Graphs

Overview

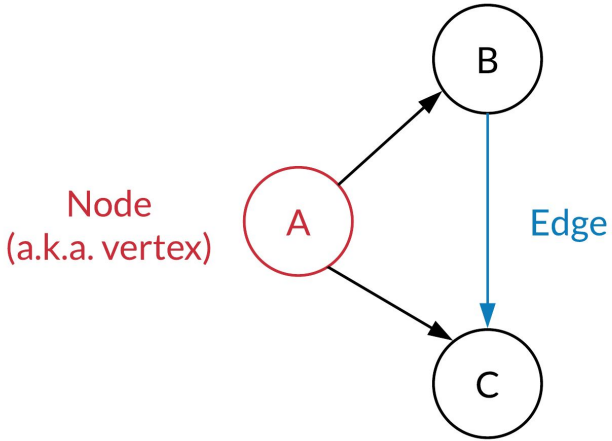
1. Overview of Graphs (15 minutes)
2. Graph question walkthrough (15 minutes)
3. In class exercises (60 minutes)
4. Topological Sort, Disjoint Sets, Advanced Graph Algorithms (15 minutes)

Types of Graph Questions

1. Explicitly stated in the interview question that you have to work with a graph
2. No mention of a graph, but the problem can be framed as a graph problem
    - Notice if there are binary relationships between objects that can be modeled as nodes and edges
    - Common themes: dependencies, grid questions

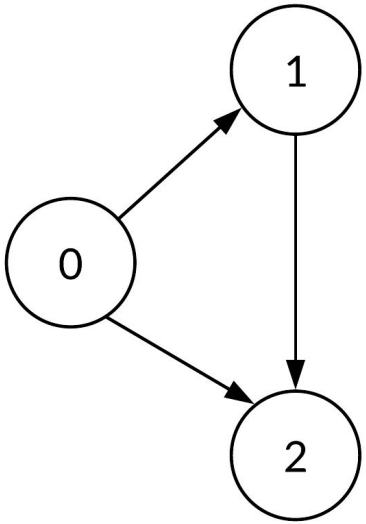
Graph terminology

Graph consists of a set of nodes connected by edges.



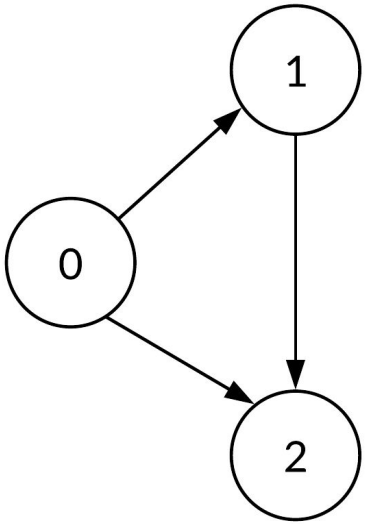
Types of Graphs

Directed Graph

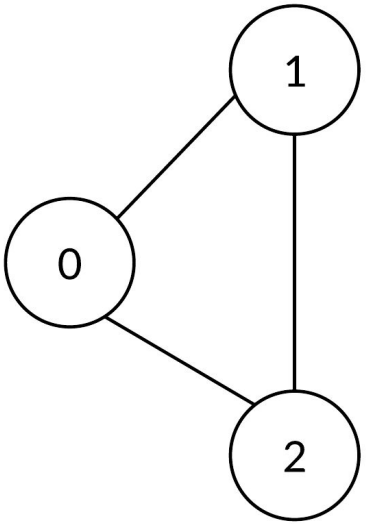


Types of Graphs

Directed Graph

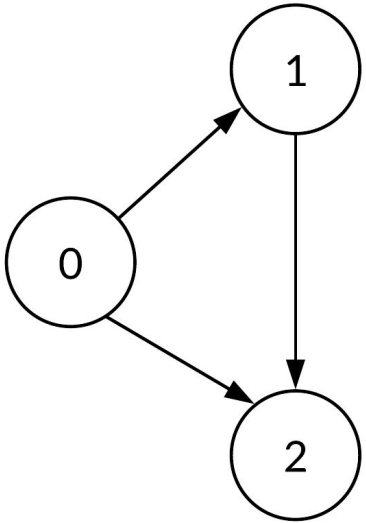


Undirected Graph



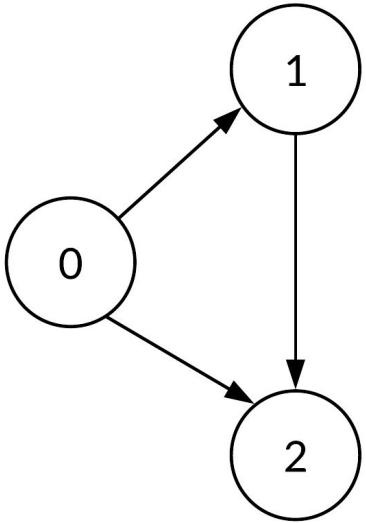
Properties of Graphs

Acyclic Graph

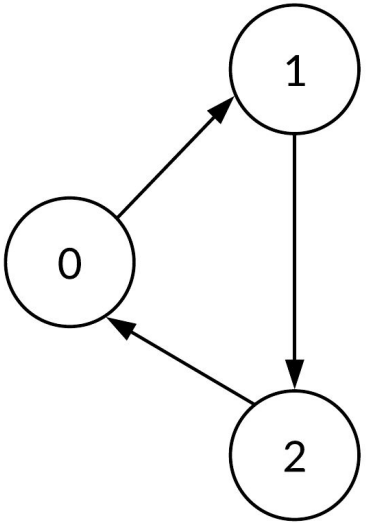


Properties of Graphs

Acyclic Graph

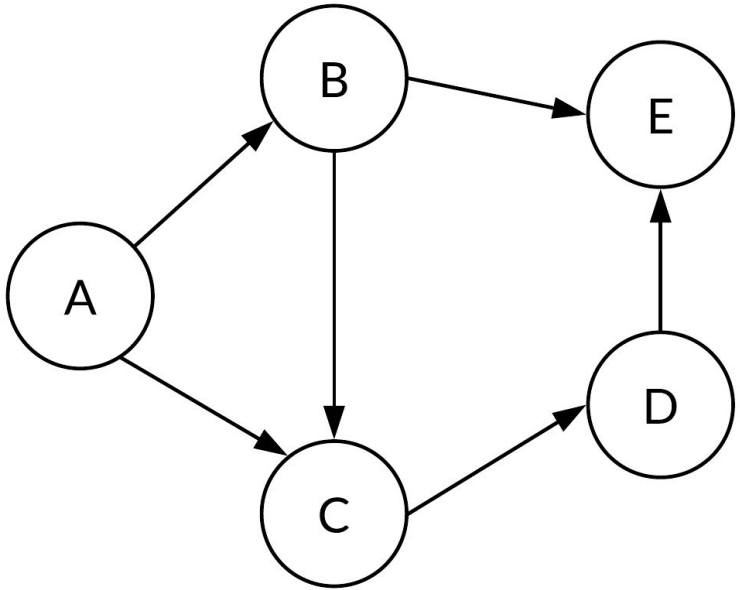


Cyclic Graph



Graph Representations

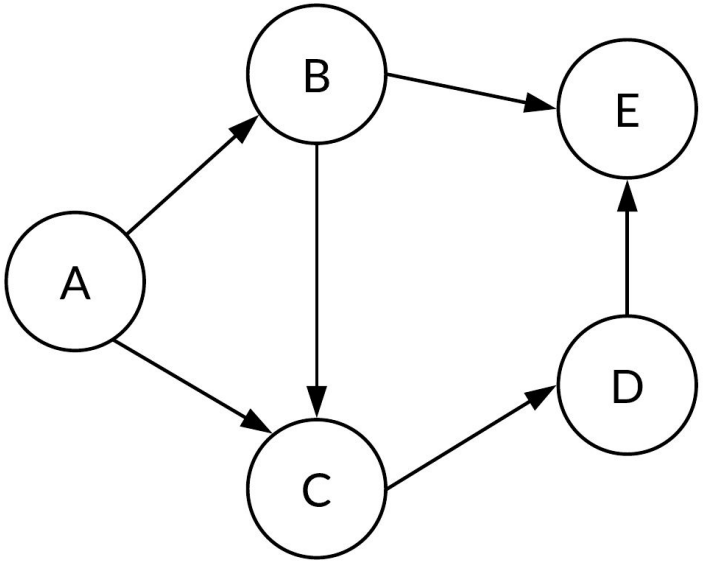
Representation #1: Adjacency List



Adjacency list:

```
graph = {
'A': ['B', 'C'],
'B': ['C', 'E'],
'C': ['D'],
'D': ['E'],
}
```

Representation #2: Edge List

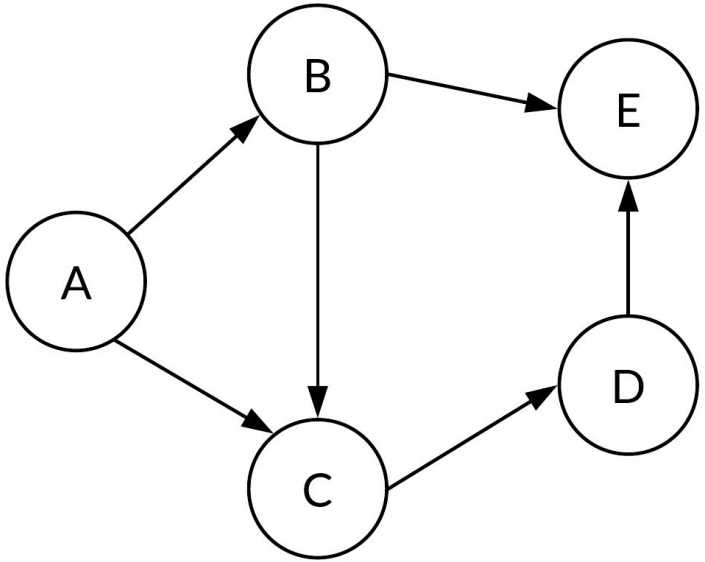



```
Edge list:

graph = [('A', 'B'),
('A', 'C'),
('B', 'C'),
('B', 'E'),
('C', 'D'),
('D', 'E')
]
```

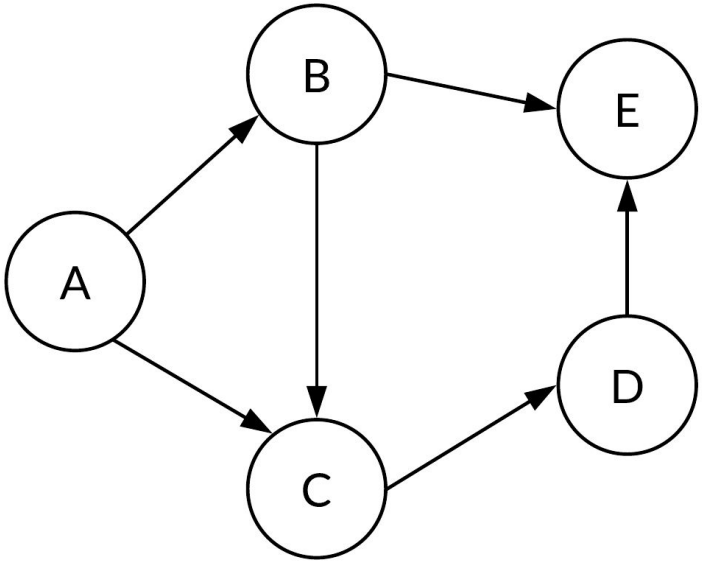
Representation #3: Adjacency Matrix

`matrix[i][j]` is true when there is an edge from `i` to `j`



Representation #3: Adjacency Matrix

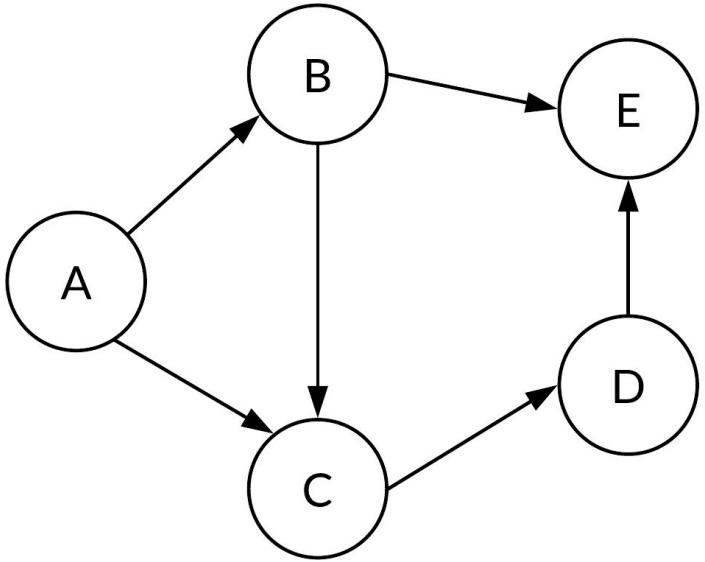
`matrix[i][j]` is true when there is an edge from `i` to `j`



	A	B	C	D	E
A	0	1	1	0	0
B	0	0	1	0	1
C	0	0	0	1	0
D	0	0	0	0	1
E	0	0	0	0	0

Representation #3: Adjacency Matrix

`matrix[i][j]` is true when there is an edge from `i` to `j`



	A	B	C	D	E
A	0	1	1	0	0
B	0	0	1	0	1
C	0	0	0	1	0
D	0	0	0	0	1
E	0	0	0	0	0

Graph Representations

Runtime of some basic operations for each representation:

- V represents # of vertices
- E represents # of edges

Graph Representations

Runtime of some basic operations for each representation:

- V represents # of vertices
- E represents # of edges

Graph Structure	Getting all adjacent edges	hasEdge(s, t)	space used
adjacency matrix	O(V)	O(1)	$O(V^2)$
list of edges	O(E)	O(E)	O(E)
adjacency list	O(1)	O(V)	O(E+V)

Graph Representations

Runtime of some basic operations for each representation:

- V represents # of vertices
- E represents # of edges

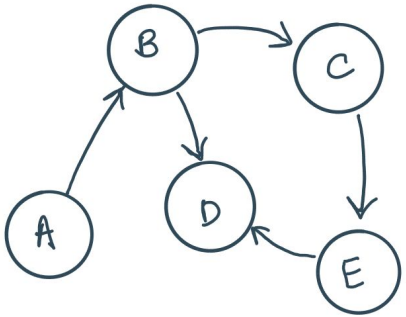
Graph Structure	Getting all adjacent edges	hasEdge(s, t)	space used
adjacency matrix	O(V)	O(1)	$O(V^2)$
list of edges	O(E)	O(E)	O(E)
adjacency list	O(1)	O(V)	O(E+V)

In practice, **adjacency lists** are most common because most graphs have few edges.

Graph traversals

Depth First Traversal (DFS)

Start with an arbitrary node as a root and explore each child node fully before exploring the next one

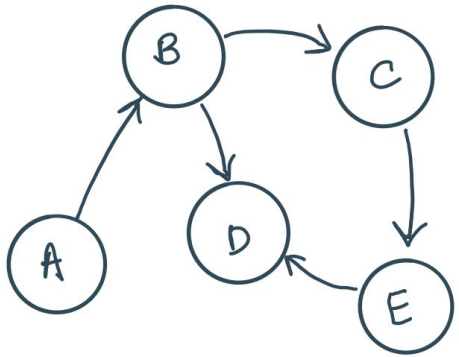


Depth first search starting from node A

Node A
Node B
Node D
Node C
Node E

Breadth First Traversal (BFS)

Pick an arbitrary node as the root and explore each of its neighbors before visiting their children.



Breadth first search starting at node A

Node A
Node B
Node C
Node D
Node E

DFS vs BFS

- BFS is better for optimization
  - ex. shortest path algorithms
- DFS is better for analyzing structure of graphs
  - ex. looking for cycles, bipartite graph

Walkthrough

Walkthrough: Friend Circles

There are N students in a class. Some of them are friends, while some are not. Their friendship is transitive in nature.

For example:

- A is a direct friend of B
- B is a direct friend of C
- A is an indirect friend of C

A **friend circle** is a group of students who are direct or indirect friends.

You are given a N*N matrix `M` that represents the friend relationships between students in the class. If `M[i][j]` = 1, then the `i`th and `j`th students are direct friends with each other.

Output the total number of maximal friend circles among all the students.

Example

You are given a N*N matrix `M` that represents the friend relationships between students in the class. If `M[i][j]` = 1, then the `i`th and `j`th students are direct friends with each other.

Output the total number of maximal friend circles among all the students.

Input

```
[[1,1,0],
 [1,1,0],
 [0,0,1]]
```

Output

2

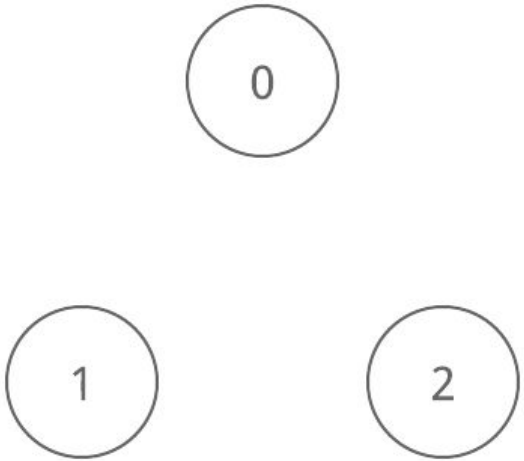
Example

Input

```
[[1,1,0],
 [1,1,0],
 [0,0,1]]
```

Output

2



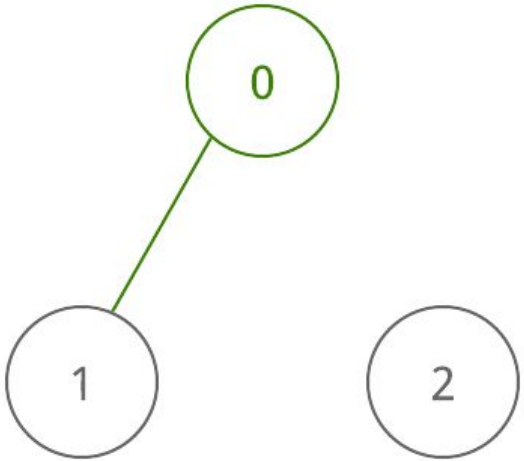
Example

Input

```
[[1,1,0],
 [1,1,0],
 [0,0,1]]
```

Output

2



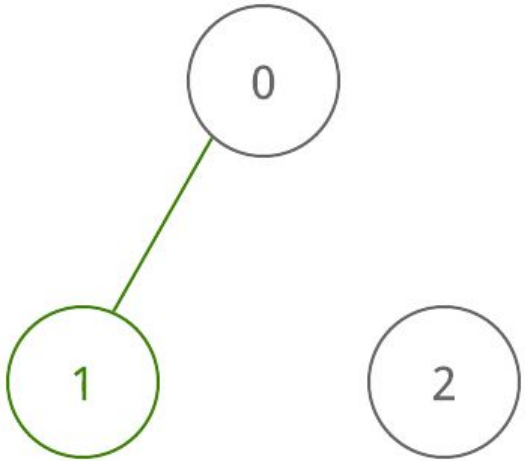
Example

Input

```
[[1,1,0],
 [1,1,0],
 [0,0,1]]
```

Output

2



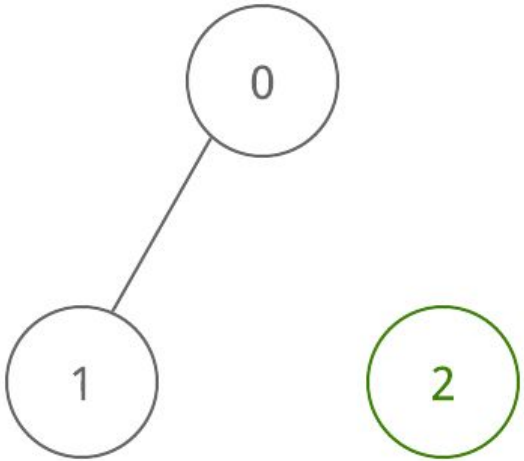
Example

Input

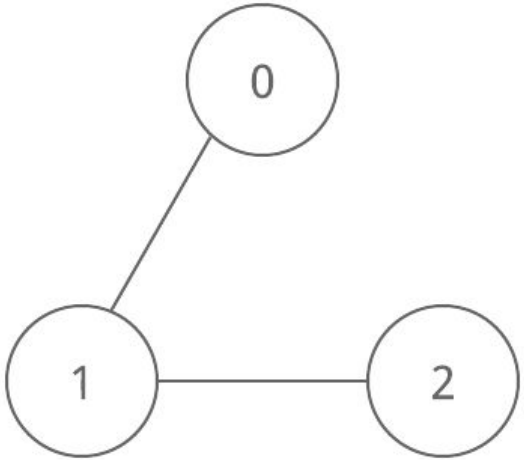
```
[[1,1,0],
 [1,1,0],
 [0,0,1]]
```

Output

2



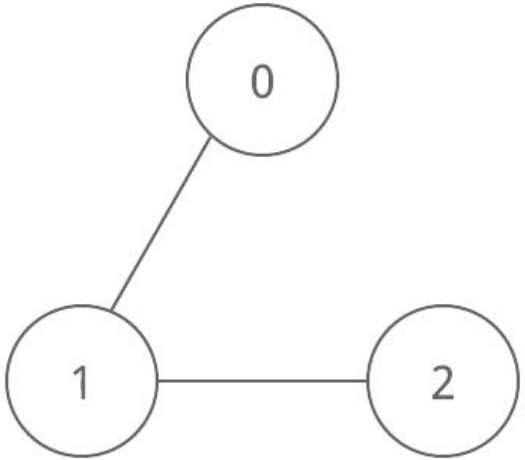
Understand



Understand

Input

`[1,1,0]`

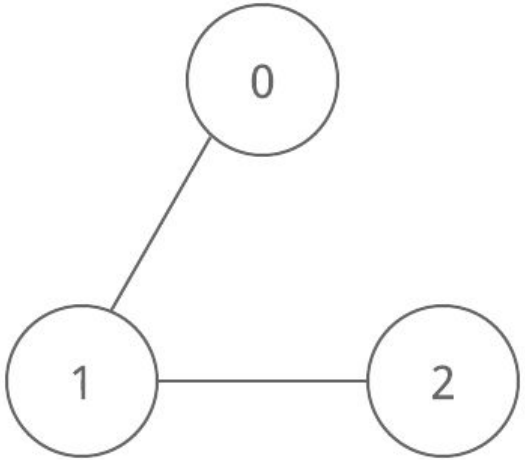


Understand

Input

`[1,1,0]`

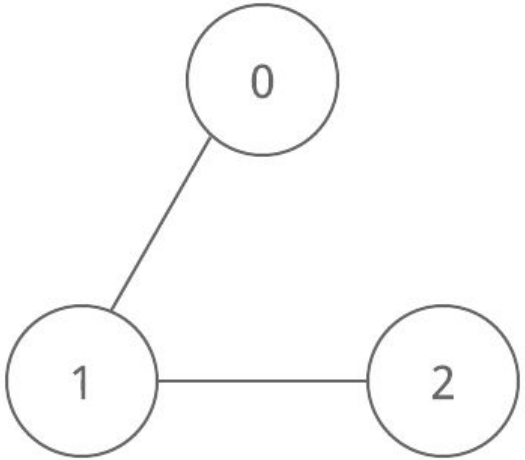
`[1,1,1]`



Understand

Input

```
[1,1,0]
[1,1,1]
[0,1,1]
```



Understand

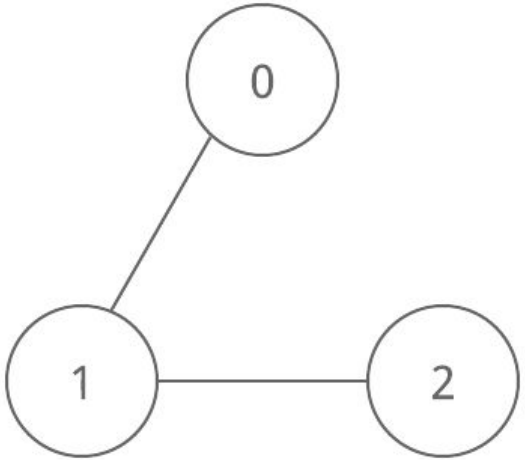
Input

`[1,1,0]`

`[1,1,1]`

`[0,1,1]`

Output 1



Understand

Input / Output

- Case #1: No one has friends
- Case #2: Matrix with all 1s returns 1
- Other cases: use the ones we just created

Clarifying questions

- Is a person who is not friends with anyone else in their own group?

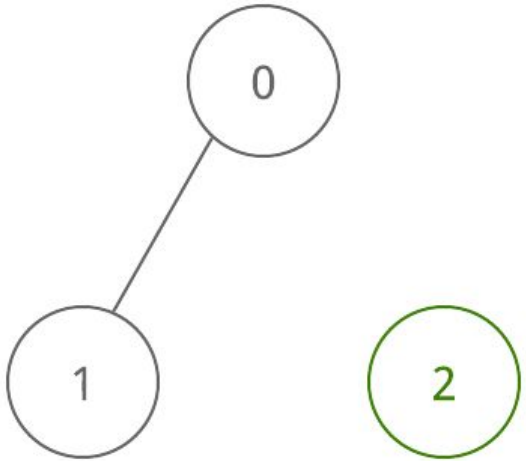
Match

Input

```
[[1,1,0],
 [1,1,0],
 [0,0,1]]
```

Output

2



Match

Graph algorithms

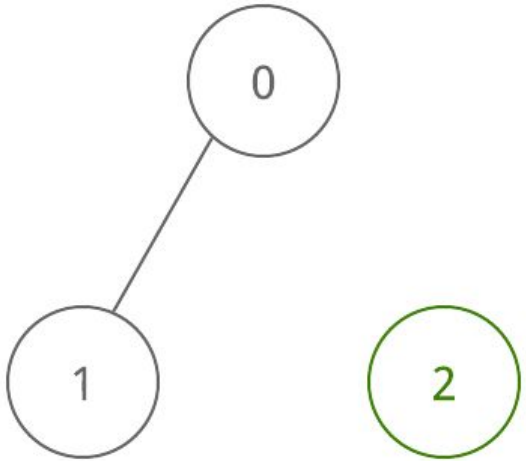
- BFS?
- DFS?

Input

```
[[1,1,0],
 [1,1,0],
 [0,0,1]]
```

Output

2



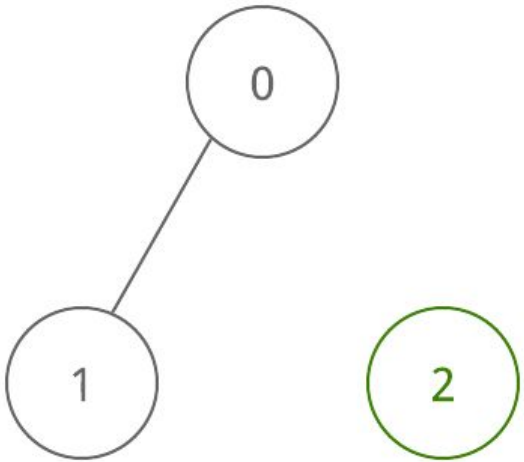
Plan

Input

```
[[1,1,0],
 [1,1,0],
 [0,0,1]]
```

Output

2



Plan

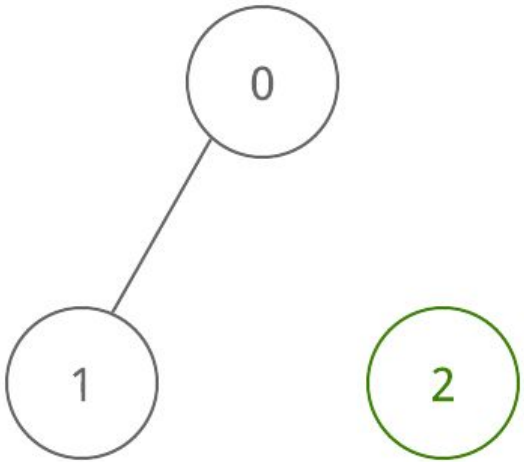
```
for each node:
  if node not visited:
    groups += 1
    mark as visited
    run DFS on node
return groups
```

Input

```
[[1,1,0],
 [1,1,0],
 [0,0,1]]
```

Output

2



Plan

```
for each node:
  if node not visited:
    groups += 1
    mark as visited
    run DFS on node
return groups
```

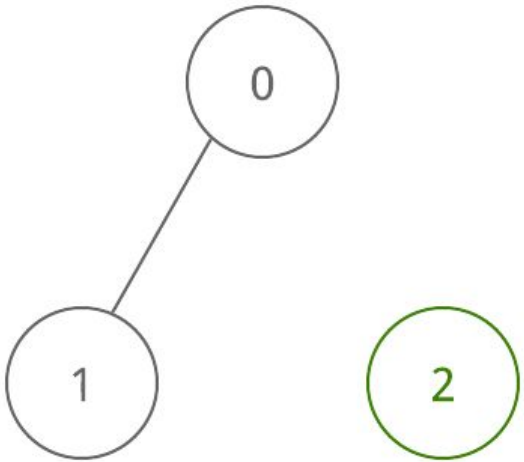
```
DFS(node, matrix, visited):
  for each node's neighbor:
     run DFS if not visited
```

Input

```
[[1,1,0],
 [1,1,0],
 [0,0,1]]
```

Output

2



In class exercises

In class exercise instructions

- 60 minutes
  - 15 minutes self prep
  - 45 minutes discuss with your group
- Questions
  - Reconstruct Itinerary
  - **BONUS:** Max Area of Island

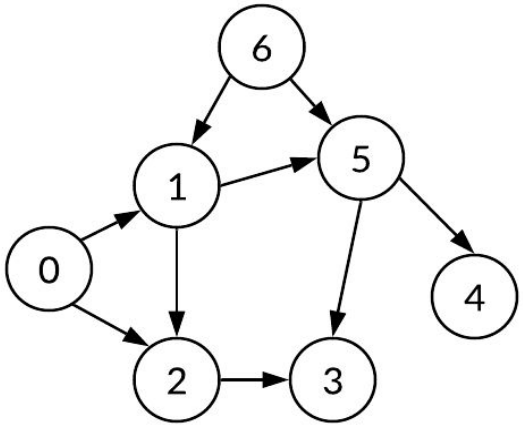
Topological sorting

Topological Sort

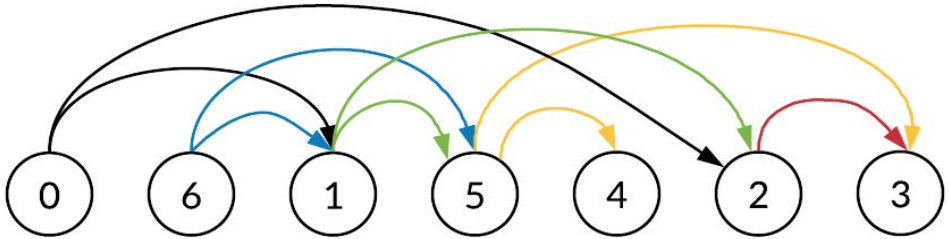
- **Objective:** Create an ordering of nodes, such that for each node, all the dependencies come before that node
- Modification of DFS, only works for **directed acyclic graphs (DAG)**
- Know the implementation of topological sort and why it works
- Real life applications
  - Figuring out what order to install libraries in build systems

Topological Sort

Directed Graph



Topologically sorted graph



Disjoint Sets

Disjoint Sets

- Data structure that represents a collection of sets that are disjoint, meaning that any item in this data structure is found in no more than one set.
- Resources
  - Union Find, Disjoint Sets Guide
  - Union Find, Disjoint Sets Interview question bank

Advanced Graph Algorithms

Advanced Graph Algorithms

Generally not expected to code these algorithms

1. Dijkstra's shortest paths
    a. Starting with a source node, find shortest paths to all other nodes
2. Minimum spanning trees
    a. Given an weighted undirected graph that is connected, find the subset of edges with the minimum total weight such that the graph constructed from these edges is still connected

Helpful Links

- Guides
  - Graph representations
  - Graph traversals (with code)
  - Topological sort (with code)
- Video walkthrough of detect a cycle in a graph

Wrap up

- Be **very comfortable** with BFS, DFS, Topological Sort
- Practice!
- No class on Saturday