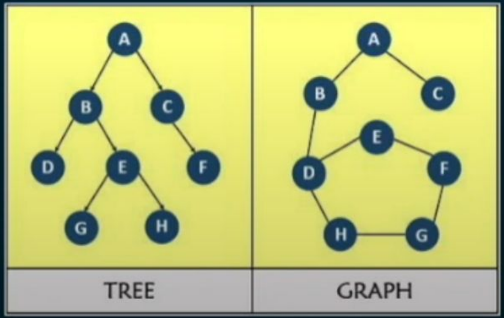
Algorithms: Graph Data Structure with 3 Javascript Implementations

What is a graph?

- A data structure made up of *nodes* or *vertices* and *edges* or the connections between nodes
- Typically, a visualization of a graph will be of nodes represented by circles and edges as lines between the circles

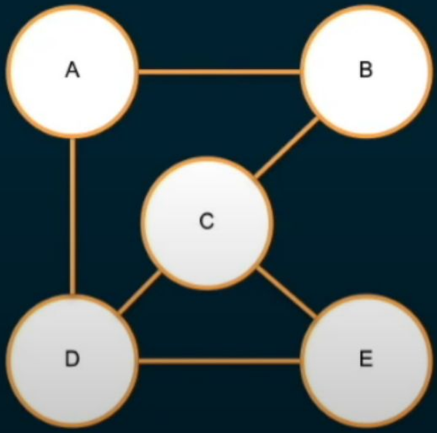
Trees: A special kind of graph

- Trees are just a special kind of graph with one root and only one unique path between any two nodes
- A graph can go beyond that and have any number of root elements and multiple paths between nodes



coderbyte

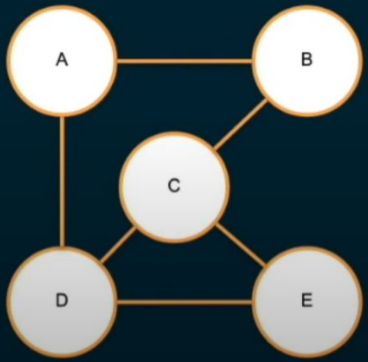
Vertex list + Edge list



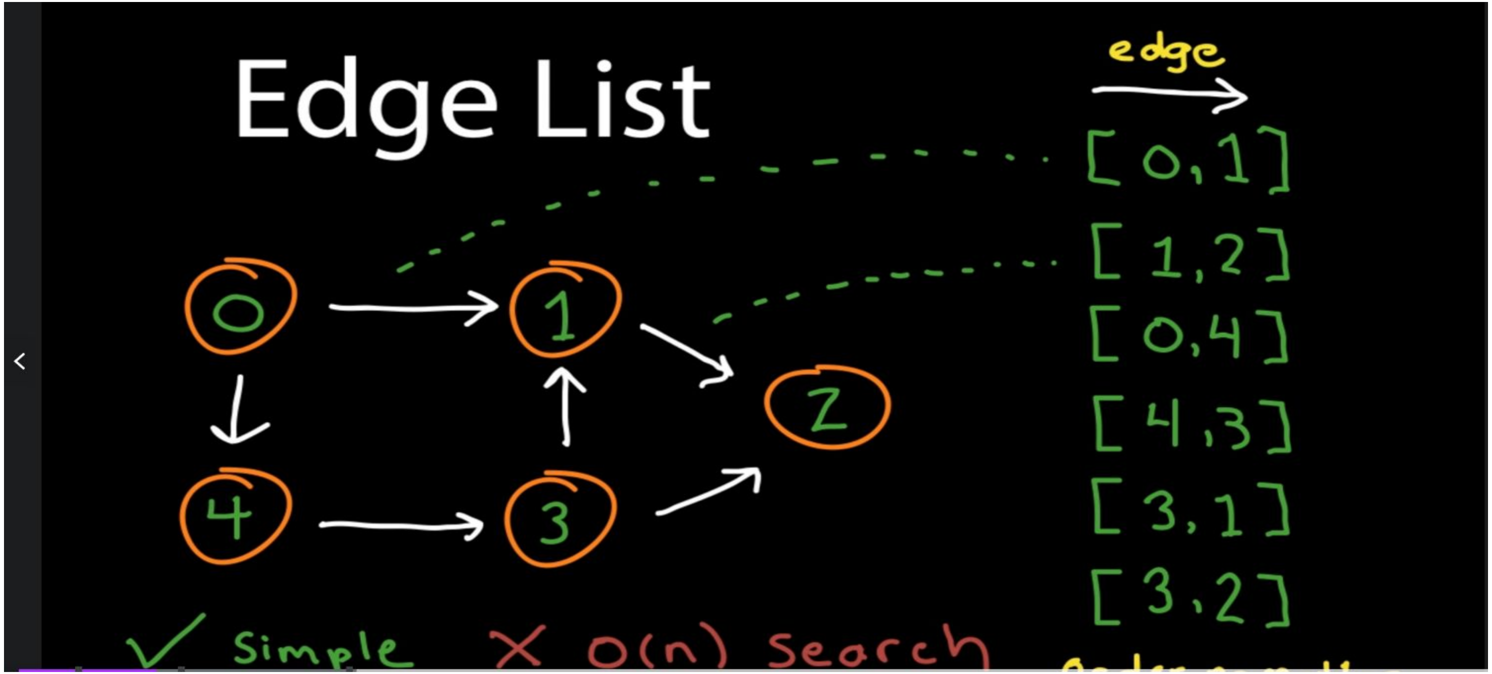

```
const vertices = ['A', 'B', 'C', 'D', 'E']
const edges = [
  ['A', 'B'],
  ['A', 'D'],
  ['B', 'C'],
  ['C', 'D'],
  ['C', 'E'],
  ['D', 'E']
]
```

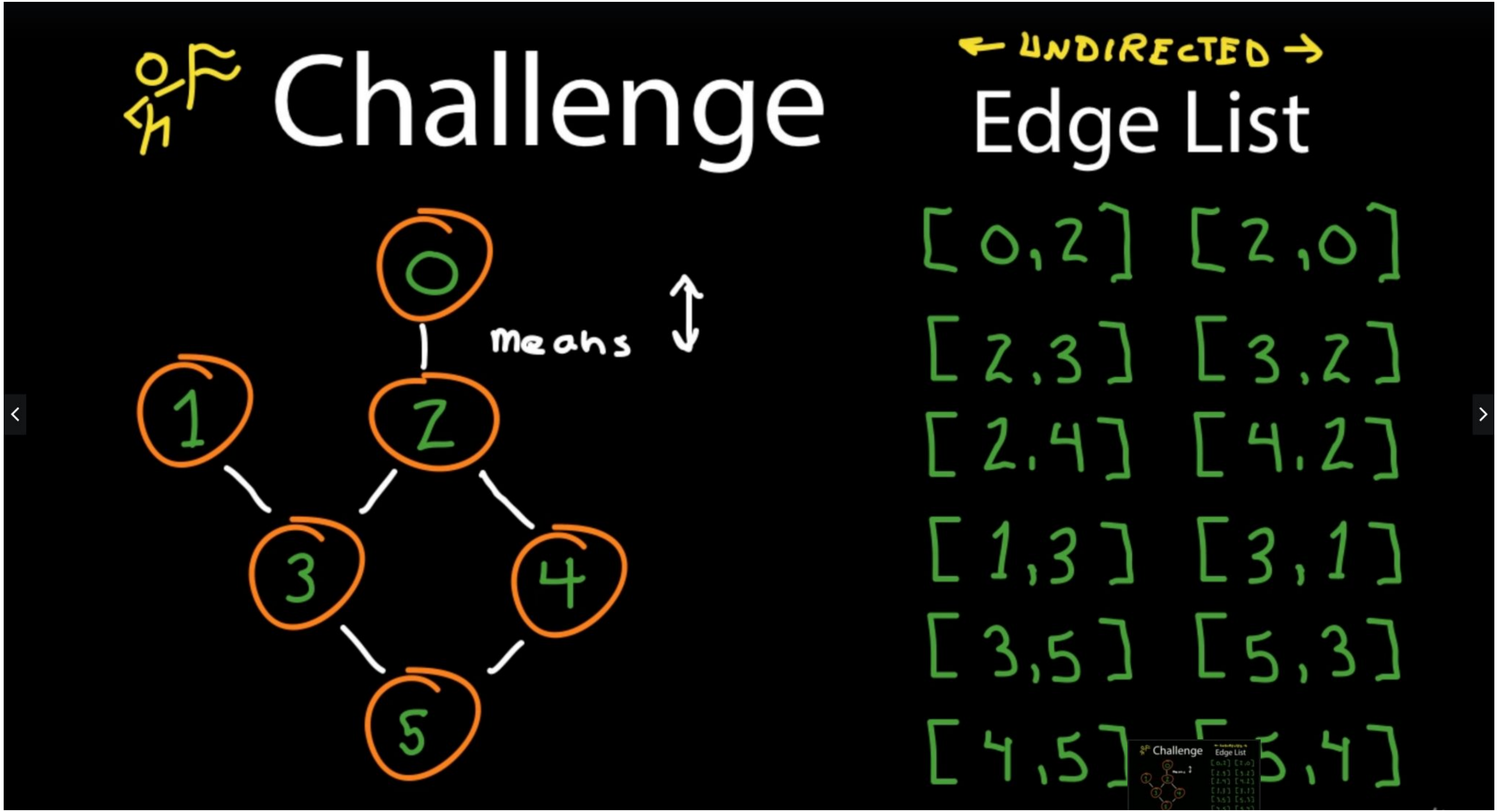
Vertex list + Edge list

- Time complexity to find adjacent nodes → *O(e)* where *e* is the number of edges
- Time complexity to check if two nodes are connected → *O(e)*
- Space complexity → *O(v + e)* where *v* is number of vertices and *e* is number of edges




```
const vertices = ['A', 'B', 'C', 'D', 'E']
const edges = [
  ['A', 'B'],
  ['A', 'D'],
  ['B', 'C'],
  ['C', 'D'],
  ['C', 'E'],
  ['D', 'E']
]
```

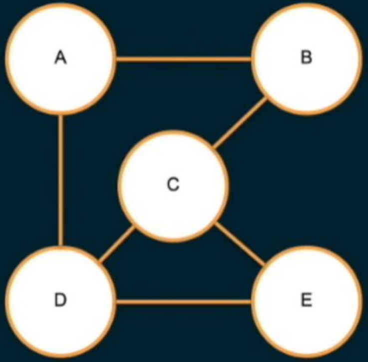

Edge List
edge
[0,1]
[1,2]
[0,4]
[4,3]
[3,1]
[3,2]
0
1
2
3
4
✓ Simple
✗ O(n) search



Challenge
← UNDIRECTED →
Edge List
0
1
2
3
4
5
means ↕
[0,2] [2,0]
[2,3] [3,2]
[2,4] [4,2]
[1,3] [3,1]
[3,5] [5,3]
[4,5] [5,4]

Adjacency Matrix

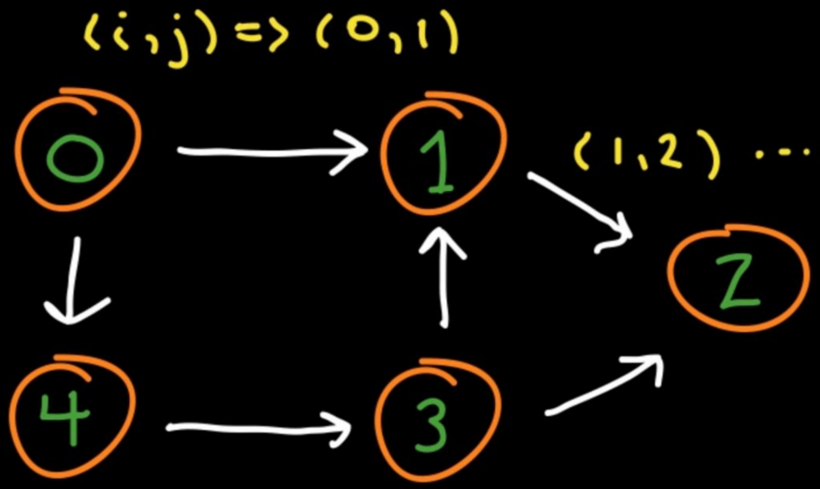
- A 2-D array filled out with 1's and 0's where each array represents a node and each index in the subarray, represents a potential connection to another node
- The value at *adjacencyMatrix[node1][node2]* indicates where there is a connection between node1 and node2




```
const vertices = ['A', 'B', 'C', 'D', 'E']

const adjacencyMatrix = [
  [0, 1, 0, 1, 0],
  [1, 0, 1, 0, 0],
  [0, 1, 0, 1, 1],
  [1, 0, 1, 0, 1],
  [0, 0, 1, 1, 0]
]
```

Adjacency Matrix

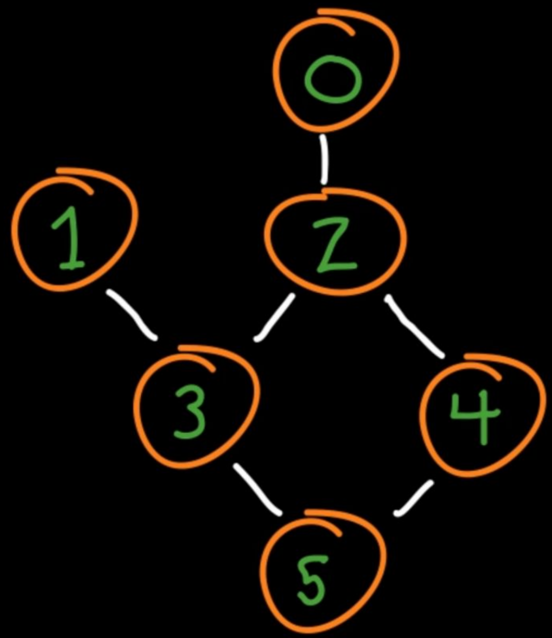


	0	1	2	3	4
0	0	1	0	0	1
1	0	0	1	0	0
2	0	0	0	0	0
3	0	1	1	0	0
4	0	0	0	1	0

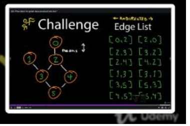
✓ O(1) lookup ✗ Not space efficient

Challenge

← UNDIRECTED →
Adjacency Matrix

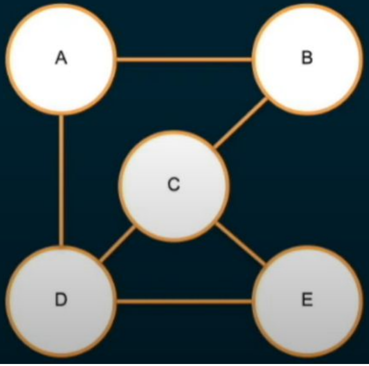


$$
\begin{array}{c|cccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 \\
\hline
0 & & & 1 & & & \\
1 & & & & 1 & & \\
2 & 1 & & & 1 & 1 & \\
3 & & 1 & 1 & & & 1 \\
4 & & & 1 & & & 1 \\
5 & & & & 1 & 1 &
\end{array}
$$



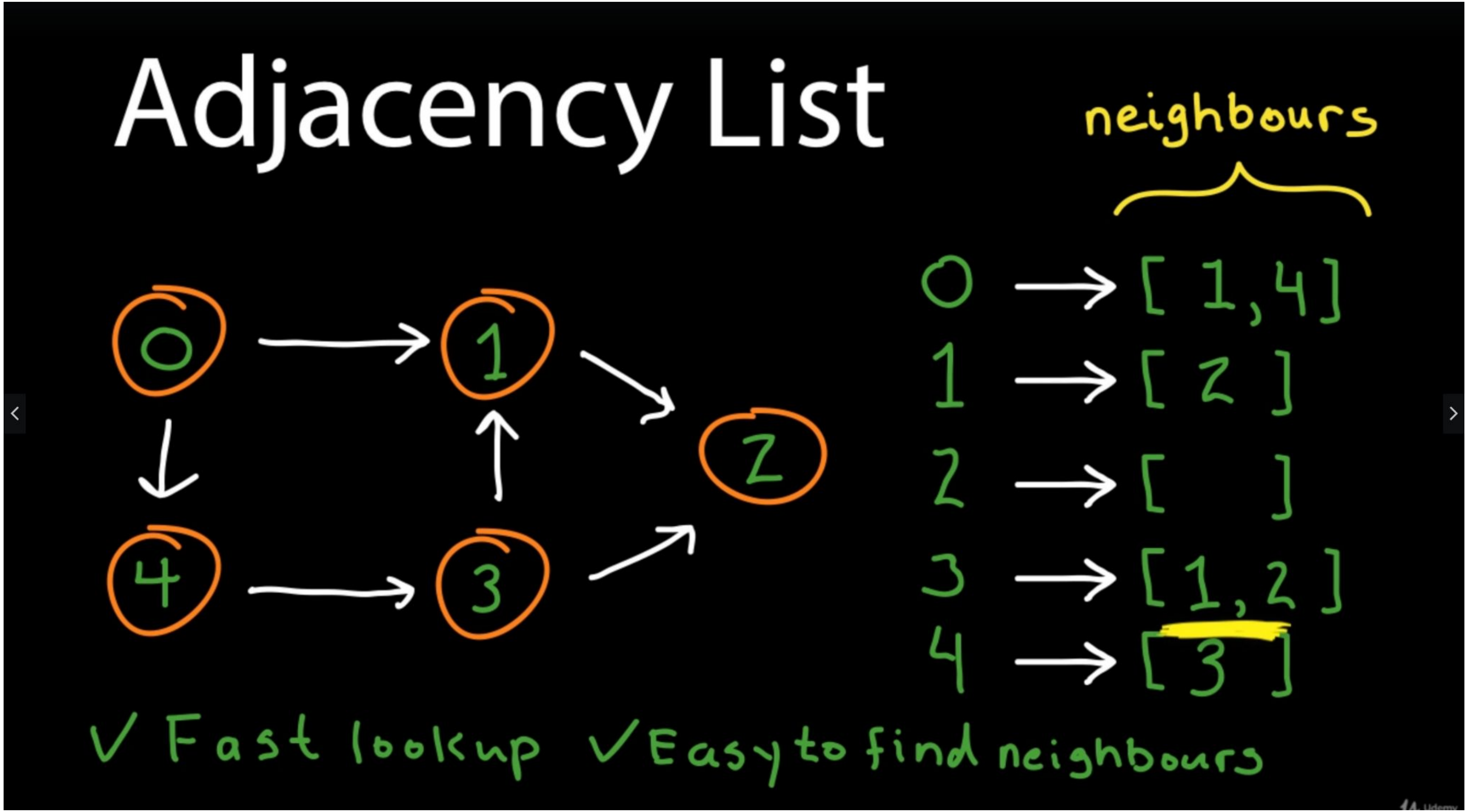
Adjacency List

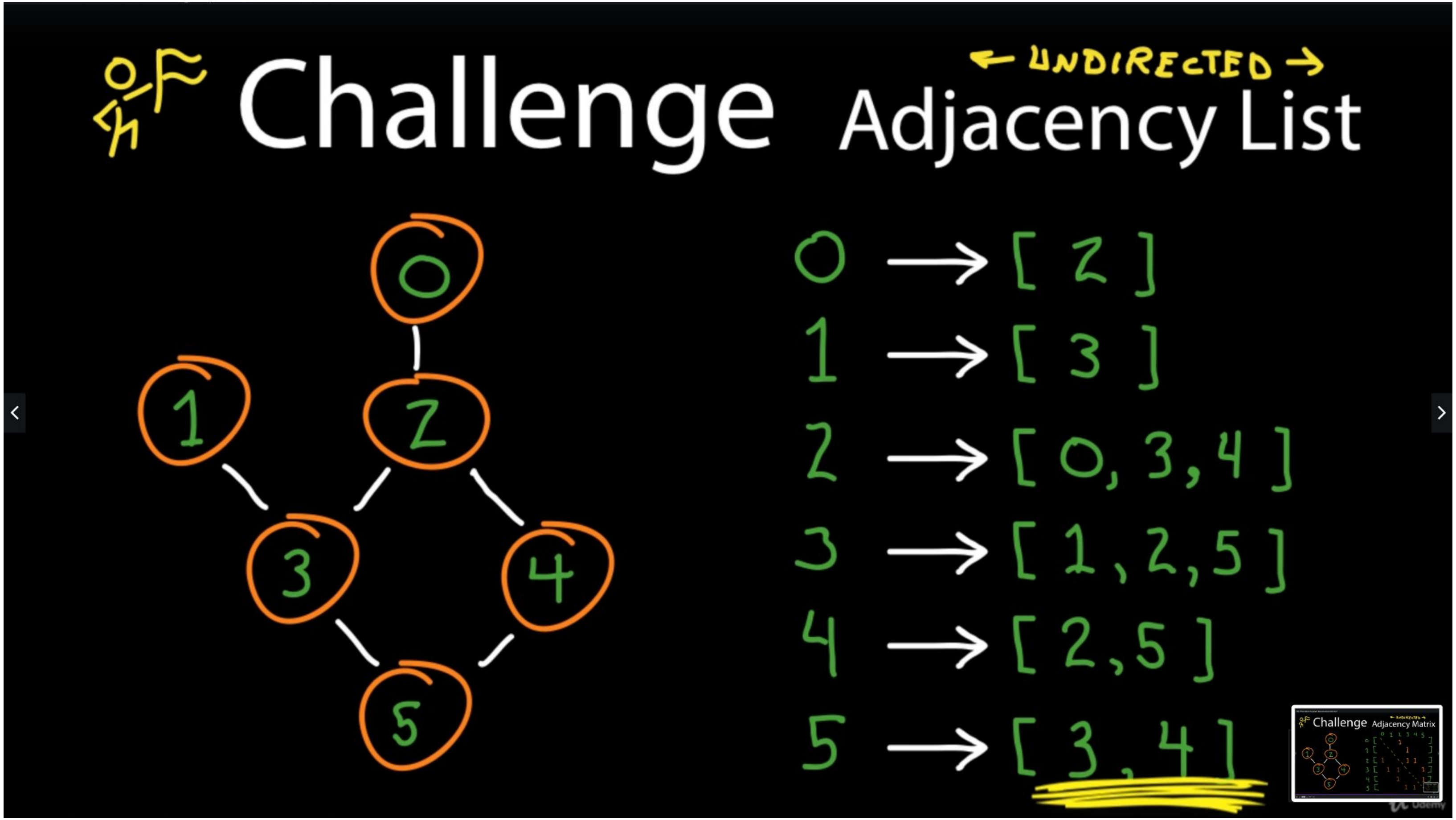
- For every node, store a list of what nodes it's connected to
- Time complexity to find adjacent nodes → *O(1)*
- Time complexity to check if two nodes are connected → *O(logv)* if each adjacent row is sorted
- Space complexity → O(e)




```
const vertices = ['a', 'b', 'c', 'd', 'e']

const adjacency_list = [
  ['b', 'd'],
  ['a', 'c'],
  ['b', 'd', 'e'],
  ['a', 'c', 'e'],
  ['c', 'd']
]
```


Adjacency List
neighbours
0
1
2
4
3
0 → [1,4]
1 → [2]
2 → [ ]
3 → [1,2]
4 → [3]
✓ Fast lookup ✓ Easy to find neighbours

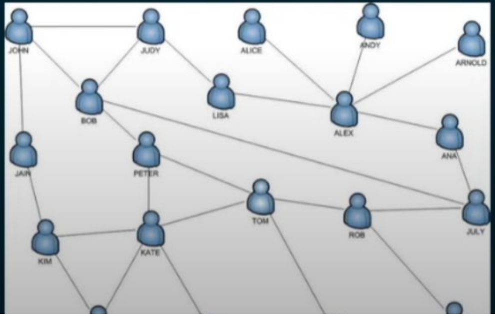

Challenge
← UNDIRECTED →
Adjacency List
0
1
2
3
4
5
0 → [2]
1 → [3]
2 → [0, 3, 4]
3 → [1, 2, 5]
4 → [2, 5]
5 → [3, 4]
Challenge Adjacency Matrix

Directed vs Undirected Graphs

- In an *Undirected Graph*, when there is a connection between nodes, it goes both ways
- Facebook and its users and the relationship between the users can be modeled as an undirected graph
- Users are *nodes* and friendships between the users are *edges*
- There may be many ways two users are connected on Facebook

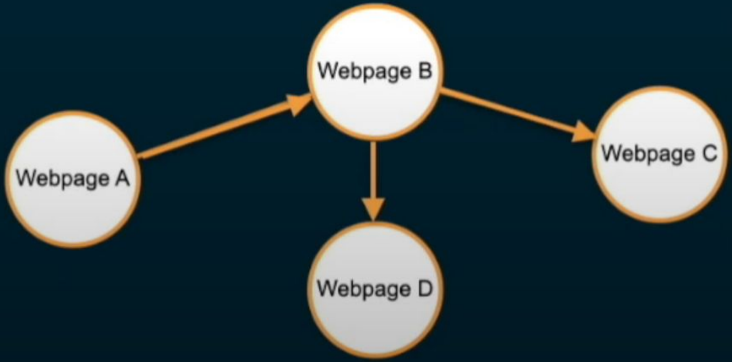
Directed vs Undirected Graphs

- In an *Undirected Graph*, when there is a connection between nodes, it goes both ways
- Facebook and its users and the relationship between the users can be modeled as an undirected graph
- Users are *nodes* and friendships between the users are *edges*
- There may be many ways two users are connected on Facebook
- The graph is undirected because if you are friends with someone on Facebook, they are by definition friends with you in return



Directed Graphs

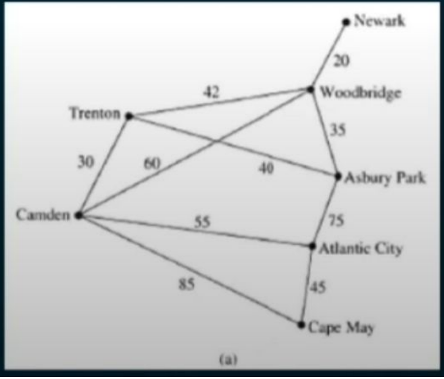
- In a *Directed Graph*, connections between nodes have direction
- The internet can be modeled as a directed graph, where individual web pages are nodes and links between the pages are directed edges
- Links are directed - just because one page links to another, doesn't mean that page links back in return
- The *degree* of a node is the number of edges connected to the node.
- In a directed graph, nodes have an *indegree* or edges pointing to it and an *outdegree* or edges pointing from it



coderbyte

Weighted vs Unweighted Graphs

- A *Weighted Graph* is a graph in which the edges have values corresponding to weights
- An intercity road network could be an example of a weighted graph, where each city is a node and each road is an edge, labeled with their lengths.
- An *Unweighted Graph* has no weights to its edges, so every edge is the same as any other edge

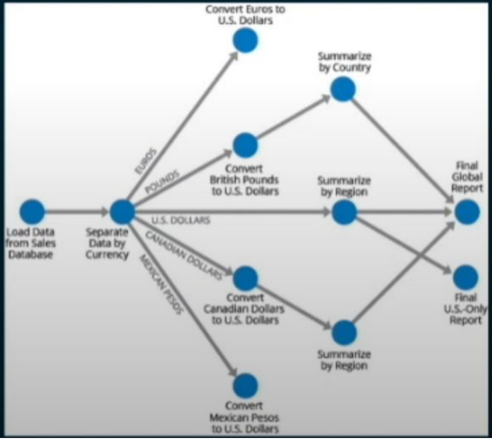


(a)

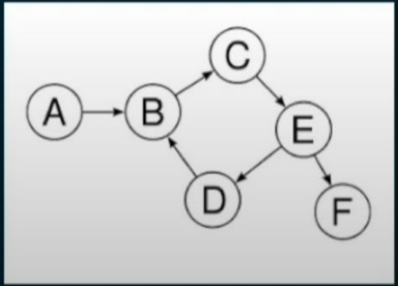
Cyclic vs Acyclic Graphs

- In a *Cyclic Graph*, there is at least one cycle, meaning that there is a path from at least one node back to itself
- An *Acyclic Graph*, means the graph contains no cycles aka no node can be traversed back to itself
- Both of these graphs can be directed or undirected. *Directed Acyclic Graphs (DAG's)* have special applications in computer science and can often be used to represent any complex data processing flows

Example of a DAG



A Cyclic Graph



coderbyte

Dense vs Sparse Graphs

- *Dense graph* - close to the maximum number of edges
- *Sparse graph* - the number of edges is close to the number of nodes in the graph

- *Self-loop* - when an edge has just one vertex (like a web page linking to itself)
- *Multi-edge graphs* - there are multiple edges between two vertices
- *Simple graph* - A graph with no self-loops and no multi-edges

- In a simple directed graph, the maximum number of edges will be equal to $n * (n-1)$ where n is the number of nodes

Dense vs Sparse Graphs

- *Dense graph* - close to the maximum number of edges
- *Sparse graph* - the number of edges is close to the number of nodes in the graph

- *Self-loop* - when an edge has just one vertex (like a web page linking to itself)
- *Multi-edge graphs* - there are multiple edges between two vertices
- *Simple graph* - A graph with no self-loops and no multi-edges

- In a simple directed graph, the maximum number of edges will be equal to $n * (n-1)$ where n is the number of nodes
- In a simple undirected graph, the maximum number of edges is $n * (n-1) / 2$ (because there are no directions, there can only be one edge between two nodes)

Common Graph Interview Questions

- Find the shortest path between two nodes
- Check if an undirected graph contains a cycle
- Given an undirected graph with maximum degree *D*, find a graph coloring using at most *D* + 1 colors