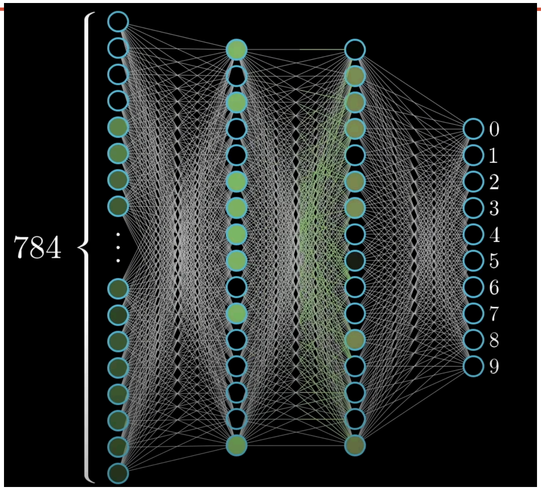
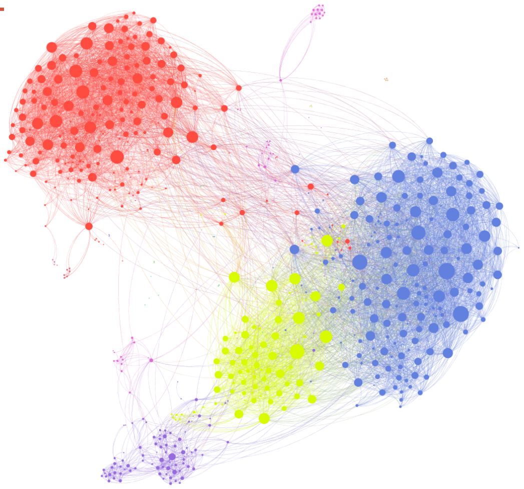
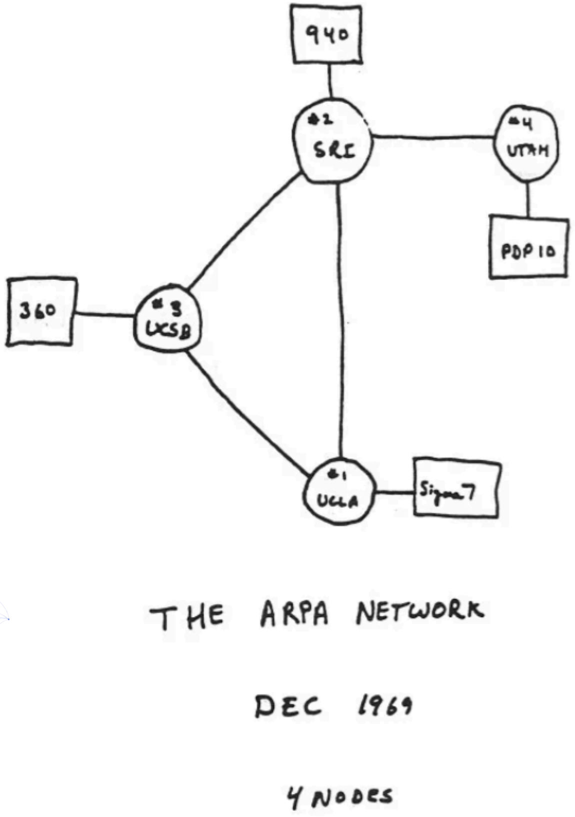
Link to hand out: https://bit.ly/CS24W24-GraphsHandout

GRAPH SEARCH







Adjacency List: Weighted graph

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

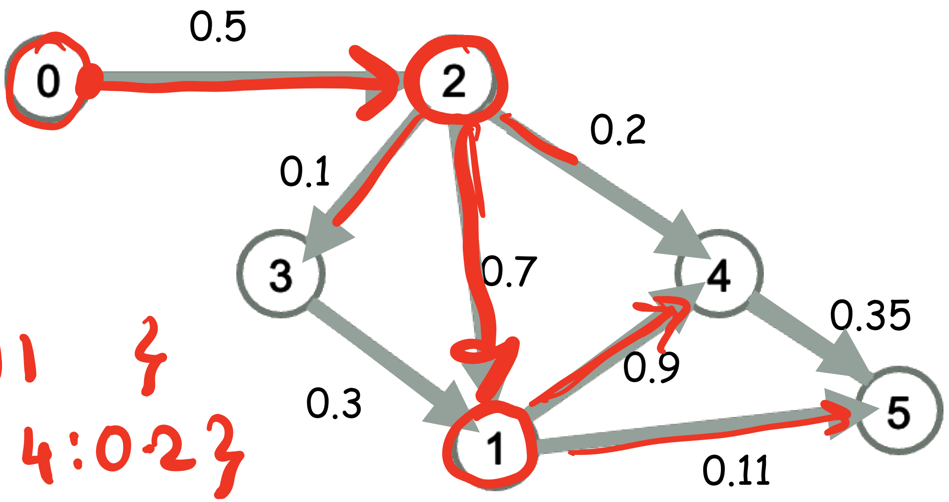
adjacenlist

0	
1	
2	
3	
4	
5	

0 → { 2 : 0.5 }

1 → { 4 : 0.9 , 5 : 0.11 }

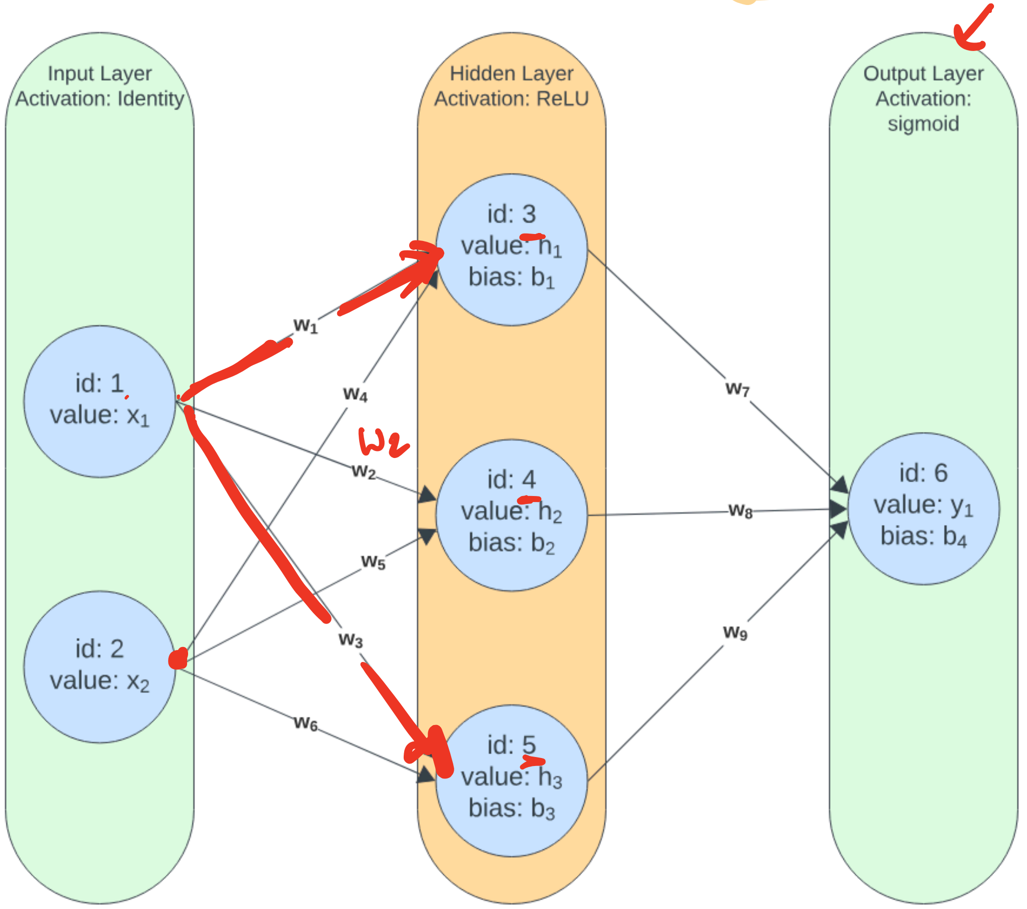
2 → { 3 : 0.1 , 1 : 0.7 , 4 : 0.2 }

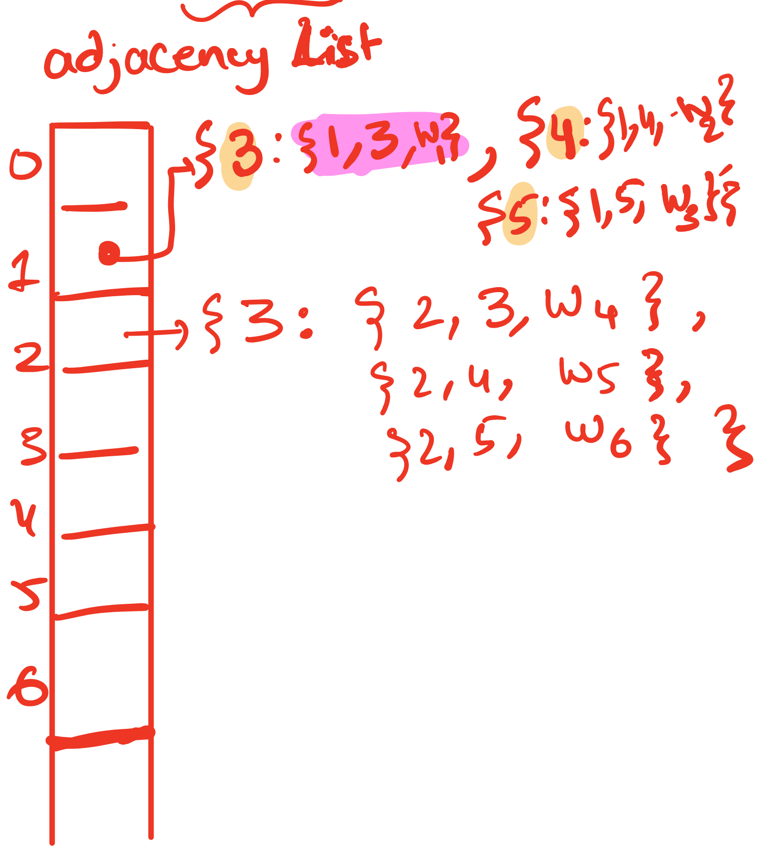


vector< unordered_map <int, double>> adjacency list

Neural Network structure for upcoming assignment

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





Understanding the Graph and NeuralNetwork classes

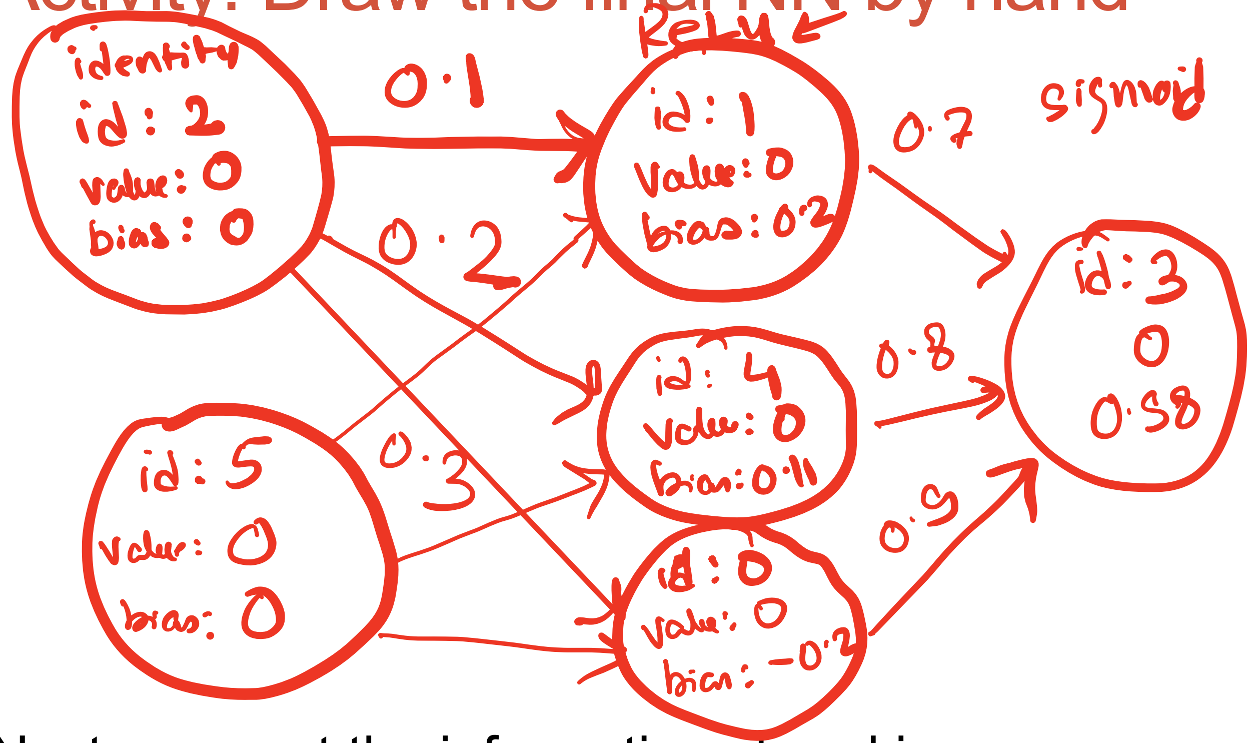
```
typedef std::vector<std::unordered_map<int, Connection> > AdjList;
```

```
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});
}
```

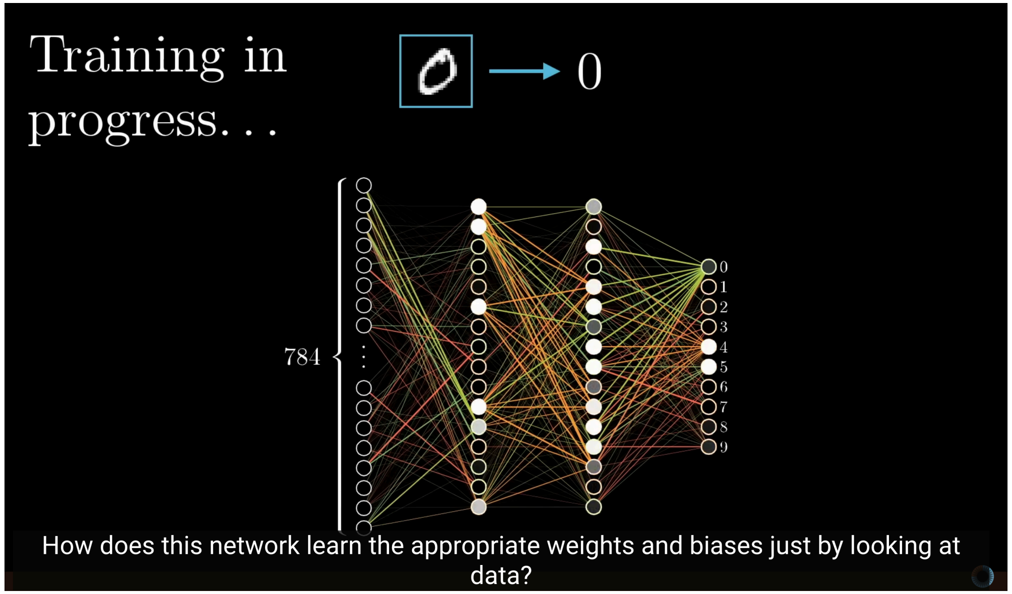
Activity: Draw the final NN by hand



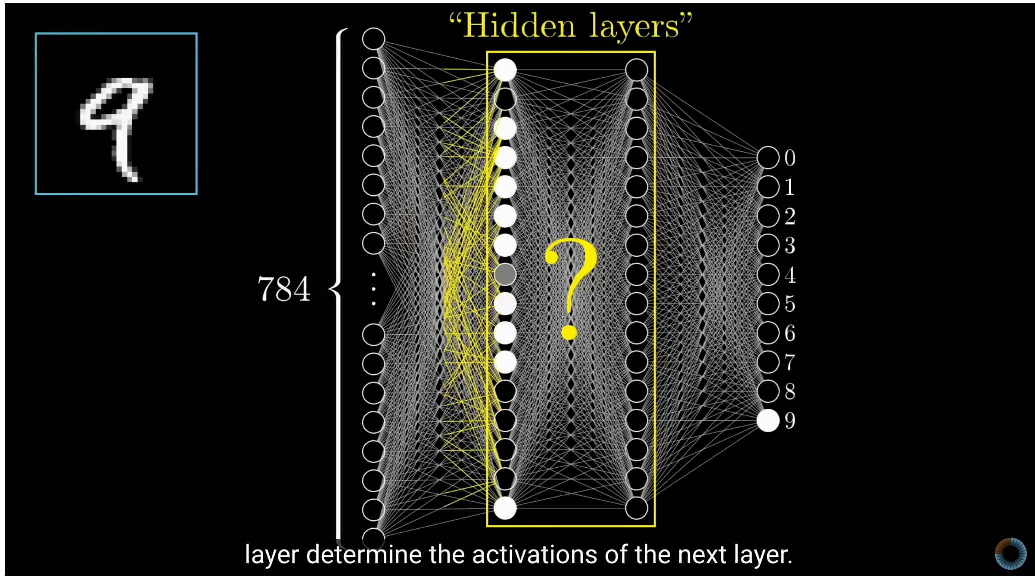
Next, map out the information stored in:

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

How does information flow in a NeuralNetwork ?



How does this network learn the appropriate weights and biases just by looking at data?



layer determine the activations of the next layer.

Training
Learn network parameters

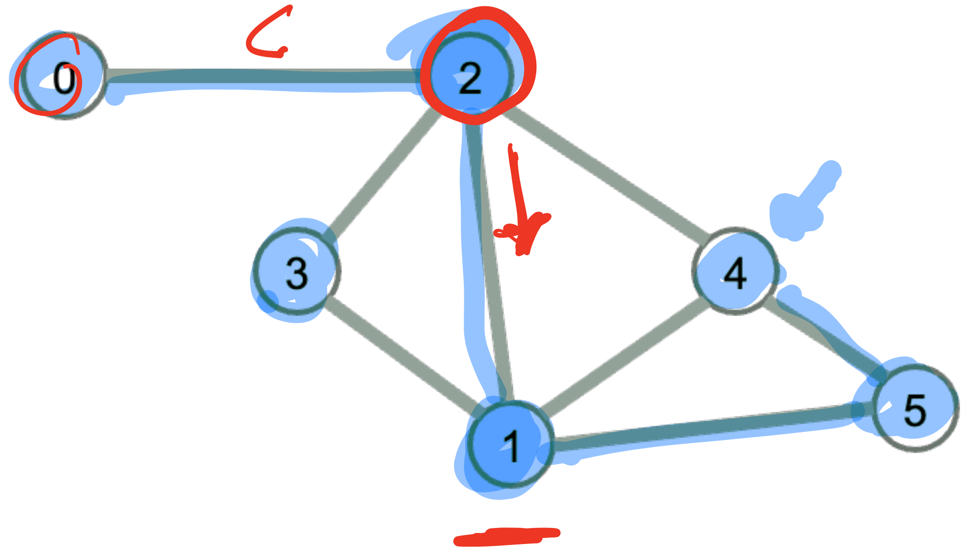
Evaluation/Prediction
Network produces outputs from inputs

Credits: 3Blue1Brown

Graph search: general approach

Starting with a source node

- find everything that can be explored
- don't explore anything twice



Breadth First Traversal: Sketch of Algorithm

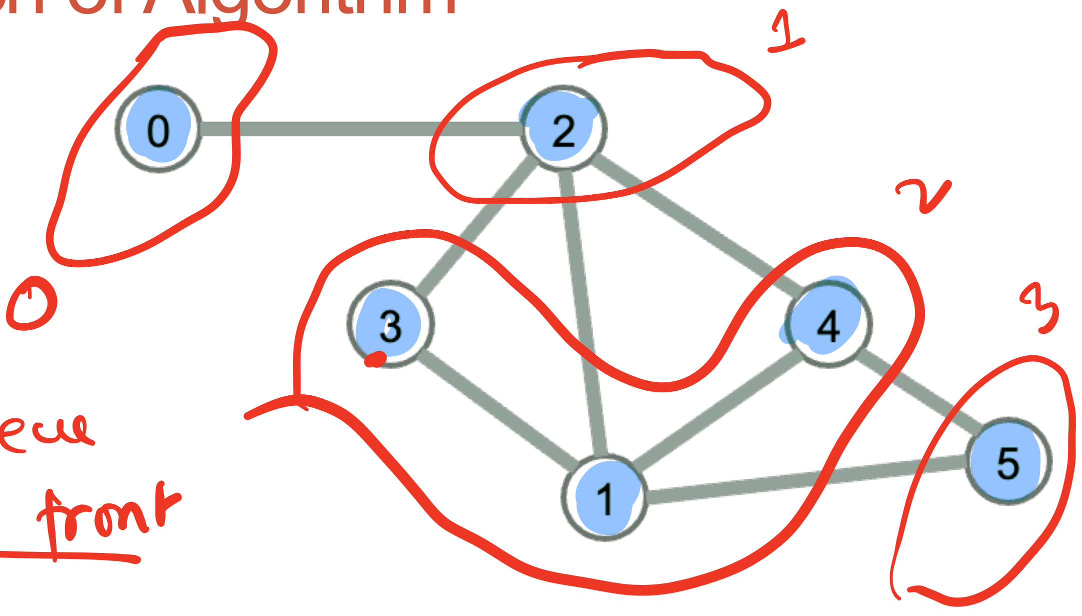
Explore all the nodes reachable from a given node before moving on to the next node to explore

3

back

Queue

front

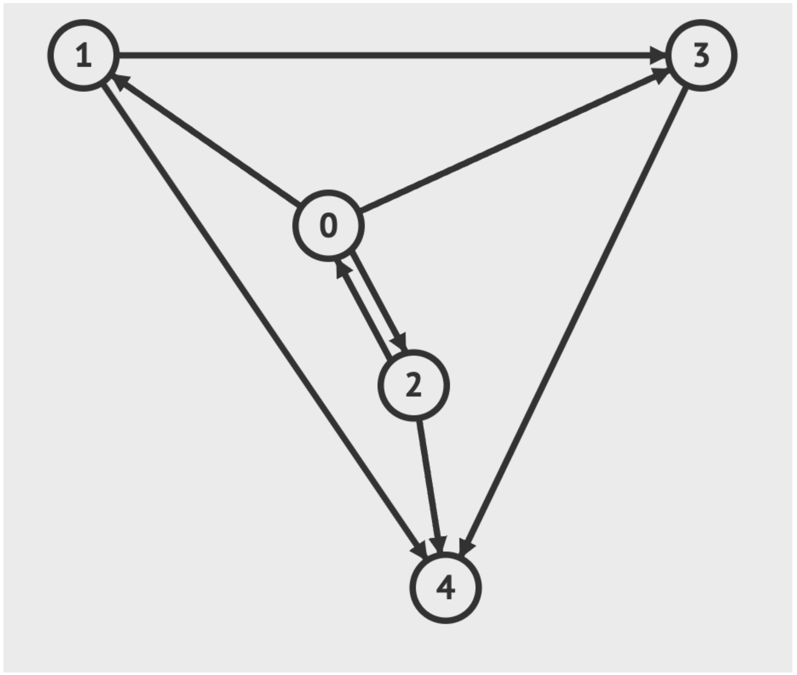


- In general, a search algorithm would explore (or "visit") from a source vertex
  - all the vertices reachable ,
  - never exploring out from the same vertex twice
- How does the Breadth First Search/Traversal algorithm ensure this?

Breadth First Algorithm and Visual Demo

Input: Graph G = (V, E), source ***s***

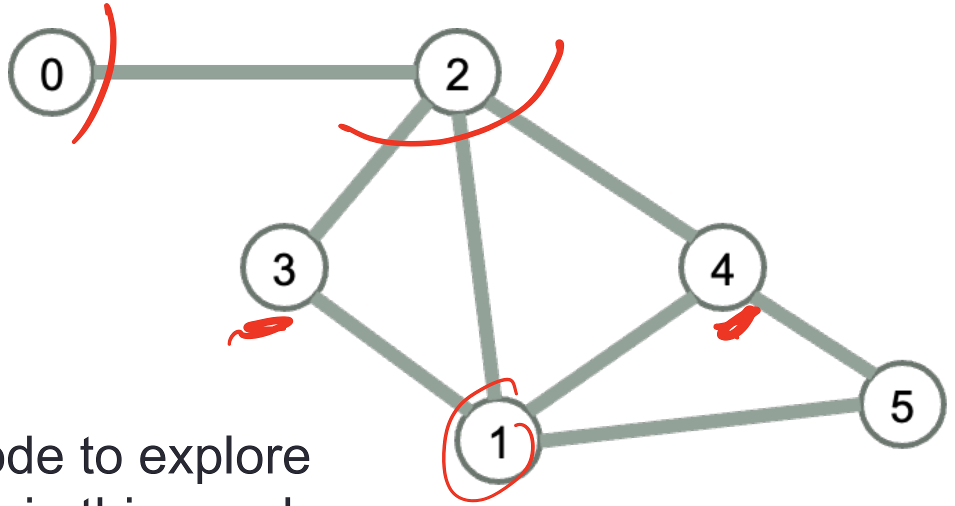
- Start at source ***s***;
- Mark all the vertices as "not visited"
- Mark ***s*** as visited
- push ***s*** into a queue
- while the queue is not empty:
    - pop the vertex ***u*** from the front of the queue
    - for each of ***u***'s neighbor (***v***)
        - If ***v*** has not yet been visited:
            - Mark ***v*** as visited
            - Push ***v*** in the queue



https://visualgo.net/en/dfsbfs

Graph search: breadth first (BFS)

Explore all the nodes reachable from a given node before moving on to the next node to explore



Assume BFS chooses the lower number node to explore first, in what order does BFS visit the nodes in this graph

A. 0, 1, 2, 3, 4, 5
B. 0, 1, 3, 2, 4, 5
C. 0, 2, 3, 1, 4, 5
[✓ D. 0, 2, 1, 3, 4, 5]
E. Something else

There are n rooms labeled from 0 to n - 1 and all the rooms are locked except for room 0. Your goal is to visit all the rooms. However, you cannot enter a locked room without having its key.

When you visit a room, you may find a set of distinct keys in it. Each key has a number on it, denoting which room it unlocks, and you can take all of them with you to unlock the other rooms.

Given an array rooms where rooms[i] is the set of keys that you can obtain if you visited room i, return true if you can visit all the rooms, or false otherwise.

https://leetcode.com/problems/keys-and-rooms/description/

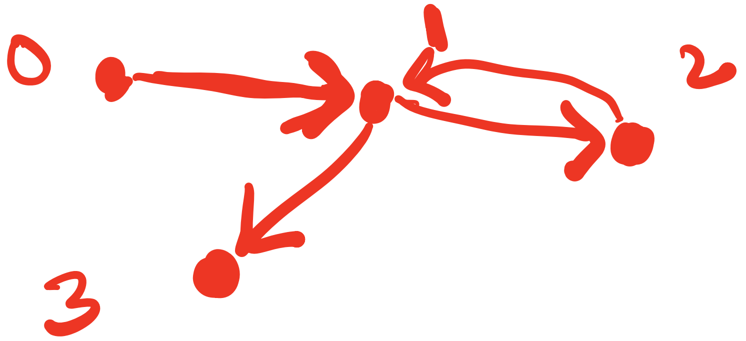
```
Input: rooms = [[1],[2,3],[1],[]]
                 0     1    2   3
Output: true
Explanation:
We visit room 0 and pick up key 1.
We then visit room 1 and pick up keys 2 and 3.
We then visit room 2 and pick up key 1.
We then visit room 3.
Since we were able to visit every room, we return true.
```

0

Can we cast this as a graph exploration problem?

A. Yes

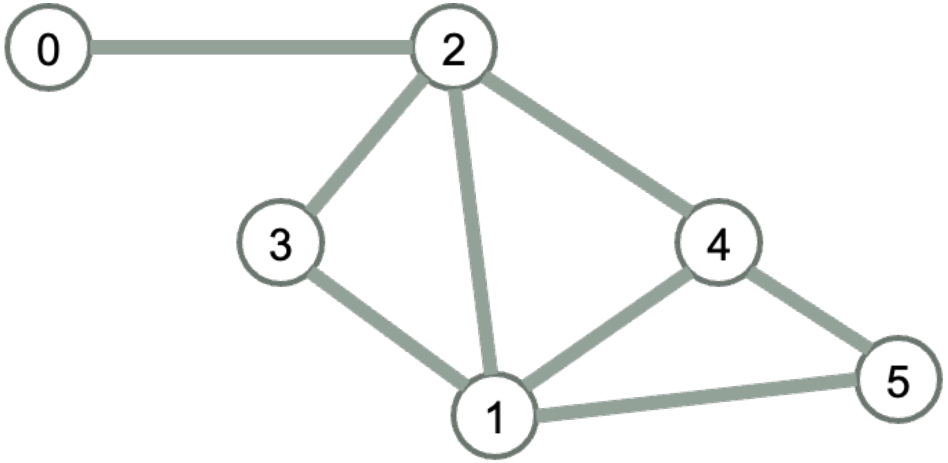
B. No



https://leetcode.com/problems/keys-and-rooms/description/

Implement these functions in your handout

```
class graph{
  public:
     graph(int n = 0) { // n is the number of vertices
       ________________________________
     }
A    void addEdge(int from, int to);
B    bool hasEdge(int i, int j) const;
C    vector<bool> bfs(int source) const;
D    bool isValidPath(const vector<int> & path) const; // returns true if the input path exists
E    bool isReachable(int source, int dest) const;  // returns true if a path exists from source to dest

  private:
     vector<________________________> adjList;
};
```



Link to hand out:https://bit.ly/CS24W24-GraphsHandout