Graphs

Overview

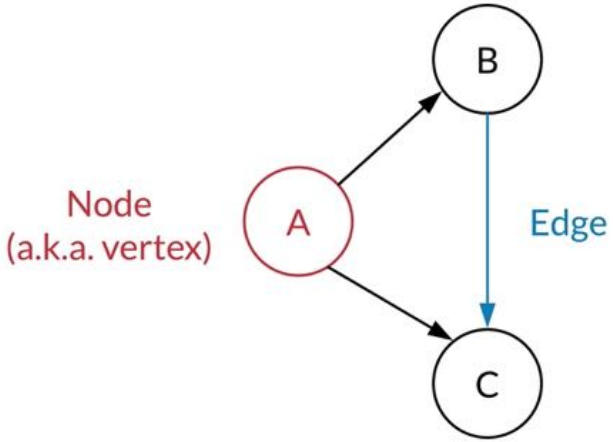
1. Overview of Graphs
2. In class walkthrough
3. Breakout Session
4. Recap

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
    - Common themes: dependencies, grid/matrix questions, finding groupings

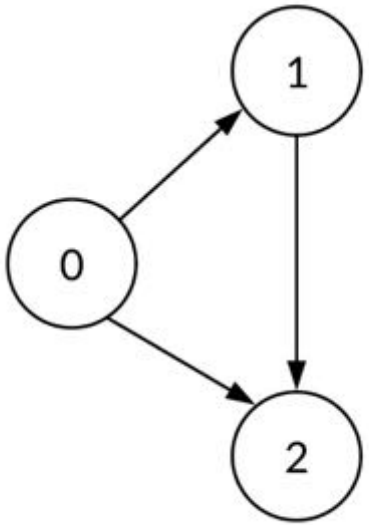
Graph terminology

Graph consists of a set of nodes connected by edges.

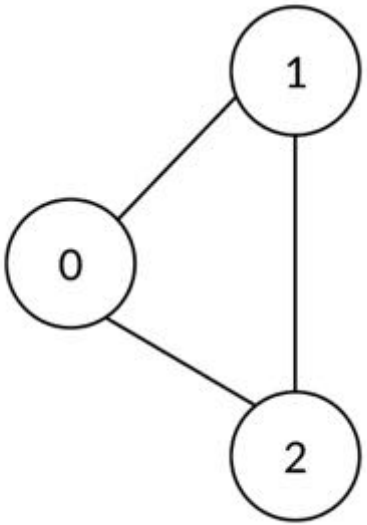


Types of Graphs

Directed Graph

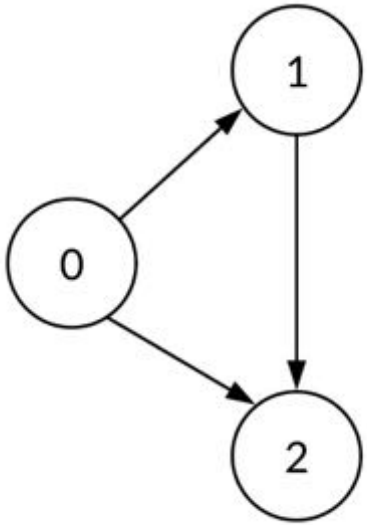


Undirected Graph

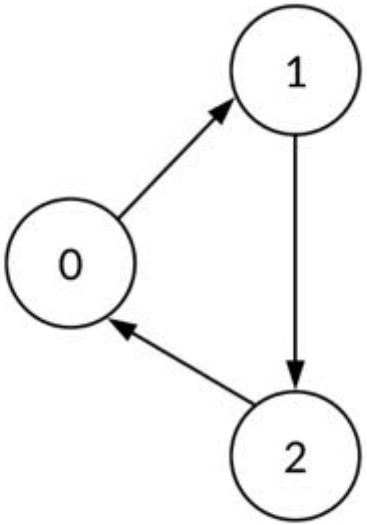


Properties of Graphs

Acyclic Graph

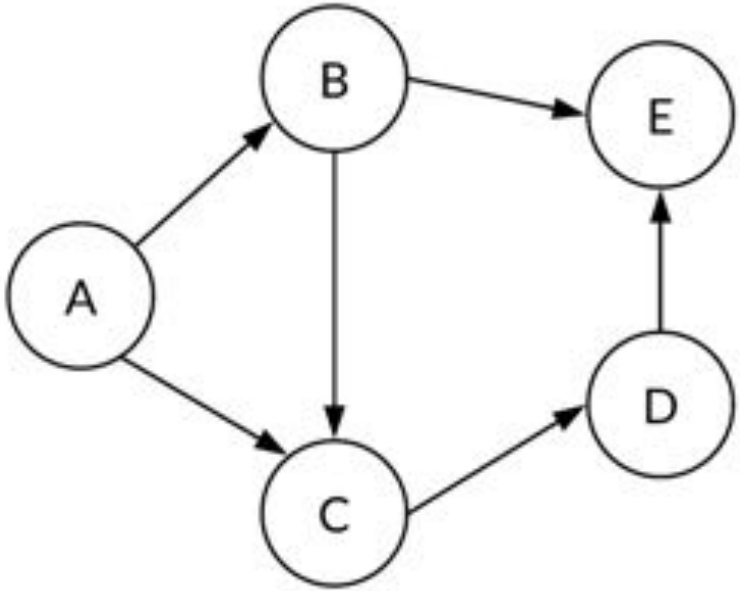


Cyclic Graph



Graph Representations

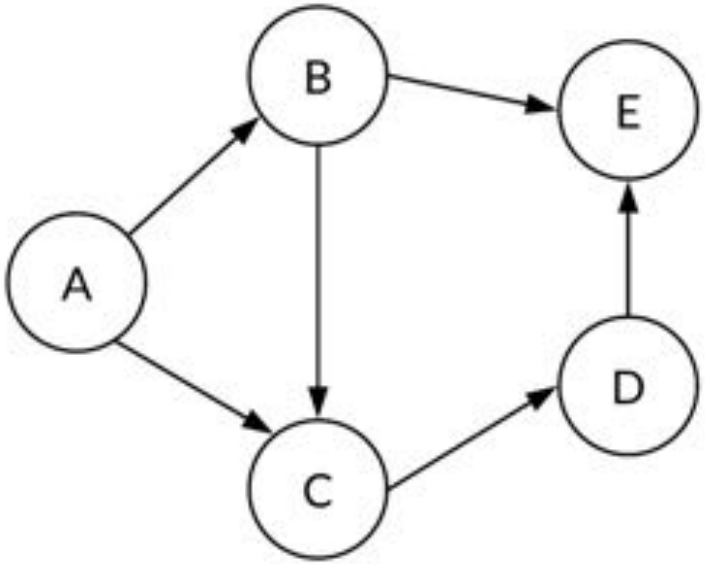
Representation #1: Adjacency List



Adjacency list:

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

Representation #2: Edge List

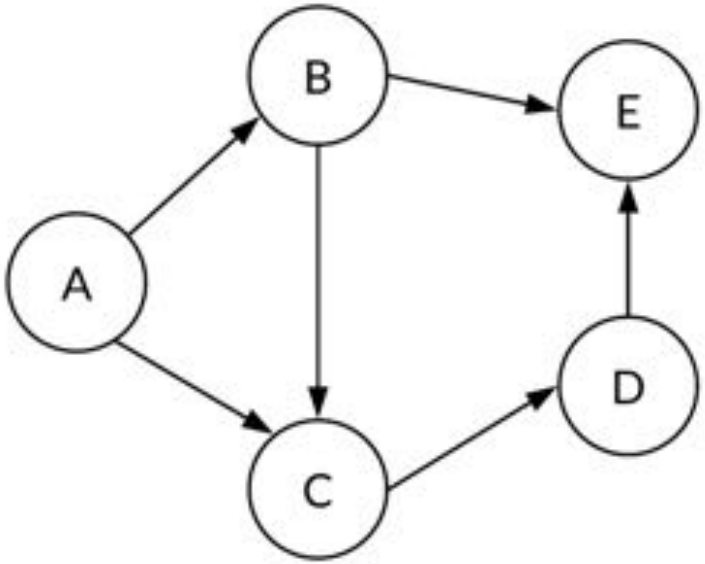


Edge list:

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

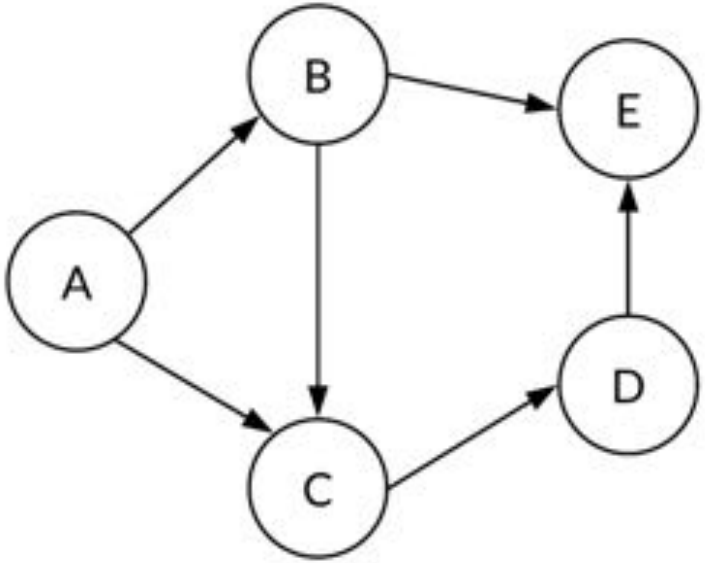
Representation #3: Adjacency Matrix

`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

Graph Representations

Runtime of some basic operations for each representation:

- V represents the number of vertices
- E represents the **TOTAL number of edges**

Graph Representations

Runtime of some basic operations for each representation:

- V represents the number of vertices
- E represents the **TOTAL number of edges**

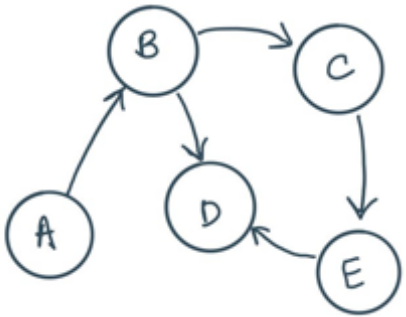
Graph Structure	**Getting all adjacent edges**	**hasEdge(s, t)**	**space used**
adjacency matrix	O(V)	O(1)	$O(V^2)$
list of edges	O(E)	O(E)	O(E)
adjacency list	O(1)	O(V)	O(E+V)

In practice, **adjacency lists** are most common because getting all adjacent edges is very useful for graph traversals.

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

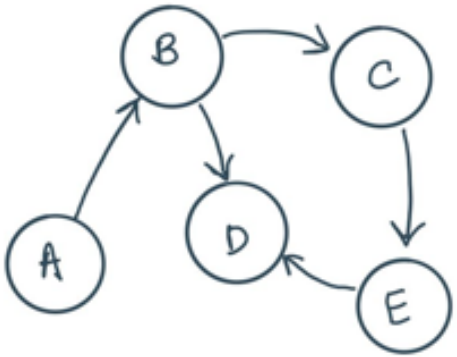
Node A
Node B
Node D
Node C
Node E

Depth First Traversal (DFS)

```
'''
Assuming we have a directed graph represented with an adjacency list.
'''

def depth_first_search(graph, start):

    visited, stack = set(), [start]

    while stack:
        curr_node = stack.pop()

        visited.add(curr_node)

        for neighbor in graph[curr_node]:
            if neighbor not in visited:
               stack.append(neighbor)
```

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

Breadth First Traversal (BFS)

```
'''
Assuming we have a directed graph represented with an adjacency list.
'''

def breadth_first_search(graph, start):

    visited, queue = set(), deque(start)

    while queue:
        curr_node = queue.popleft()

        visited.add(curr_node)

        for neighbor in graph[curr_node]:
            if neighbor not in visited:
               queue.append(neighbor)
```

DFS vs BFS

- BFS is better for shortest path algorithms
- DFS is better for analyzing structure of graphs
- Generally, both BFS and DFS can be used to solve most problems

Working with Graph Problems

- Have a pen and paper for diagramming
- Work with one node at a time to build up a solution

Walkthrough

Walkthrough: Find Provinces

There are `n` cities. Some of them are connected, while some are not.

If city `a` is connected directly with city `b`, and city `b` is connected directly with city `c`, then city `a` is connected indirectly with city `c`.

A province is a group of directly or indirectly connected cities and no other cities outside of the group.

You are given an `n x n` matrix `isConnected` where `isConnected[i][j] = 1` if the `ith` city and the `jth` city are directly connected, and `isConnected[i][j] = 0` otherwise.

Walkthrough: Find Provinces

There are `n` cities. Some of them are connected, while some are not.

If city `a` is connected directly with city `b`, and city `b` is connected directly with city `c`, then city `a` is connected indirectly with city `c`.

A province is a group of directly or indirectly connected cities and no other cities outside of the group.

You are given an `n x n` matrix `isConnected` where `isConnected[i][j] = 1` if the `ith` city and the `jth` city are directly connected, and `isConnected[i][j] = 0` otherwise.

Adjacency matrix😲!!

Example

There are `n` cities. Some of them are connected, while some are not.

If city `a` is connected directly with city `b`, and city `b` is connected directly with city `c`, then city `a` is connected indirectly with city `c`.

A province is a group of directly or indirectly connected cities and no other cities outside of the group.

You are given an `n x n` matrix `isConnected` where `isConnected[i][j] = 1` if the `ith` city and the `jth` city are directly connected, and `isConnected[i][j] = 0` otherwise.

Input

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

Output

2

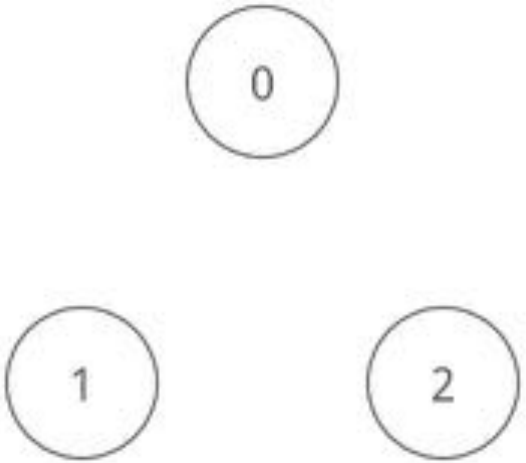
Example

Input

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

Output

2



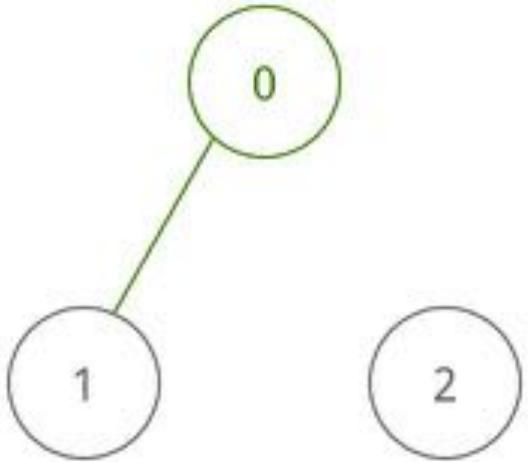
Example

Input

```
[[1,1,0], # city 0
 [1,1,0],
 [0,0,1]]
```

Output

2



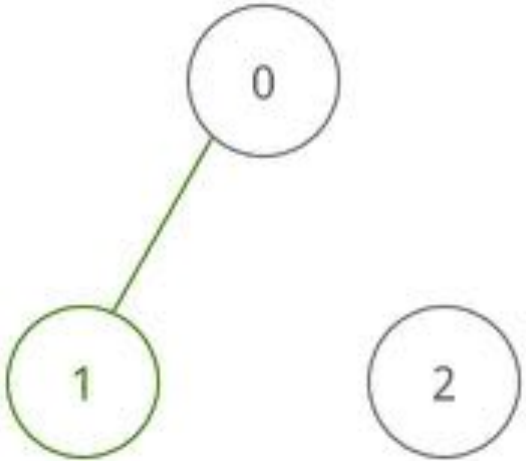
Example

Input

```
[[1,1,0],
 [1,1,0], # city 1
 [0,0,1]]
```

Output

2



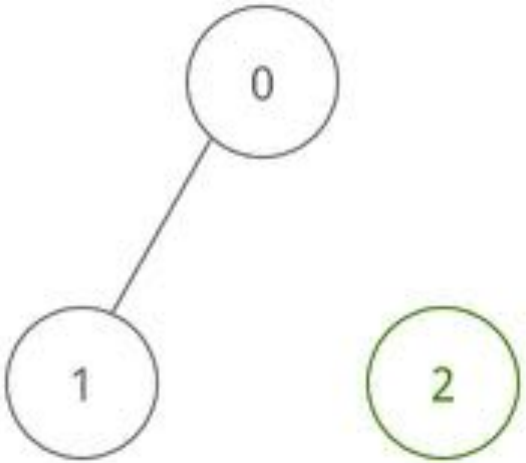
Example

Input

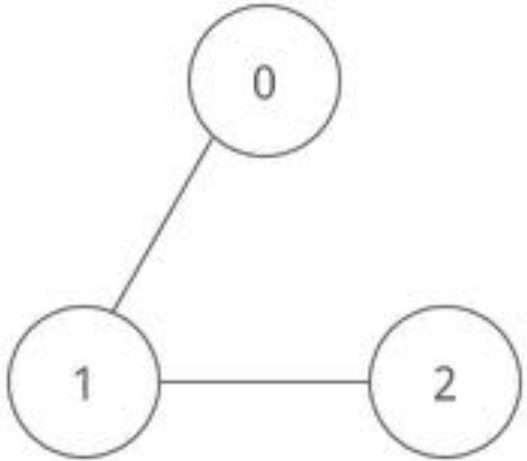
```
[[1,1,0],
 [1,1,0],
 [0,0,1]] # city 2
```

Output

2



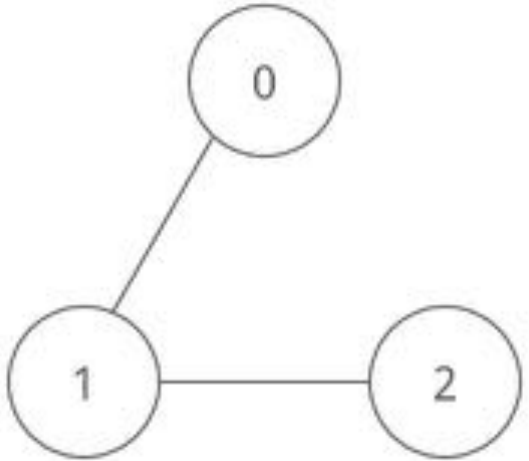
Understand



Understand

Input

`[1,1,0]`

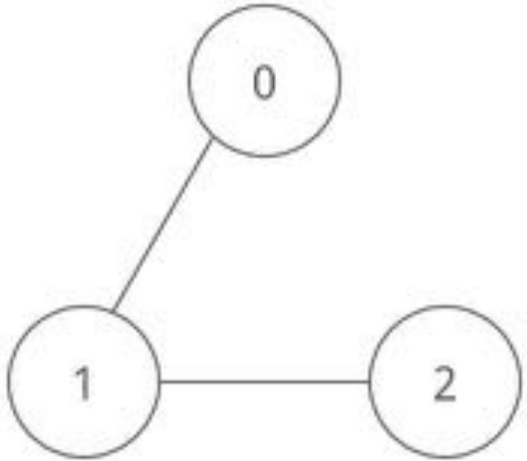


Understand

Input

[1,1,0]

[1,1,1]



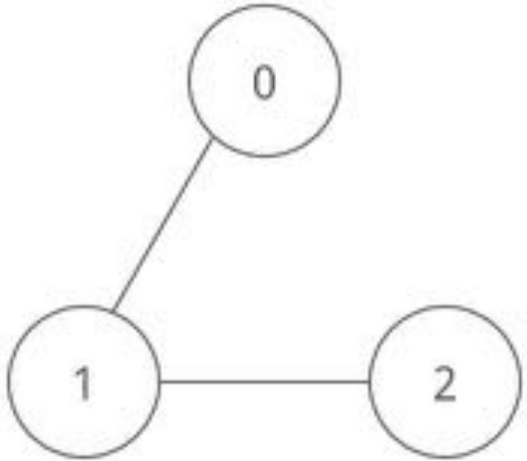
Understand

Input

[1,1,0]

[1,1,1]

[0,1,1]



Understand

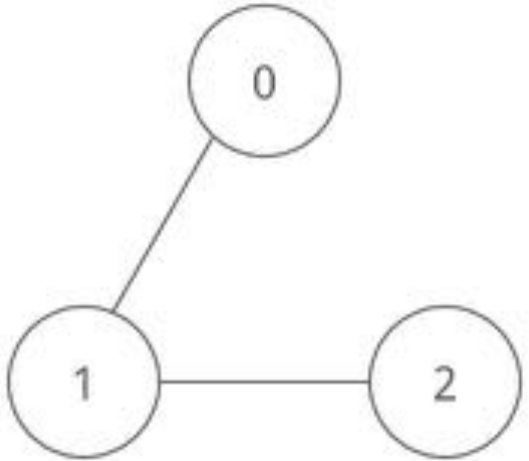
Input

`[1,1,0]`

`[1,1,1]`

`[0,1,1]`

Output 1



Understand

Input / Output

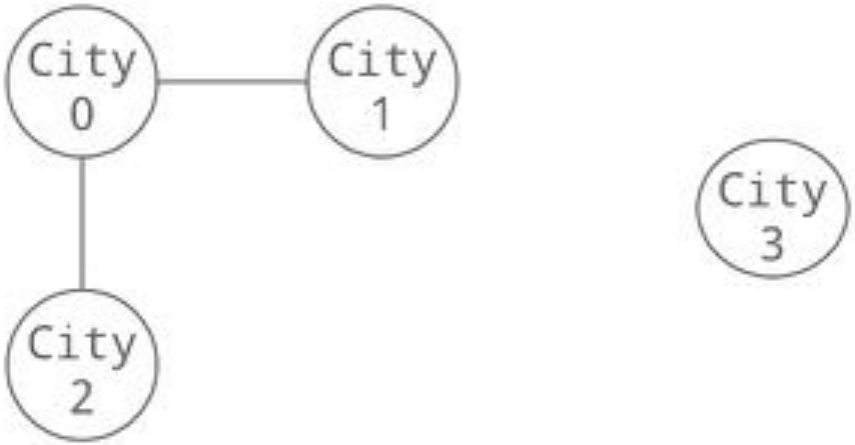
- Case #1: No city is connected to each other
- Case #2: Matrix with all 1s returns 1
- Other cases: use the ones we just created

Match

Graph algorithms

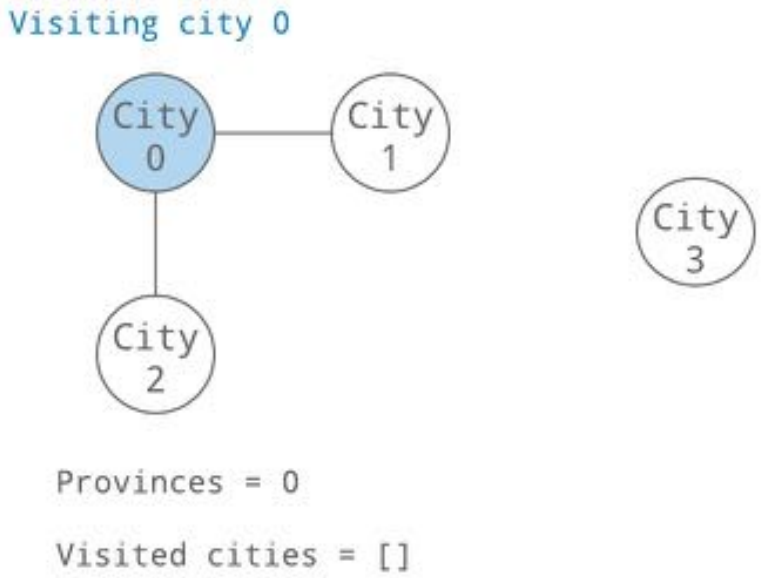
- BFS?
- DFS?

Plan



Plan

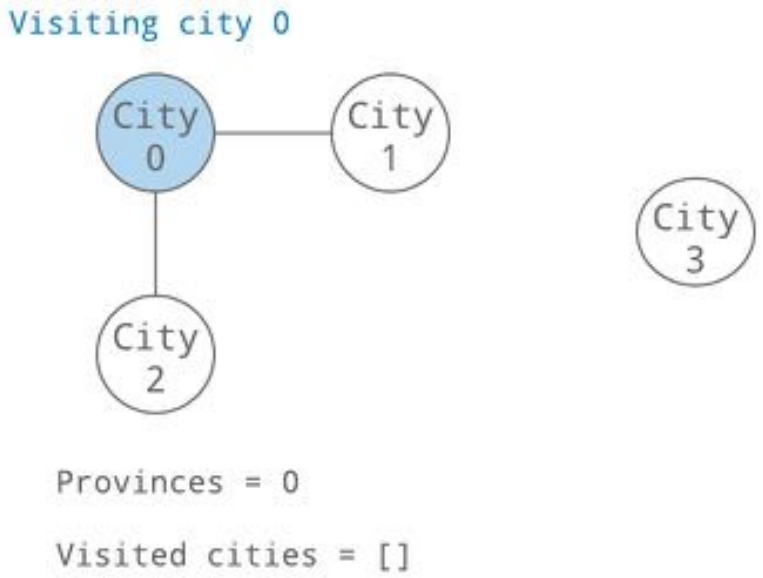
Start with exploring city 0



Plan

Start with exploring city 0

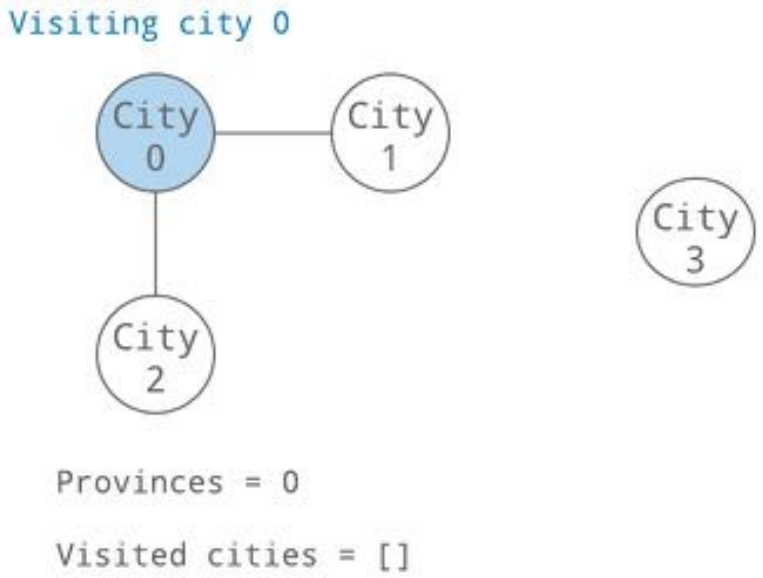
- **Have we visited this city before?**



Plan

Start with exploring city 0

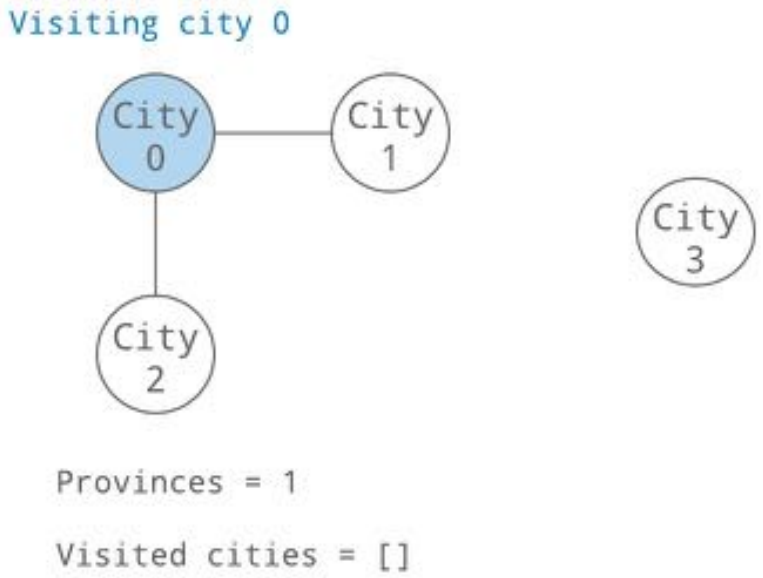
- Have we visited this city before?
- **If not, we're visiting a new province**



Plan

Start with exploring city 0

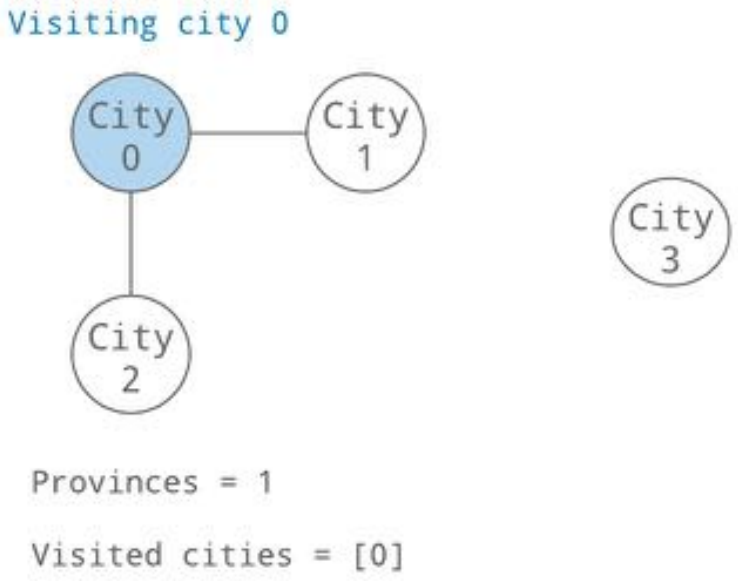
- Have we visited this city before?
- If not, we're visiting a new province
    - **Increment province count by 1**



Plan

Start with exploring city 0

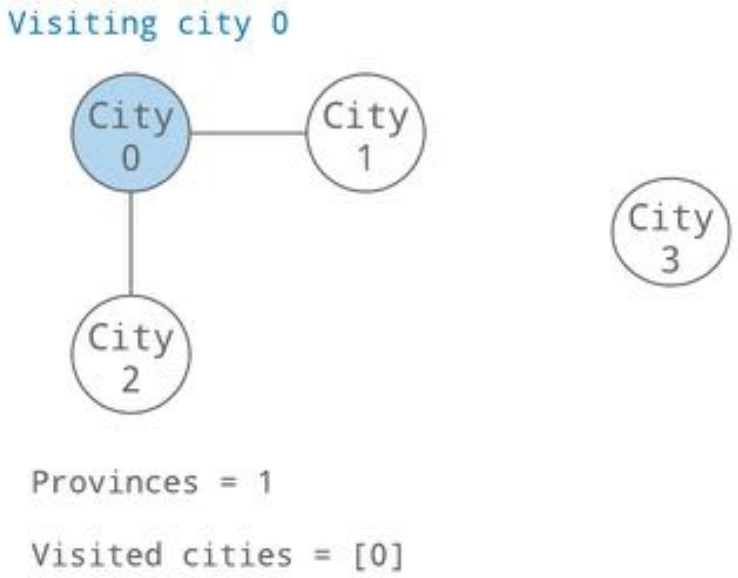
- Have we visited this city before?
- If not, we're visiting a new province
  - Increment province count by 1
  - **Put city 0 in the visited set**



Plan

Start with exploring city 0

- Have we visited this city before?
- If not, we're visiting a new province
  - Increment province count by 1
  - Put city 0 in the visited set
  - **Fully explore the rest of this province**

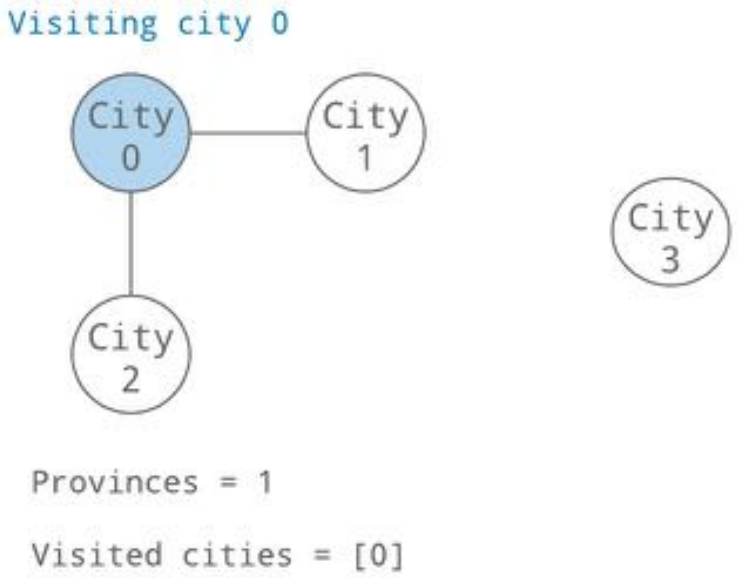


Plan

Start with exploring city 0

- Have we visited this city before?
- If not, we're visiting a new province
  - Increment province count by 1
  - Put city 0 in the visited set
  - Fully explore the rest of this province

Fully explore the province starting at city 0



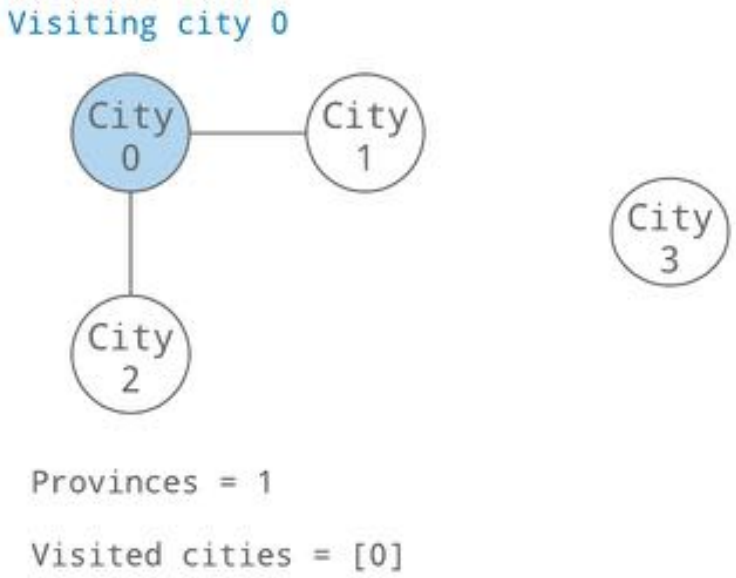
Plan

Start with exploring city 0

- Have we visited this city before?
- If not, we're visiting a new province
  - Increment province count by 1
  - Put city 0 in the visited set
  - Fully explore the rest of this province

Fully explore the province starting at city 0

- **Get the cities that city 0 is connected to**



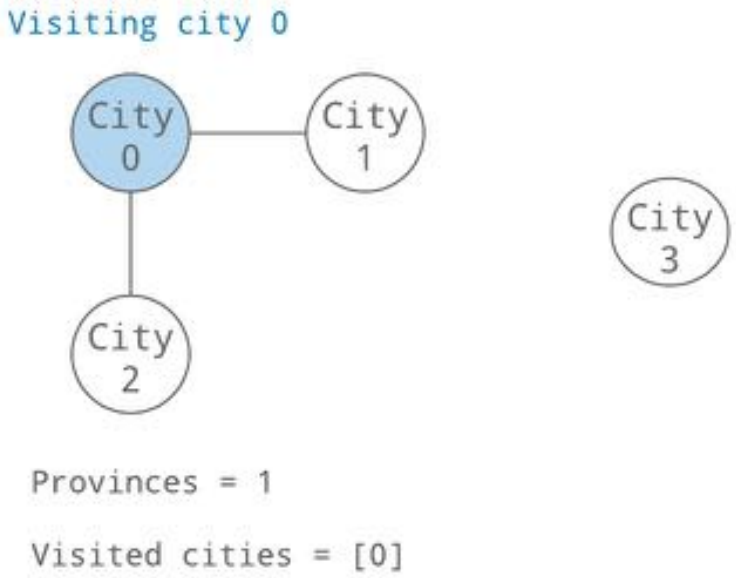
Plan

Start with exploring city 0

- Have we visited this city before?
- If not, we're visiting a new province
  - Increment province count by 1
  - Put city 0 in the visited set
  - Fully explore the rest of this province

Fully explore the province starting at city 0

- Get the cities that city 0 is connected to
- **For each of these cities connected to city 0**



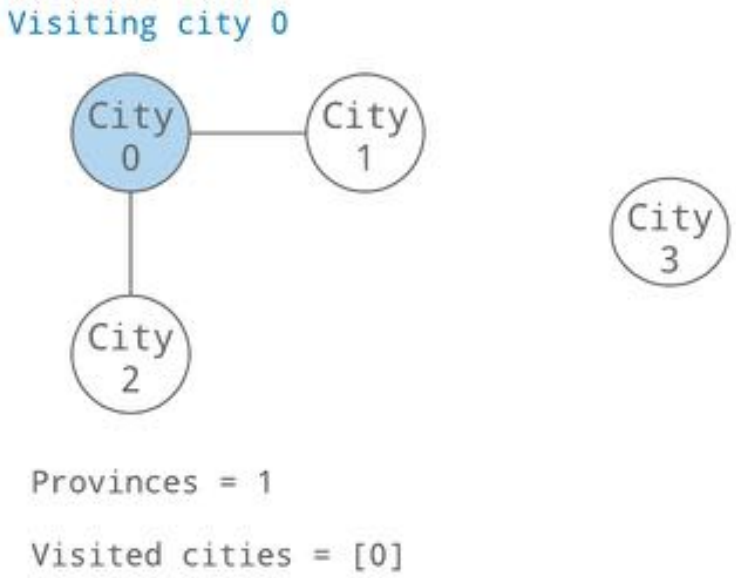
Plan

Start with exploring city 0

- Have we visited this city before?
- If not, we're visiting a new province
  - Increment province count by 1
  - Put city 0 in the visited set
  - Fully explore the rest of this province

Fully explore the province starting at city 0

- Get the cities that city 0 is connected to
- For each of these cities connected to city 0
  - **Skip the city if it has been visited**



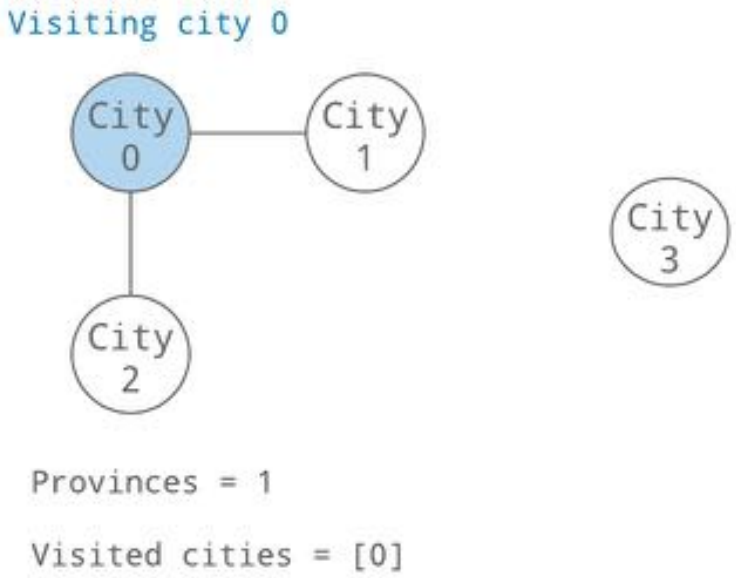
Plan

Start with exploring city 0

- Have we visited this city before?
- If not, we're visiting a new province
  - Increment province count by 1
  - Put city 0 in the visited set
  - Fully explore the rest of this province

Fully explore the province starting at city 0

- Get the cities that city 0 is connected to
- For each of these cities connected to city 0
  - Skip the city if it has been visited
  - Mark the city as visited



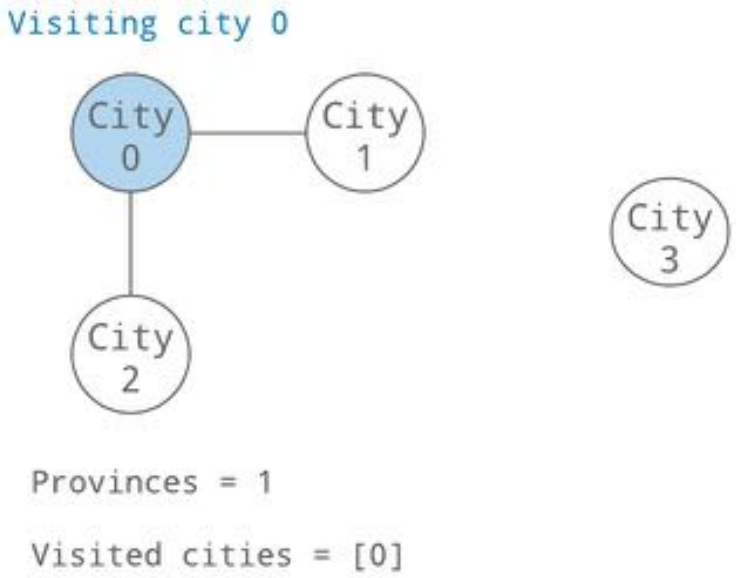
Plan

Start with exploring city 0

- Have we visited this city before?
- If not, we're visiting a new province
  - Increment province count by 1
  - Put city 0 in the visited set
  - Fully explore the rest of this province

Fully explore the province starting at city 0

- Get the cities that city 0 is connected to
- For each of these cities connected to city 0
  - Skip the city if it has been visited
  - Mark the city as visited
  - Repeat this process on the new city



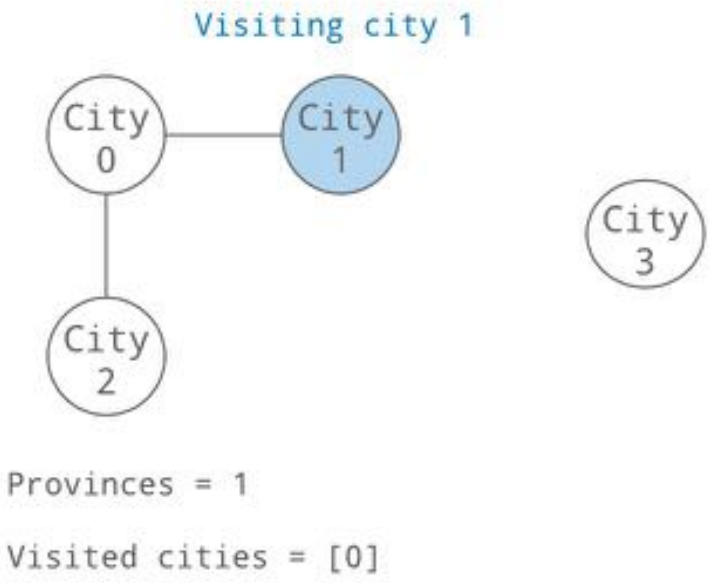
Plan

Start with exploring city 0

- Have we visited this city before?
- If not, we're visiting a new province
  - Increment province count by 1
  - Put city 0 in the visited set
  - Fully explore the rest of this province

Fully explore the province starting at city 0

- Get the cities that city 0 is connected to
- **For each of these cities connected to city 0**
  - Skip the city if it has been visited
  - Mark the city as visited
  - Repeat this process on the new city



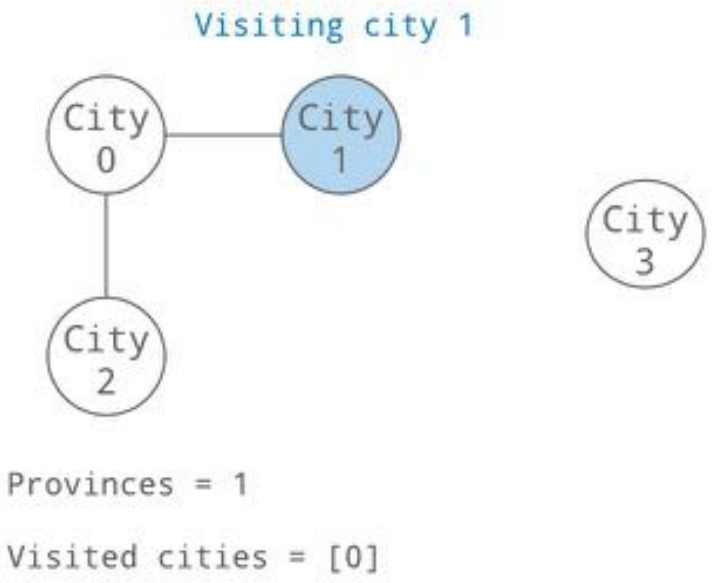
Plan

Start with exploring city 0

- Have we visited this city before?
- If not, we're visiting a new province
  - Increment province count by 1
  - Put city 0 in the visited set
  - Fully explore the rest of this province

Fully explore the province starting at city 0

- Get the cities that city 0 is connected to
- For each of these cities connected to city 0
  - **Skip the city if it has been visited**
  - Mark the city as visited
  - Repeat this process on the new city



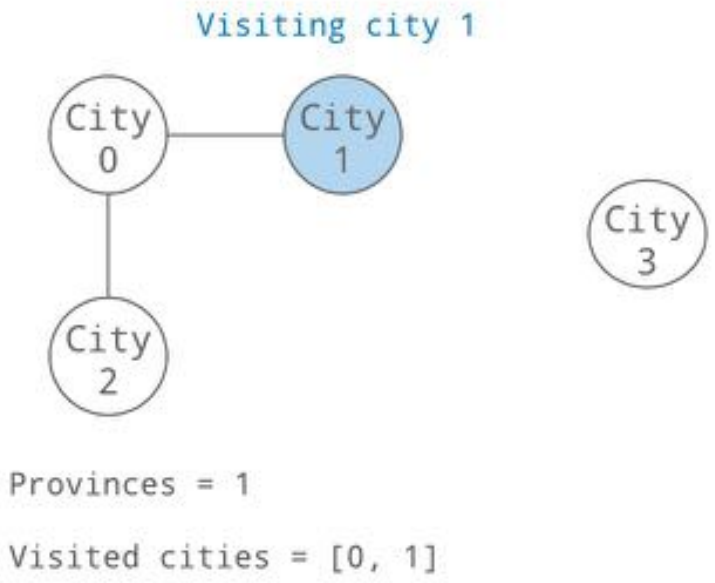
Plan

Start with exploring city 0

- Have we visited this city before?
- If not, we're visiting a new province
  - Increment province count by 1
  - Put city 0 in the visited set
  - Fully explore the rest of this province

Fully explore the province starting at city 0

- Get the cities that city 0 is connected to
- For each of these cities connected to city 0
  - Skip the city if it has been visited
  - **Mark the city as visited**
  - Repeat this process on the new city



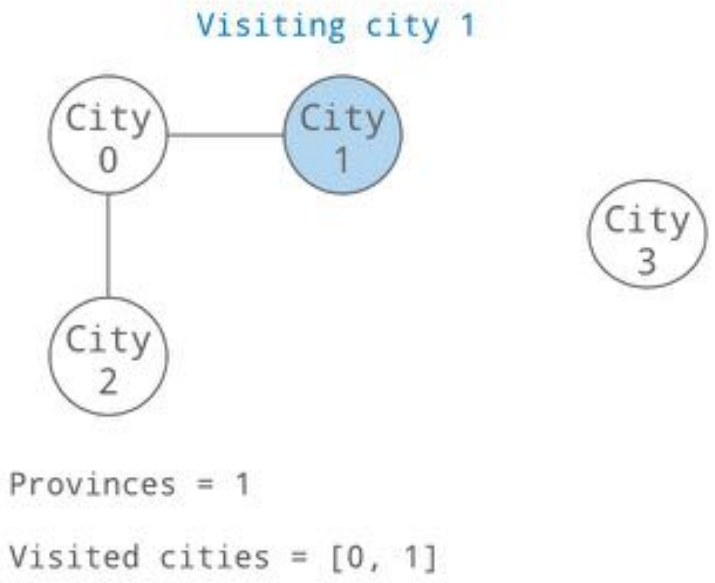
Plan

Start with exploring city 0

- Have we visited this city before?
- If not, we're visiting a new province
  - Increment province count by 1
  - Put city 0 in the visited set
  - Fully explore the rest of this province

Fully explore the province starting at city 0

- Get the cities that city 0 is connected to
- For each of these cities connected to city 0
  - Skip the city if it has been visited
  - Mark the city as visited
  - **Repeat this process on the new city (city 1)**



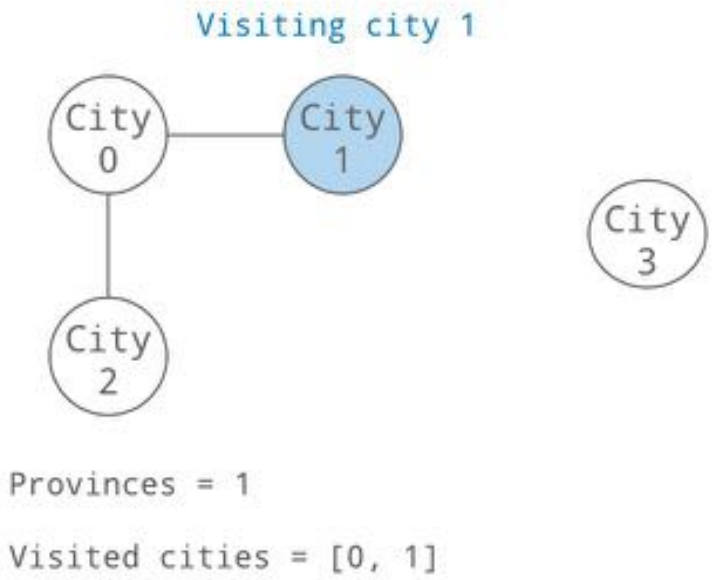
Plan

Start with exploring city 0

- Have we visited this city before?
- If not, we're visiting a new province
  - Increment province count by 1
  - Put city 0 in the visited set
  - Fully explore the rest of this province

Fully explore the province starting at city 1

- Get the cities that **city 1** is connected to
- For each of these cities connected to **city 1**
  - Skip the city if it has been visited
  - Mark the city as visited
  - Repeat this process on the new city



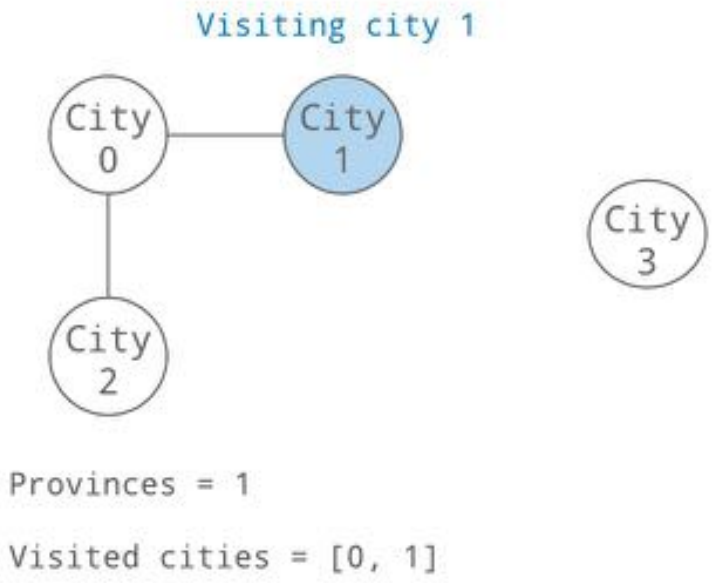
Plan

Start with exploring city 0

- Have we visited this city before?
- If not, we're visiting a new province
  - Increment province count by 1
  - Put city 0 in the visited set
  - Fully explore the rest of this province

Fully explore the province starting at city 1

- Get the cities that **city 1** is connected to
- For each of these cities connected to **city 1**
  - Skip the city if it has been visited
  - Mark the city as visited
  - Repeat this process on the new city



We've finished exploring city 1!

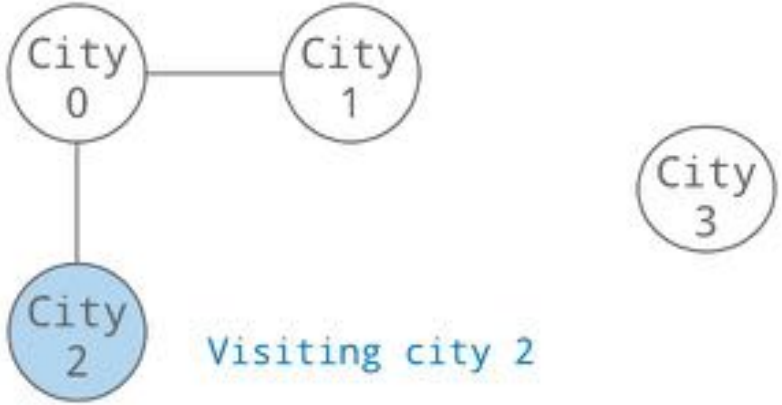
Plan

Start with exploring city 0

- Have we visited this city before?
- If not, we're visiting a new province
  - Increment province count by 1
  - Put city 0 in the visited set
  - Fully explore the rest of this province

Fully explore the province starting at city 0

- Get the cities that **city 0** is connected to
- For each of these cities connected to **city 0**
  - Skip the city if it has been visited
  - Mark the city as visited
  - **Repeat this process on the new city (city 2)**




```
Provinces = 1

Visited cities = [0,1,2]
```

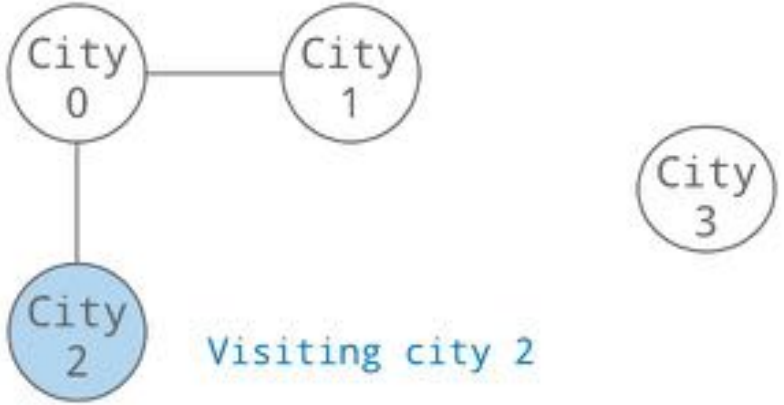
Plan

Start with exploring city 0

- Have we visited this city before?
- If not, we're visiting a new province
  - Increment province count by 1
  - Put city 0 in the visited set
  - Fully explore the rest of this province

Fully explore the province starting at city 2

- Get the cities that **city 2** is connected to
- For each of these cities connected to **city 2**
  - Skip the city if it has been visited
  - Mark the city as visited
  - **Repeat this process on the new city**




```
Provinces = 1

Visited cities = [0,1,2]
```

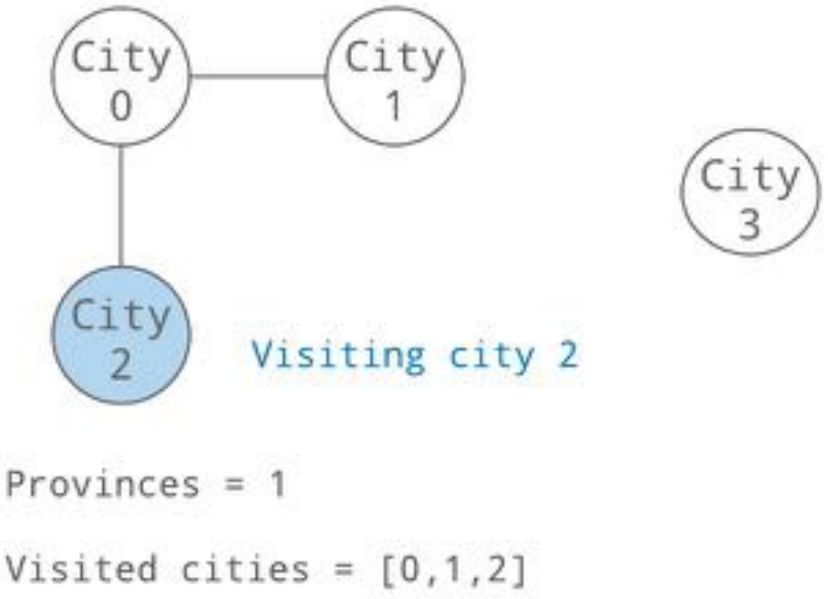
Plan

Start with exploring city 0

- Have we visited this city before?
- If not, we're visiting a new province
  - Increment province count by 1
  - Put city 0 in the visited set
  - Fully explore the rest of this province

Fully explore the province starting at city 2

- Get the cities that **city 2** is connected to
- For each of these cities connected to **city 2**
  - Skip the city if it has been visited
  - Mark the city as visited
  - **Repeat this process on the new city**



We've finished exploring city 2!

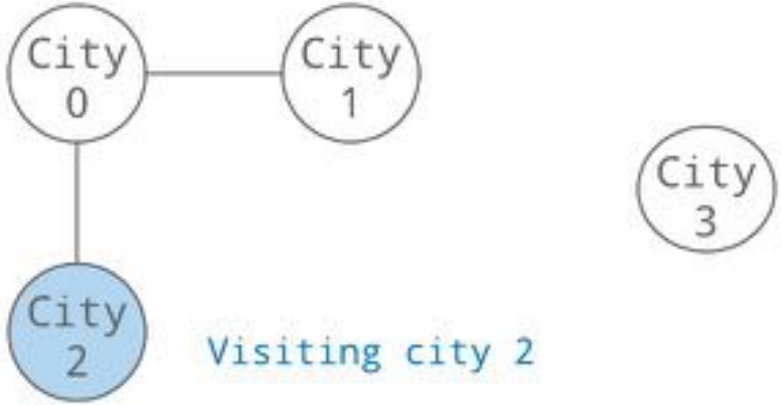
Plan

Start with exploring city 0

- Have we visited this city before?
- If not, we're visiting a new province
  - Increment province count by 1
  - Put city 0 in the visited set
  - Fully explore the rest of this province

Fully explore the province starting at city 0

- Get the cities that **city 0** is connected to
- For each of these cities connected to **city 0**
  - Skip the city if it has been visited
  - Mark the city as visited
  - **Repeat this process on the new city**




```
Provinces = 1

Visited cities = [0,1,2]
```

We've finished exploring city 0 and the whole province!

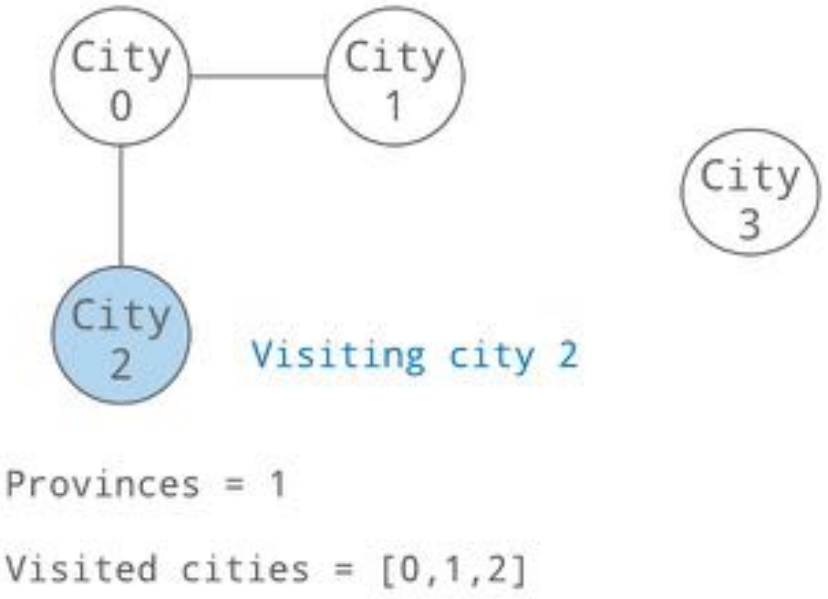
Plan

Start with exploring city 0

- Have we visited this city before?
- If not, we're visiting a new province
  - Increment province count by 1
  - Put city 0 in the visited set
  - Fully explore the rest of this province

Fully explore the province

- Get the cities that city 0 is connected to
- For each of these cities connected to city 0
  - Skip the city if it has been visited
  - Mark the city as visited
  - **Repeat this process on the new city**



Repeat this whole process for every city to make sure we cover all cities!

Plan

Start with exploring a city

- Have we visited this city before?
- If not, we're visiting a new province
    - Increment province count by 1
    - Put the city in the visited set
    - Fully explore the rest of this province

Plan

Start with exploring a city

- Have we visited this city before?
- If not, we're visiting a new province
  - Increment province count by 1
  - Put the city in the visited set
  - Fully explore the rest of this province

```
for each city:
    if city is not visited:
        provinces += 1
        add city to visited
        dfs(city)
```

Plan

Start with exploring a city

- Have we visited this city before?
- If not, we're visiting a new province
  - Increment province count by 1
  - Put the city in the visited set
  - Fully explore the rest of this province

Fully explore the province

- Get the cities that starting city is connected to
- For each of these cities connected to starting city
  - Skip the city if it has been visited
  - Mark the city as visited
  - Repeat this process on the new city

```
for each city:
    if city is not visited:
        provinces += 1
        add city to visited
        dfs(city)
```

Plan

Start with exploring a city

- Have we visited this city before?
- If not, we're visiting a new province
  - Increment province count by 1
  - Put the city in the visited set
  - Fully explore the rest of this province

Fully explore the province

- Get the cities that starting city is connected to
- For each of these cities connected to starting city
  - Skip the city if it has been visited
  - Mark the city as visited
  - Repeat this process on the new city

```
for each city:
    if city is not visited:
        provinces += 1
        add city to visited
        dfs(city)


dfs(city):
    For every other city:
        if the other city is connected:
            if other city in visited:
                skip
            add other city to visited
            dfs(other city)
```

Runtime Analysis

- N = number of cities
- **Space complexity**: O(N)
  - Max size of the visited set is the number of cities

Runtime Analysis

- N = number of cities
- **Space complexity**: O(N)
  - Max size of the visited set is the number of cities
- **Time complexity**: $O(N^2)$
  - Traversing through the N by N matrix to find all the connected cities

Breakout session

In class exercise instructions

- Questions
  - Keys and Rooms
  - Maximal Network Rank
  - Bonus: Accounts Merge
- Work together as a group

Helpful Links

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