**Graphs**

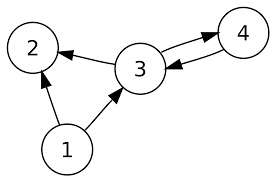
Graphs contain vertices and edges.

A vertex is also known as a node.

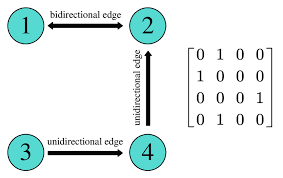
Edges is also known as connections.



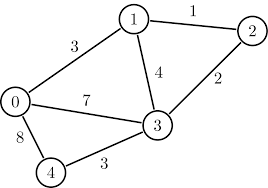
Directed Graph – the arrows point to the next vertex.



Bi-directional Graph – the arrows point both directions.



Weighted Edges – the paths from one vertex to another have a value.

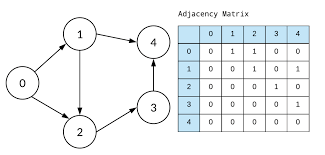


In this example, the fastest way to get from 1 🡪 3 is to go directly. However, the weight of this path would be 4.

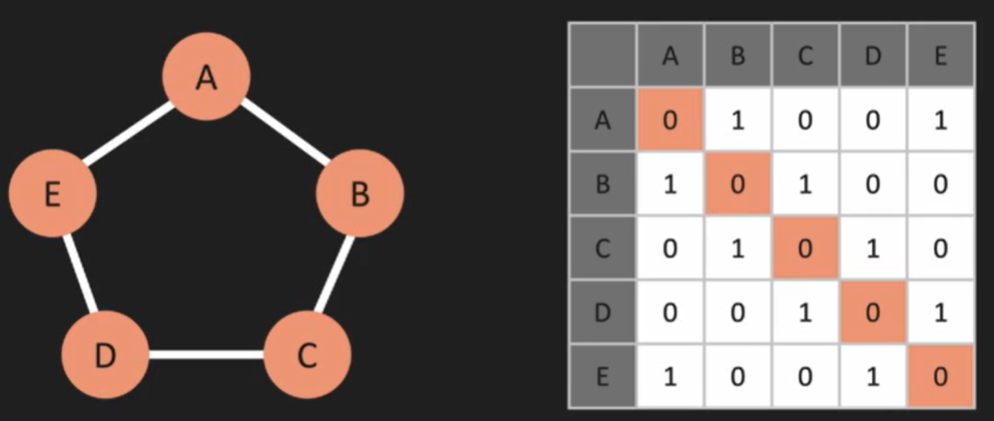
If the go from 1 🡪 2 🡪 3, the weight would be 3 which is better.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

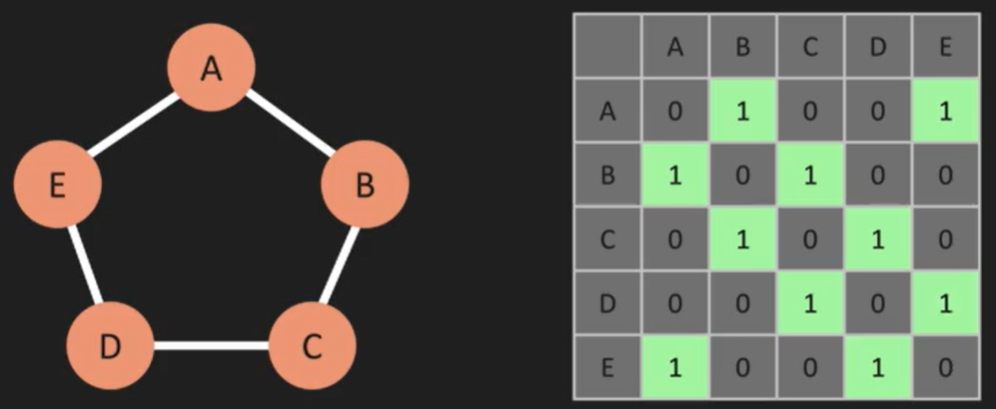
**Adjacency Matrix**



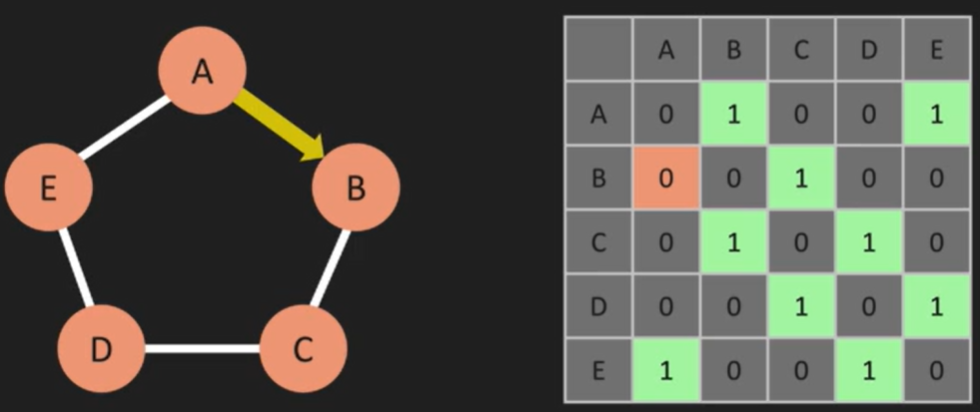
There is always 0’s in a 45 degree line in an adjacency matrix.



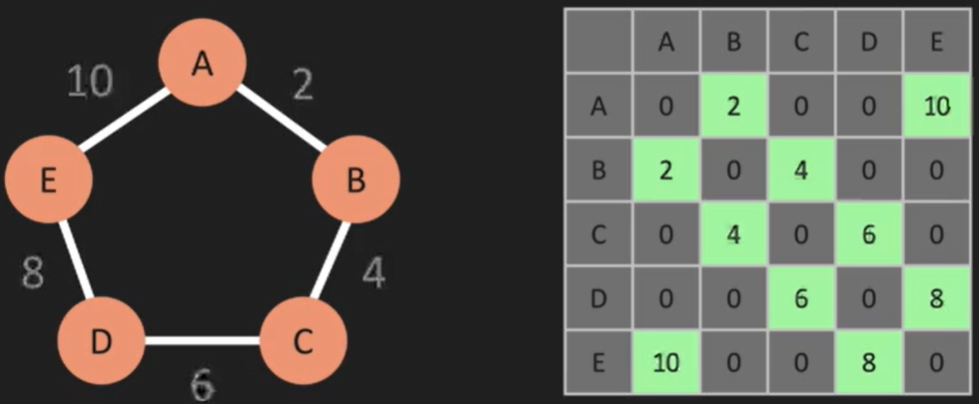
Graph with bi-directional connections are symmetrical.



If all the connections are not bidirectional, then the matrix is no longer symmetrical.



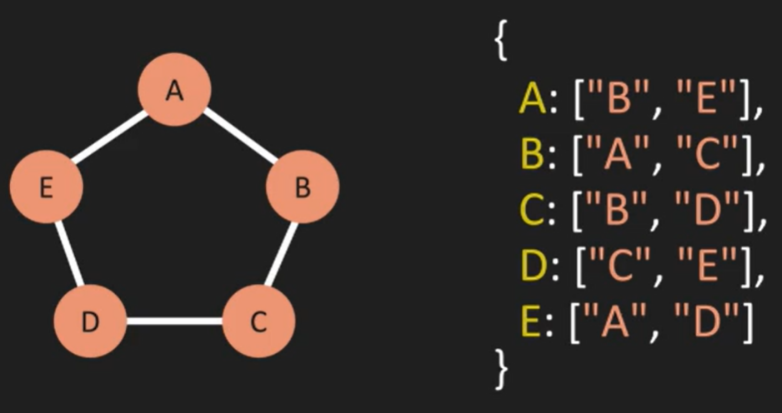
Weighted Graph



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Adjacency List**

Store the vertex and edges in an Object.



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Big O**

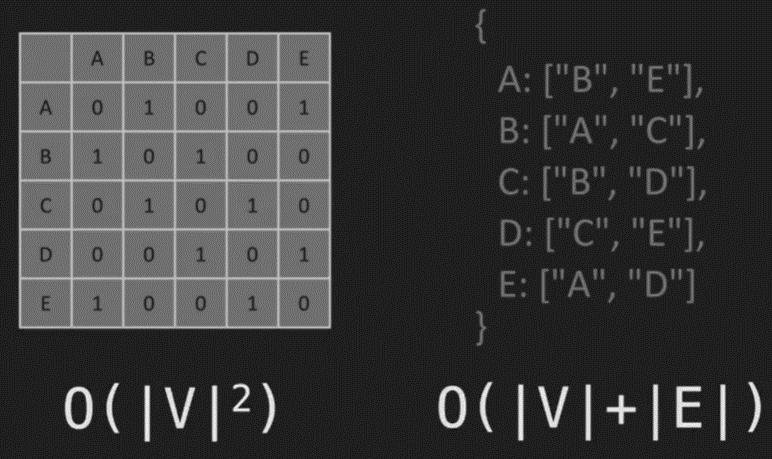
Adjacency Matrix vs Adjacency List

Space Complexity

Adjacency Matrix – We store connections of each vertices and edges in addition to storing any vertices and edges that are not connected.

Adjacency List – We only store the vertices and the connections to other vertices.

Adjacency List is more efficient.

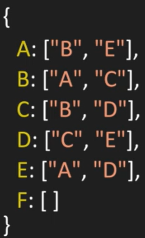


Adjacency Matrix – the number of vertices^2 (O(|V|^2). 5 vertices would mean that we need to store 5^2 = 25.

Adjacency List – we only store the vertices and edges.

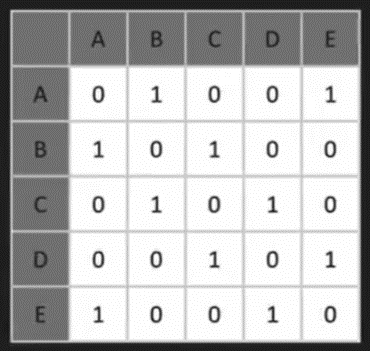
Add Vertex

Adjacency List – we just need to add the vertex to the list.

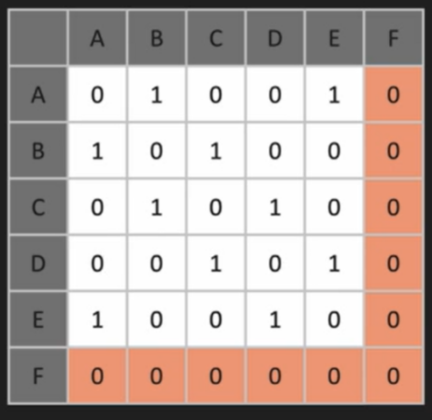


Adjacency Matrix – we need to essentially create a new matrix.

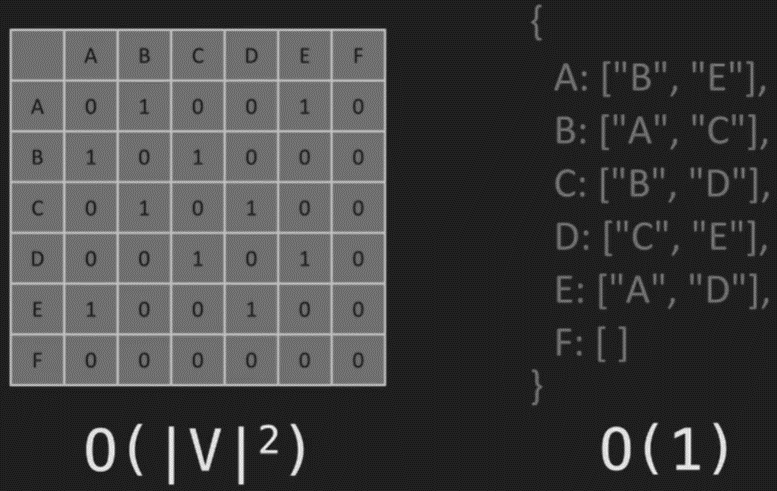
We need to change the matrix from:



to this:

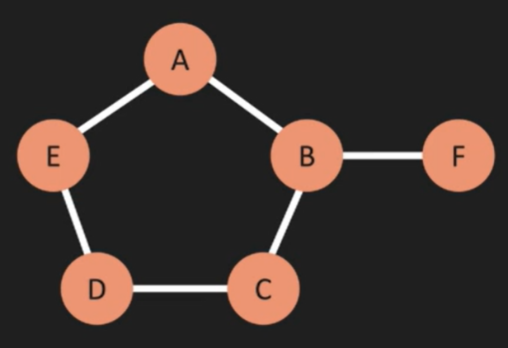


The time complexity is better with an adjacency list when adding a vertex.

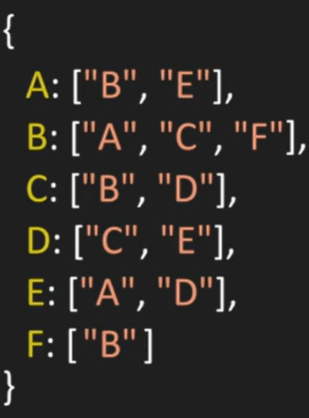


Connecting Vertices

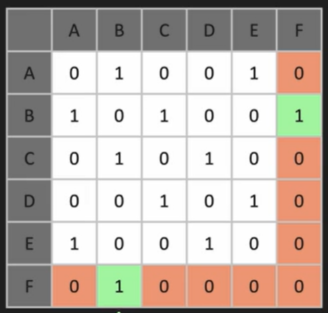
Creating an edge between B and F:



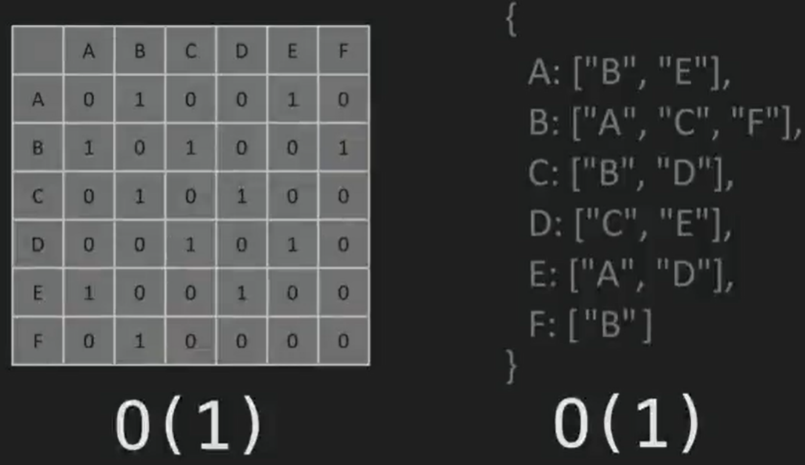
Adjacency List



Adjacency Matrix



O(1) for both

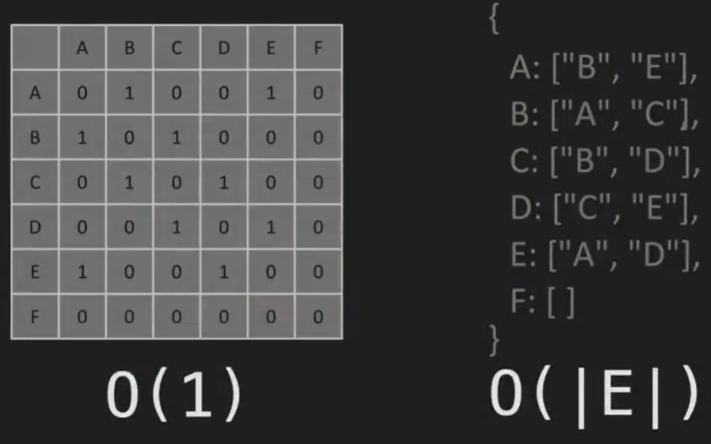


Remove Edge

Removing F:

Adjacency List – we need to iterate through each vertex array[] and remove the vertex which can be long depending on the number of elements in the array.

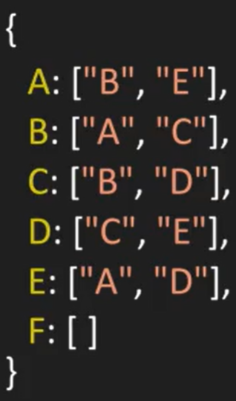
Adjacency Matrix – we only need to go to the vertex we want to remove (F) and set the value to zero.



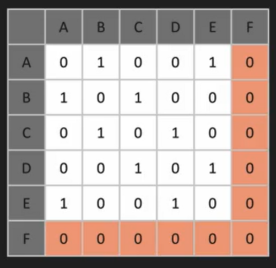
Remove Vertex

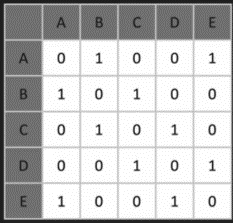
Remove F

Adjacency list – we would remove the vertex then iterate through the list and at each vertex, iterate through that list to remove any connection to the removed vertex.

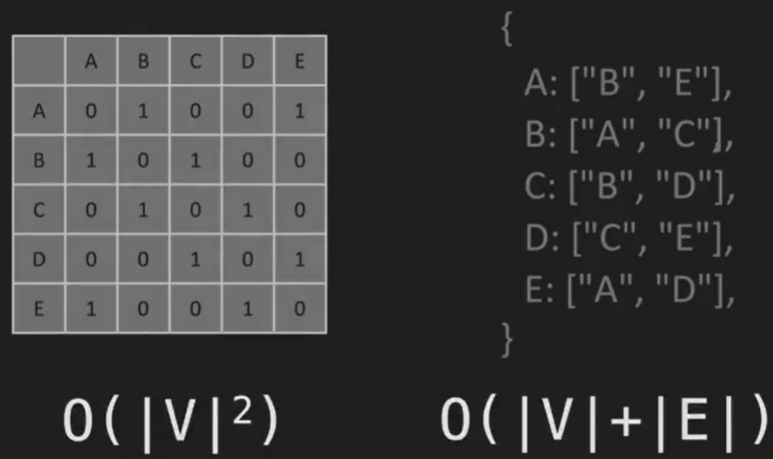


Adjacency matrix – we would essentially create a new matrix that does not include the removed vertex.





Adjacency list is more efficient.



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

class Graph {

    constructor() {

        this.adjacencyList = {};

    }

    printGraph() {

        if (Object.keys(this.adjacencyList).length !== 0) {

            console.log("{");

            for (const [key, value] of Object.entries(this.adjacencyList)) {

                console.log(" ", `${key}: ${value}`);

            }

            console.log("}");

        } else {

            console.log("{}");

        }

    }

    addVertex(vertex) {

        if (!this.adjacencyList[vertex]) {

            this.adjacencyList[vertex] = [];

            return true;

        }

        return false;

    }

    addEdge(vertex1, vertex2) {

        if (this.adjacencyList[vertex1] && this.adjacencyList[vertex2]) {

            this.adjacencyList[vertex1].push(vertex2);

            this.adjacencyList[vertex2].push(vertex1);

            return true;

        }

        return false;

    }

    removeEdge(vertex1, vertex2) {

        if (this.adjacencyList[vertex1] && this.adjacencyList[vertex2]) {

            this.adjacencyList[vertex1] = this.adjacencyList[vertex1]

                .filter(v => v !== vertex2);

            this.adjacencyList[vertex2] = this.adjacencyList[vertex2]

                .filter(v => v !== vertex1);

            return true;

        }

        return false;

    }

    removeVertex(vertex) {

        if (!this.adjacencyList[vertex]) return undefined;

        while (this.adjacencyList[vertex].length) {

            let temp = this.adjacencyList[vertex].pop();

            this.removeEdge(vertex, temp);

        }

        delete this.adjacencyList[vertex];

        return this;

    }

}