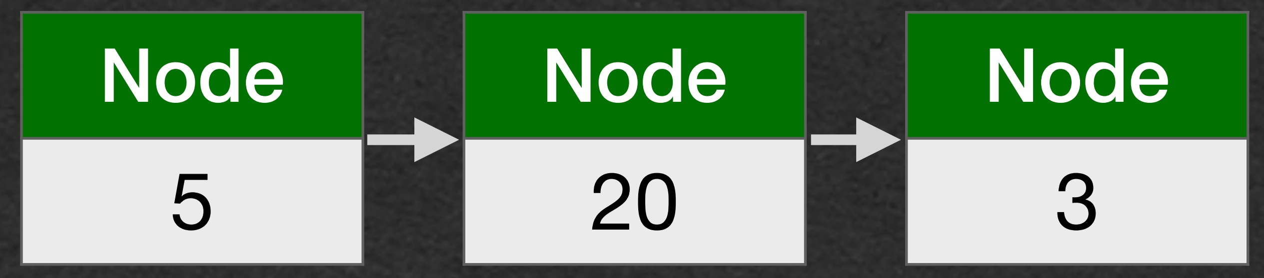
Graphs

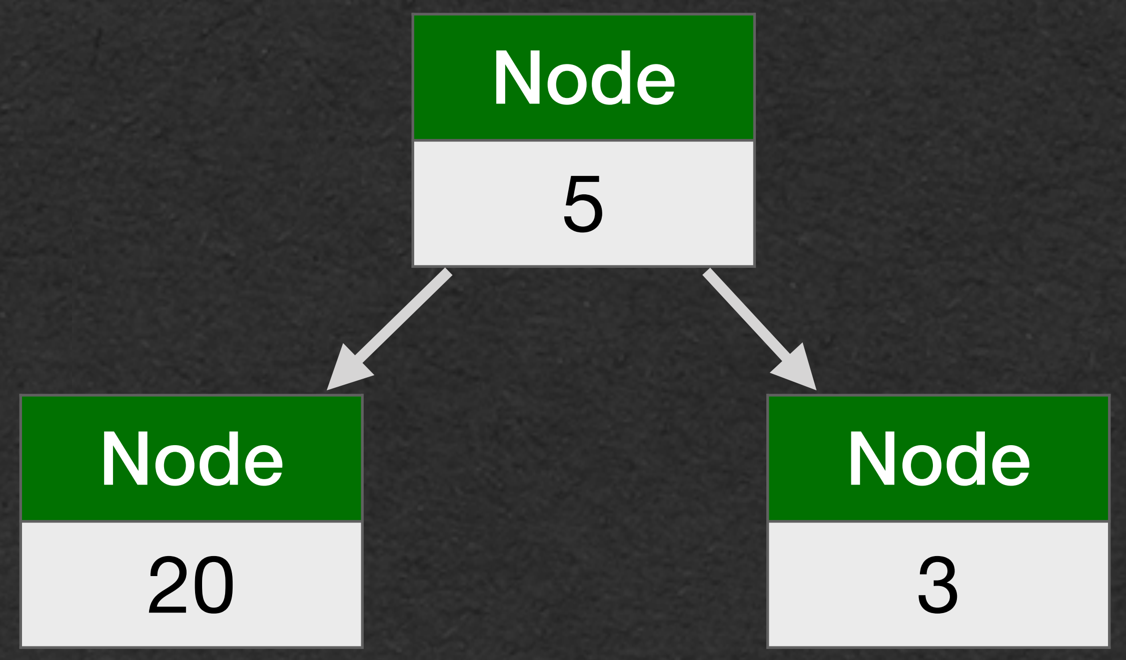
Data Structures: Review

- Sequential Data Structures
  - Elements stored in a specific order
  - Ex: Array, List
- Key-Value Store
  - Stores pairs of elements with no particular order
  - Each key is associated with one value
  - Ex. Map, Dictionary, Object
- Tree
  - Non-linear structure
  - Each element can be associated with multiple other elements

Index	0	1	2
Value	5	20	3

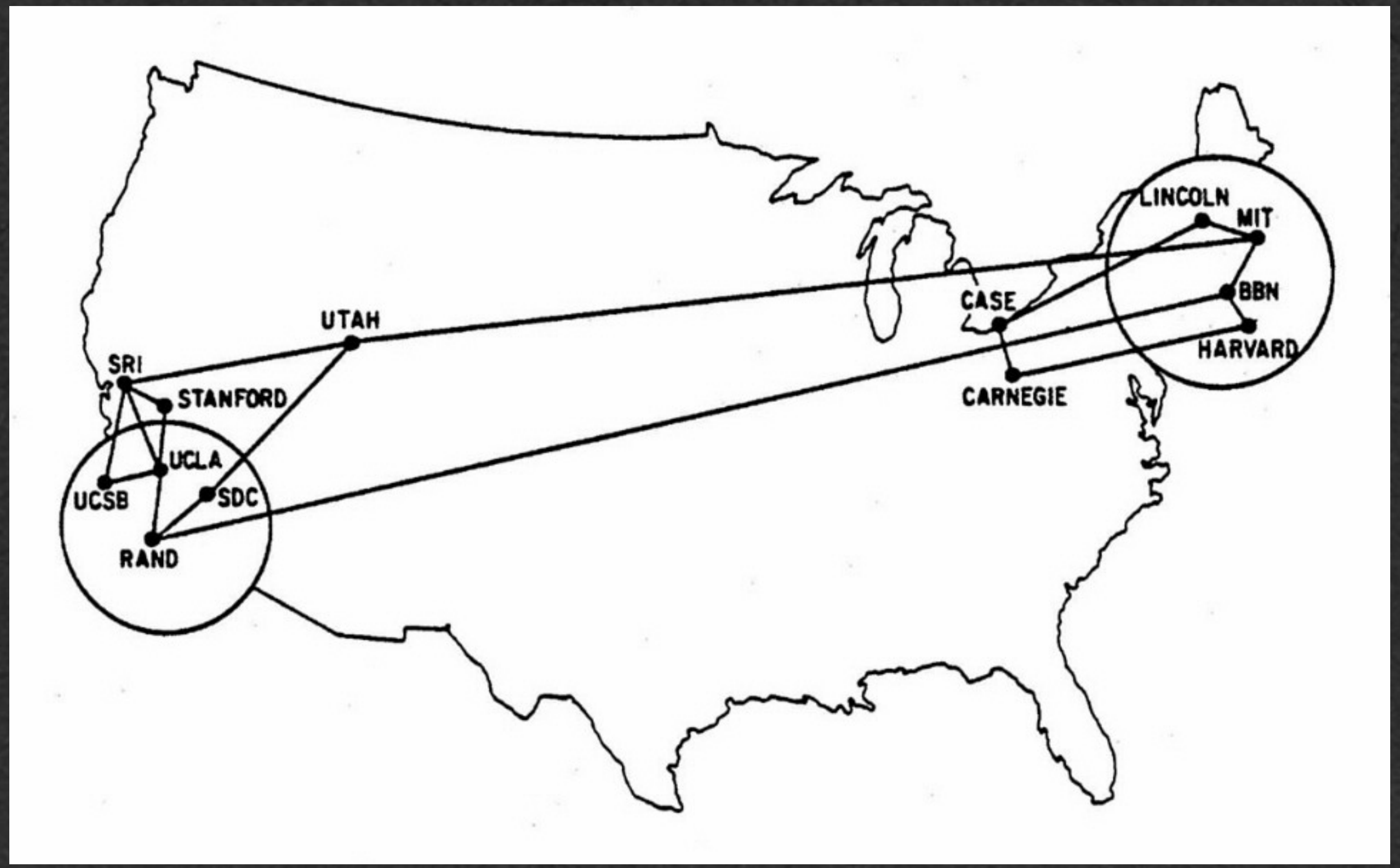


Key	"cse"	"mga"	"geo"
Value	20	3	5



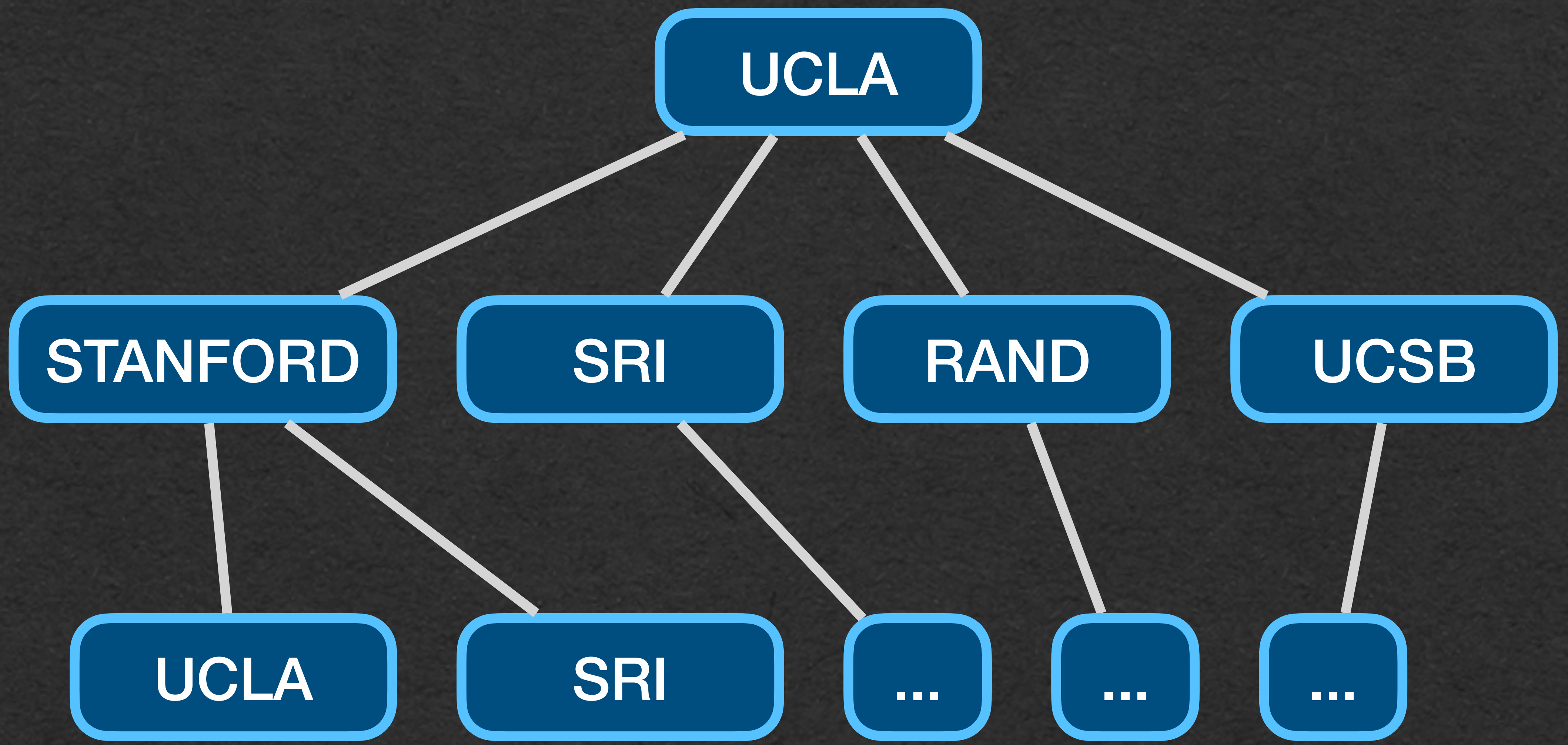
Data Structures

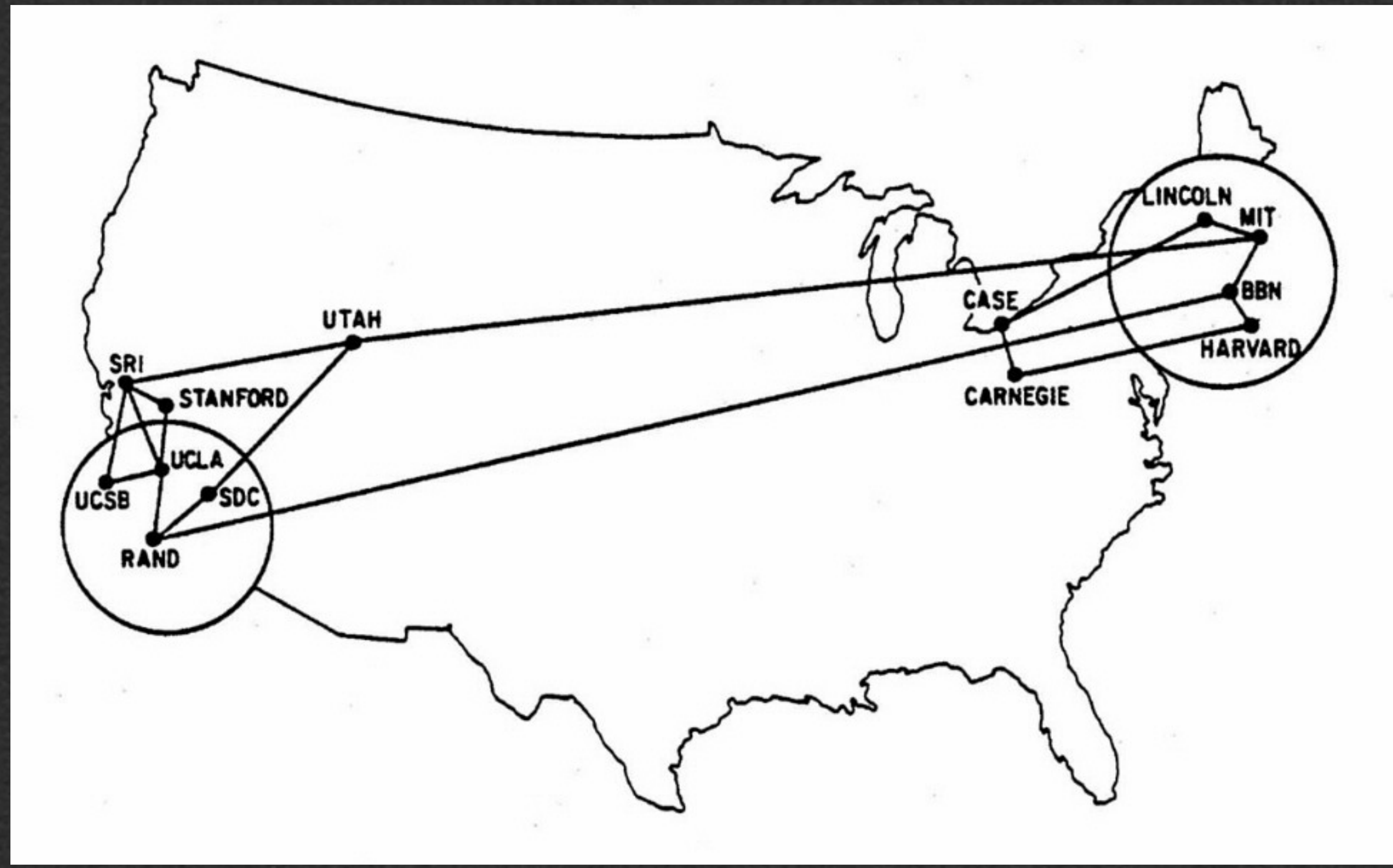
- How do we store data with multiple interconnected associations?
- A [station, intersection, city] can have multiple connections



Data Structures

- Let's use trees
- Start with UCLA as the root
- Recursively add all children

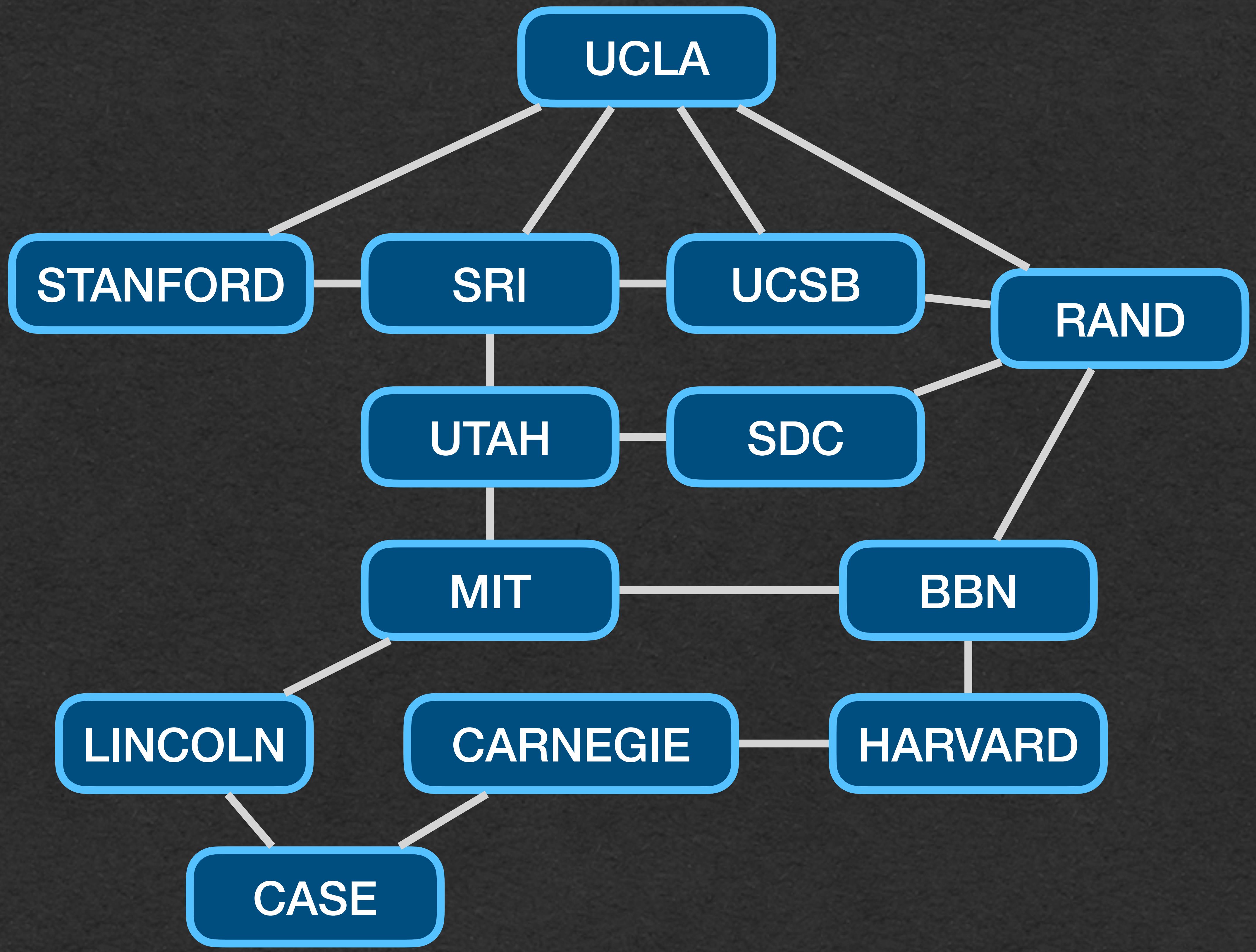


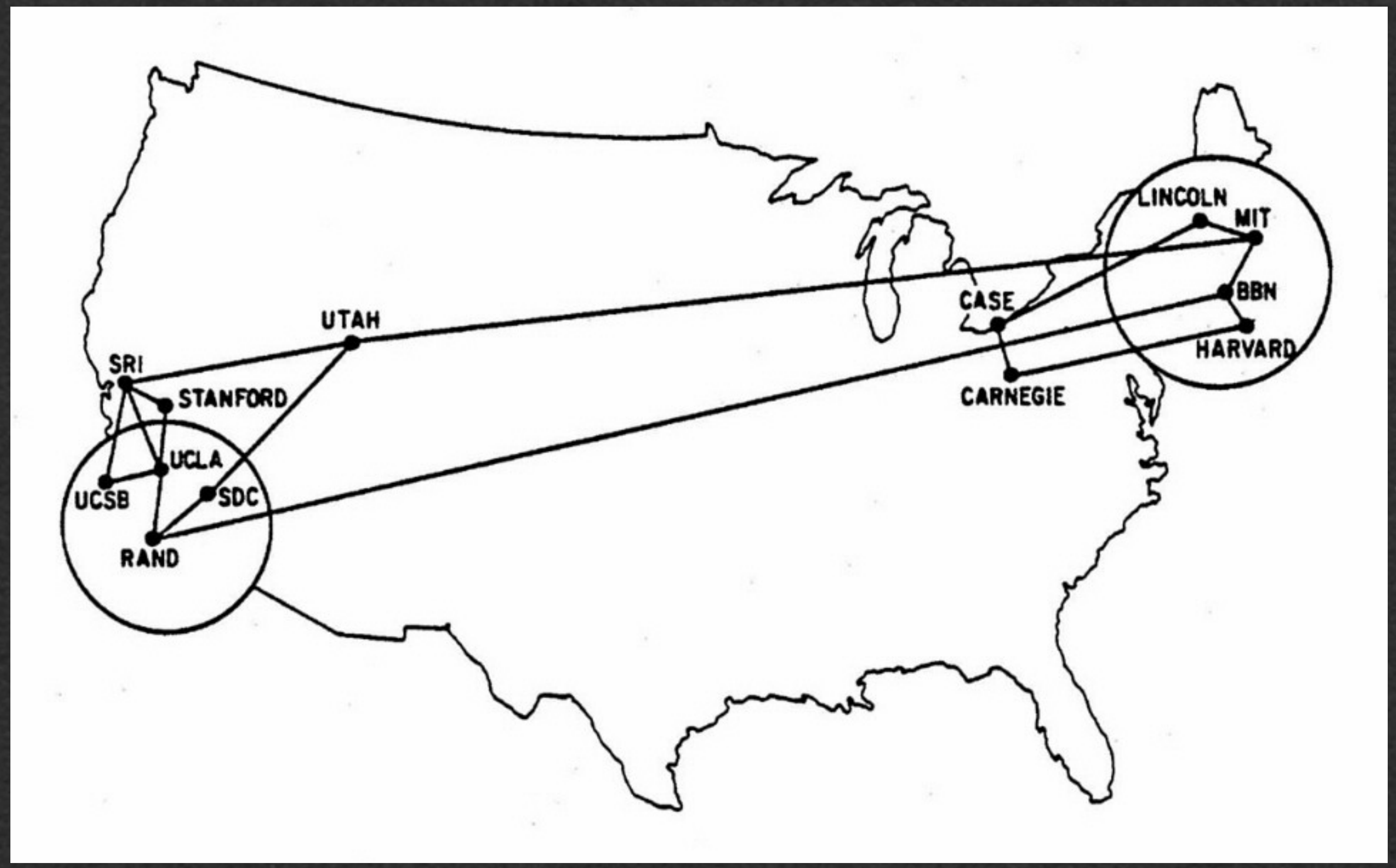


- Oops
    - We have duplicates in our data structure

Data Structures

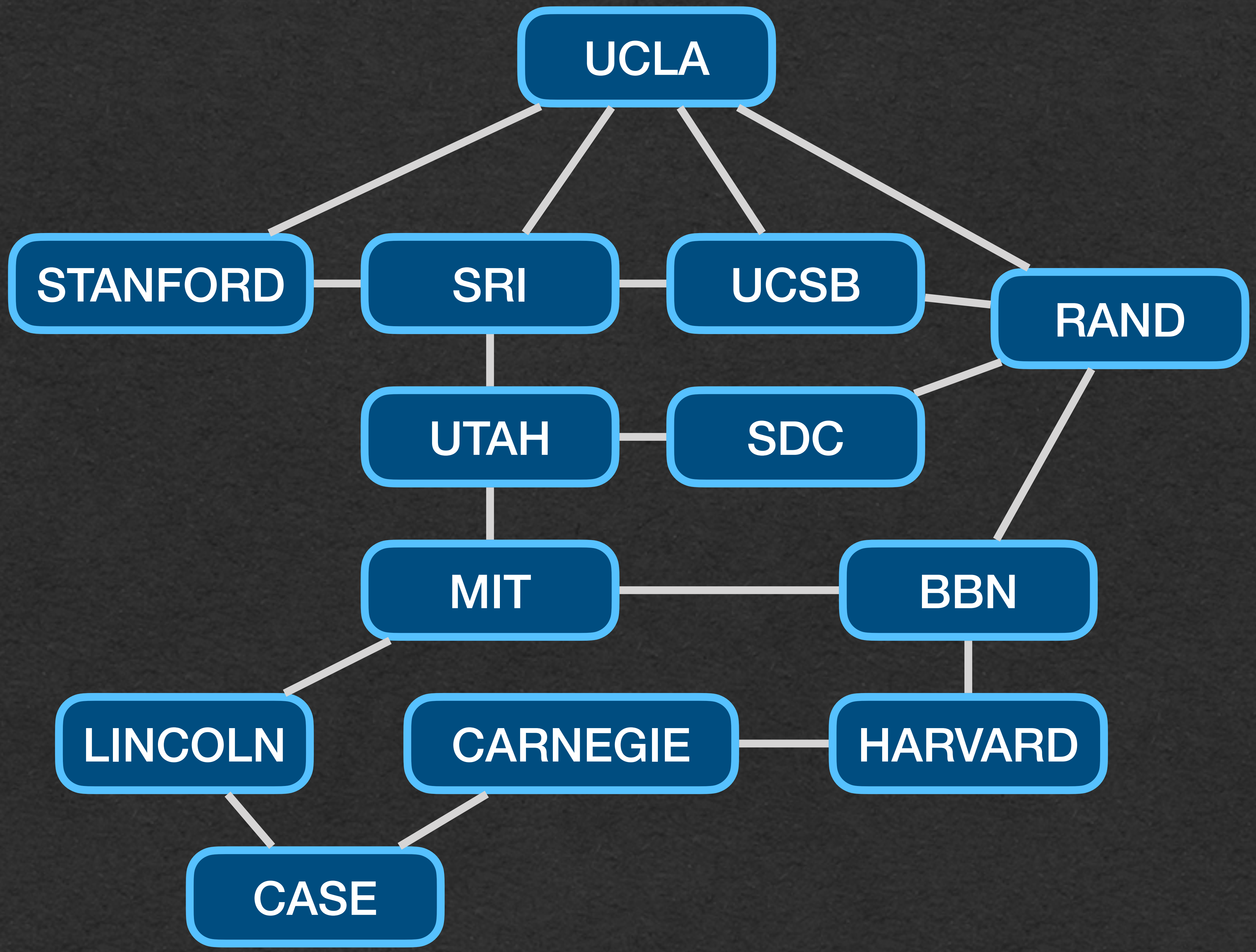
- Let's try again
  - When we try to add a duplicate, add a reference to the existing node

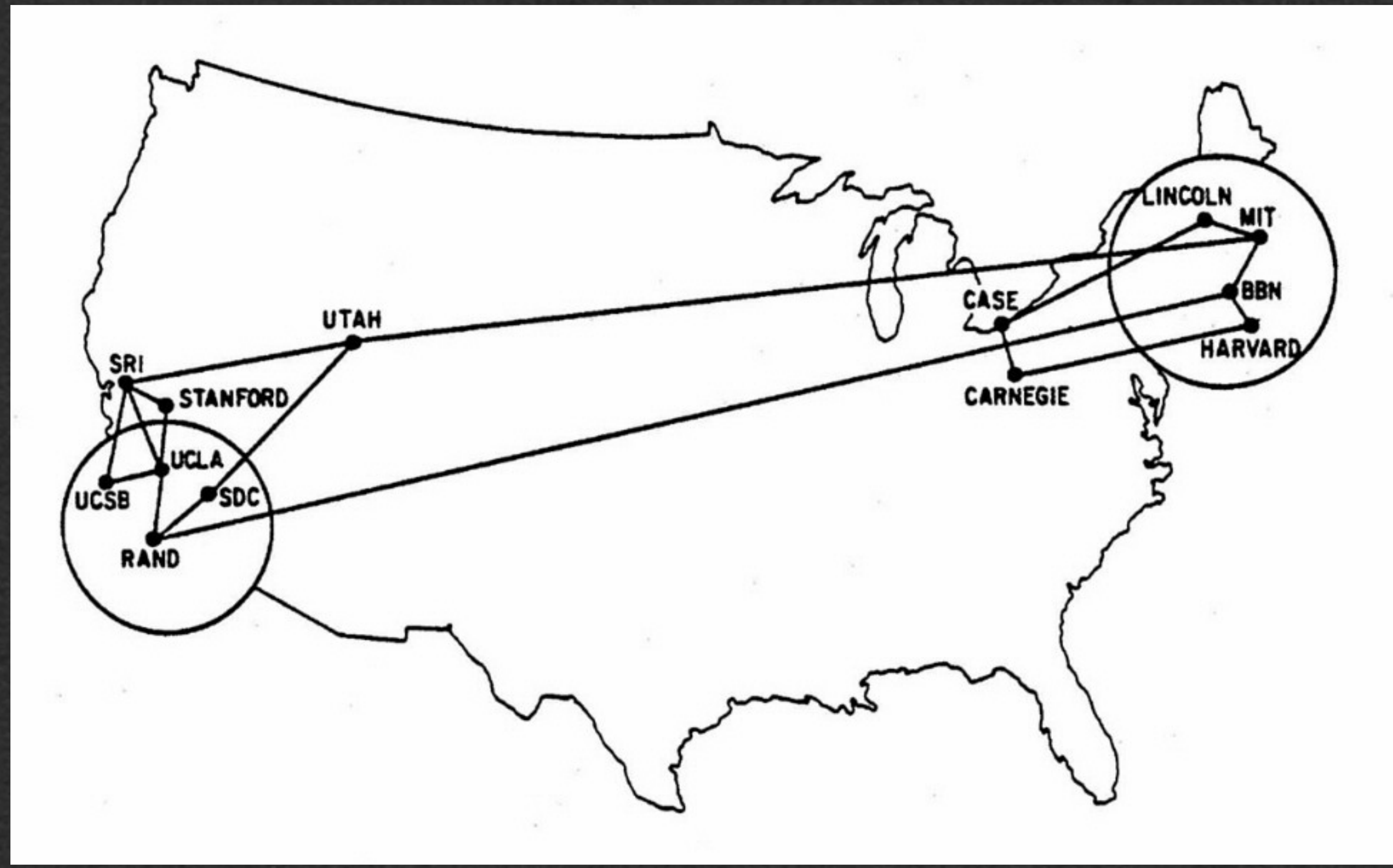




Graphs

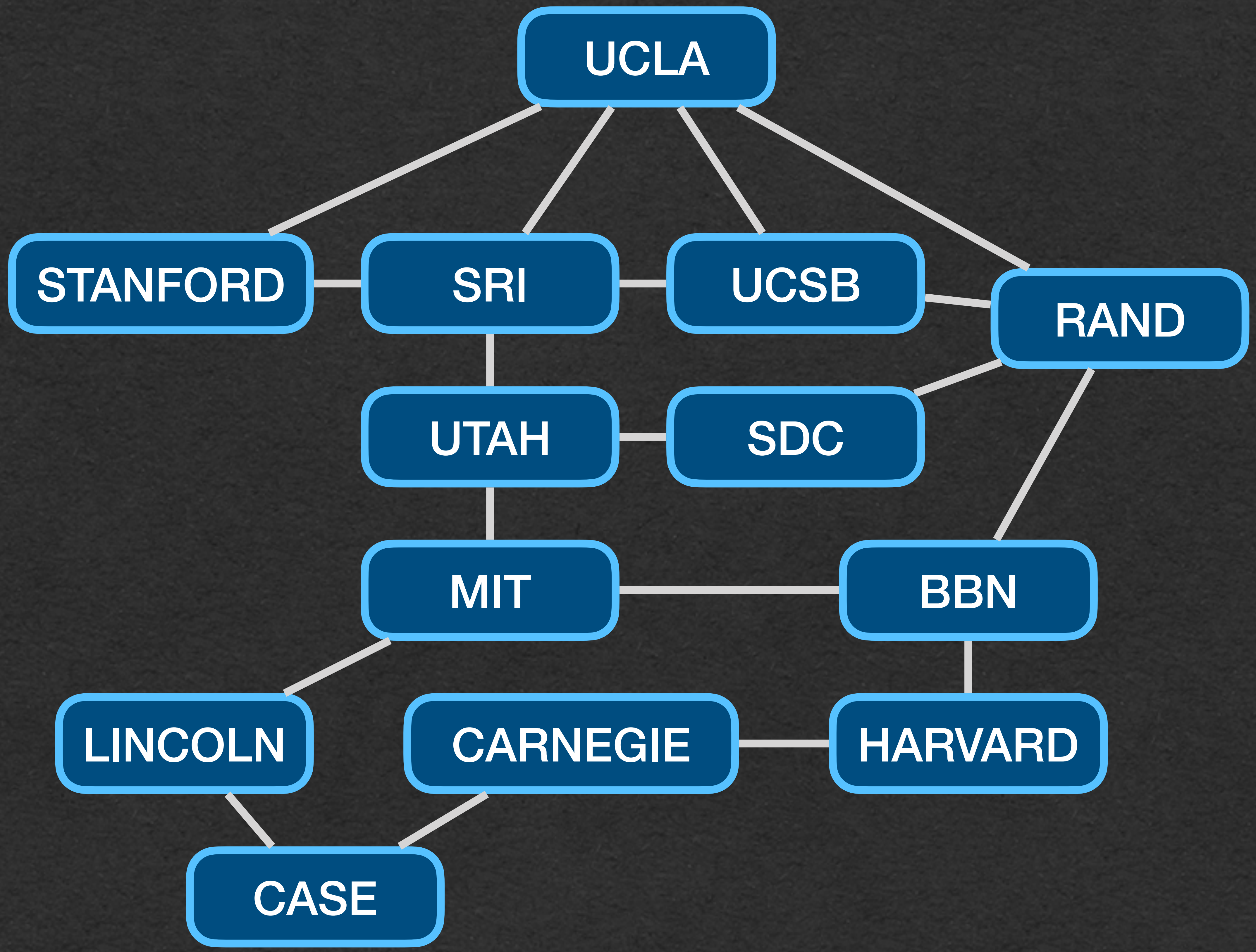
- This is a graph
- Similar to a tree, except cycles are allowed
    - Cycle: Can "travel" from a node back to itself without repeating a node

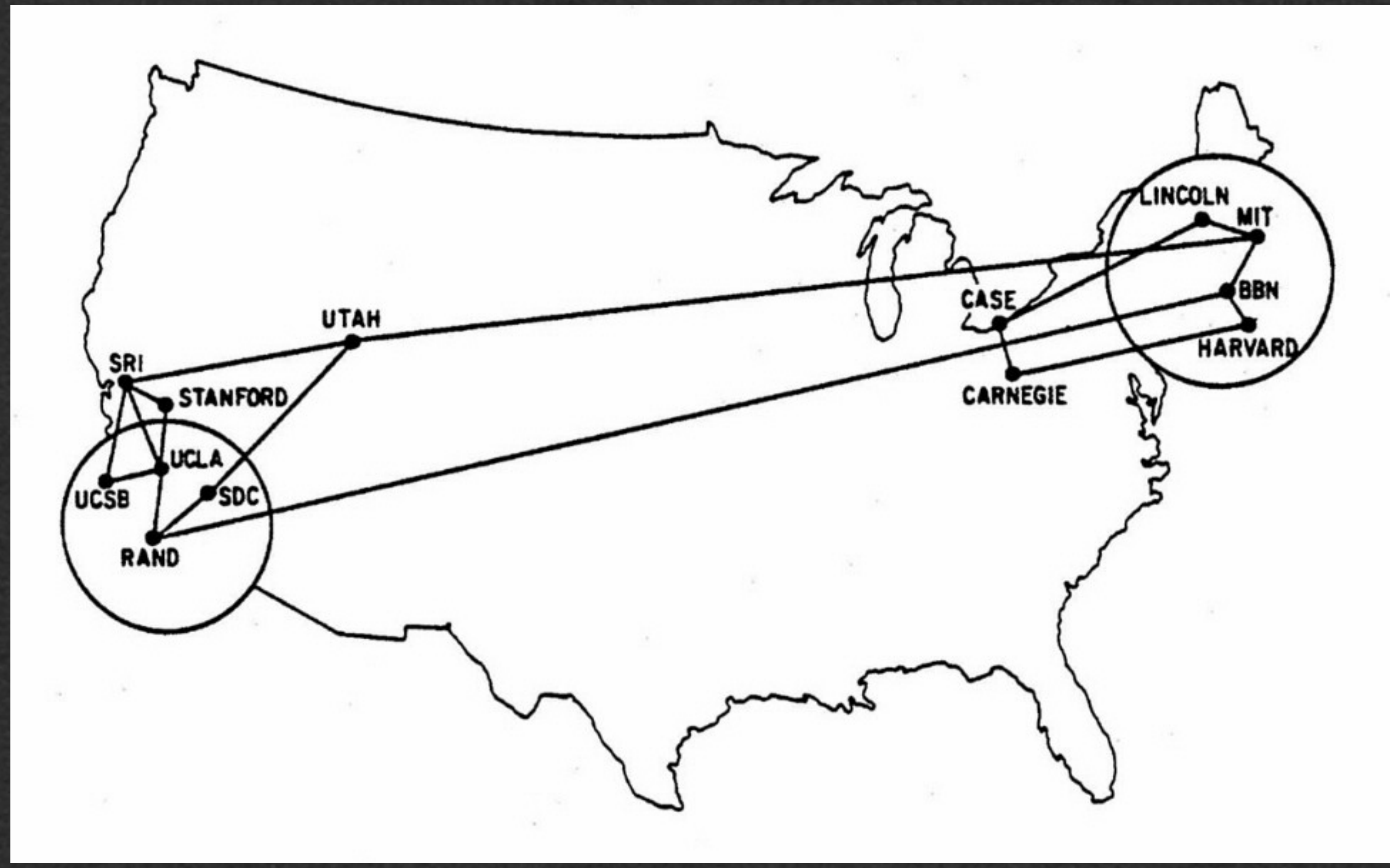




Graphs

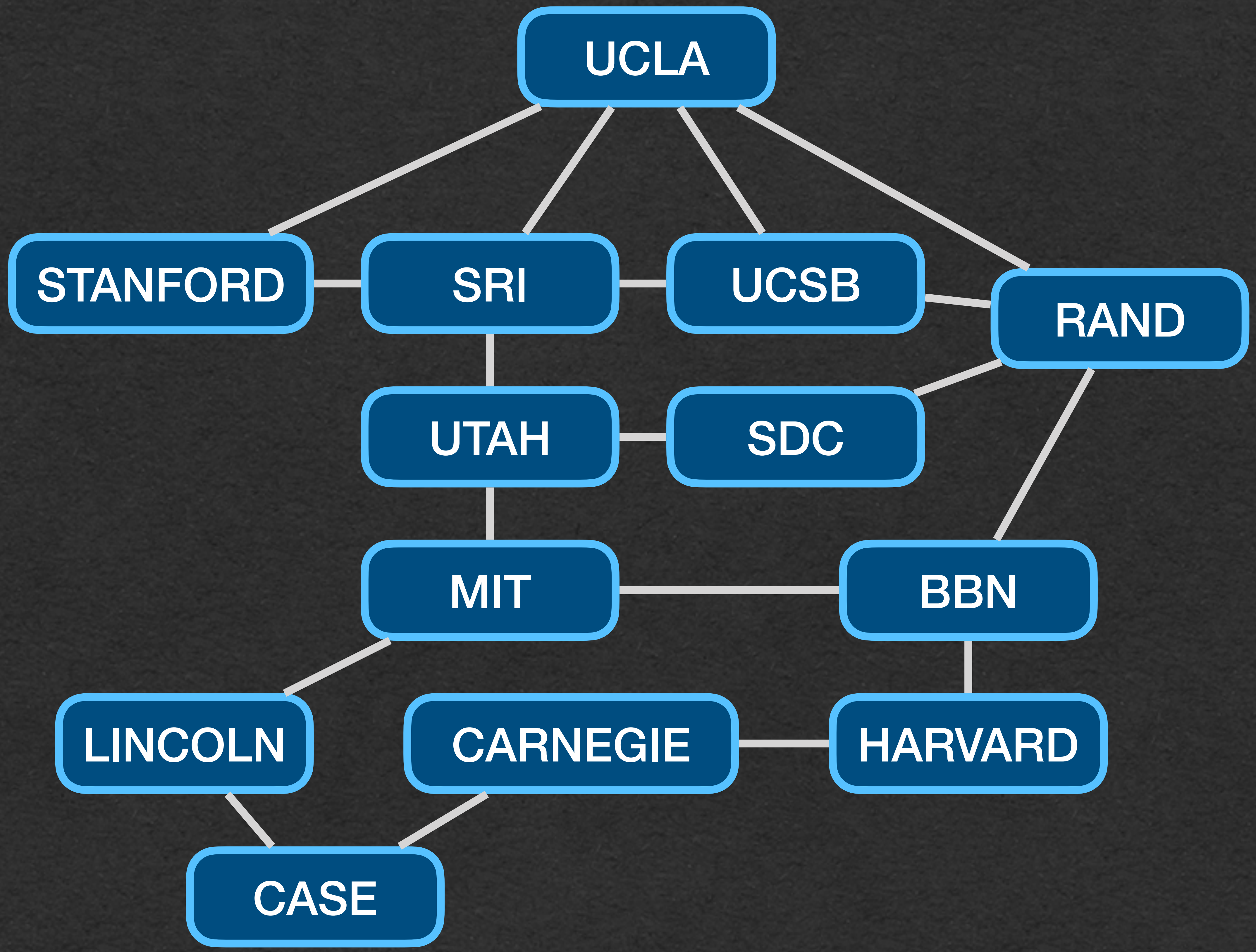
- Because of the cycles, our tree traversals will get stuck in infinite recursion
  - No leaves (node with no children) to terminate the recursion

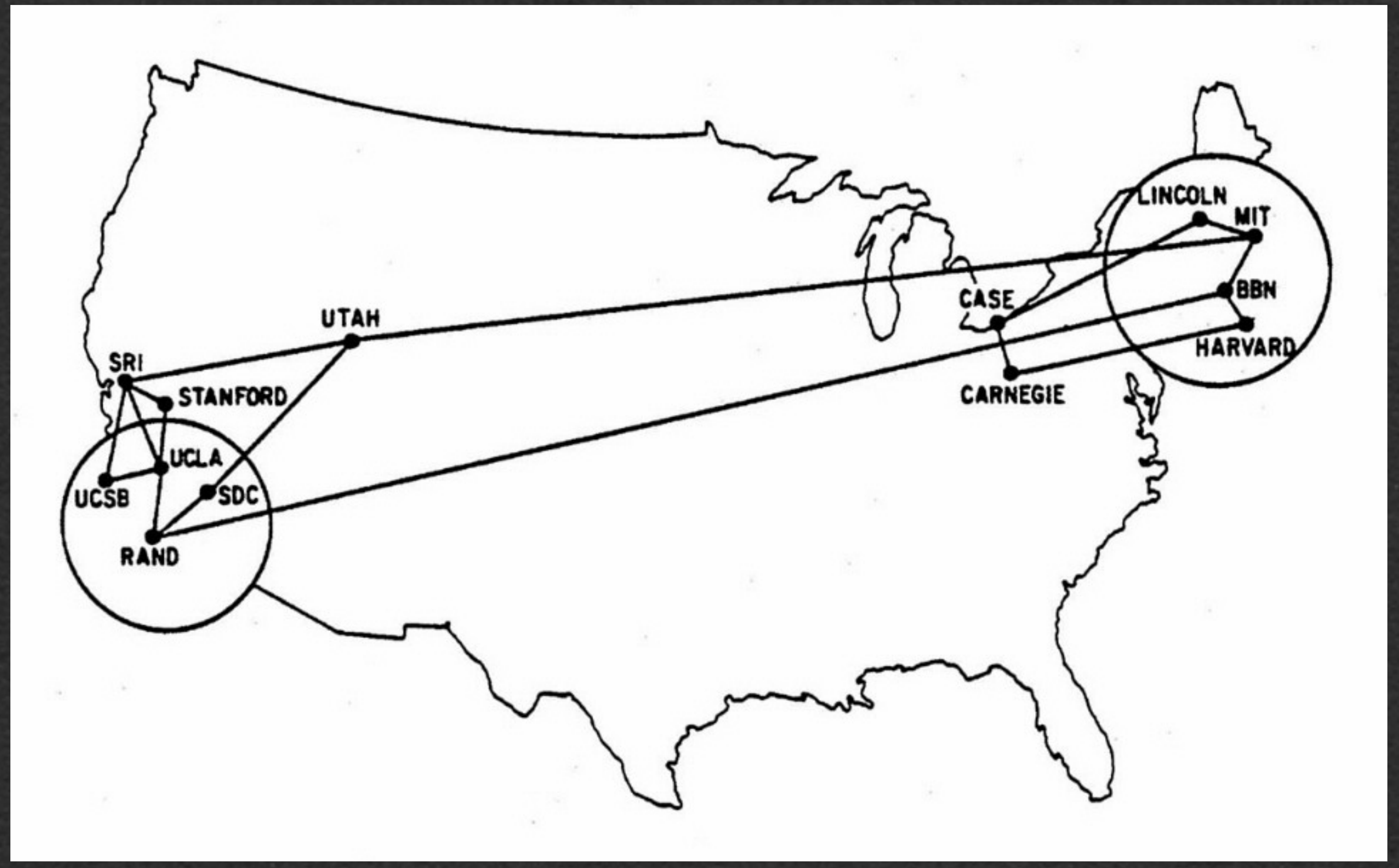




Graphs

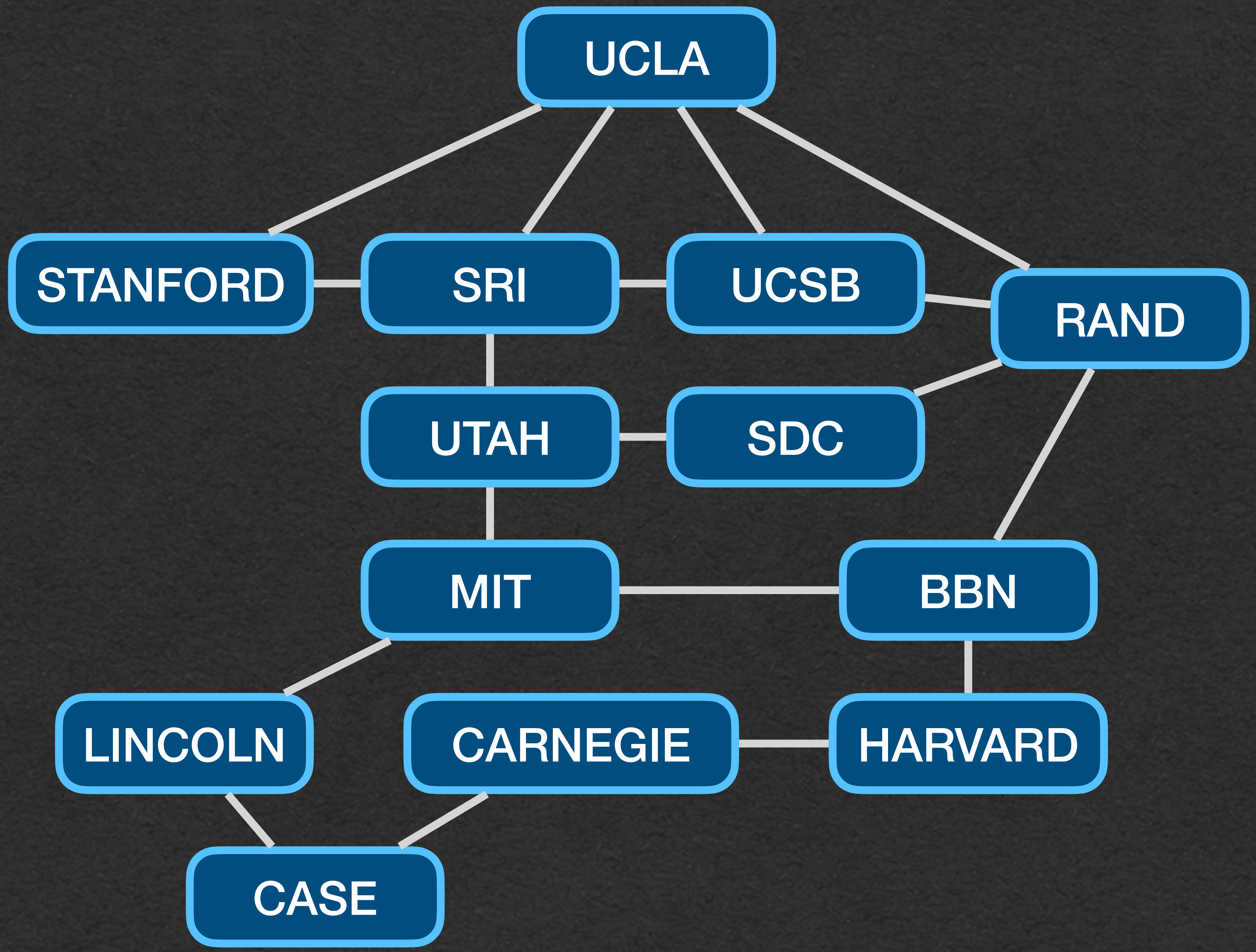
- We'll need a new way of representing this data structure and new algorithms to work with the data
- Store the nodes and edges

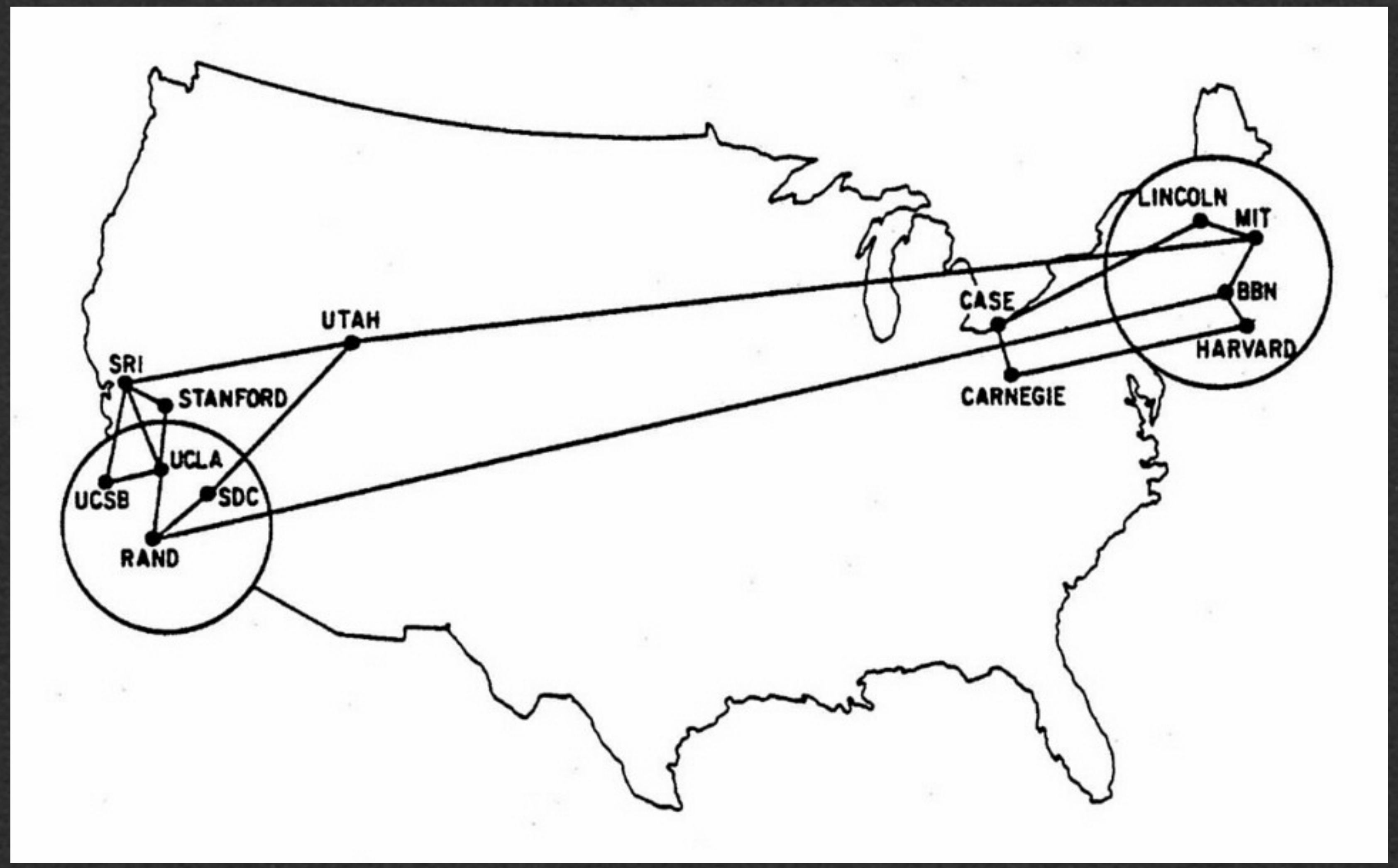




Graphs - Nodes and Edges

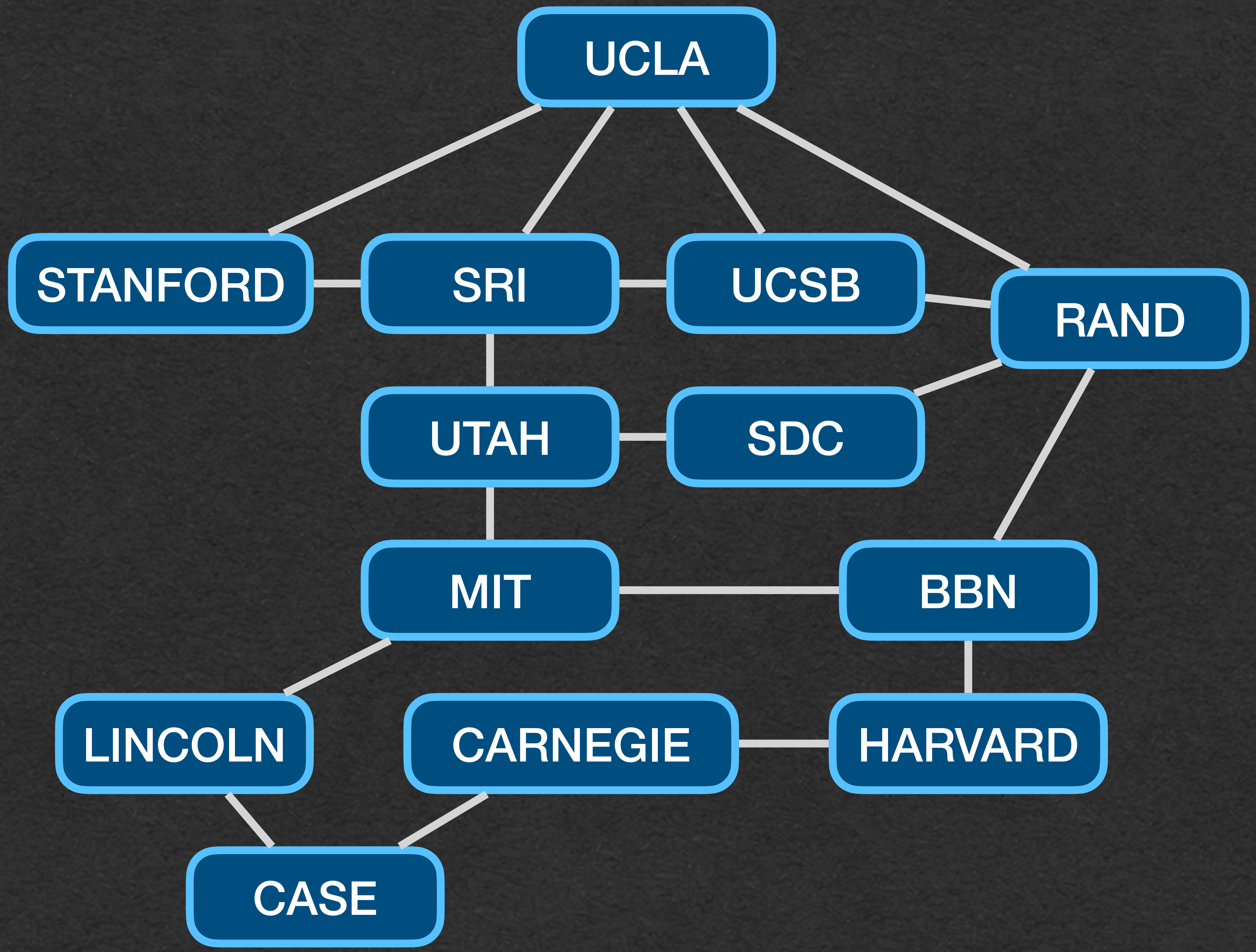
- Node: Each data element is stored in a node, similar to linked lists and trees
- Edge: A connection between two nodes





Graphs - Adjacency List

- A map of nodes to all nodes connected to it through an edge
- This is how we'll represent graphs



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

Graphs - Adjacency List

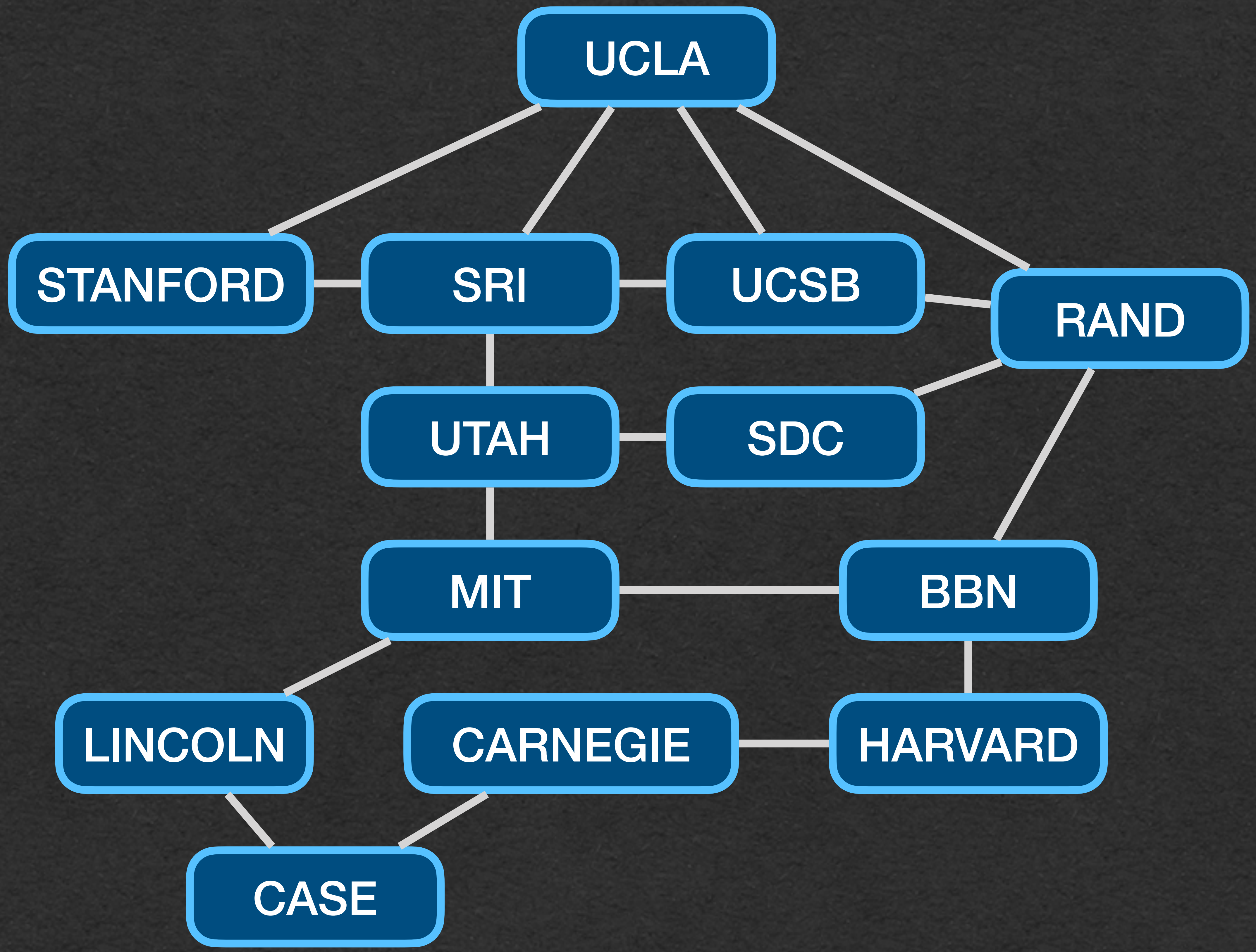
- We use generics in our code so we can create a graph of any type

```
public class Graph<N> {

    private HashMap<N, ArrayList<N>> adjacencyList;

    public Graph() {
        this.adjacencyList = new HashMap<>();
    }
    public void addEdge(N from, N to) {
        this.addNode(from);
        this.addNode(to);
        this.adjacencyList.get(from).add(to);
    }
    public void addBidirectionalEdge(N node1, N node2) {
        this.addNode(node1);
        this.addNode(node2);
        this.adjacencyList.get(node1).add(node2);
        this.adjacencyList.get(node2).add(node1);
    }
    private void addNode(N a) {
        if (!this.adjacencyList.containsKey(a)) {
            this.adjacencyList.put(a, new ArrayList<>());
        }
    }
}
```

```
public static void example(){
    Graph<String> graph = new Graph<>();
    graph.addBidirectionalEdge("UCLA","STANFORD");
    graph.addBidirectionalEdge("UCLA","SRI");
    graph.addBidirectionalEdge("UCLA","UCSB");
    graph.addBidirectionalEdge("UCLA","RAND");
    graph.addBidirectionalEdge("STANFORD","SRI");
    graph.addBidirectionalEdge("SRI","UCSB");
    graph.addBidirectionalEdge("UCSB","RAND");
    graph.addBidirectionalEdge("SRI","UTAH");
    graph.addBidirectionalEdge("RAND","SDC");
    graph.addBidirectionalEdge("UTAH","SDC");
    graph.addBidirectionalEdge("UTAH","MIT");
    graph.addBidirectionalEdge("RAND","BBN");
    graph.addBidirectionalEdge("MIT","BBN");
    graph.addBidirectionalEdge("MIT","LINCOLN");
    graph.addBidirectionalEdge("LINCOLN","CASE");
    graph.addBidirectionalEdge("CASE","CARNEGIE");
    graph.addBidirectionalEdge("CARNEGIE","HARVARD");
    graph.addBidirectionalEdge("HARVARD","BBN");
}
```

Paths

- A path is a sequence of nodes where each pair of adjacent nodes are connected by an edge
- ["UCLA", "SRI", "UTAH", "MIT", "BBN", "RAND"] is a path in this graph
- ["SRI", "UTAH", "BBN"] is not a path since UTAH and BBN are not connected by an edge

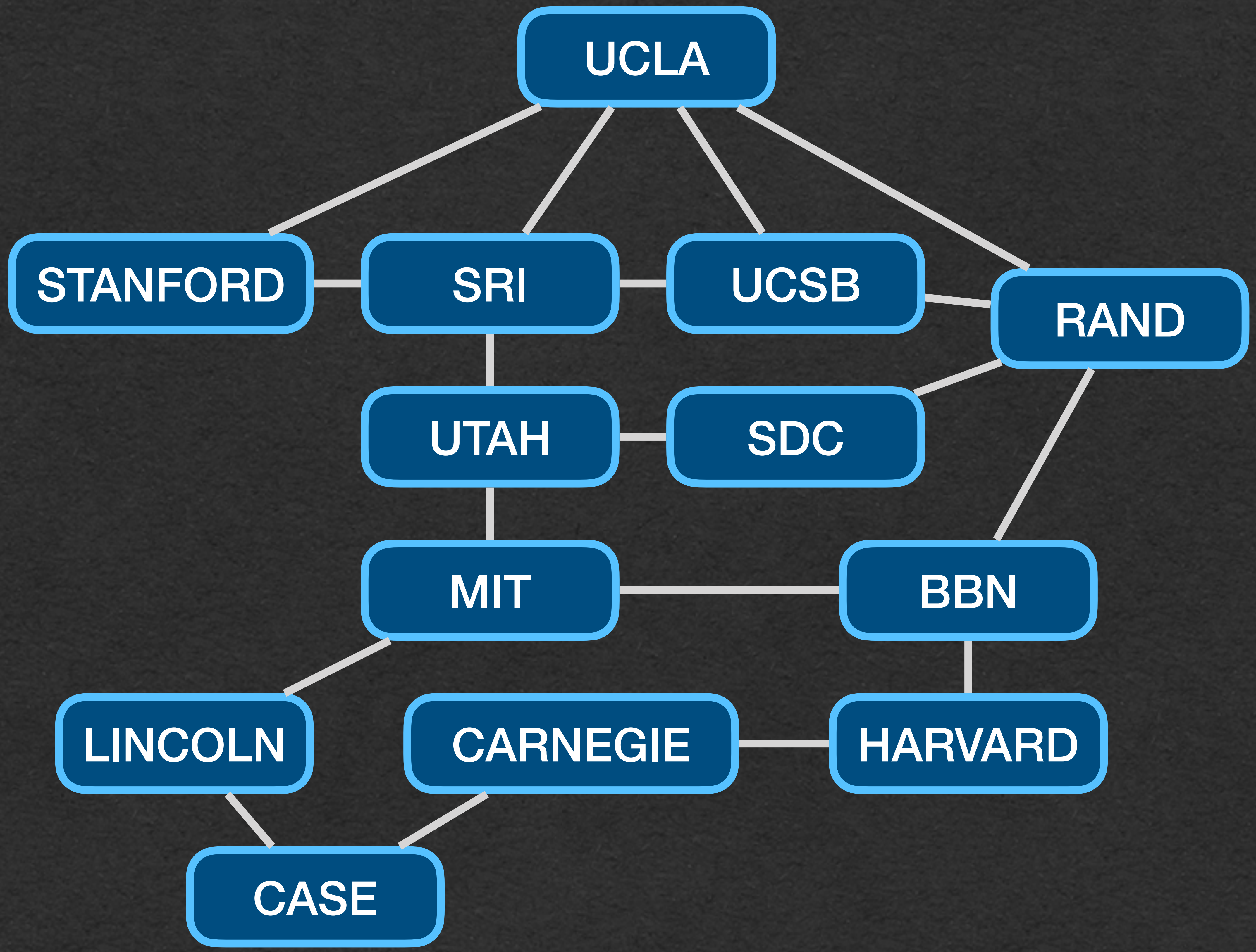


UCLA	STANFORD, SRI, UCSB, RAND
STANFORD	UCLA, SRI
SRI	UCLA, UCSB, UTAH, STANFORD
UCSB	UCLA, SRI, RAND
RAND	UCLA, UCSB, SDC, BBN
UTAH	SRI, SDC, MIT
SDC	UTAH, RAND
MIT	UTAH, BBN, LINCOLN
BBN	RAND, MIT, HAVARD
LINCOLN	MIT, CASE
CARNEGIE	CASE, HARVARD
HARVARD	BBN, CARNEGIE
CASE	LINCOLN, CARNEGIE

Breadth-First Search (BFS)

Connected Component

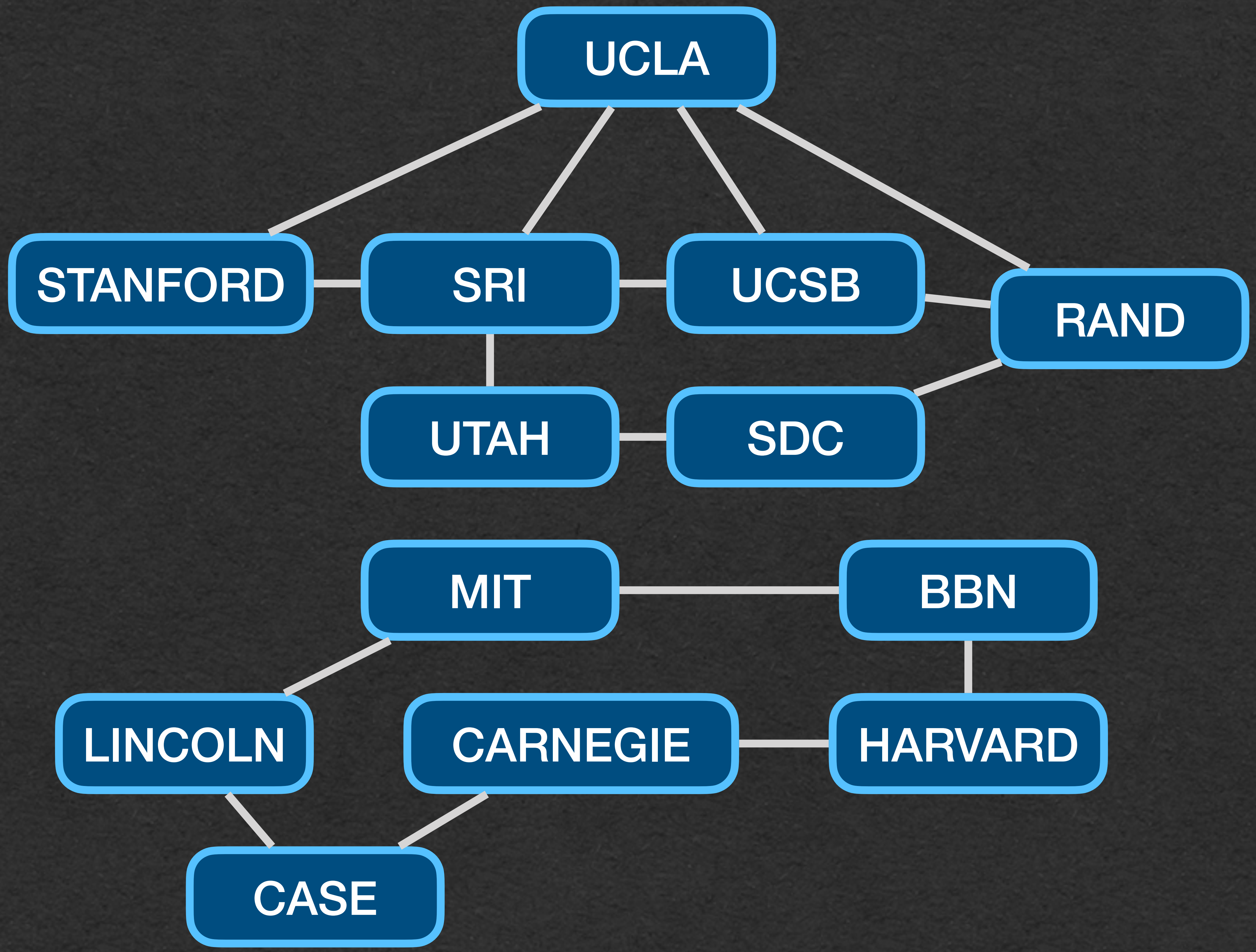
- This graph is connected
  - There exists a path between any 2 nodes in the graph



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

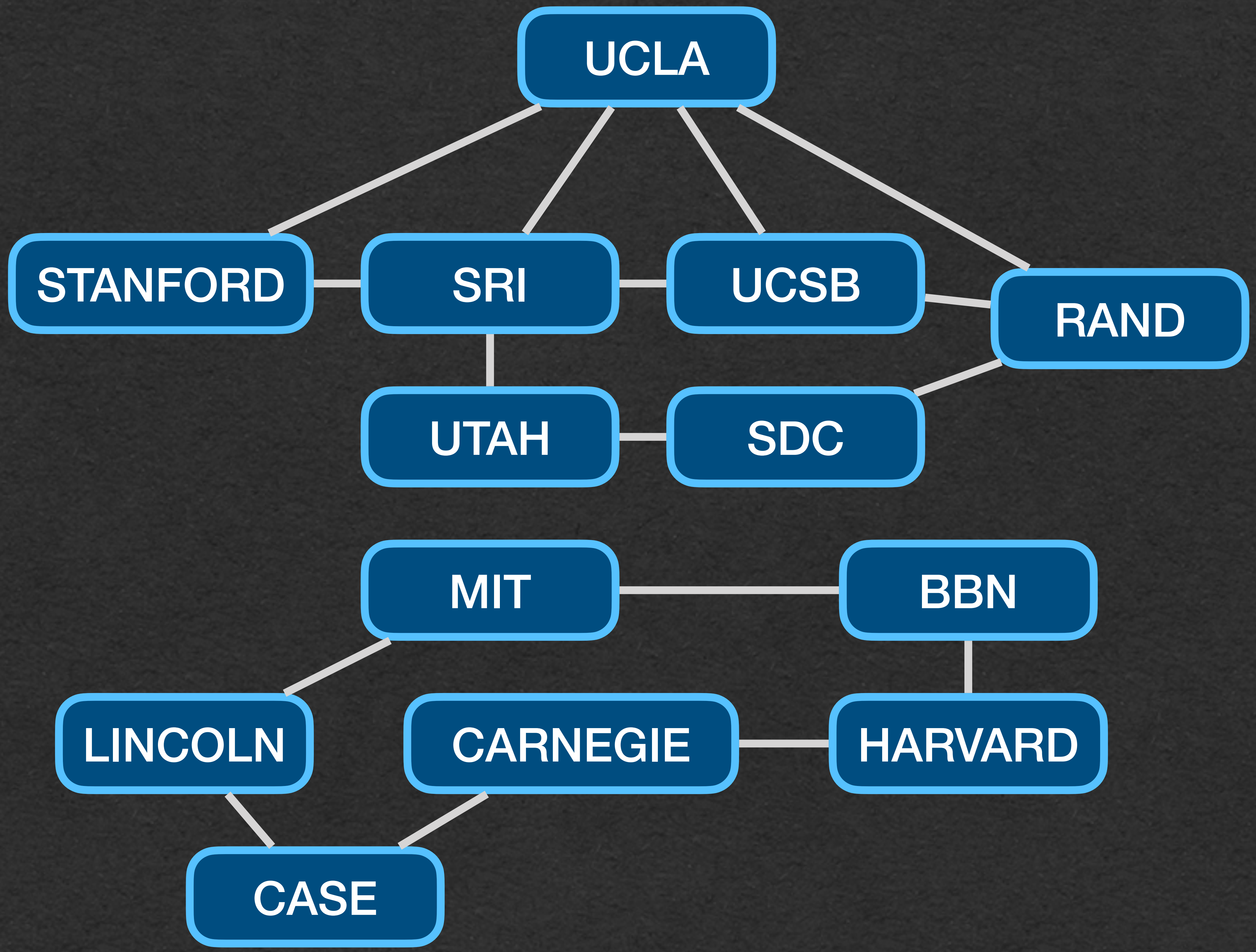
- What if a few connections are broken?
  - How can we tell if two nodes are connected?

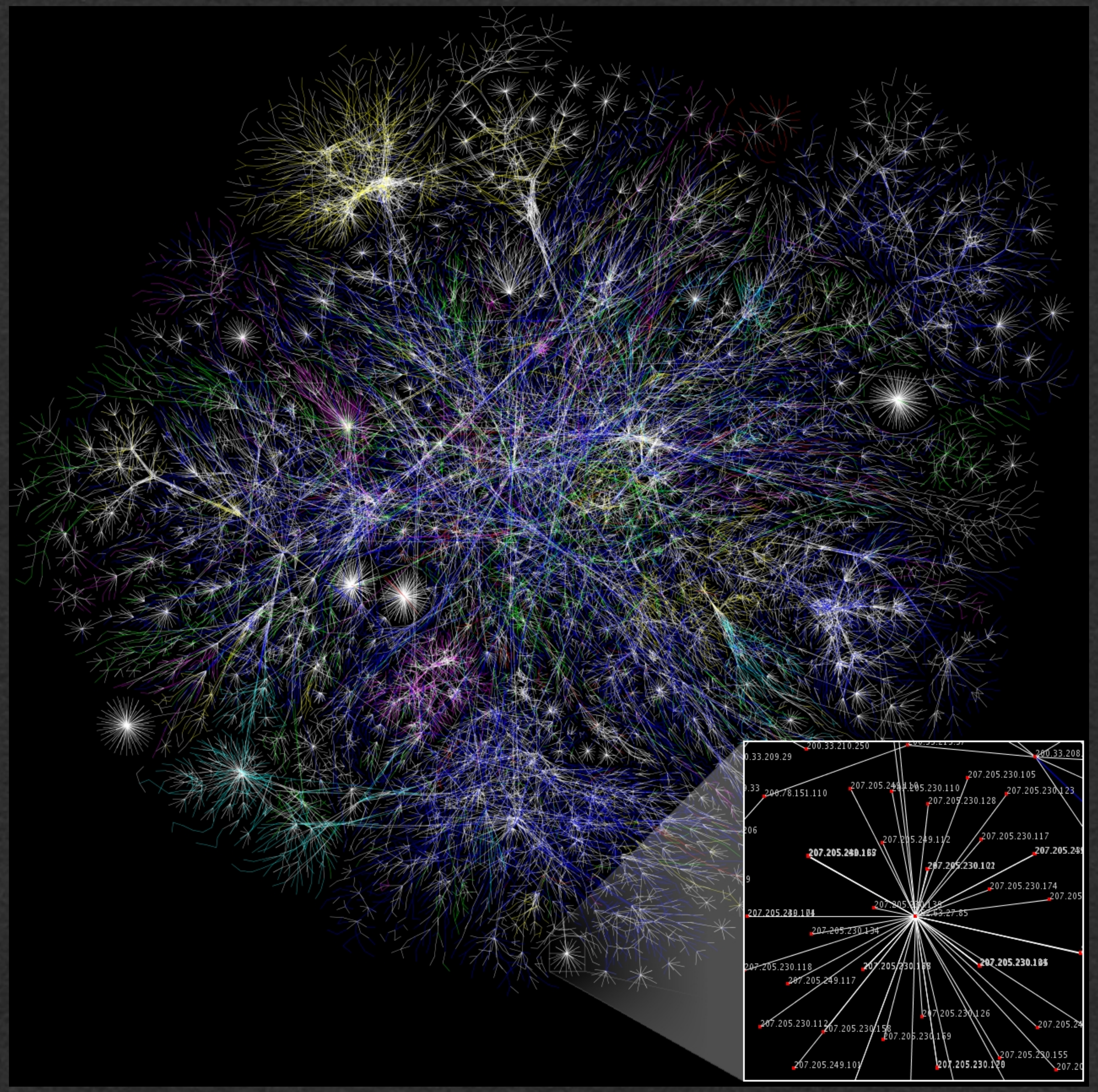


UCLA	STANFORD, SRI, UCSB, RAND
STANFORD	UCLA, SRI
SRI	UCLA, UCSB, UTAH, STANFORD
UCSB	UCLA, SRI, RAND
RAND	UCLA, UCSB, SDC
UTAH	SRI, SDC
SDC	UTAH, RAND
MIT	BBN, LINCOLN
BBN	MIT, HAVARD
LINCOLN	MIT, CASE
CARNEGIE	CASE, HARVARD
HARVARD	BBN, CARNEGIE
CASE	LINCOLN, CARNEGIE

Connected Component

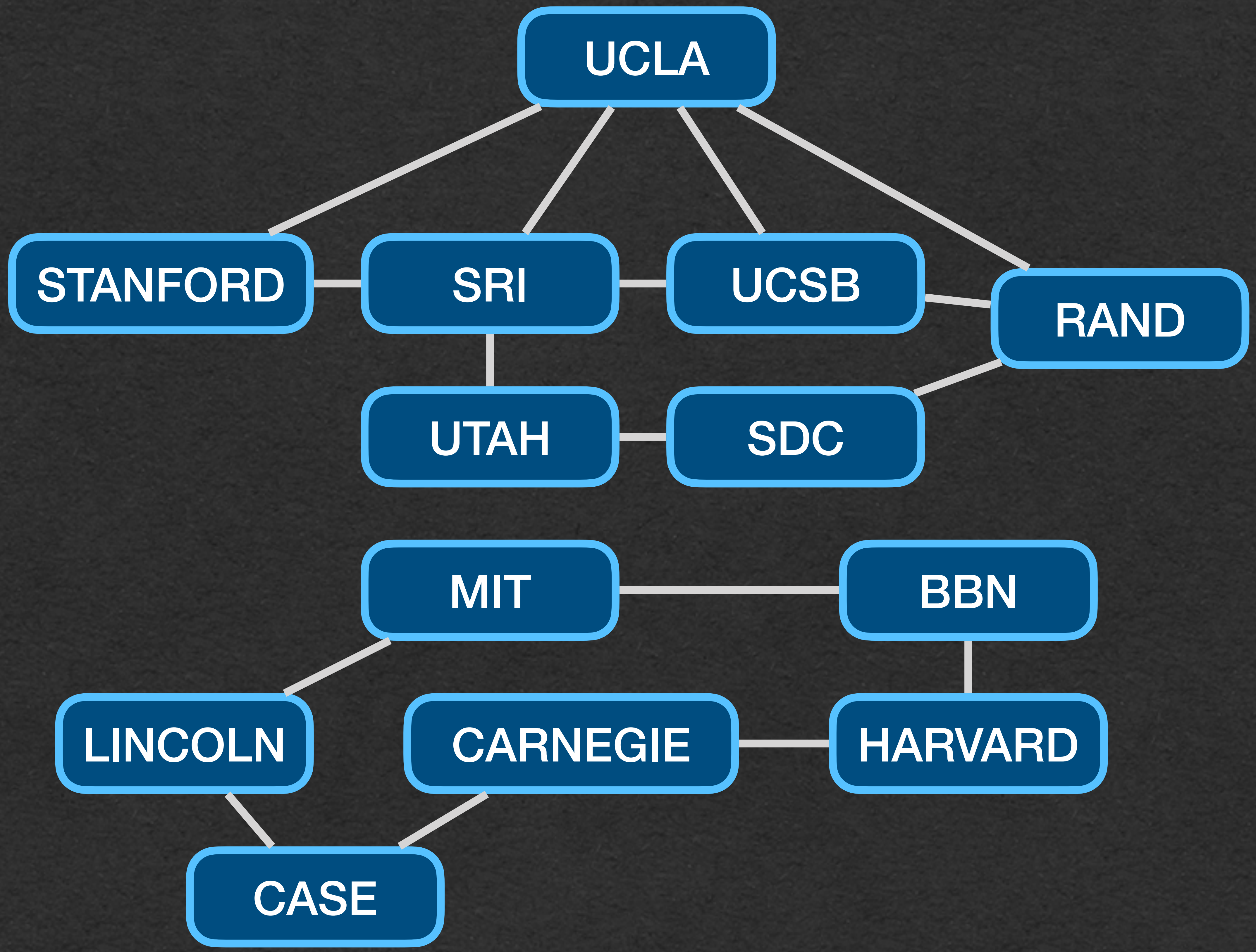
- We could verify manually for this graph
- But the Internet has gotten a *little* bigger over time
- Need to code an algorithm to solve this for us





BFS

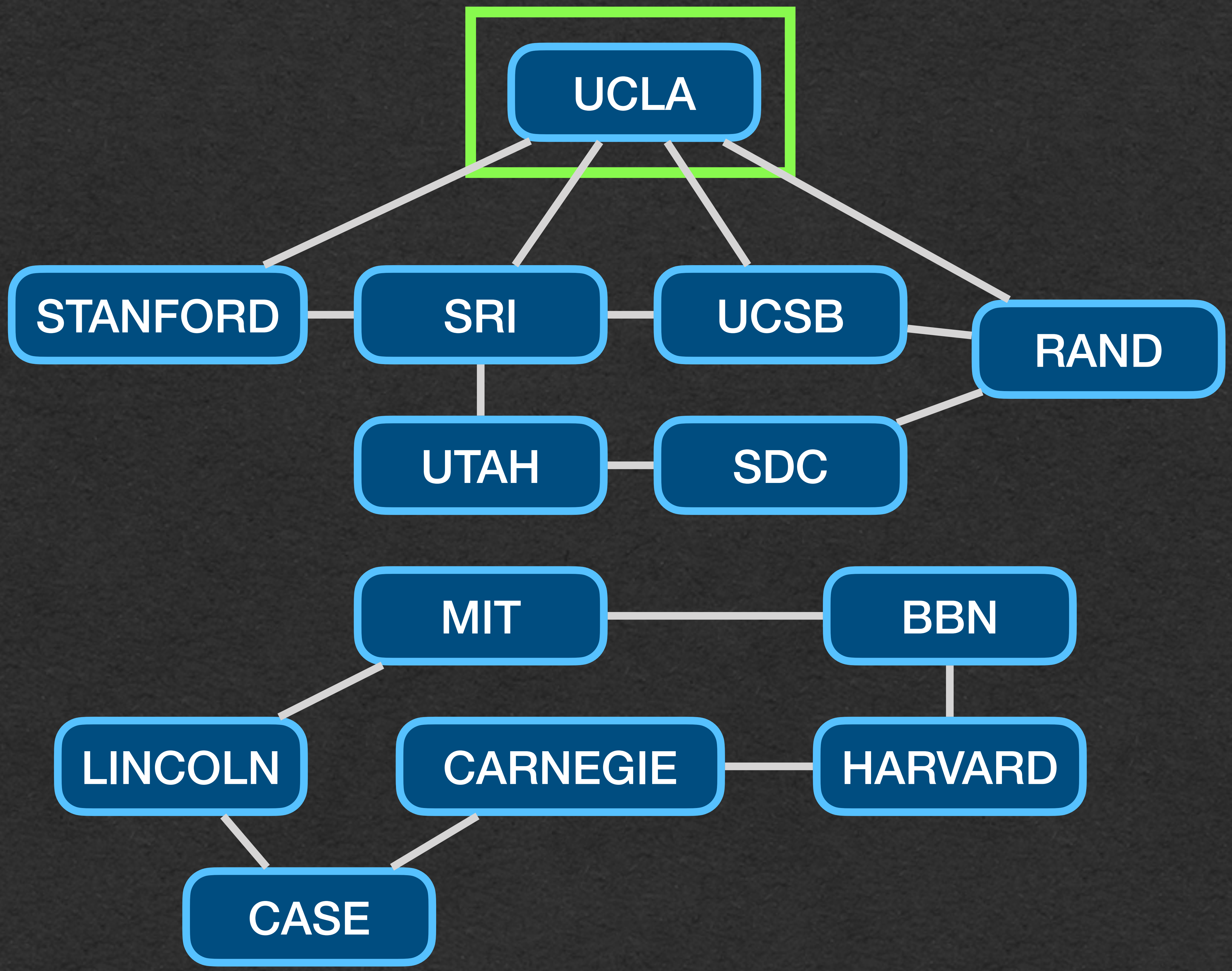
- The Algorithm: Breadth-First Search (BFS)
  - Choose a starting node
  - Continuously explore connected nodes



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

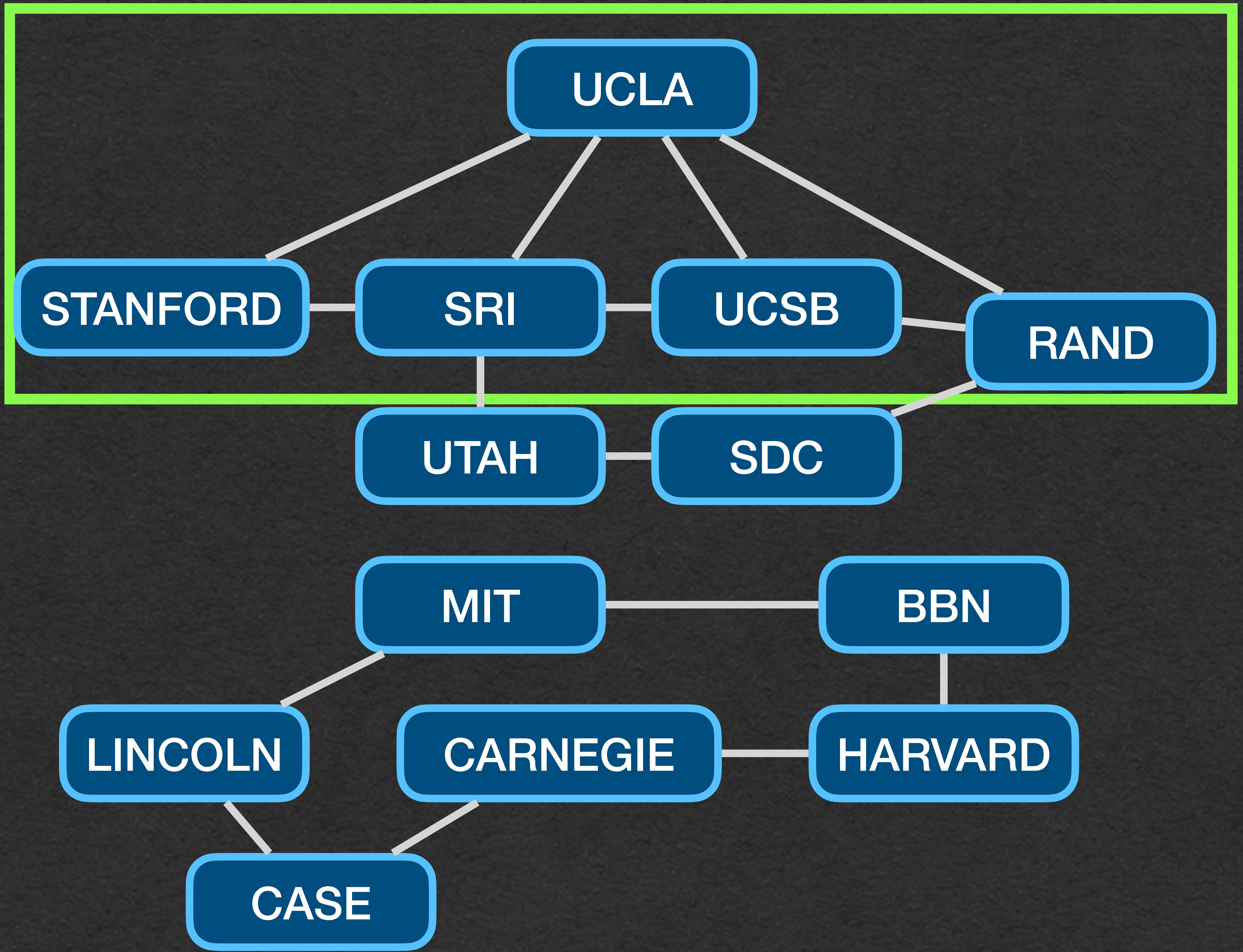
- Choose a starting node



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

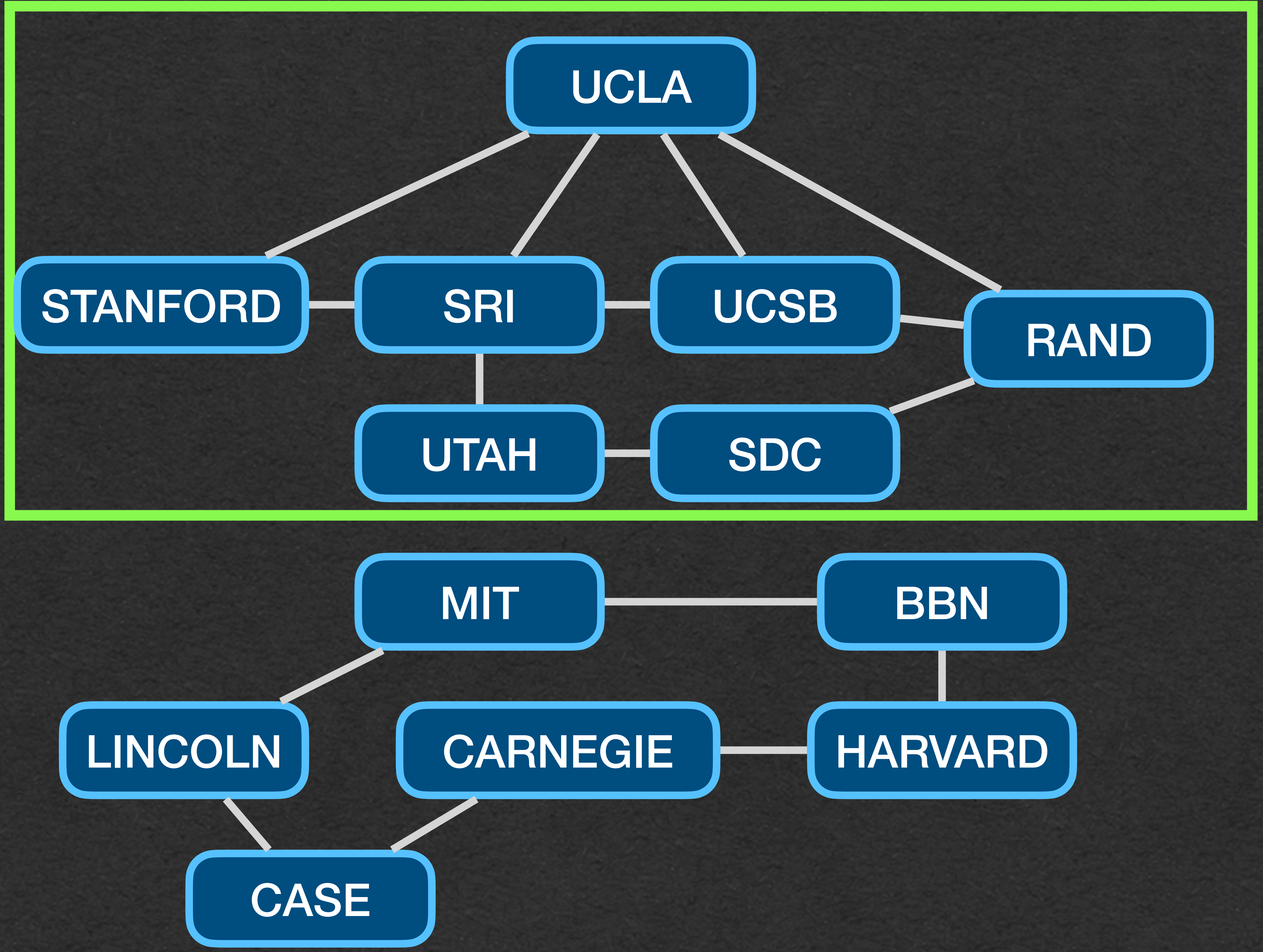
- Explore all nodes connected to the striating node



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

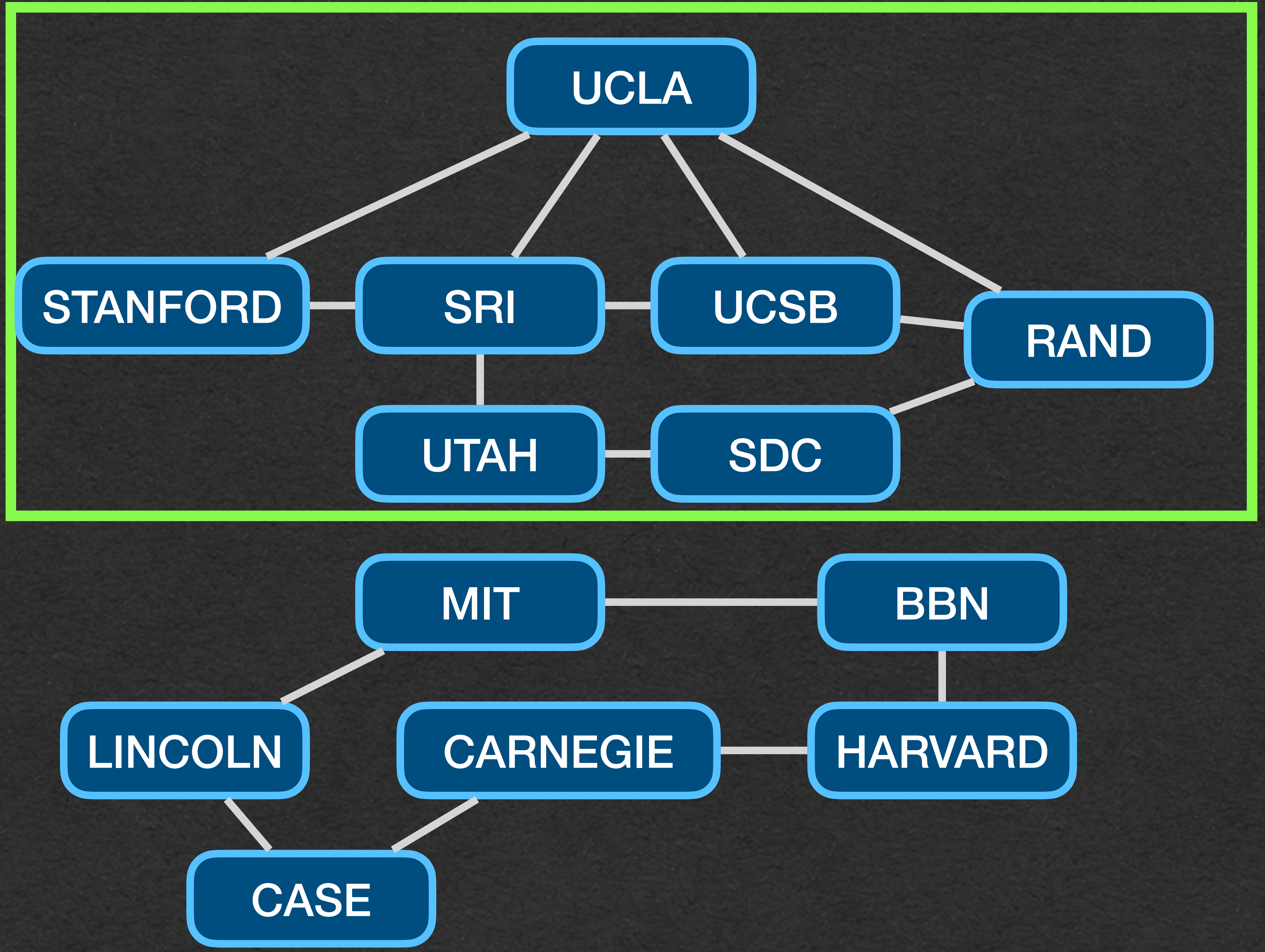
- Repeatedly explore nodes that were visited in the last round



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

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

- More BFS details to come

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 nodeA and explore nodeB during the algorithm
  - nodeA and nodeB are connected