no new nodes are added
- Never visit a node twice



UCLA	STANFORD, SRI, UCSB, RAND
STANFORD	UCLA, SRI
SRI	UCLA, UCSB, UTAH, STANFORD
UCSB	UCLA, SRI, RAND
RAND	UCLA, UCSB, SDC
UTAH	SRI, SDC
SDC	UTAH, RAND
MIT	BBN, LINCOLN
BBN	MIT, HAVARD
LINCOLN	MIT, CASE
CARNEGIE	CASE, HARVARD
HARVARD	BBN, CARNEGIE
CASE	LINCOLN, CARNEGIE

BFS

- Use a queue to track the order of nodes to visit
- Start with starting node in the queue
- When visiting a node, add all unexplored neighbors to the queue
- Visit neighbors of the node at the front of the queue until the queue is empty

```scala
def bfs[A](graph: Graph[A], startID: Int): Unit = {

  var explored: Set[Int] = Set(startID)

  val toExplore: Queue[Int] = new Queue()
  toExplore.enqueue(startID)

  while (!toExplore.empty()) {
    val nodeToExplore = toExplore.dequeue()
    for (node <- graph.adjacencyList(nodeToExplore)) {
      if(!explored.contains(node)){
        println("exploring: " + graph.nodes(node))
        toExplore.enqueue(node)
        explored = explored + node
      }
    }
  }
}
```

UCLA	STANFORD, SRI, UCSB, RAND
STANFORD	UCLA, SRI
SRI	UCLA, UCSB, UTAH, STANFORD
UCSB	UCLA, SRI, RAND
RAND	UCLA, UCSB, SDC
UTAH	SRI, SDC
SDC	UTAH, RAND
MIT	BBN, LINCOLN
BBN	MIT, HAVARD
LINCOLN	MIT, CASE
CARNEGIE	CASE, HARVARD
HARVARD	BBN, CARNEGIE
CASE	LINCOLN, CARNEGIE

Connectivity

- If you start at one nodeA and explore nodeB during the algorithm
  - nodeA and nodeB are connected

Lecture Question

Task: Determine if two nodes connected

- In the Graph class
    - Write a method named areConnected that takes two node indices (Ints) and determines if the two nodes are connected in the graph
    - Return true if they are connected, false if they are not

* This question will be open until midnight