**Graph**

*Definition:* A graph data structure consists of a finite (and possibly mutable) set of vertices or nodes or points, together with a set of unordered pairs of these vertices for an undirected graph or a set of ordered pairs for a directed graph.

*Essential Graph Terms:*

**Vertex** – a node

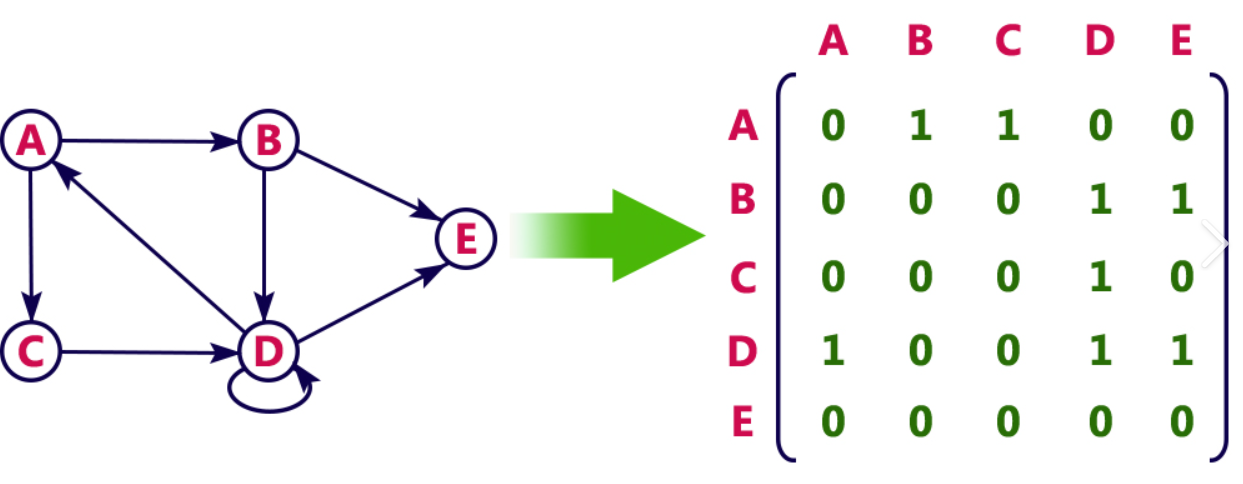
**Edge** – connection between nodes

**Weighted/Unweighted** – values assigned to distances between vertices

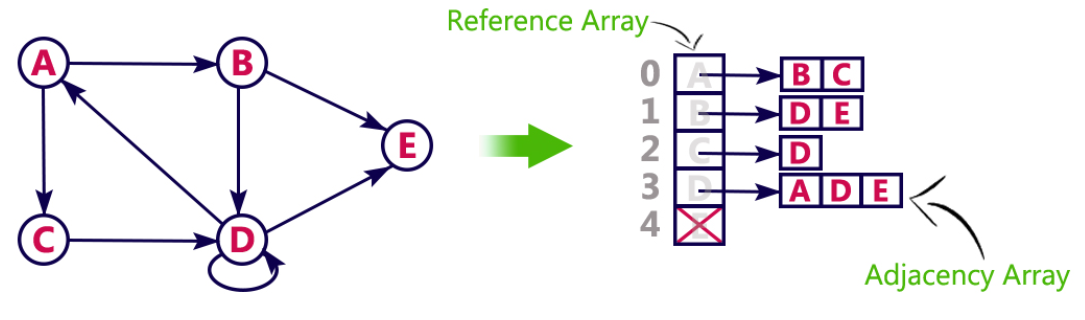
**Directed/Undirected** – directions assigned to distanced between vertices

*Storing Graphs:*

Adjacent Matrix:



Adjacent List (or Array):



*Big O Comparison:* |V| - number of vertices, |E| - number of edges

|  |  |  |
| --- | --- | --- |
| OPERATION | ADJACENT LIST | ADJACENT MATRIX |
| Add Vertex | O(1) | O(|V^2|) |
| Add Edge | O(1) | O(1) |
| Remove Vertex | O(|V| + |E|) | O(|V^2|) |
| Remove Edge | O(|E|) | O(1) |
| Query | O(|V| + |E|) | O(1) |
| Storage | O(|V| + |E|) | O(|V^2|) |
|  | * **Can take up less space (in sparse graphs)** * **Faster to iterate over all edges** * **Can be slower to lookup specific edge** | * **Takes up more space (in sparse graphs)** * **Slower to iterate over all edges** * **Faster to lookup specific edge** |

*Pseudocode for Graph:*

addVertex:

* This function accepts a name of a vertex
* It should add a key to the adjacent list with the name of the vertex and set its value to be an empty array

addEdge:

* This function accepts two vertices
* The function should find in the adjacent list the key of vertex1 and push vertex2 to the array

removeEdge:

* This function accepts two vertices
* The function should reassign the key of vertex1 to be an array that does not contain vertex2
* The function should reassign the key of vertex2 to be an array that does not contain vertex1

removeVertex:

* This function should accept a vertex to remove
* The function should loop as long as there are any other vertices in the adjacent list for that vertex
* Inside of the loop, call our removeEdge() with the vertex we are removing and any values in the adjacent list for that vertex
* Delete the key in the adjacent list for that vertex

*Solution for Graph:*

class Graph {

constructor() {

this.adjacentList = {};

}

addVertex(vertex) {

if (!this.adjacentList[vertex]) {

this.adjacentList[vertex] = [];

}

}

addEdge(v1, v2) {

this.adjacentList[v1].push(v2);

this.adjacentList[v2].push(v1);

}

removeEdge(v1, v2) {

this.adjacentList[v1] = this.adjacentList[v1].filter(

v => v !== v2;

);

this.adjacentList[v2] = this.adjacentList[v2].filter(

v => v !== v1;

);

}

removeVertex(vertex) {

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

const adjacentVertex = this.adjacentList[vertex].pop();

this.removeEdge(vertex, adjacentVertex);

}

delete this.adjacentList[vertex];

}

}