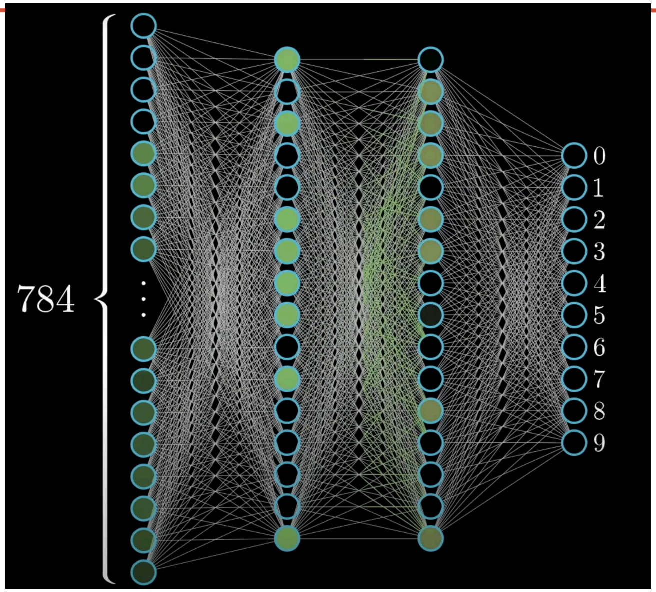
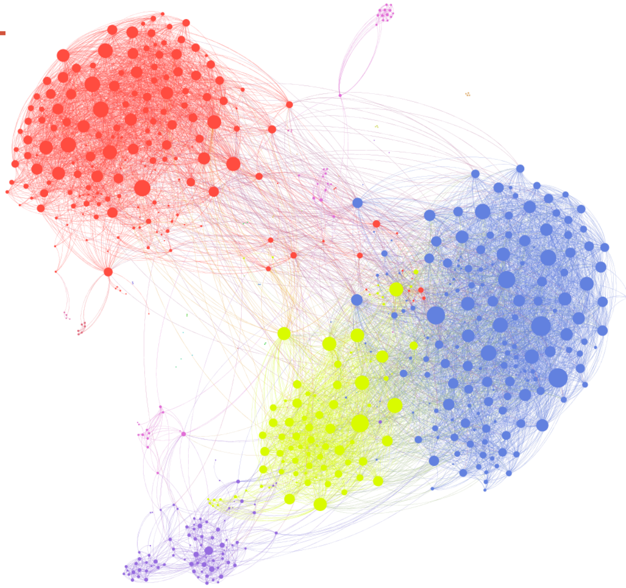
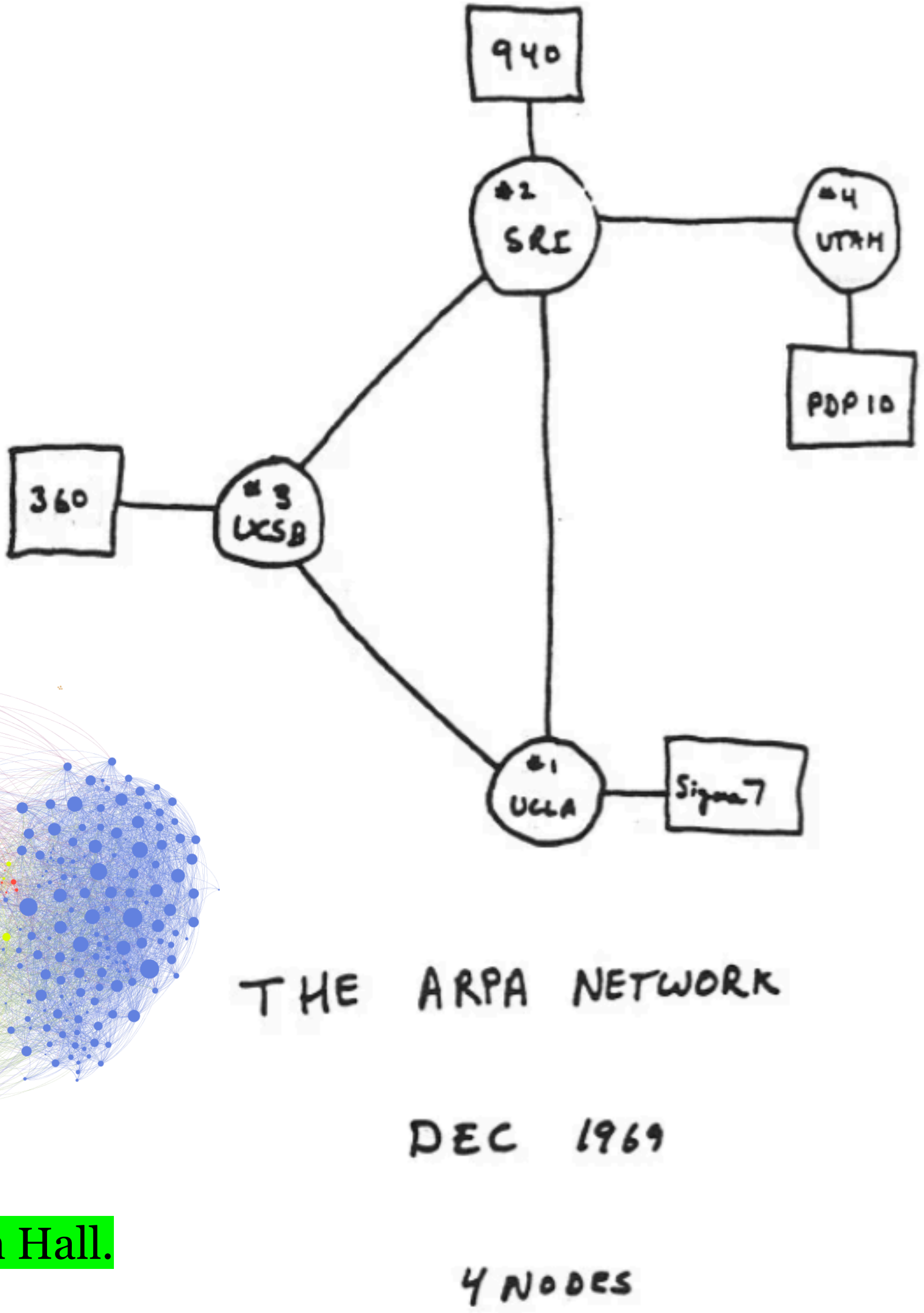
GRAPHS



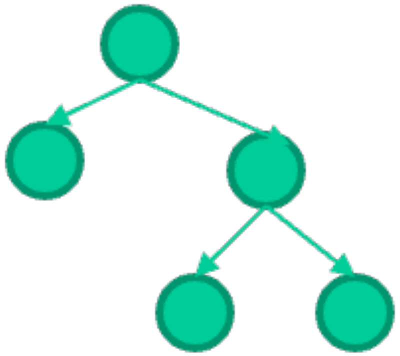




*The IBM 360, the IMP, and the workstations were all located in North Hall.
https://jeweledplatypus.org/news/text/ucsbnet.html

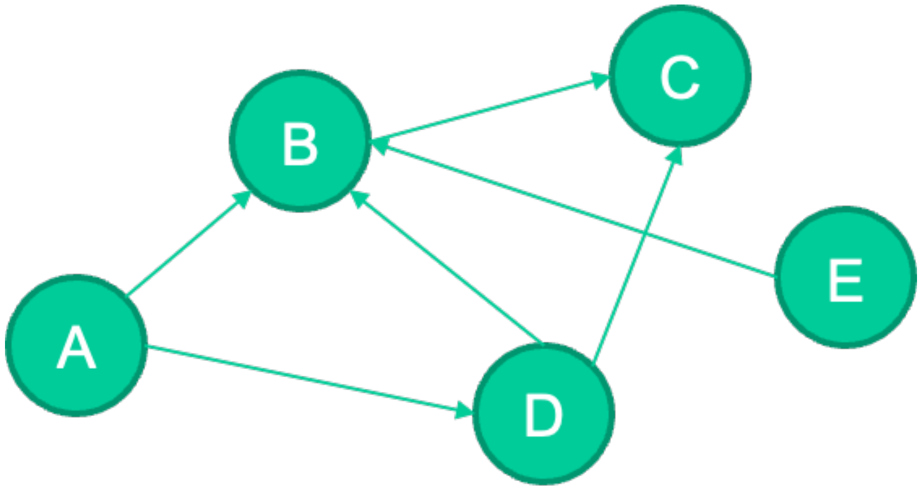
Kinds of data structures

Sequential, linear structures
(arrays, linked lists)



Hierarchical structures
(trees)

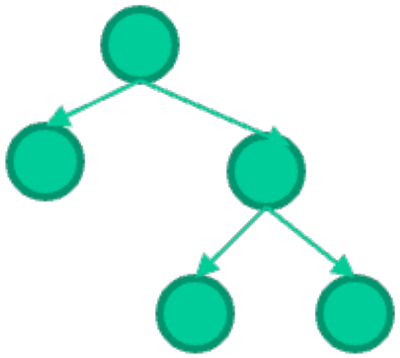
Graphs



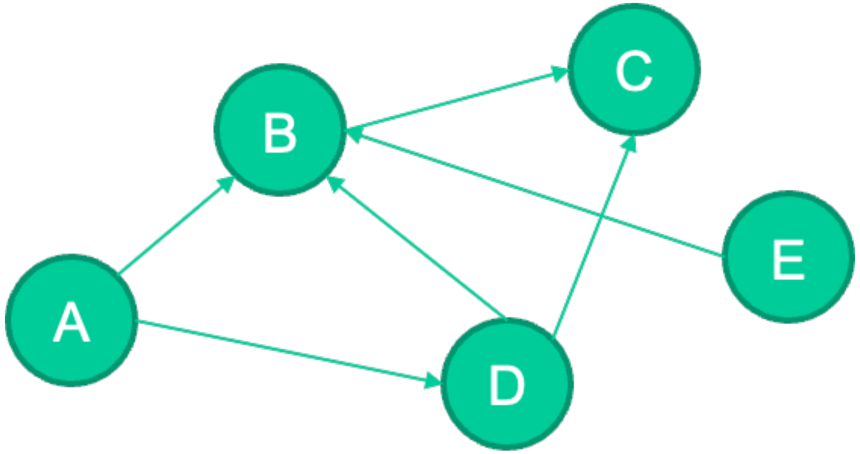
Graphs are not hierarchical or sequential, no requirements for a “root” or “parent/child” relationships between nodes

Kinds of data structures

Sequential, linear structures
(arrays, linked lists)



Hierarchical structures
(trees)



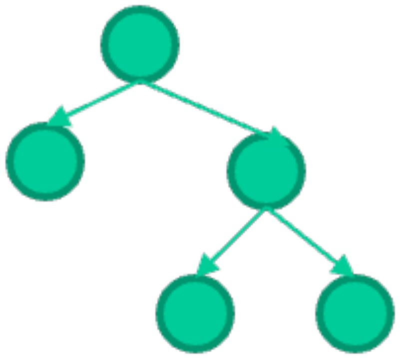
Graphs consist of

- A collection of elements ("nodes" or "vertices")
- A set of connections ("edges" or "links" or "arcs") between pairs of vertices.

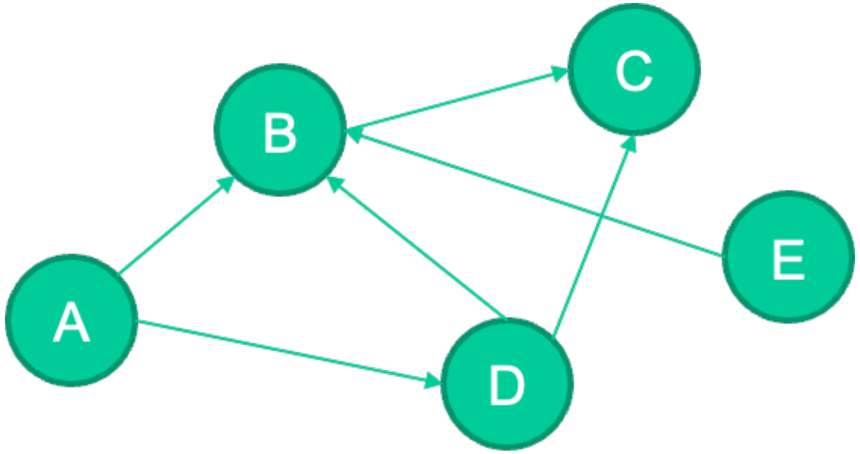
Edges may be directed or undirected
Edges may have weight associated with them

Kinds of data structures

Sequential, linear structures
(arrays, linked lists)



Hierarchical structures
(trees)

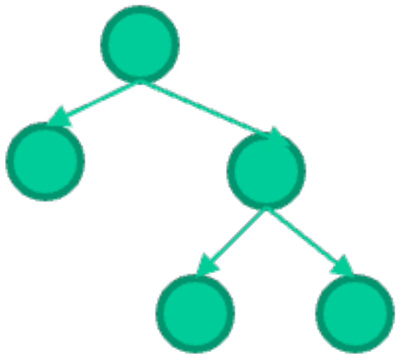


Graphs

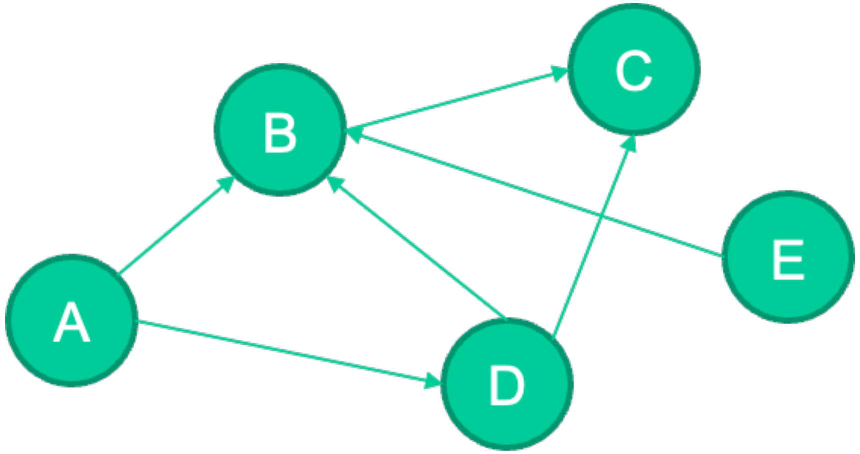
- They consist of both vertices and edges
- They do NOT have an inherent order
- Edges may be weighed or unweighted
- Edges may be directed or undirected
- They may contain cycles

Kinds of data structures

Sequential, linear structures
(arrays, linked lists)



Hierarchical structures
(trees)



Graphs

Which of the following is true about graphs?

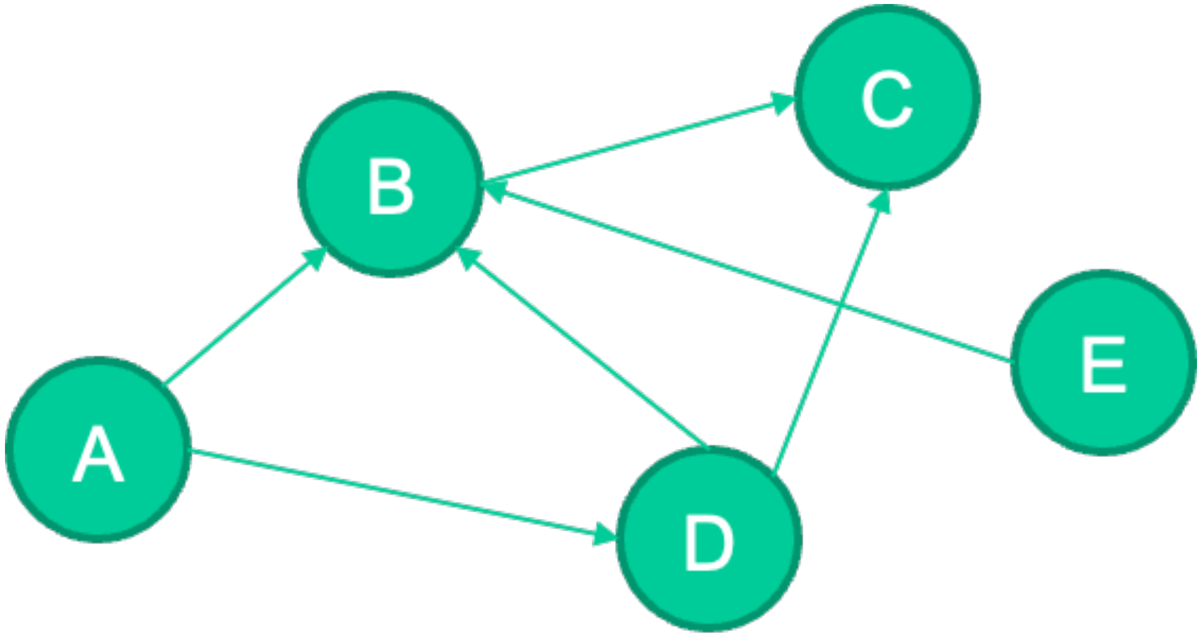
A. A graph can always be represented as a tree

B. A tree can always be represented as a graph

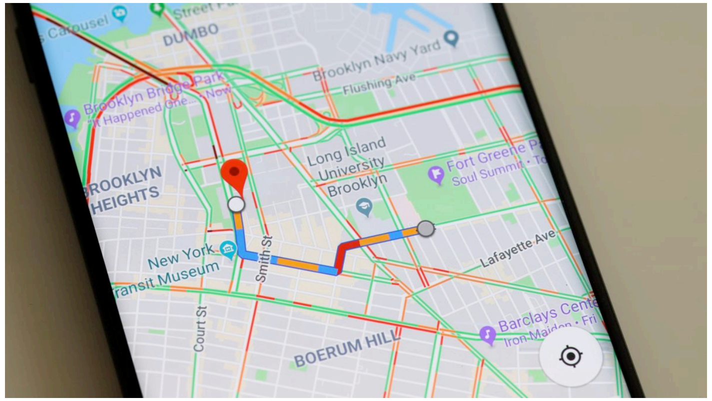
C. Both A and B

D. Neither A or B

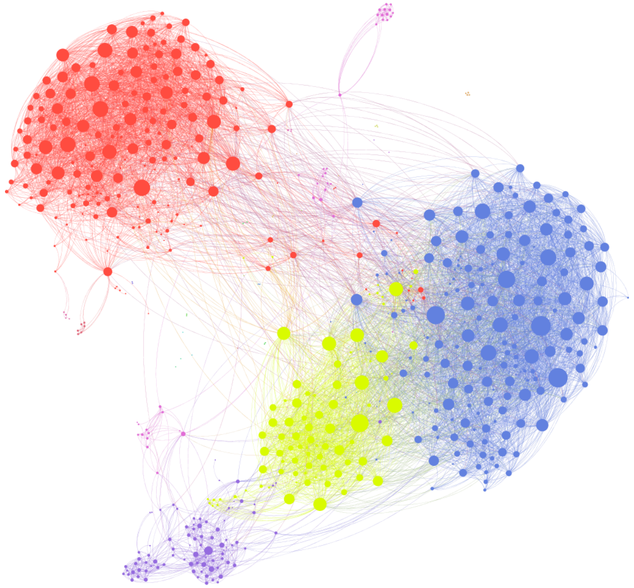
Why Graphs?



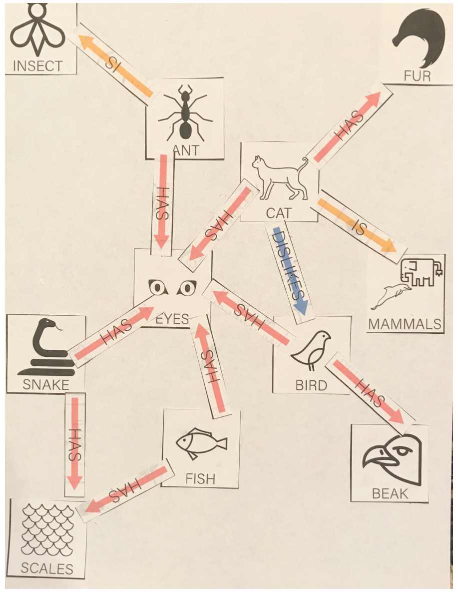
Why Graphs?



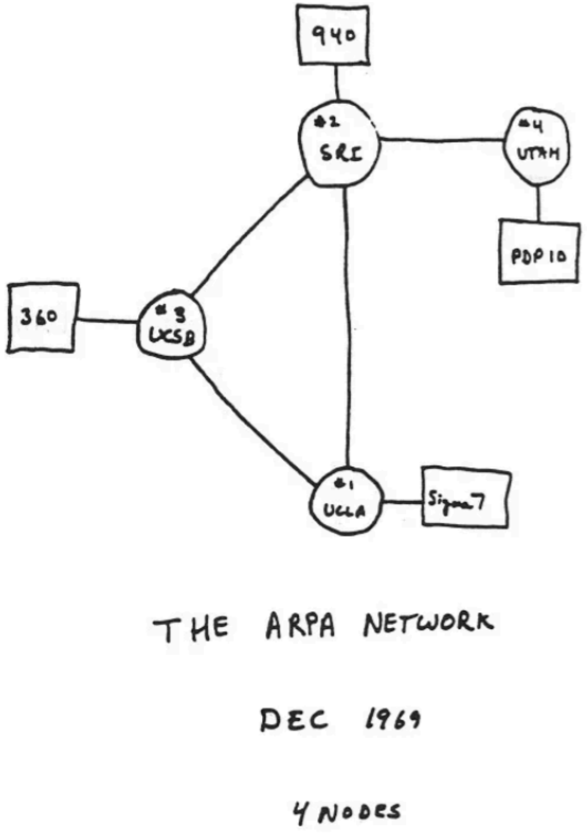
Road networks



Social networks



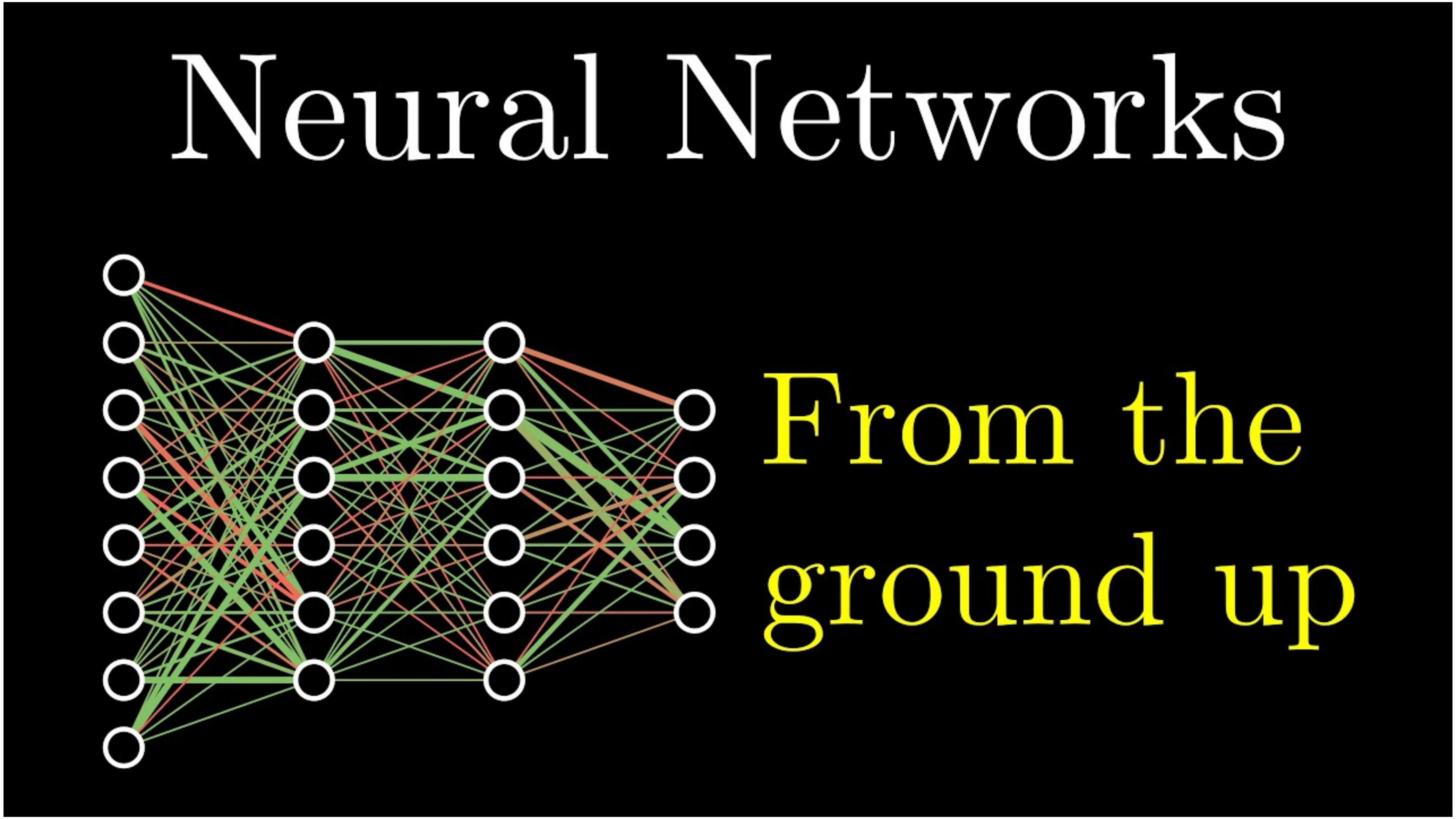
Semantic networks



Computer networks*

Remember: If you can map your problem to a well-known graph problem, it usually means you can solve it fast!

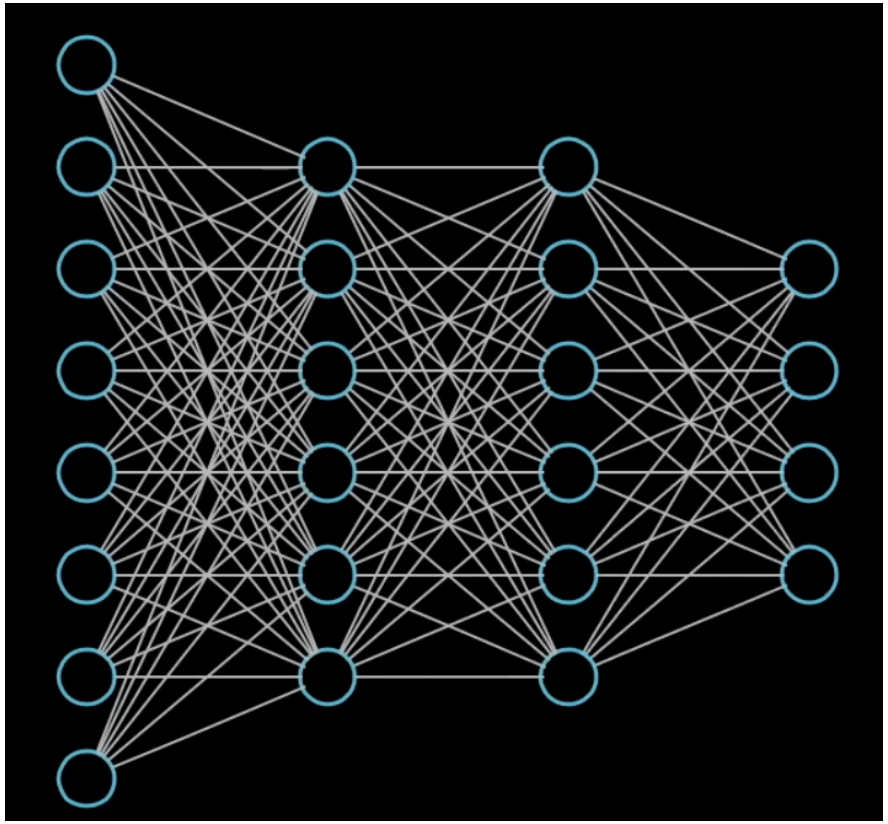
Next assignment: Graph applications to Machine Learning



https://youtu.be/aircAruvnKk?feature=shared   We'll listen to the first 5 minutes

In a graph representing a neural network, which of the following is FALSE? Discuss why in each case.

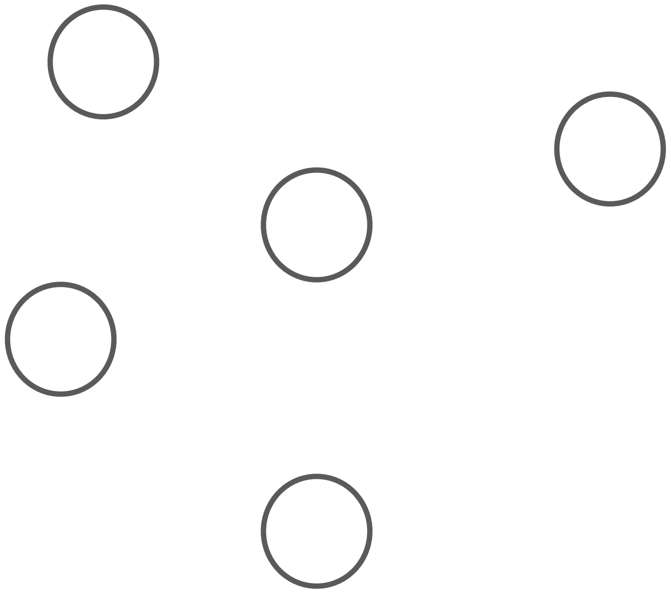
A. Vertices represent neurons
B. Edges represent layers
C. Edges are directed
D. Edges have weights
E. None of the above



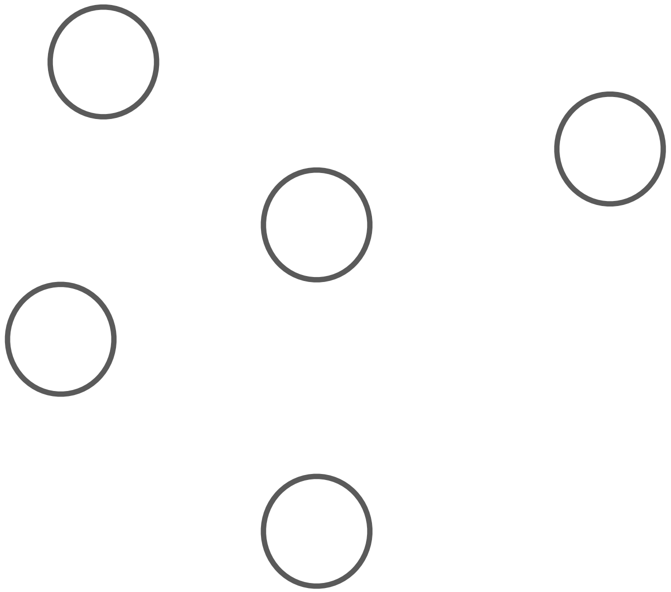
Neural Network

Types of Graphs

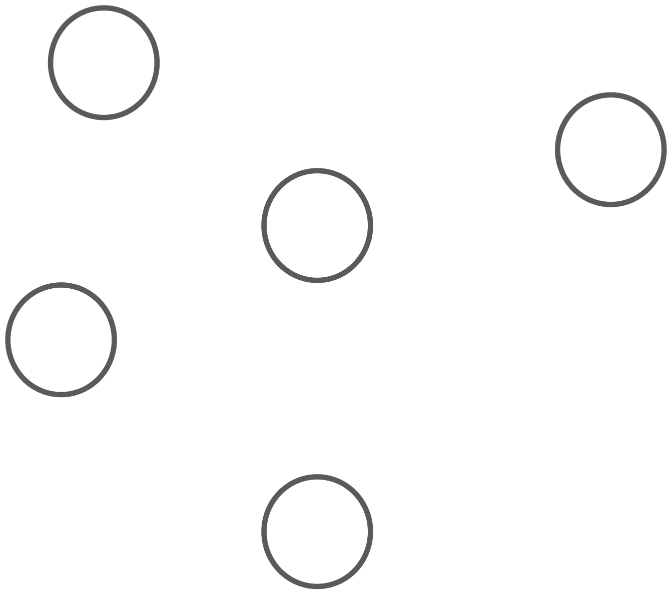
Disconnected



Connected



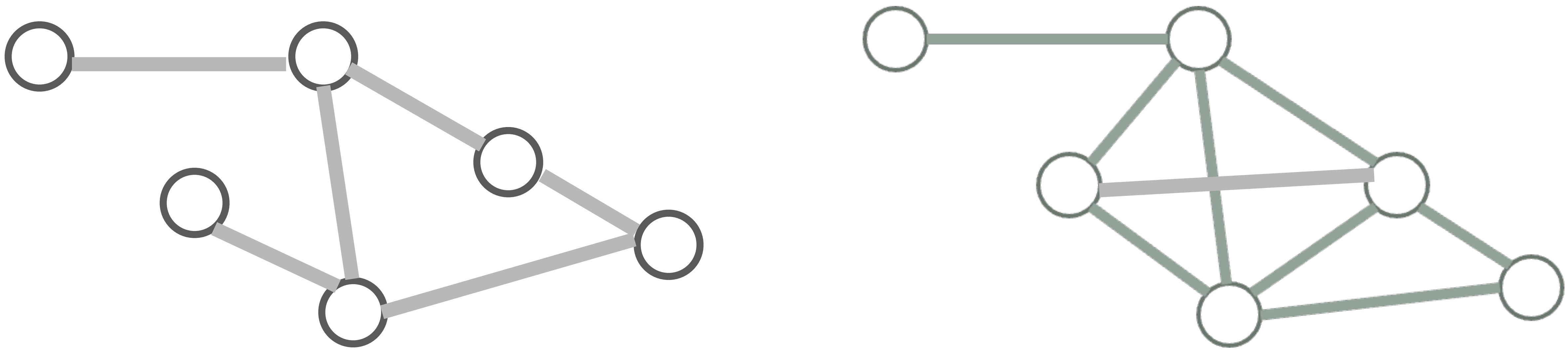
Fully connected



What is minimum and maximum number of edges in a connected undirected graph with n vertices (with no self-loops)?

A. 0 and n
B. (n - 1) and n (n - 1) / 2
C. (n - 1) and n^2
D. (n - 1) and 2^n

Sparse vs. Dense Graphs

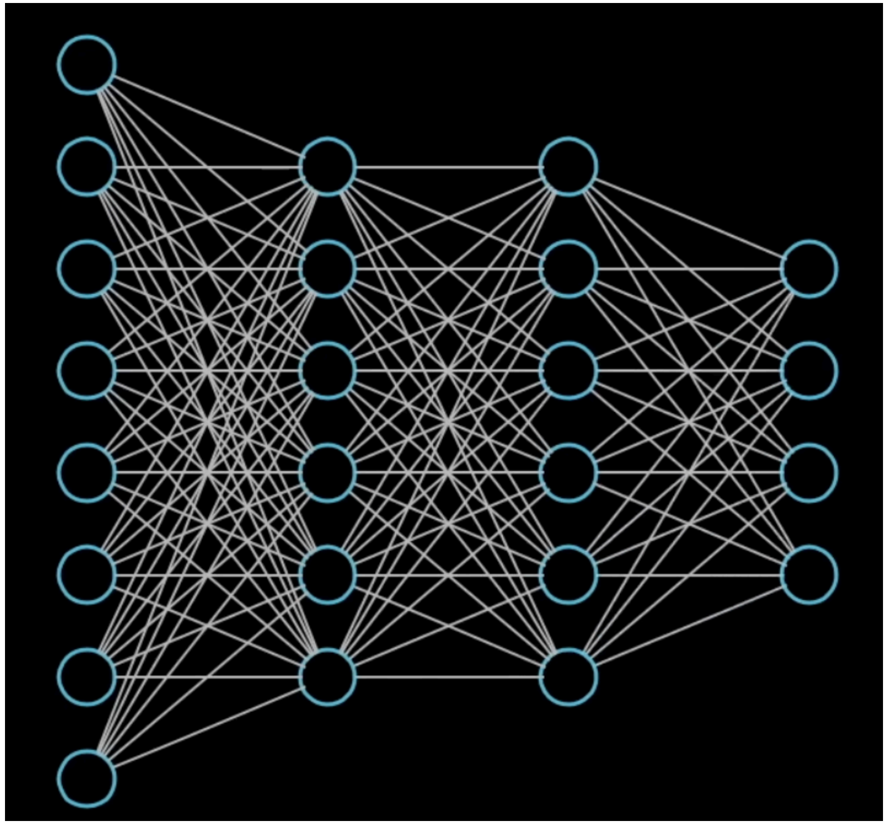


A dense graph is one where $|E|$ is "close to" $|V|^2$.

A sparse graph is one where $|E|$ is "closer to" $|V|$.

Is the neural network a sparse or dense graph?

A. Sparse
B. Dense
C. Can’t say!



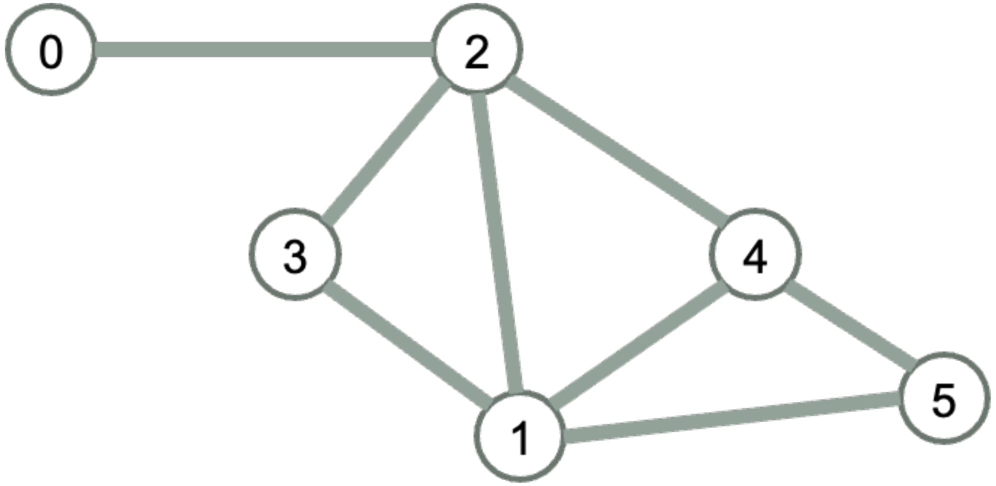
Neural Network

Adjacency Matrix Representation of a Graph

Represent the graph by a n x n binary/integer/float valued adjacency matrix, A

n: number of vertices or |V|

m: number of edges or |E|

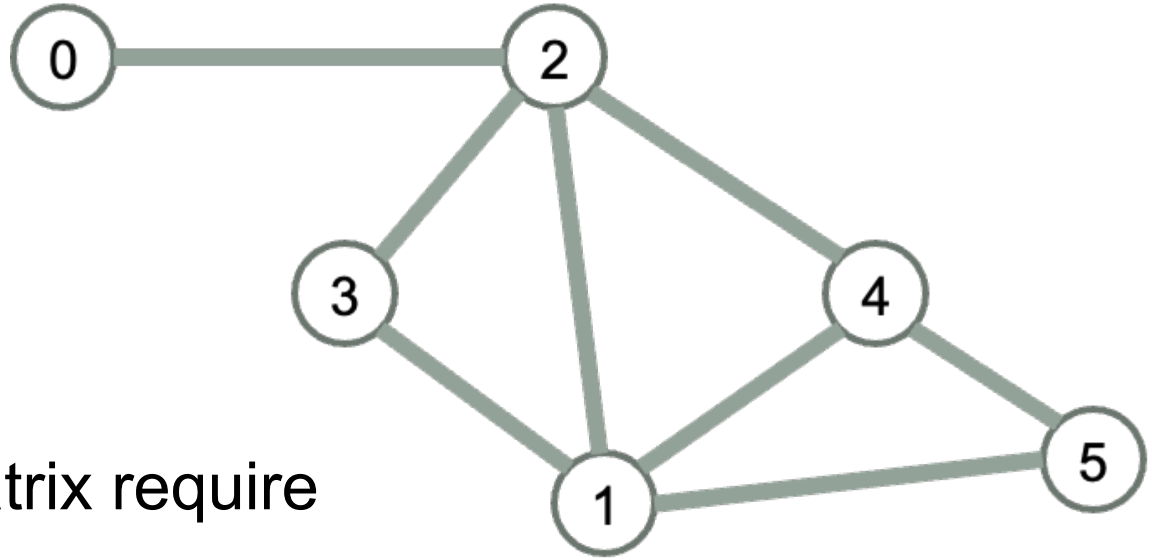


How much space does an adjacency matrix require to represent a graph?

A. O(n)
B. O(m)
C. O(n + m)
D. O(n^2)
E. O(mn)

Adjacency Matrix

Represent the graph by a n x n binary valued adjacency matrix, A
A[i, j] = 1, if there is an edge from i to j



How much space does an adjacency matrix require to represent a graph?

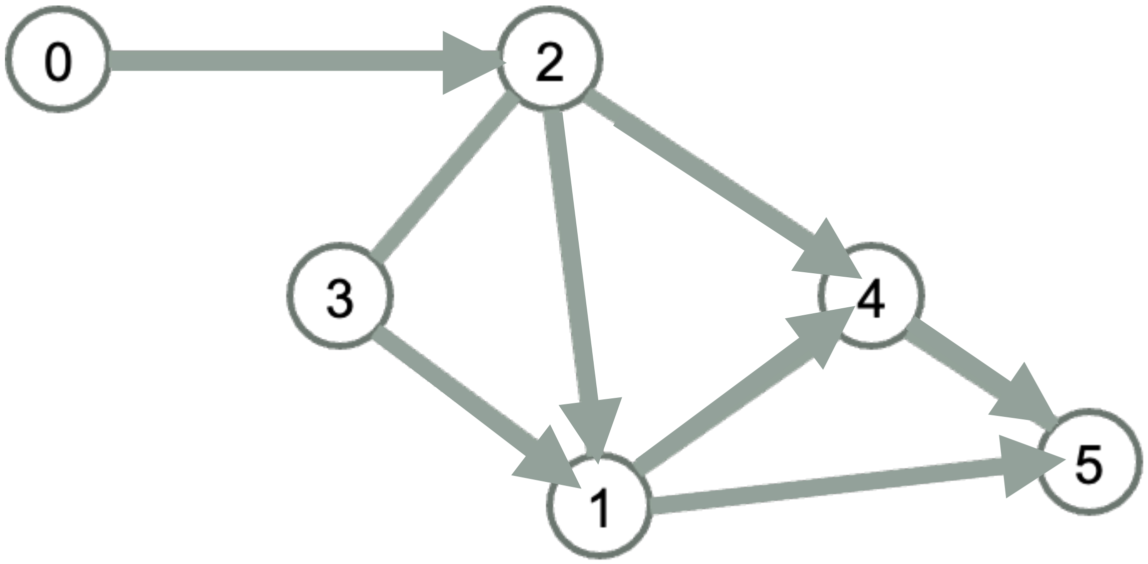
A. O(n)
B. O(m)
C. O(n + m)
D. O(n^2)
E. O(mn)

n: number of vertices or |V|
m: number of edges or |E|

Adjacency List Representation of a Graph

- Vertices and edges stored as lists
- Each vertex points to all its edges

0	
1	
2	
3	
4	
5	



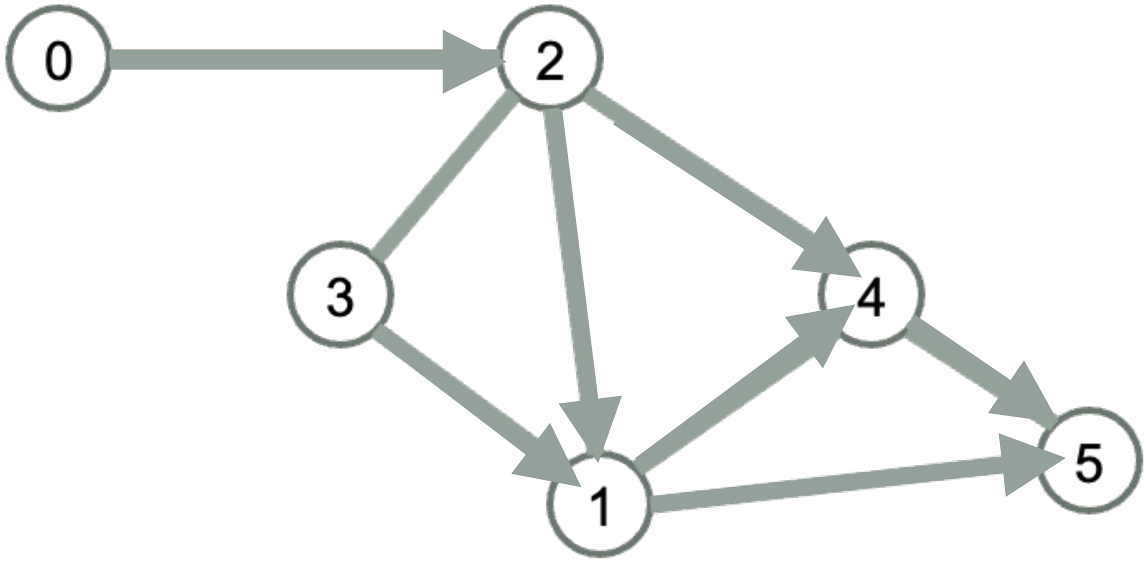
How much space does an adjacency list require to represent a graph?

A. O(n)
B. O(m)
C. O(n + m)
D. O(n^2)
E. O(m.n)

Assume each vertex has a unique id between 0 and 5

```
class graph{

  private:
  _________ adjlist;
};
```



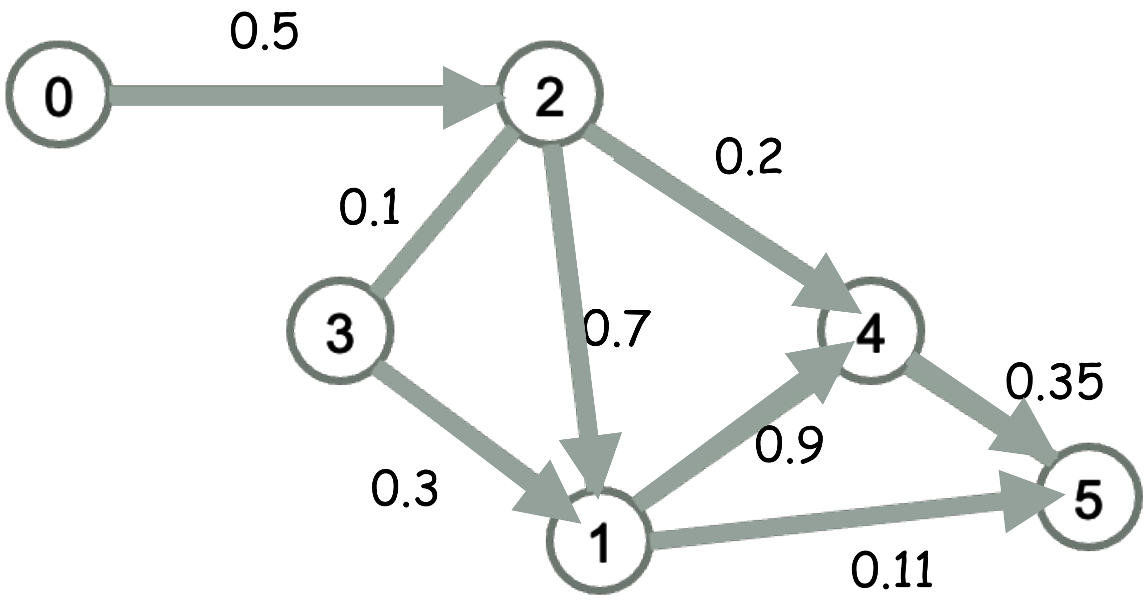
Choose the ADT to represent the adjacency list

A. `vector<int>`
B. `vector<unordered_set<int>>`
C. `list<vector<int>>`
D. `vector<list<int>>`
E. `set<list<int>>`

Adjacency List: Weighted graph

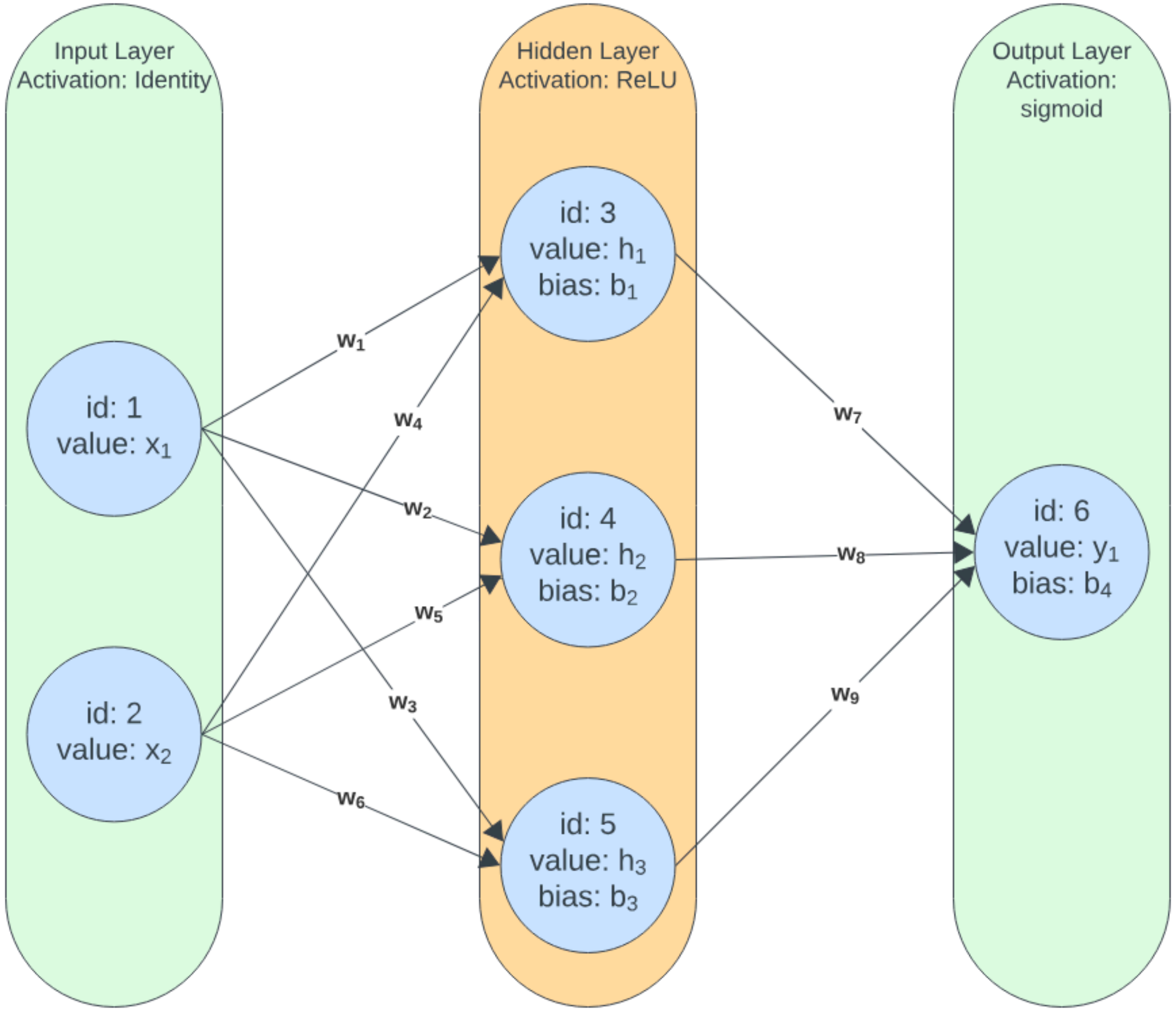
- Vertices and edges stored as lists
- Each vertex points to all its edges

0	
1	
2	
3	
4	
5	



Neural Network structure for upcoming assignment

```
typedef std::vector<std::unordered_map<int, Connection> > AdjList;
```



Understanding the Graph and NeuralNetwork classes

```
typedef std::vector<std::unordered_map<int, Connection> > AdjList;
```

```
class Graph {

    public:
        Graph();
        Graph(int size);
        // Constructors and destructor

        // TODO: graph methods
        void updateNode(int id, NodeInfo n);
        NodeInfo* getNode(int id) const;
        void updateConnection(int v, int u, double w);


    protected:
        // protected to give NeuralNetwork access

        // adjacency list containing weights for edges.
        AdjList adjacencyList;

        // vector storing node info
        std::vector<NodeInfo*> nodes;

        //Other functions
};
```

```
class NeuralNetwork : public Graph {

    public:

        // Constructors and public functions


    private:

        // each index of layers holds a vector which
contains the id's of every node in that layer.
        std::vector<std::vector<int> > layers;

        // contains ids of input nodes
        std::vector<int> inputNodeIds;

        // contains ids of output nodes
        std::vector<int> outputNodeIds;


// since NeuralNetwork inherits from Graph, you can imagine
all of the graph members here as well...

};
```

```
void test_algorithm() {
    cout << "test_algorithm" << endl;
    NeuralNetwork nn(6);

    NodeInfo n0("ReLU", 0, -0.2);
    NodeInfo n1("ReLU", 0, 0.2);
    NodeInfo n2("identity", 0, 0);
    NodeInfo n3("sigmoid", 0, 0.98);
    NodeInfo n4("ReLU", 0, 0.11);
    NodeInfo n5("identity", 0, 0);

    nn.updateNode(0, n0);
    nn.updateNode(1, n1);
    nn.updateNode(2, n2);
    nn.updateNode(3, n3);
    nn.updateNode(4, n4);
    nn.updateNode(5, n5);

    nn.updateConnection(2, 1, 0.1);
    nn.updateConnection(2, 4, 0.2);
    nn.updateConnection(2, 0, 0.3);
    nn.updateConnection(5, 1, 0.4);
    nn.updateConnection(5, 4, 0.5);
    nn.updateConnection(5, 0, 0.6);
    nn.updateConnection(1, 3, 0.7);
    nn.updateConnection(4, 3, 0.8);
    nn.updateConnection(0, 3, 0.9);

    nn.setInputNodeIds({2, 5});
    nn.setOutputNodeIds({3});
}
```

Activity: Draw the final NN by hand

Next, map out the information stored in:

- `nn.nodes`
- `nn.adjacencyList`
- `nn.inputNodeIds`
- `nn.outputNodeIds`