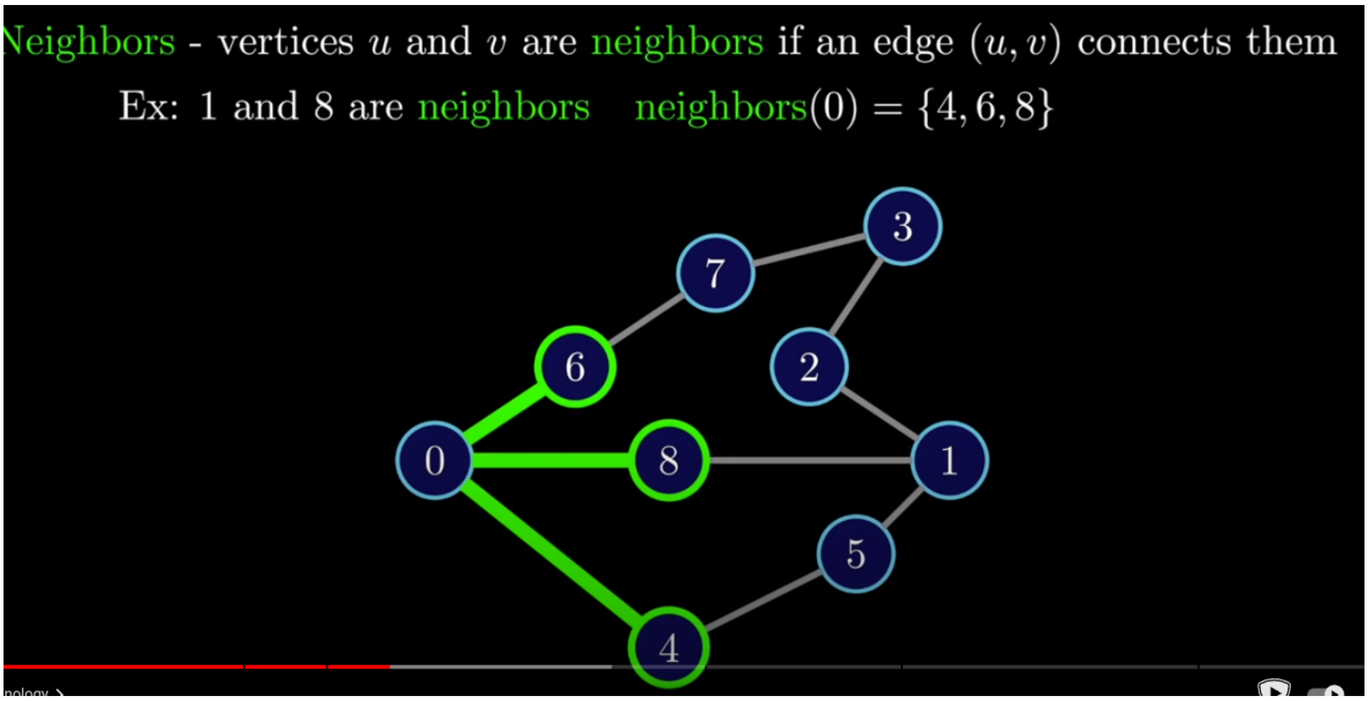
Graphs

What is a Graph?

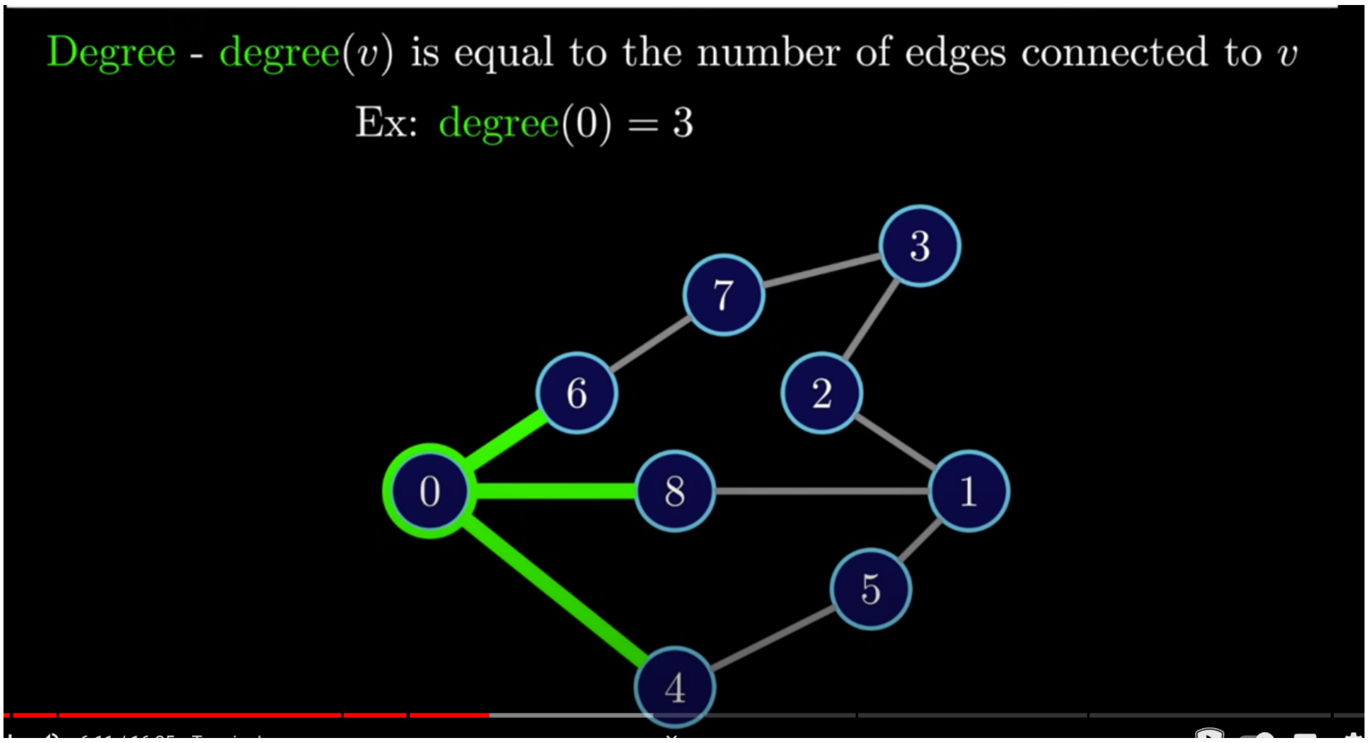
Graph - A collection of nodes with a collection of edges, the data is the node and the thing that connects them are called edges. This could be used to abstract the distance between cities.

A graph is a collection of objects that are called nodes, connected together by edges. These are non linear data structures, meaning that the data doesn't have to be in order, rather it is a hierarchical structure. It is used to represent relationships between objects or entities. The nodes in a graph can be connected by edges, which indicate a connection or relationship between the nodes.

There are different types of graphs, including directed graphs (also known as digraphs), where the edges have a specific direction, and undirected graphs, where the edges have no specific direction. Graphs can also have weighted edges, where each edge has a numerical value associated with it. Nodes can have neighbors, which are other nodes that are connected to the node by an edge.

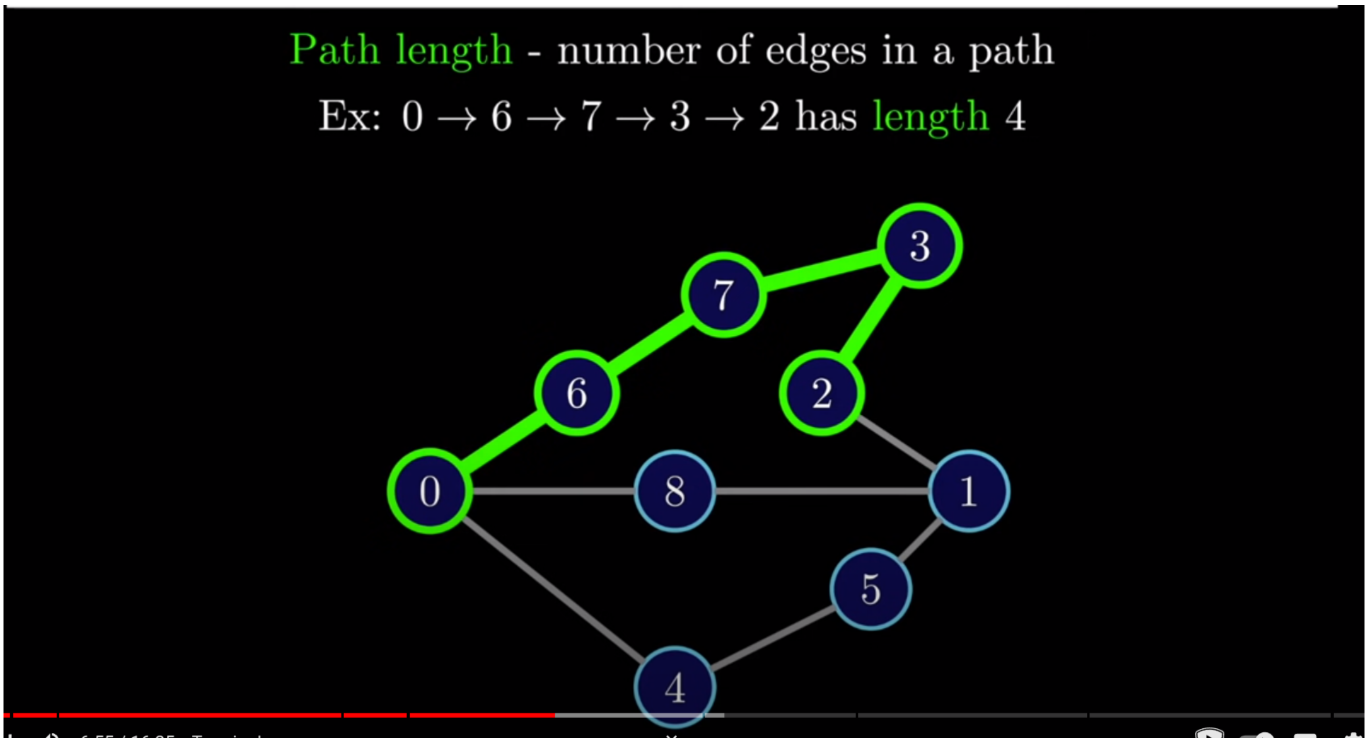

Neighbors - vertices u and v are neighbors if an edge (u, v) connects them
Ex: 1 and 8 are neighbors
$\text{neighbors}(0) = \{4, 6, 8\}$
0
1
2
3
4
5
6
7
8

The degree of a node is the number of edges connected to the node.


Degree - degree(v) is equal to the number of edges connected to v

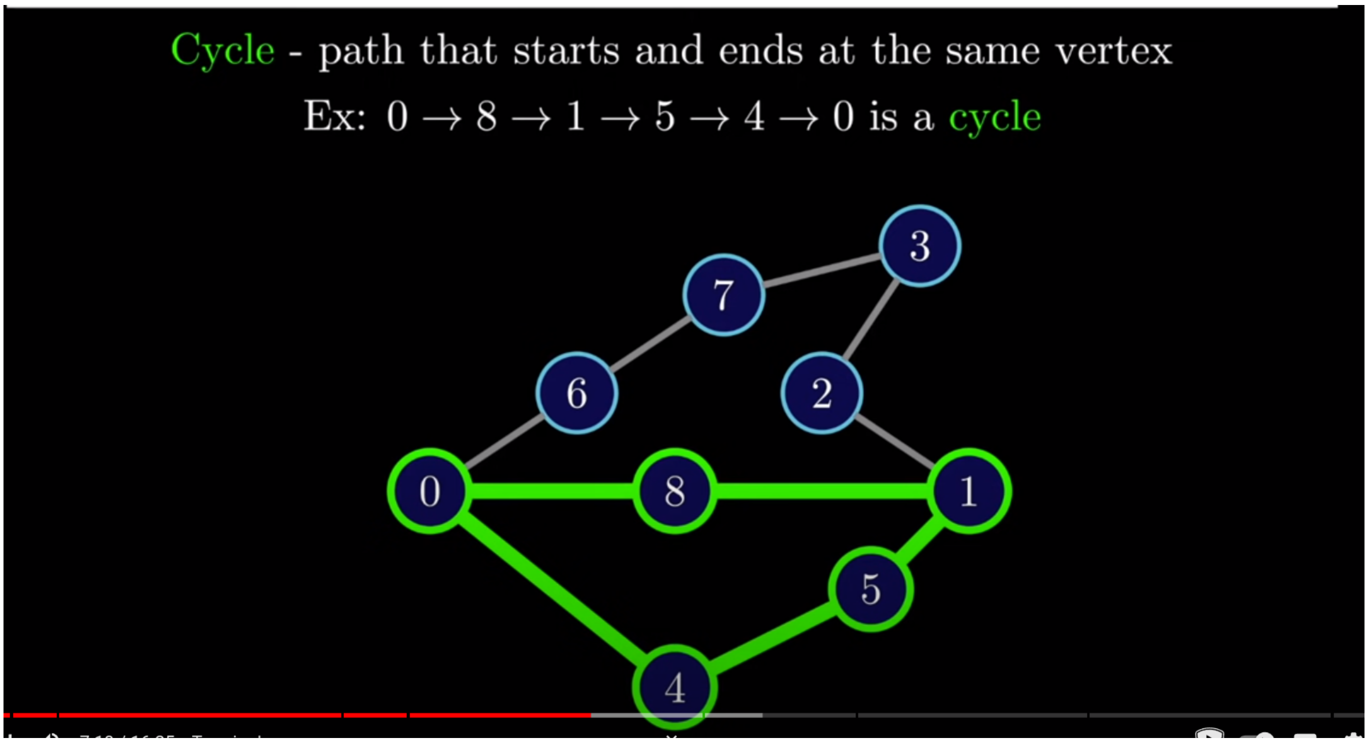
Ex: degree(0) $= 3$

The path is the sequence of nodes that one must go through to get from one node to another. Path length is the number of edges in a path.


Path length - number of edges in a path

Ex: $0 \to 6 \to 7 \to 3 \to 2$ has length 4

A circle is a path that starts and ends at the same node.


Cycle - path that starts and ends at the same vertex
Ex: 0 → 8 → 1 → 5 → 4 → 0 is a cycle
0
1
2
3
4
5
6
7
8

Algorithms for graphs, such as depth-first search (DFS) and breadth-first search (BFS), can be used to traverse and manipulate the data stored in a graph.

Depth-first search (DFS) is an algorithm that traverses a graph in a depthward motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration. As in the example given above, DFS algorithm traverses from S to A to D to G to E to B first, then to F and lastly to C. It employs the following rules.

Breadth-first search (BFS) is an algorithm that traverses a graph in a breadthward motion and uses a queue to remember to get the next vertex to start a search, when a dead end occurs in any iteration. As in the example given above, BFS algorithm traverses from A to B to E to F first then to C and G lastly. It employs the following rules.

These are the different types of graphs:

- Directed - The edges have a direction, like a one way street.
- Undirected - The edges don't have a direction, like a two way street.
- Weighted - The edges have a weight, like the distance between two cities.
- Unweighted - The edges don't have a weight, like the distance between two cities.

Example of a Graph

Graphs can be used to represent and solving problems related to network connectivity or routing. For example social network.

Say you are building a social networking application and need to implement a feature that suggests friends to users based on their connections. You can represent the network of users and their relationships using a graph.

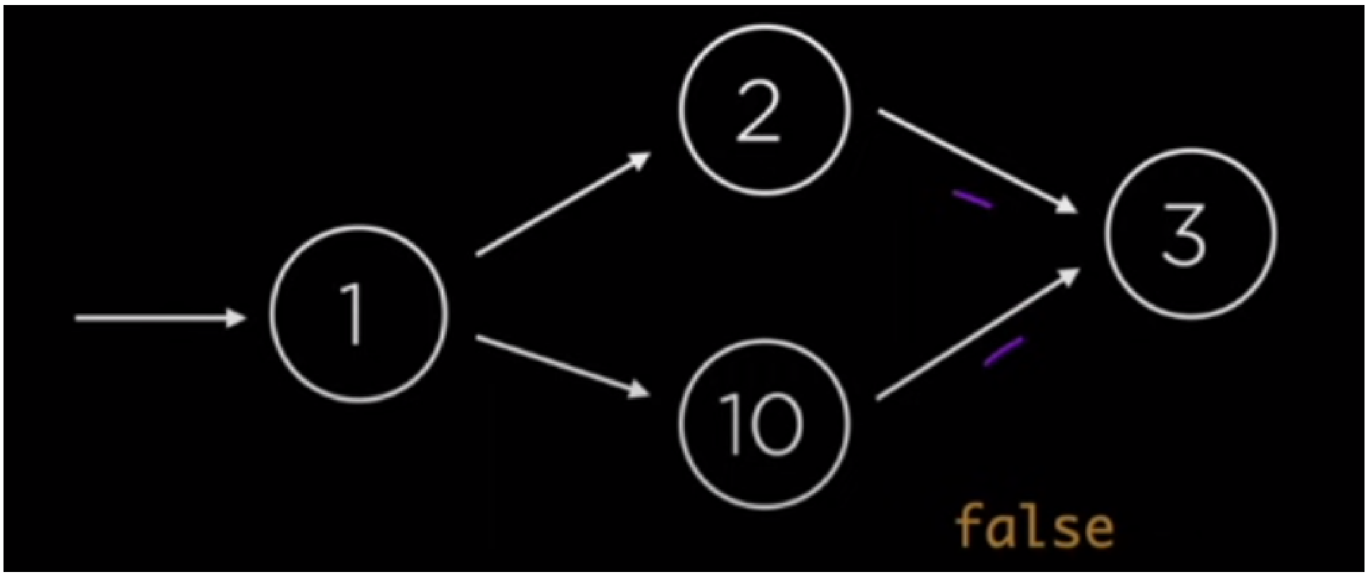
Each user can be represented as a node in the graph, and the connections between users can be represented as edges between the nodes. For example, if user A is friends with user B, there will be an edge connecting the nodes representing A and B.

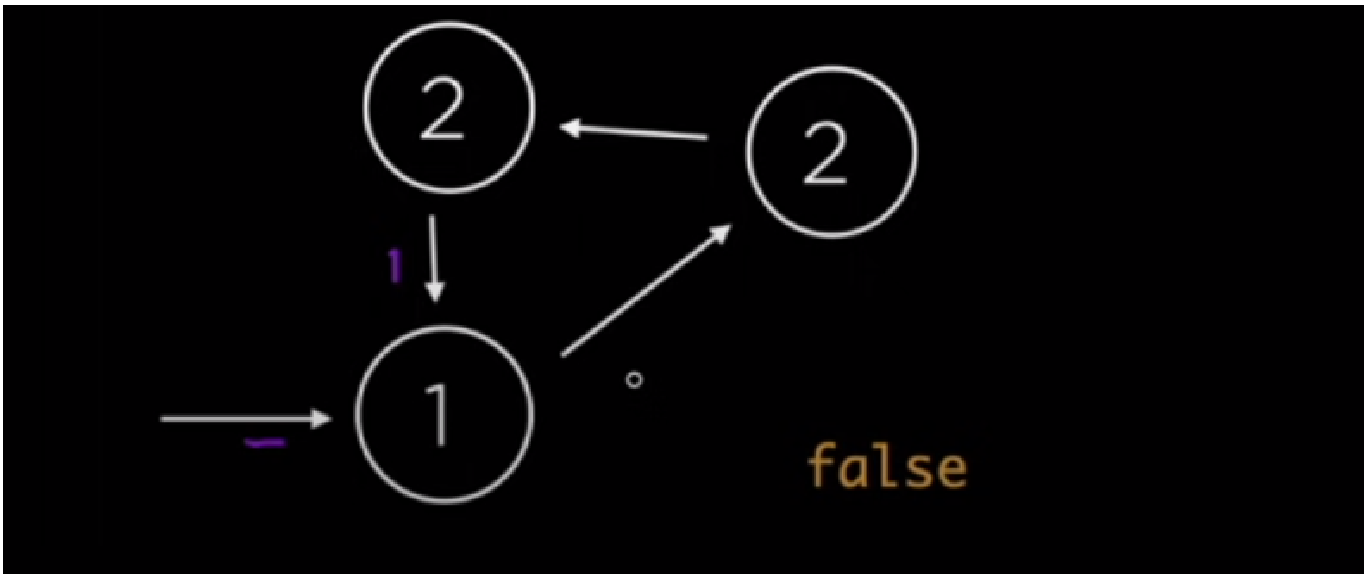
To suggest friends to a user, you can utilize graph algorithms like breadth-first search (BFS) or depth-first search (DFS) to explore the graph starting from the user's node. These algorithms help traverse the graph and find connections or friends-of-friends within a certain distance or based on certain criteria.

Tree

A tree is a special type of graph, where no cycles are allowed. A cycle is when a node can be traversed through and potentially end up back at itself. Trees do not have to be binary, they can have any number of children per node. The root note **has** to be able to access all other nodes. The root node is the only node that doesn't have a parent.

With trees, there can't be two nodes that are connected by more than one path, and there can't be two references that link to the same node.





Trees a similar to linkedlist, the difference is that trees can link to multiple node, while linkedlist can only link to one other node .

```
class Node {
    int data;
    Node node1;
    Node node2;
    Node node3;
}
```

Example of a Tree

A file system is a common example of a tree structure. In a file system, files and directories are organized in a hierarchical structure. Each directory can contain multiple files and subdirectories. This hierarchical structure can be represented using a tree.

Here's an example:

```
/
├── home
│   ├── user1
│   │   ├── file1.txt
│   │   └── file2.txt
│   └── user2
│       └── file3.txt
├── var
│   └── log
│       ├── system.log
│       └── error.log
└── usr
    └── bin
        ├── program1
        └── program2
```

In this example, the root directory ("/") is at the top of the tree, and it has three children: "home," "var," and "usr." The "home" directory has two subdirectories: "user1" and "user2." The "user1" directory contains two files: "file1.txt" and "file2.txt." Similarly, other directories and files are organized in the tree structure.

This tree representation allows us to navigate and access files and directories efficiently. We can perform operations like creating new directories, deleting files, searching for files, and traversing the file system using tree traversal algorithms like depth-first search or breadth-first search.

By using a tree data structure, file systems can maintain the hierarchical relationship between files and directories, enabling efficient storage and retrieval of data.