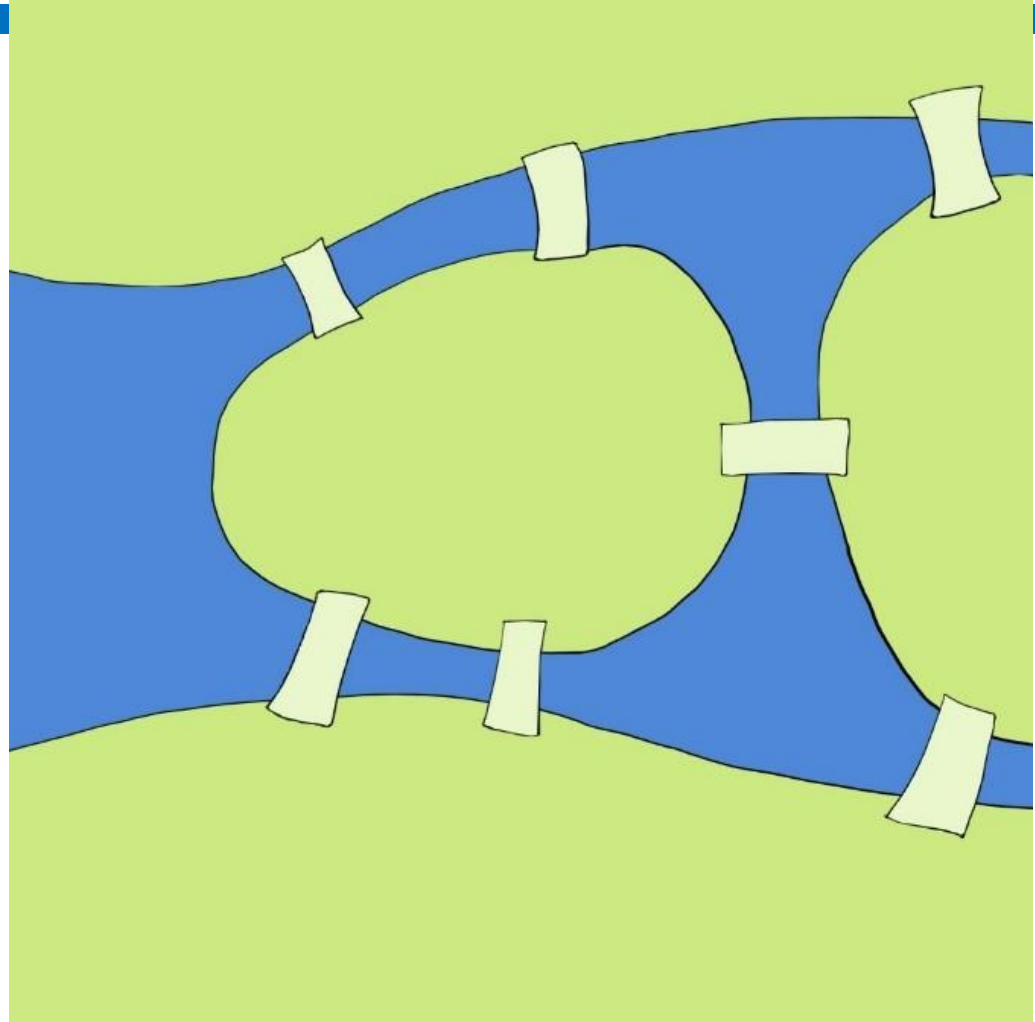


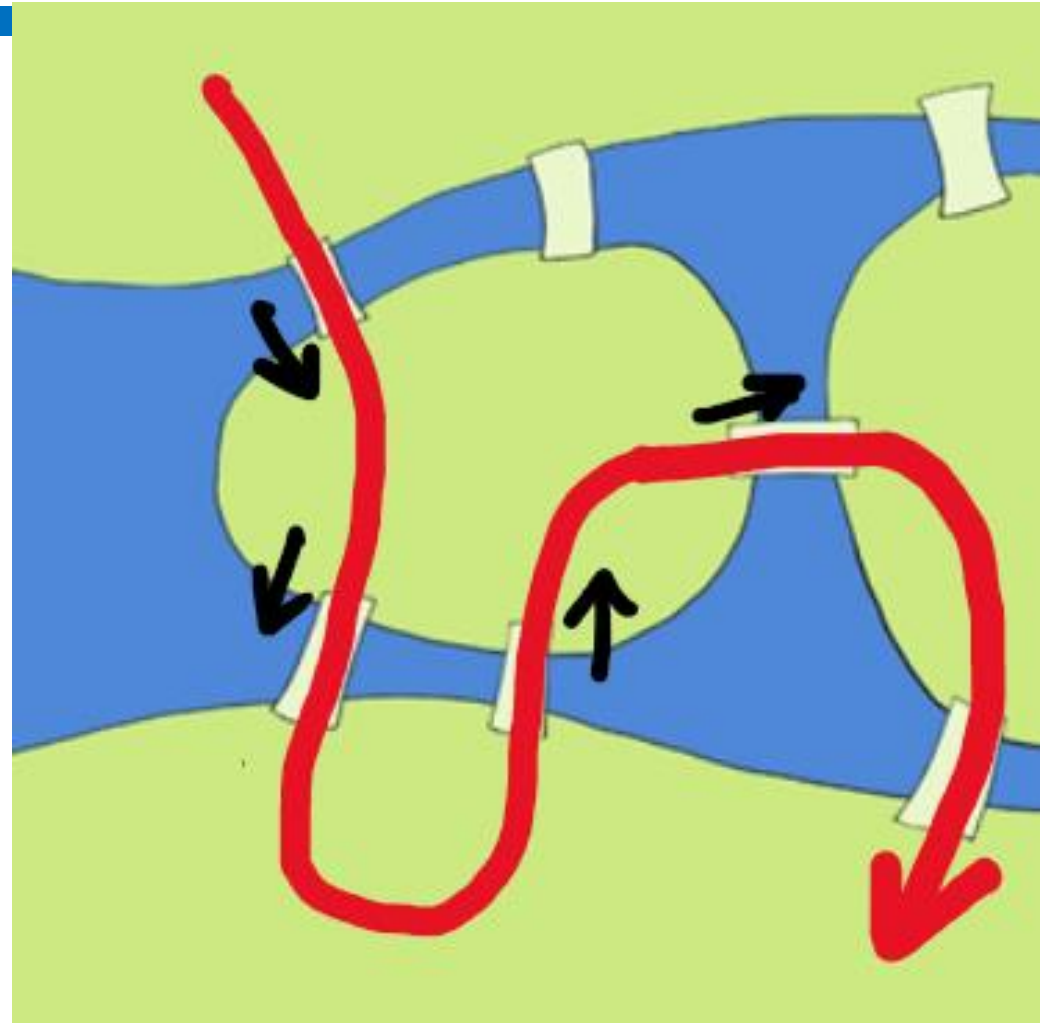
# A Puzzle

- Find a way to walk around this town, passing through each bridge exactly once
- Can start and end anywhere
- Can't go out of town
- Can't walk on water



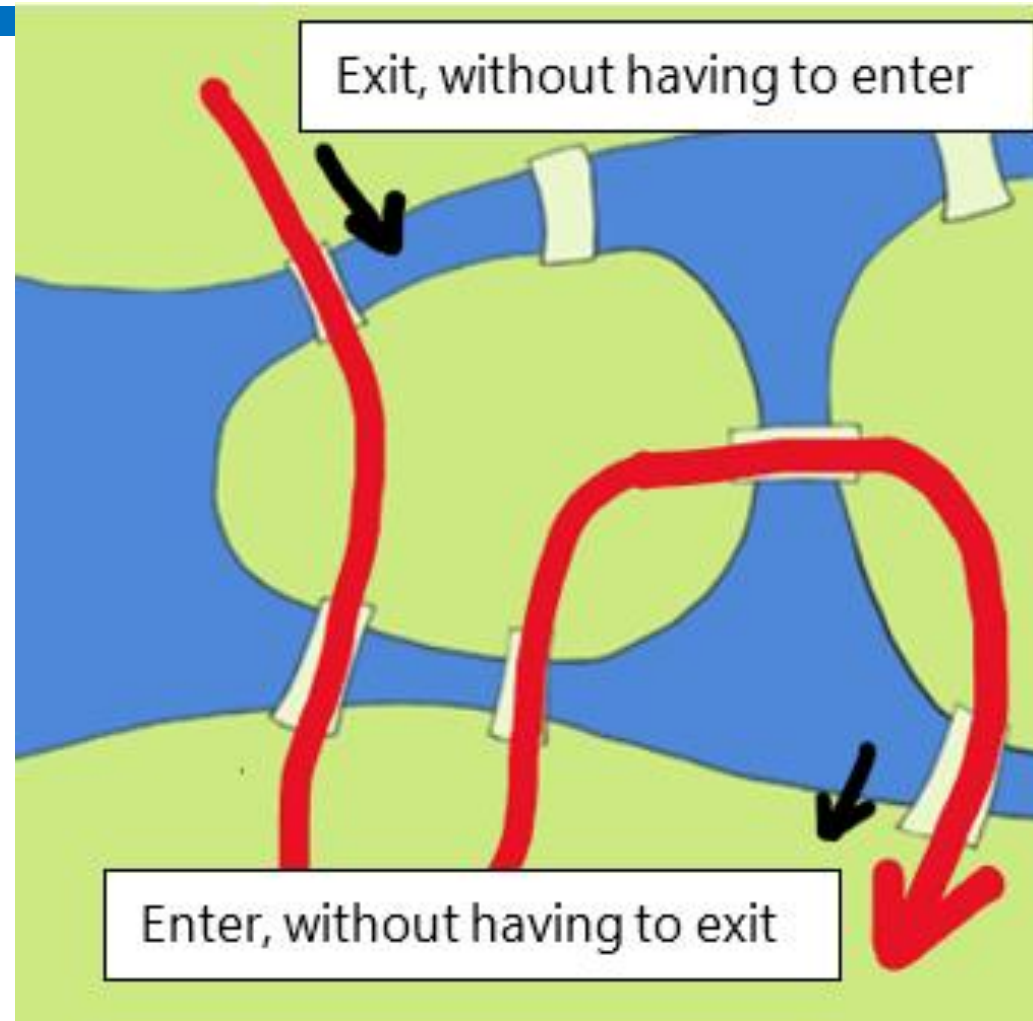
## Impossible? Why?

- Whenever you enter an island by a bridge, you must exit by a different bridge
- Only possible if all islands have an even number of bridges!



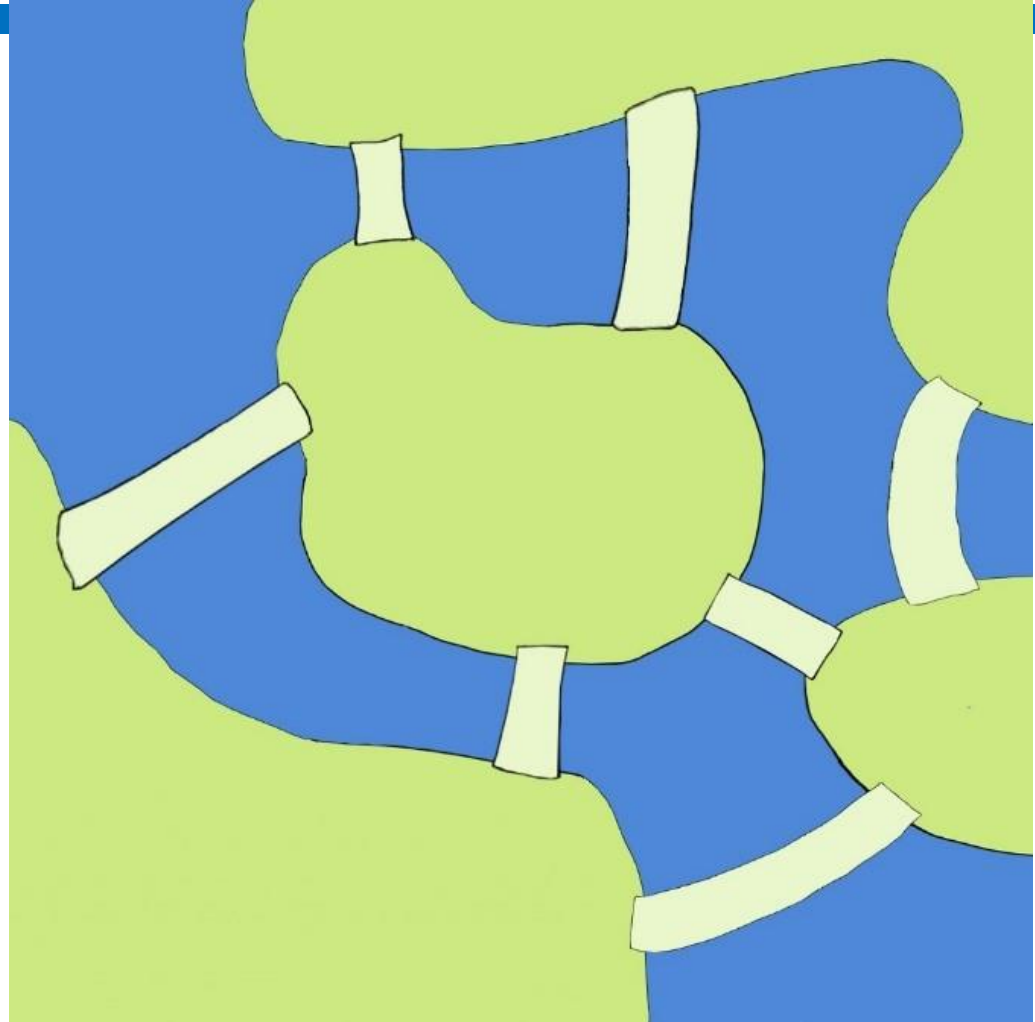
## Impossible? Why?

- Exception: starting and ending islands can have an odd number of bridges if they are different
- Only possible if the number of “odd islands” is 2 or 0
  - In fact, always possible if this is the case, though harder to prove



# A “New” Puzzle

- Is there a way to pass through each bridge exactly once?
- No, for the exact same reason as earlier
- Even more, this map is *essentially* the same as the one before

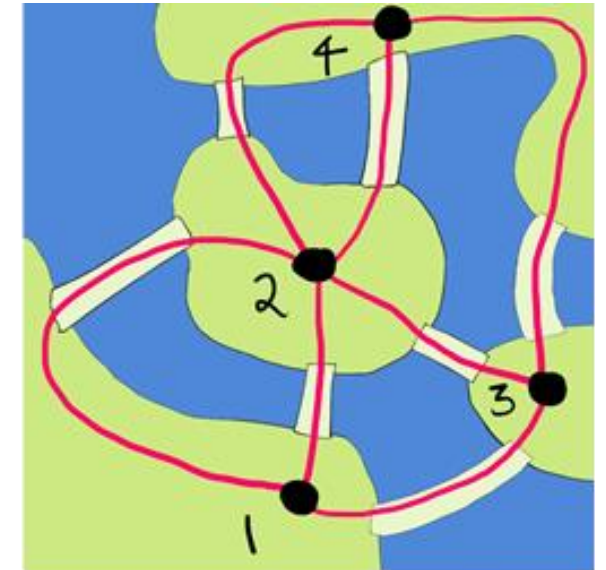
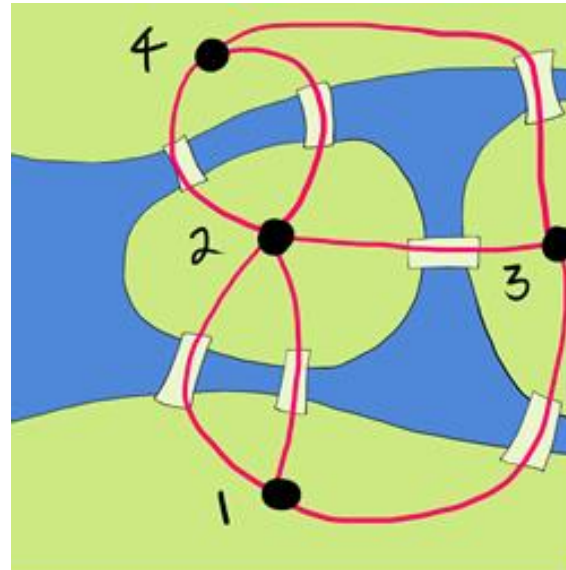


# What Makes Them “Essentially” the Same

- Shapes of landmasses and bridges have changed, but none of that is important to the problem or our argument
- What is important about these maps?
  - There is a set of islands
  - There is a set of bridges
  - Which islands does each bridge connect?
  - The number of bridges connected to each island
    - Can be inferred from previous information

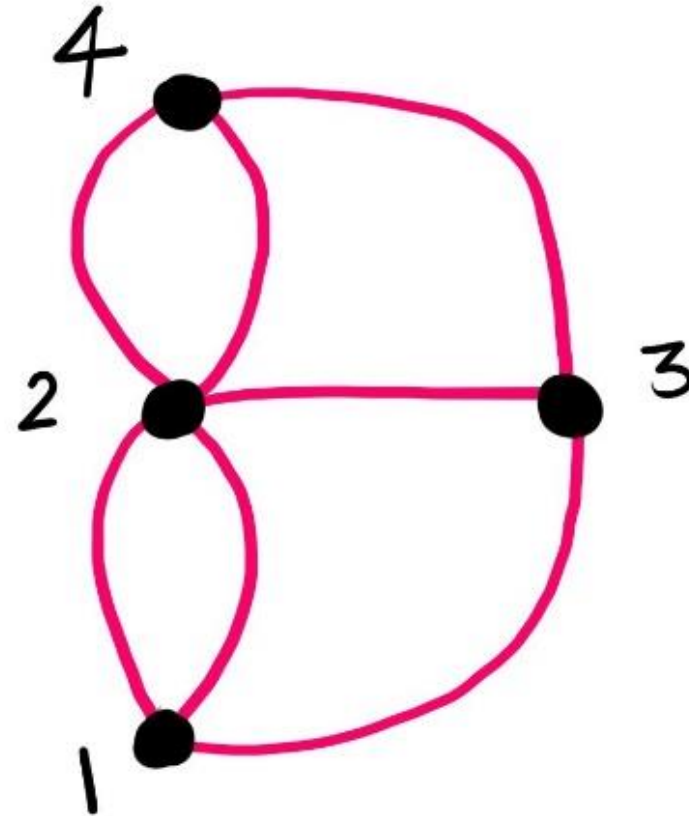
## Keep Only the Important Info

- 4 islands, labeled 1 to 4 (labels chosen carefully)
- 7 bridges
  - Two connecting 1 and 2
  - Two connecting 2 and 4
  - One connecting 1 and 3
  - One connecting 2 and 3
  - One connecting 3 and 4



## Keep Only the Important Info

- Forget about maps, towns, islands, bridges
- The truly important info:
  - Some set of **nodes**
  - How the nodes are *connected* to each other by **links**



# It's Your Birthday!

- You wish to invite your ten closest friends to your birthday party
  - And others, but the ten friends need to be there
- For simplicity, call them friend 1, friend 2, ..., friend 10
- You're too lazy to invite each one individually, but you know if you invite someone, they'll invite their friends, who'll invite their friends, who'll...



# And You Know Who Are Friends with Each Other

1 and 2 are friends

2 and 4 are friends

1 and 5 are friends

7 and 8 are friends

2 and 3 are friends

3 and 4 are friends

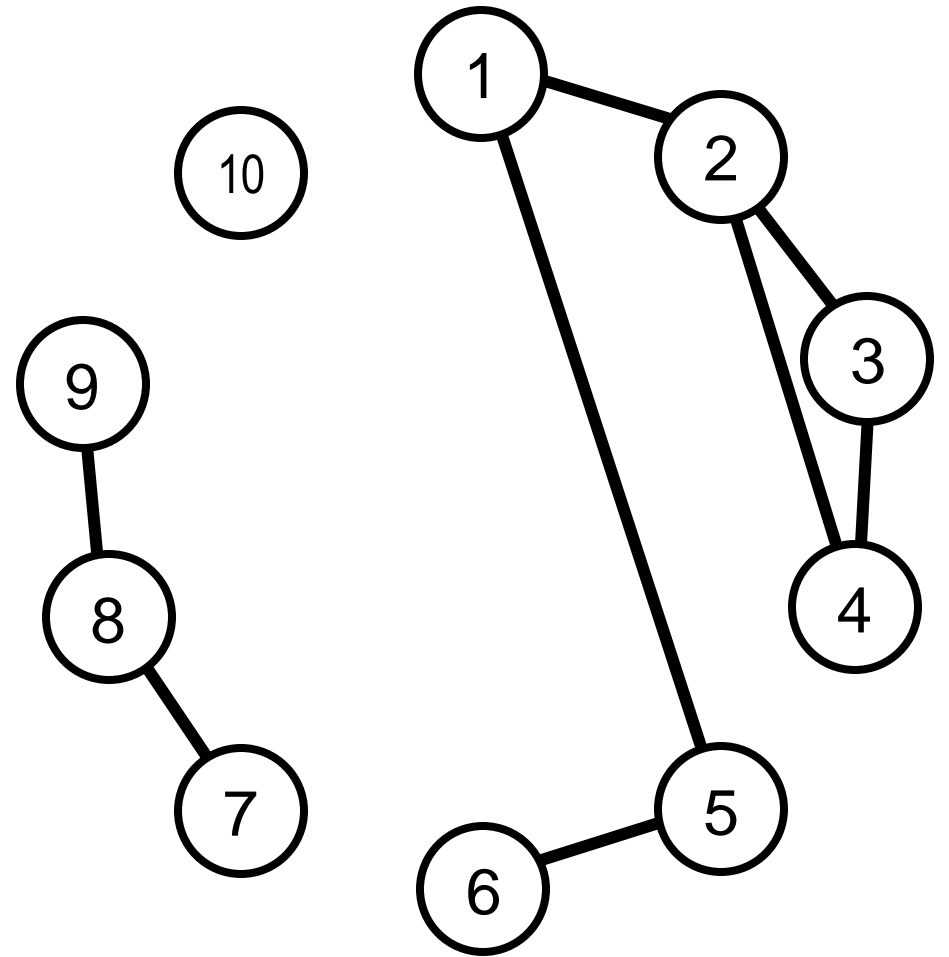
5 and 6 are friends

8 and 9 are friends

- Can you invite just one person so that all ten friends get invited?
- If not, what's the minimum number of people you need to invite personally?
- And whom?
- Hard to visualize relationships just with the list

# Circles To Represent People, Lines Between If They Are Friends

- Now it's easy to answer the questions!



# We Can Do This for Any Set of Friendships

1 and 6 are friends

4 and 10 are friends

8 and 5 are friends

5 and 3 are friends

7 and 8 are friends

3 and 7 are friends

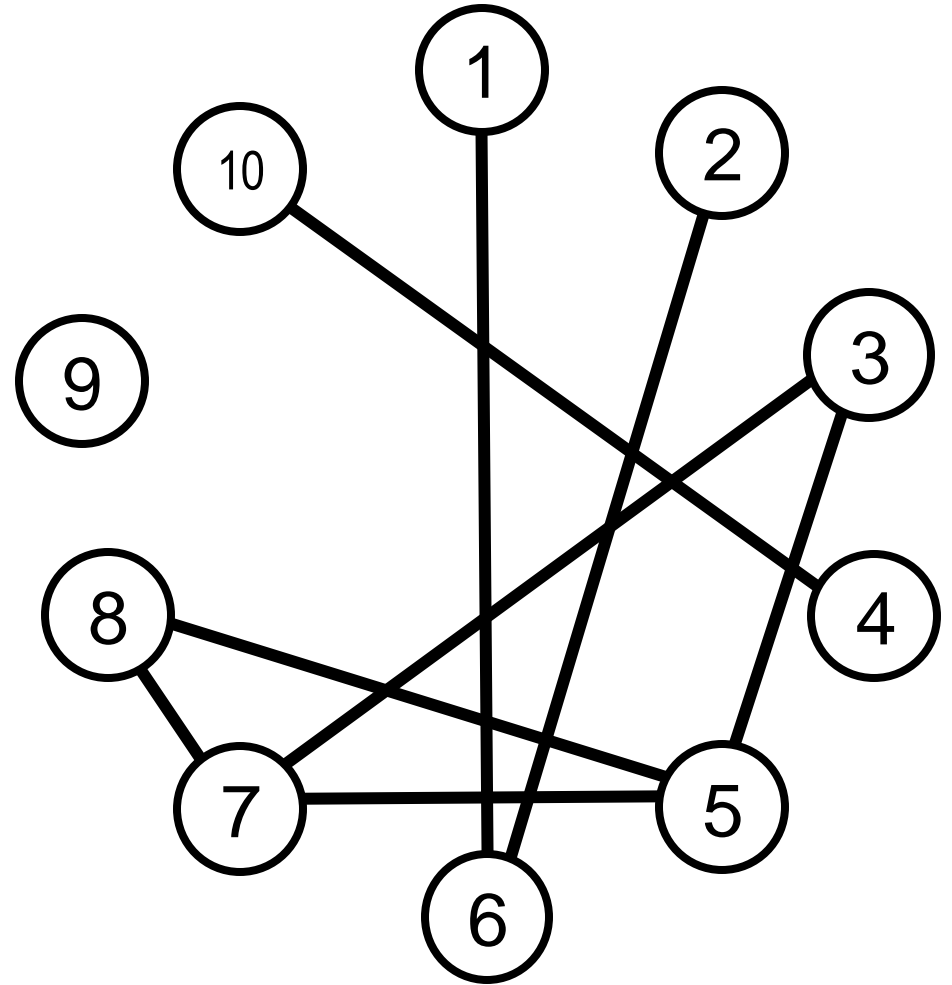
6 and 2 are friends

5 and 7 are friends

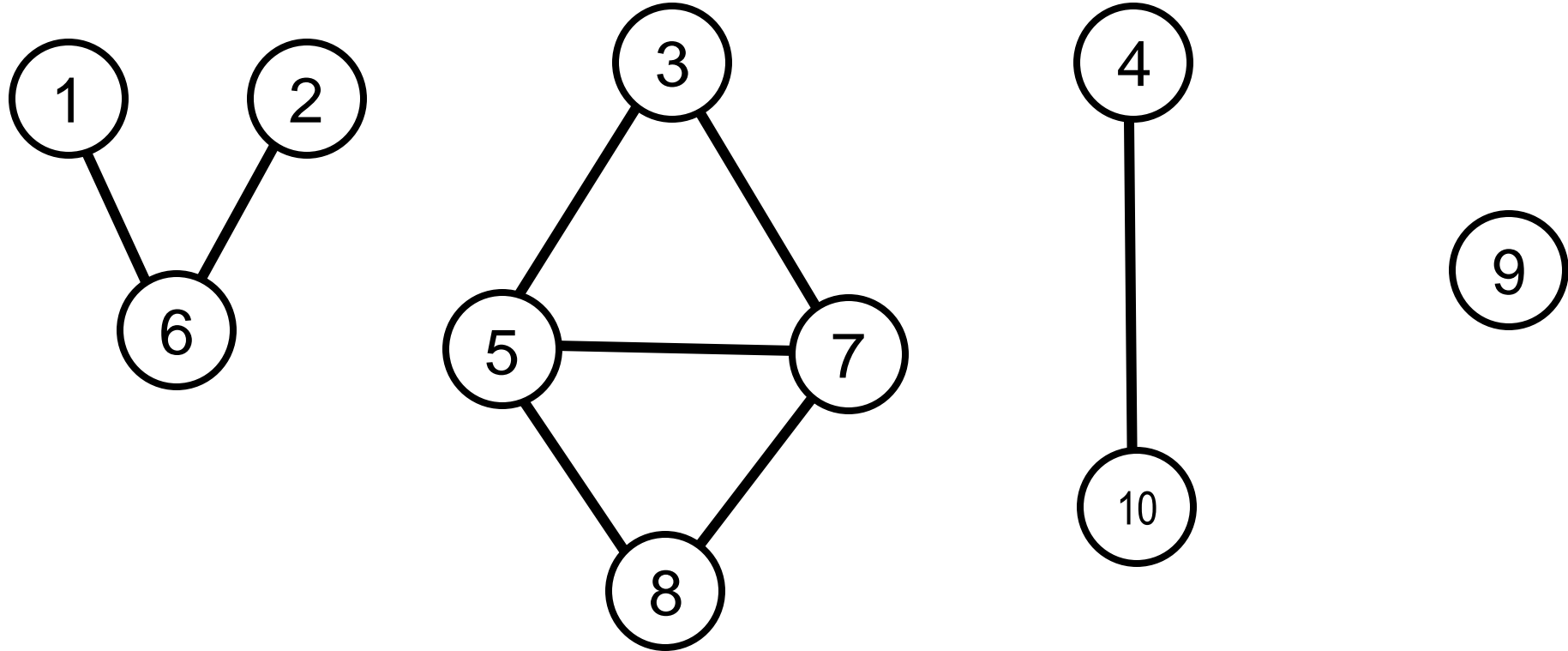
- Suppose these were the relationships between your best friends instead
- What's the minimum number of people you need to invite personally?
- And whom?

# We Can Do This for Any Set of Friendships

- This time, it's not so easy to see



# The Physical Locations of the Circles Aren't Important, As Long As the Relationships Are Preserved



- Now it's easy to answer the questions!

# Notice Anything Similar Between the Bridge Puzzle and the Birthday Problem?

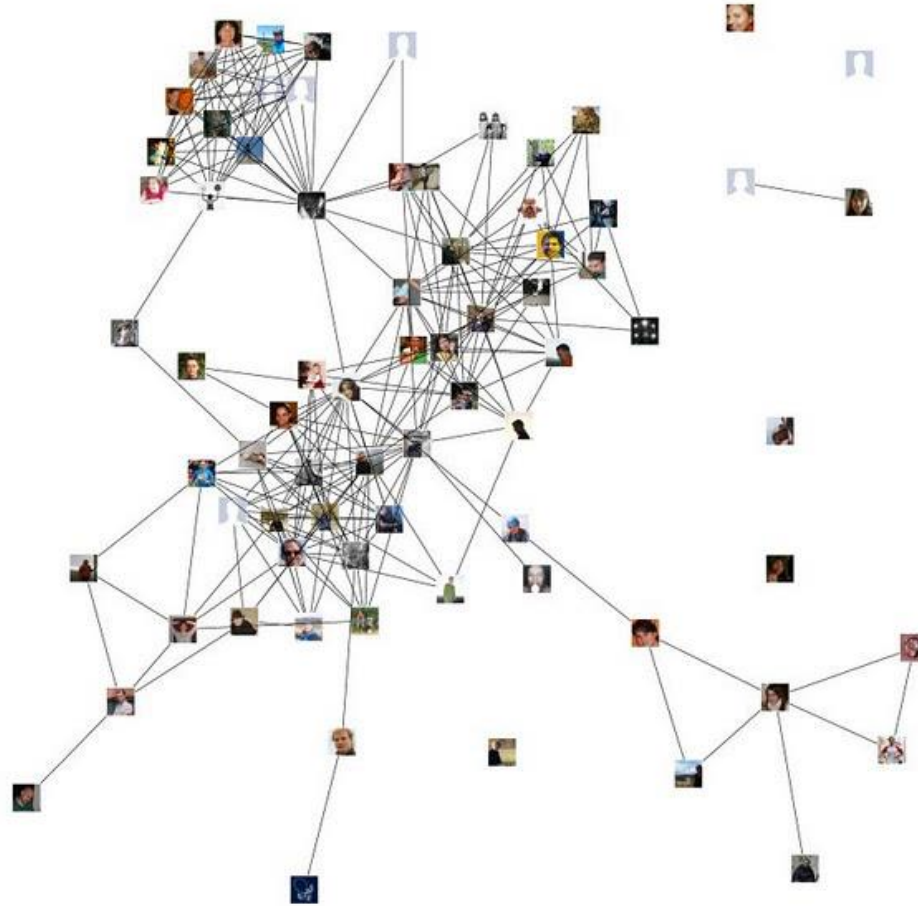
- Circles and lines!
- **Networks** with **nodes** and **links**
- Where else have you seen this?

# MANILA METRO RAIL TRANSIT MAP

www.seacitymaps.com



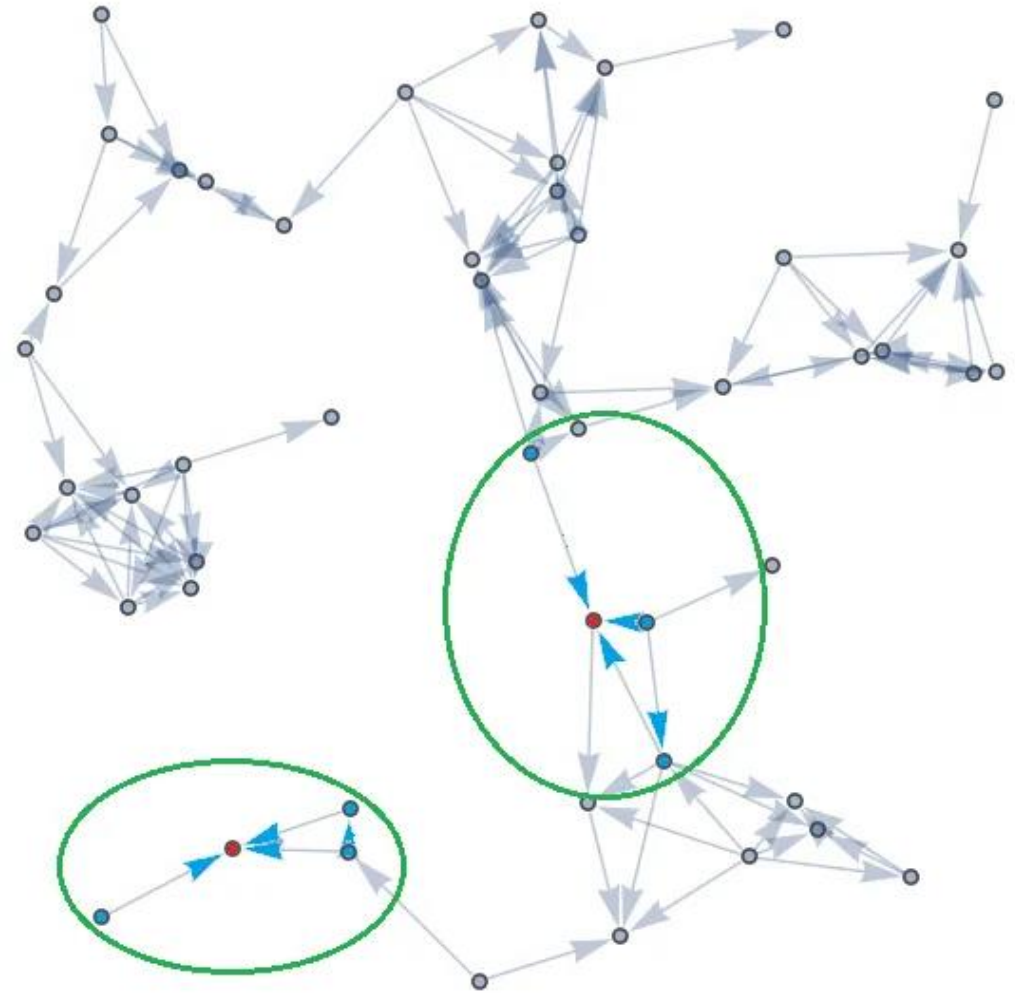
# Nodes and Links Don't Need to be Physical Objects and Connections: Facebook, For Example





## Friendship is Mutual

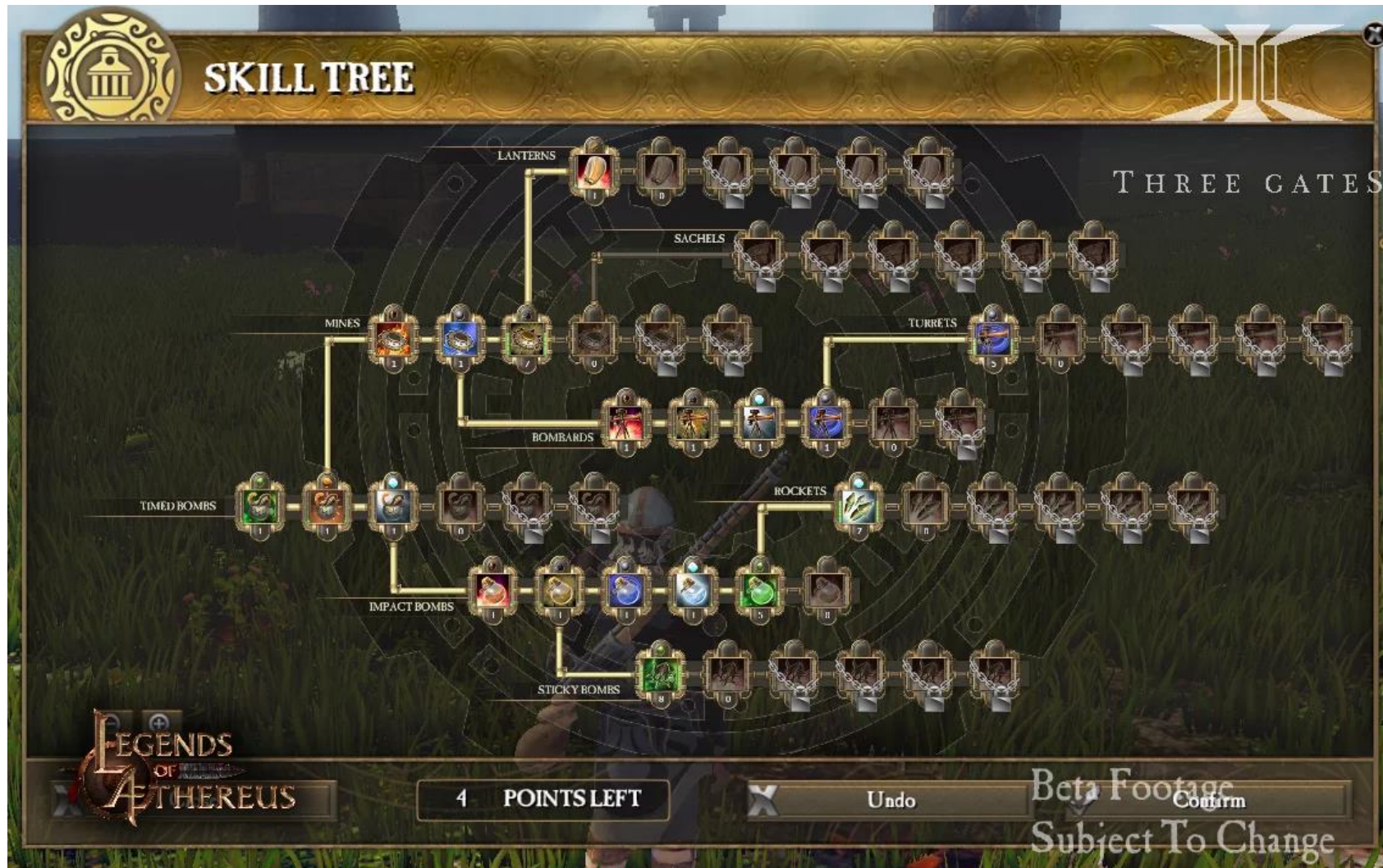
- But one-sided relationships also exist 😬
- We're talking about follow relationships in Instagram, of course



# Friendship is Mutual, But One-Sided Relationships Also Exist 😞

- One-sided links are called **directed** links
- Networks with directed links are called **directed networks**
- Directedness is useful for representing relationships that are not necessarily symmetric or two-way
- In contrast, the links and the networks in the earlier examples are called **undirected** links and **undirected networks**

# Another Example of Directed Networks: Video Game Skill Trees



# Exercise: What are the Nodes and Links in the Following Scenarios? Directed or Undirected?

- A city's road network
- The network of airports in the world
- The Wikipedia Game: given two Wikipedia articles, find a way to get from one to another by only traveling through the hyperlinks on your current webpage

# Exercise: What are the Nodes and Links in the Following Scenarios? Directed or Undirected?

- Six Degrees of Kevin Bacon
  - Start with a Hollywood actor
  - List a sequence of actors (beginning with the chosen one), but each next actor you name must have shared a show or movie with the last actor in the list
    - For example, Elizabeth Olsen and Paul Bettany both appeared in *Avengers: Age of Ultron* (Scarlet Witch and Vision)
  - Reach some other target actor using as short a sequence as possible



# There Are Many Problems We Want to Solve, For Example...

- Shortest route to go to some desired location
- Fastest way to send data over the internet
- Smallest cost to supply electricity to barangays
- Predict which people are likely to get married



# Do These Problems Have Common Features?

## What Are They?

- Shortest route to go to some desired location
- Fastest way to send data over the internet
- Smallest cost to supply electricity to barangays
- Predict which people are likely to get married

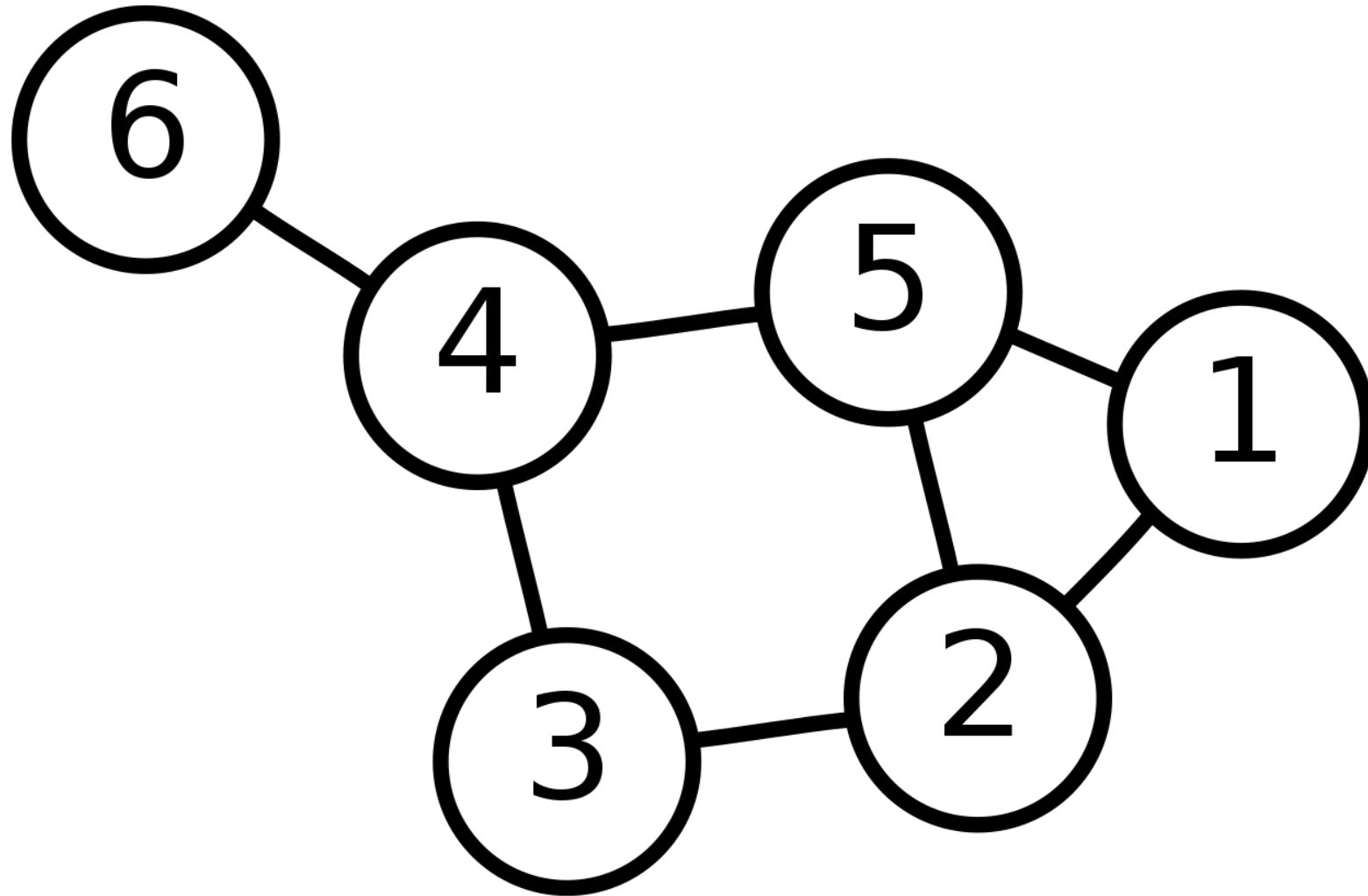


# Common Feature: They Are All Networks with Nodes and Links Between Nodes

Network	Node	Link
Communication	Phones, Computers	Cables
Transportation	Intersections, Places	Roads, Highways
Power Supply	Power Plant, Towns	Wires
Financial	Stock, Currencies	Transactions
Game	Board Positions	Legal Moves
Social Relationship	Persons	Friendships
Neural Network	Neurons	Synapses
Molecule	Atoms	Bonds



# We Can Forget About Real Life and Draw Everything With Circles and Lines



# Wait, But Why?

1. Extract a common *model* for various scenarios
2. Frame the *specific* problem in terms of the *general* model
3. Develop and improve a solution for just the general model
4. Apply the same solution for all specific scenarios

# Wait, But Why?

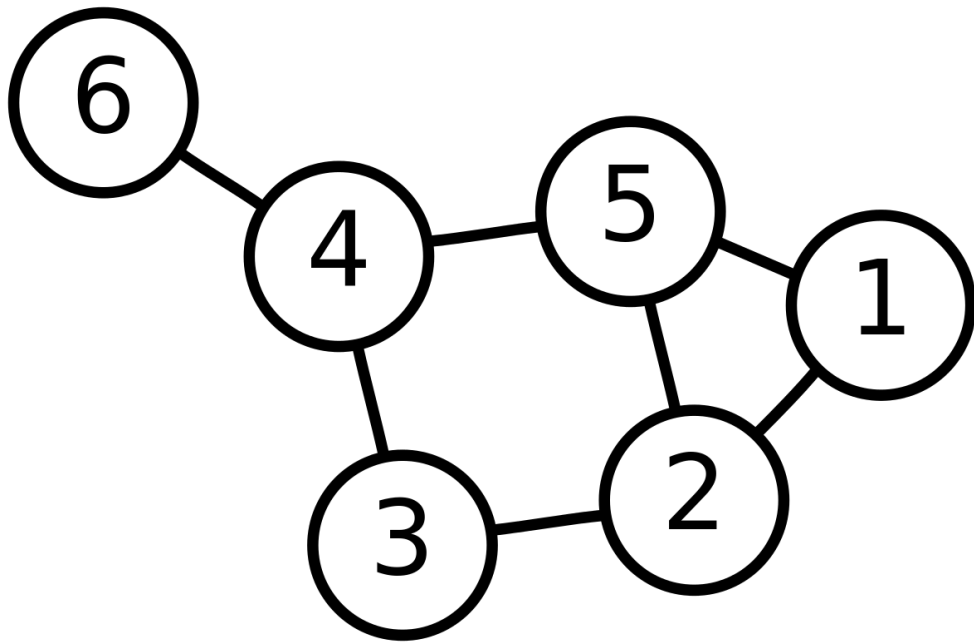
- Example: numbers used to model money, coconuts, goats, etc.
  - We only needed to learn how to add numbers – we don't need to separately learn how to add money, coconuts, goats, etc.
- The bridges puzzle and the practical real-life problems seem to have nothing to do with each other
- But in fact, they can be solved with the same idea: draw circles and lines
- Algorithms we invent on circles and lines can be used on all sorts of real-life problems, and need to be invented only once

# Representing Networks

- Many useful applications have thousands/millions/billions of nodes
- We'd rather write code and let the computer do it rather than draw circles and lines with pen and paper
- However, we can't draw circles and lines in code, so we need different ways of representing the structure of the network
- It's easy to manipulate integers with a computer, so give each node a unique integer label
- Then, how to represent links?

# List of Links (in Python)

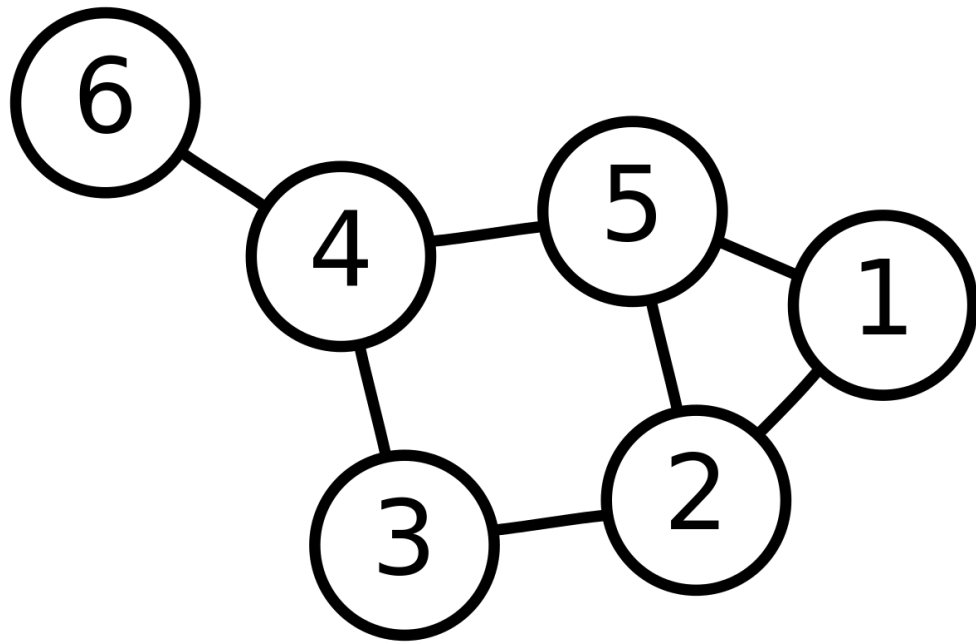
- Each link is a pair of integers
- A network is just a list of pairs



```
links = [  
    [6, 4],  
    [4, 5],  
    [4, 3],  
    [3, 2],  
    [5, 2],  
    [5, 1],  
    [1, 2]  
]
```

# List of Links (in C++)

- Each link is a pair of integers
- A network is just a list of pairs



```
vector<pair<int, int>> links = {  
    {6, 4},  
    {4, 5},  
    {4, 3},  
    {3, 2},  
    {5, 2},  
    {5, 1},  
    {1, 2}  
};
```

# Questions

- Does the order of the pairs in the list matter?
  - No, the order doesn't mean anything
- Does the order of the two numbers in one pair matter?
  - No, but only because the network is undirected

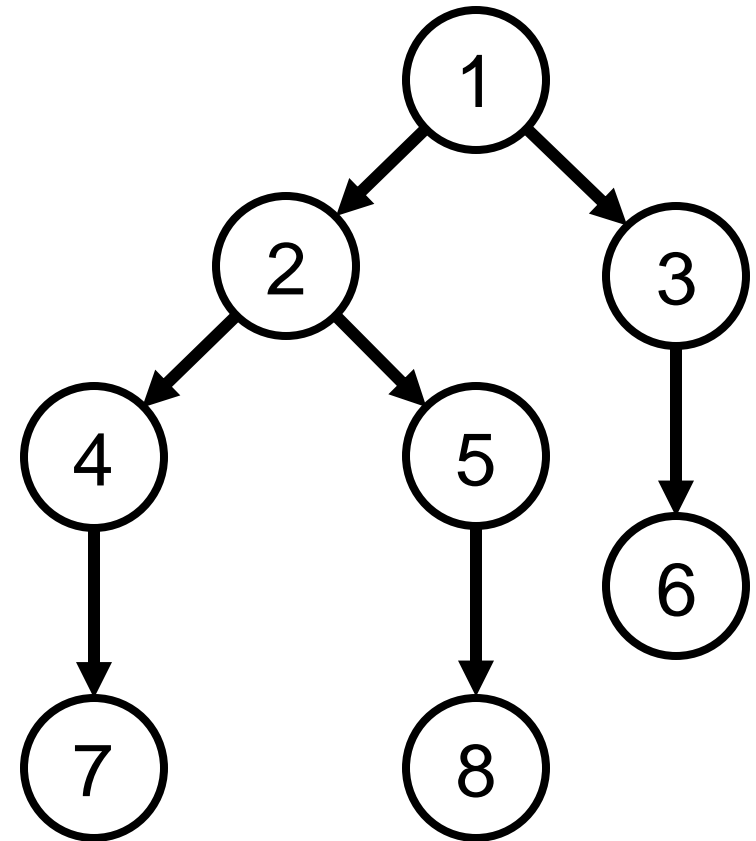
# What About Directed Networks?

- Simply declare that the order of the two numbers in each pair matters
  - E.g., the link points from the first number in the pair to the second one
- Undirected and directed networks have the same code
  - What makes it undirected or directed is what we say it is, and how we use it

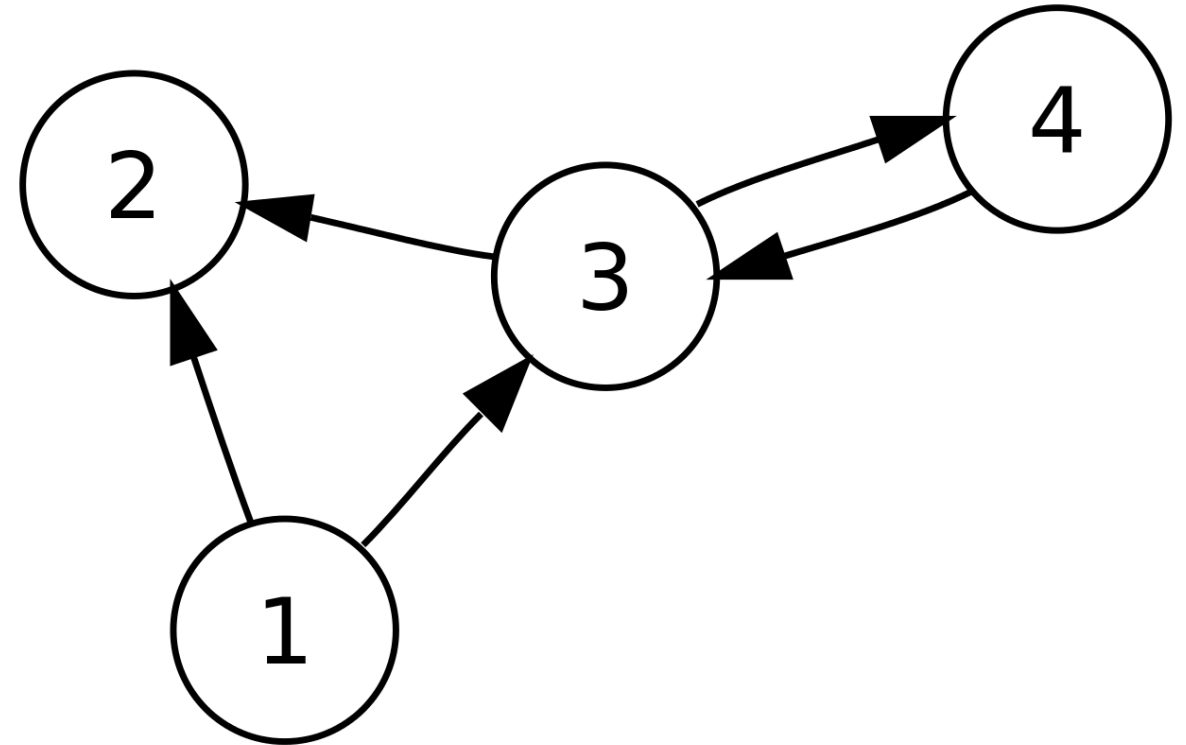
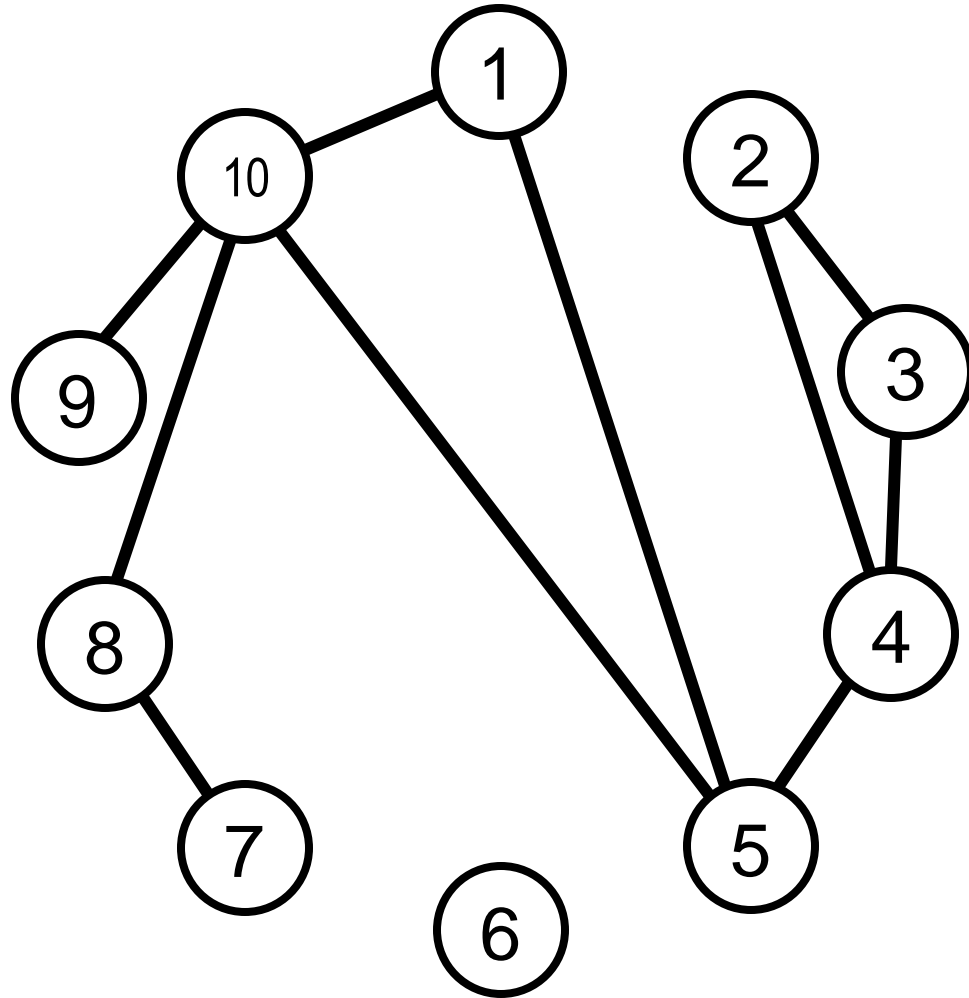


# What About Directed Networks?

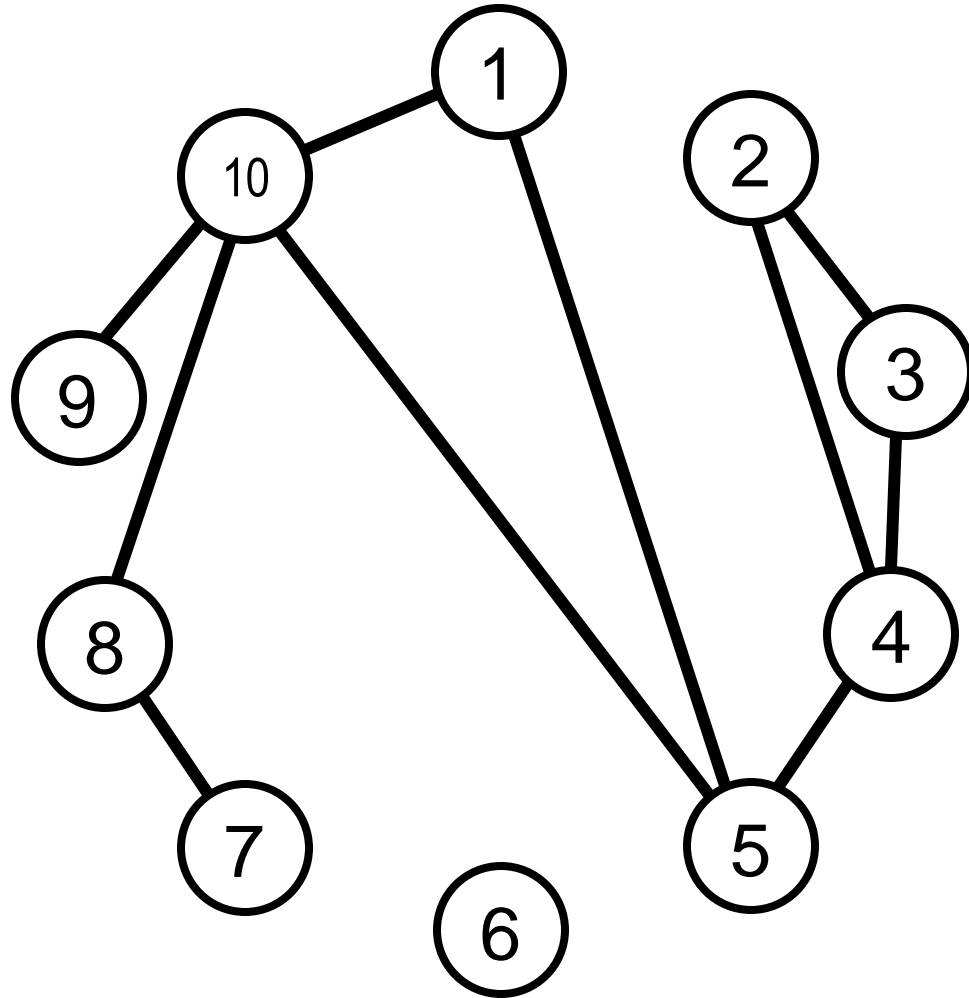
```
links = [  
    [1, 2],  
    [1, 3],  
    [2, 4],  
    [2, 5],  
    [3, 6],  
    [4, 7],  
    [5, 8]  
]
```



# Exercise: Give the List of Links of the Following Networks



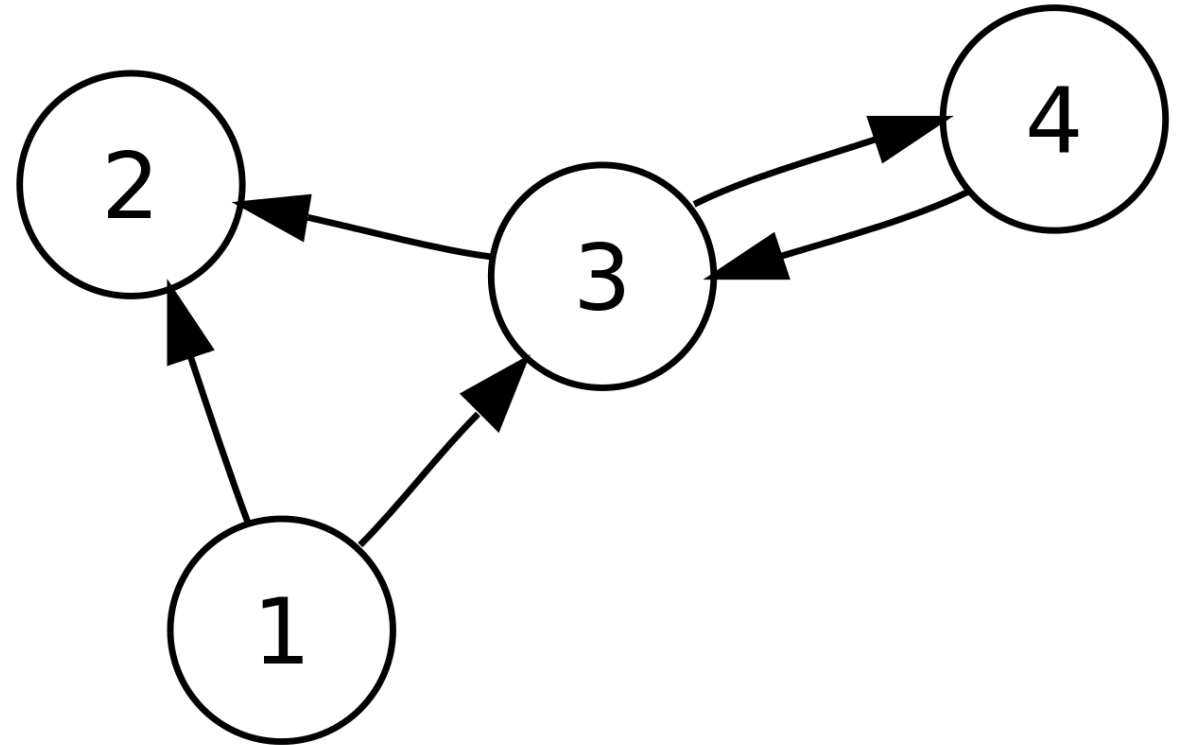
# Answers



```
links = [  
    [1, 5],  
    [1, 10],  
    [2, 3],  
    [2, 4],  
    [3, 4],  
    [4, 5],  
    [5, 10],  
    [7, 8],  
    [8, 10],  
    [9, 10],  
]
```

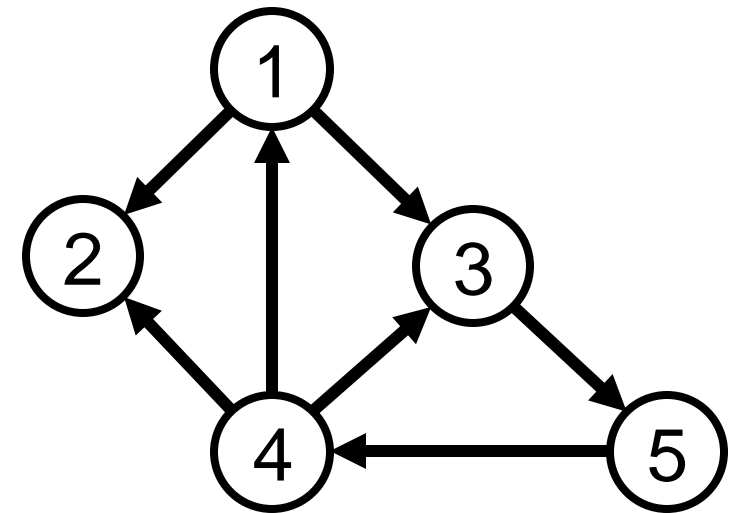
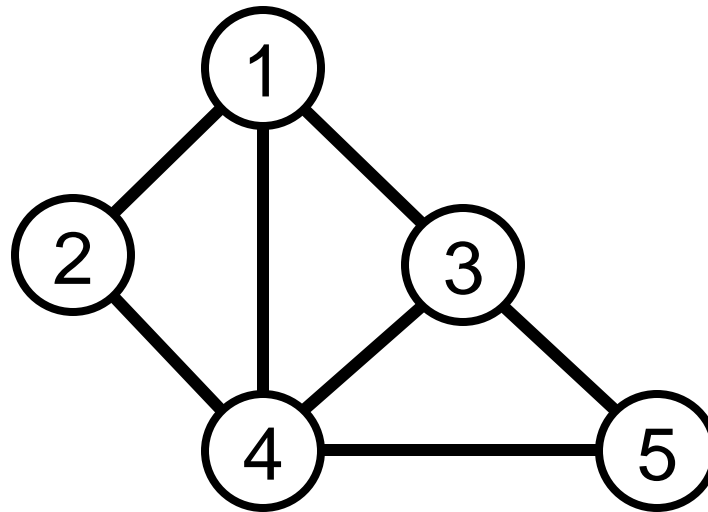
# Answers

```
links = [  
    [1, 2],  
    [1, 3],  
    [3, 2],  
    [3, 4],  
    [4, 3]  
]
```



# Exercise: Draw the Undirected/Directed Networks Corresponding to This Link of Lists

```
links = [  
    [1, 2],  
    [1, 3],  
    [4, 2],  
    [3, 5],  
    [4, 1],  
    [4, 3],  
    [1, 2]  
]
```



# Basic Computations We Might Want

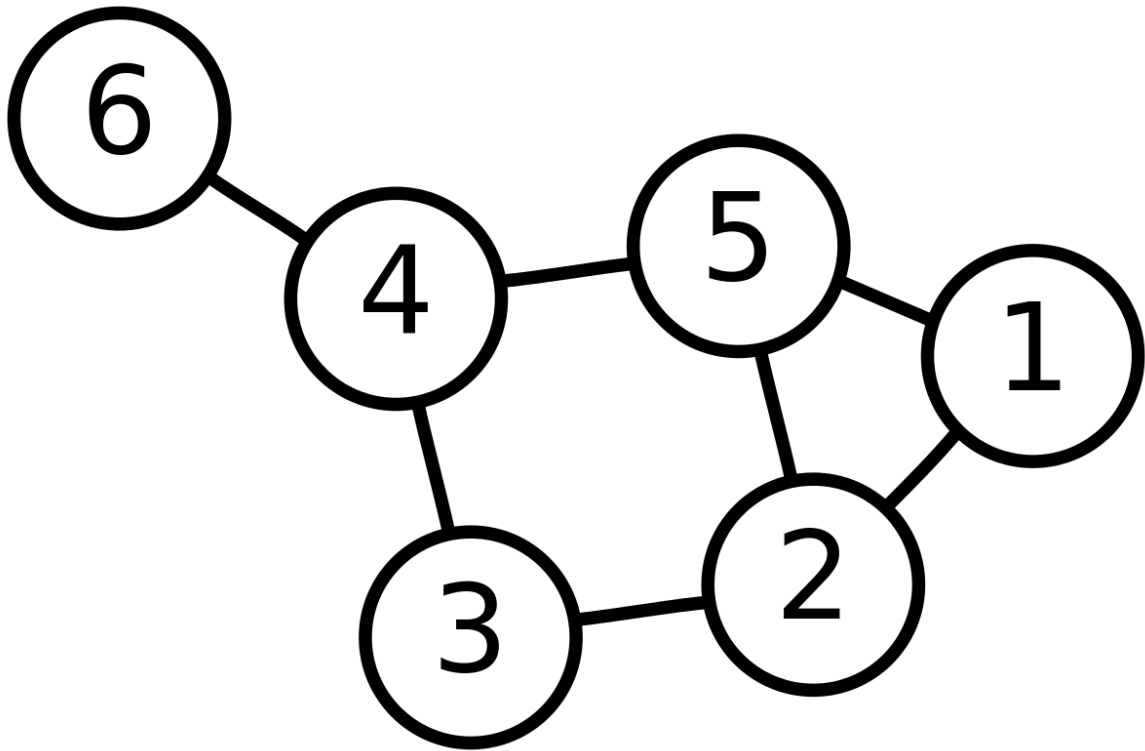
## Think: How to do Each One, and How Fast/Slow?

- Determine if two nodes  $(u, v)$  are connected by a link
    - In other words, if they are **neighbors** of each other
    - Determine if should display a new post or not in a social network
  - Find all neighbors of a node  $v$ 
    - Friends list in a social network
  - Find how many neighbors  $v$  has
    - Bridges puzzle
- Go through each pair in the list, check if it is  $(u, v)$  or  $(v, u)$
- $\Theta(L)$  where  $L$  is the total number of links in the whole network
- Also need to go through the list...
- Also  $\Theta(L)$
- Also need to go through the list...
- Also  $\Theta(L)$

# Speed Up Checking if Two Nodes Are Neighbors of Each Other: Neighbor Table

- Avoid searching through links, store table for quick lookup
- $i$ th row,  $j$ th column is true (or 1) if there is link from  $i$  to  $j$
- False (or 0) otherwise

# Neighbor Table

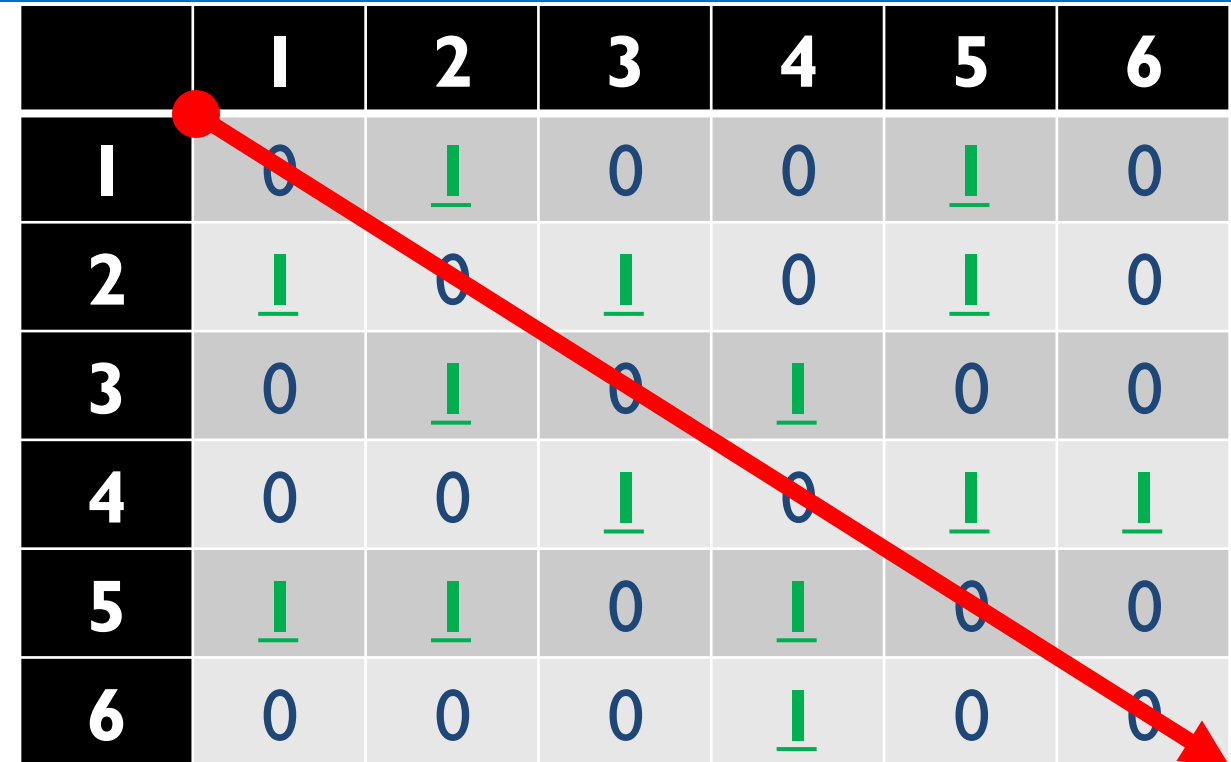


	1	2	3	4	5	6
1	0	<u>1</u>	0	0	<u>1</u>	0
2	<u>1</u>	0	<u>1</u>	0	<u>1</u>	0
3	0	<u>1</u>	0	<u>1</u>	0	0
4	0	0	<u>1</u>	0	<u>1</u>	<u>1</u>
5	<u>1</u>	<u>1</u>	0	<u>1</u>	0	0
6	0	0	0	<u>1</u>	0	0



# What About Directed Networks?

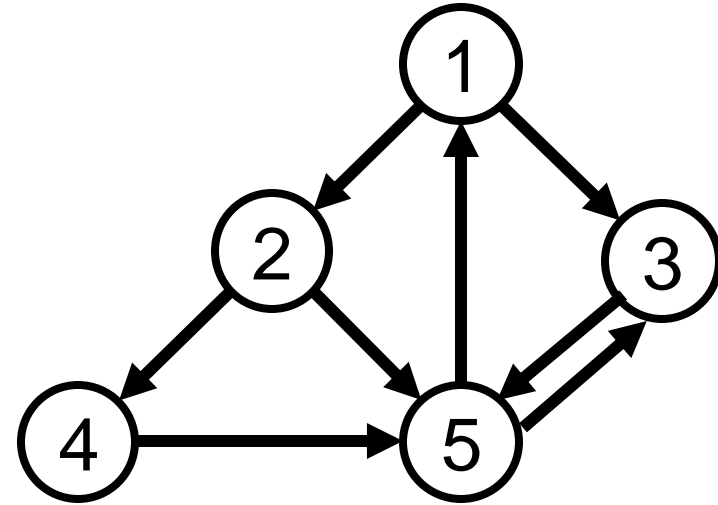
- Notice that the table for an undirected network *must be* symmetric across the top-left to bottom-right diagonal
- If the network were directed, the representation is the same, the resulting table just won't be symmetric



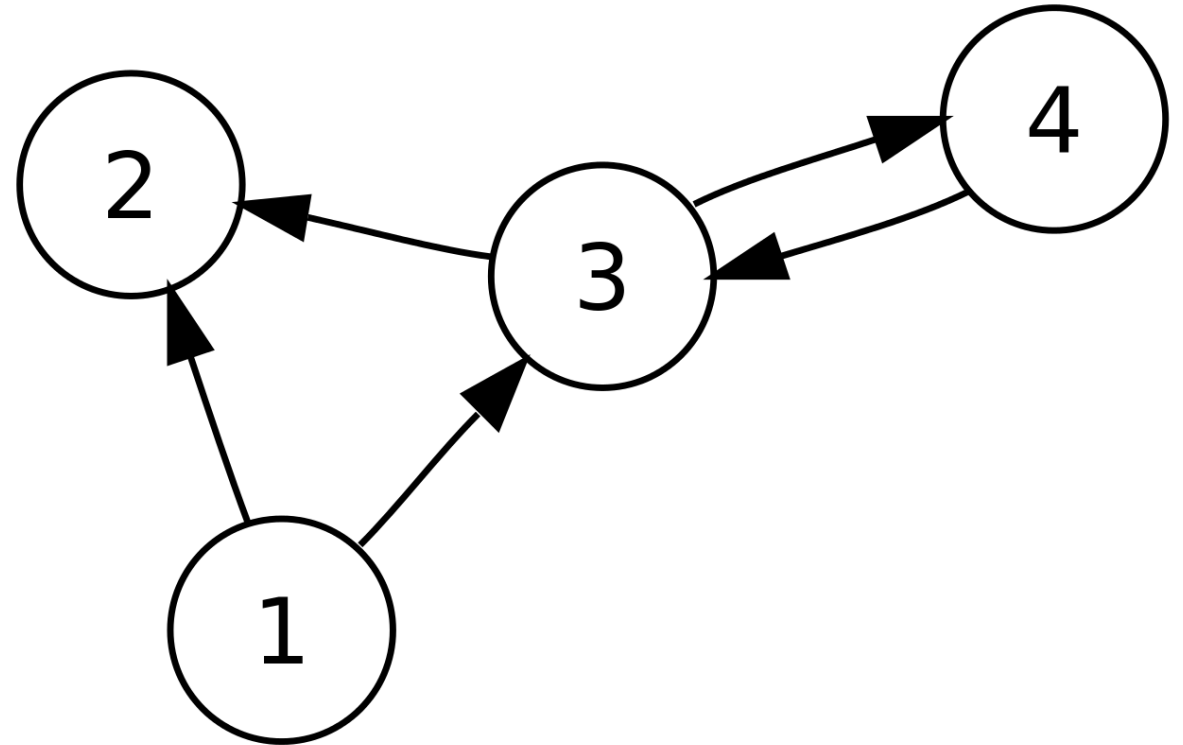
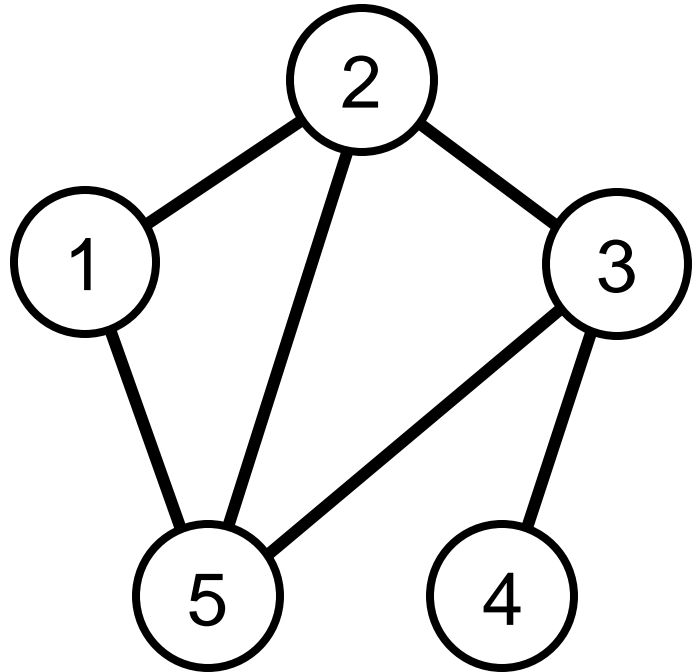
	1	2	3	4	5	6
1	0	<u>1</u>	0	0	<u>1</u>	0
2	<u>1</u>	0	<u>1</u>	0	<u>1</u>	0
3	0	<u>1</u>	0	<u>1</u>	0	0
4	0	0	<u>1</u>	0	<u>1</u>	<u>1</u>
5	<u>1</u>	<u>1</u>	0	<u>1</u>	0	0
6	0	0	0	<u>1</u>	0	0

# What About Directed Networks?

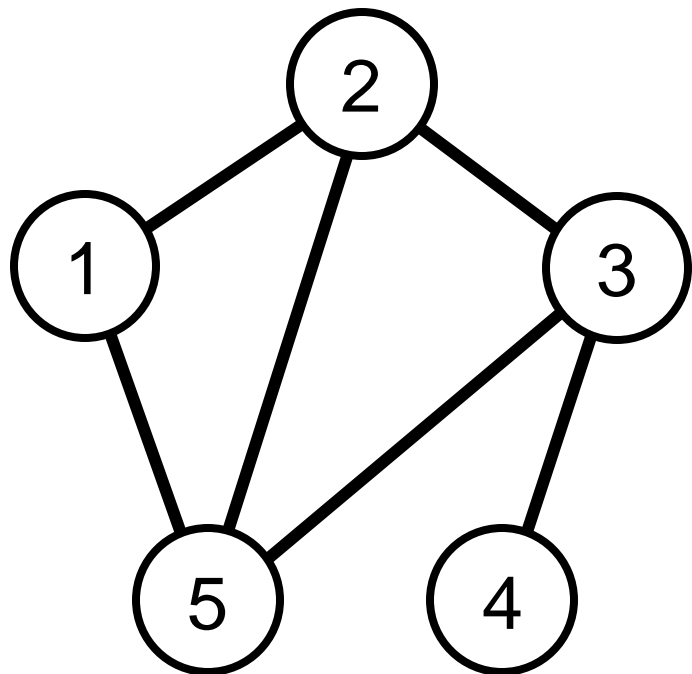
	1	2	3	4	5
1	0	<u>1</u>	<u>1</u>	0	0
2	0	0	0	<u>1</u>	<u>1</u>
3	0	0	0	0	<u>1</u>
4	0	0	0	0	<u>1</u>
5	<u>1</u>	0	<u>1</u>	0	0



# Exercise: Draw the Neighbor Tables of the Following Networks



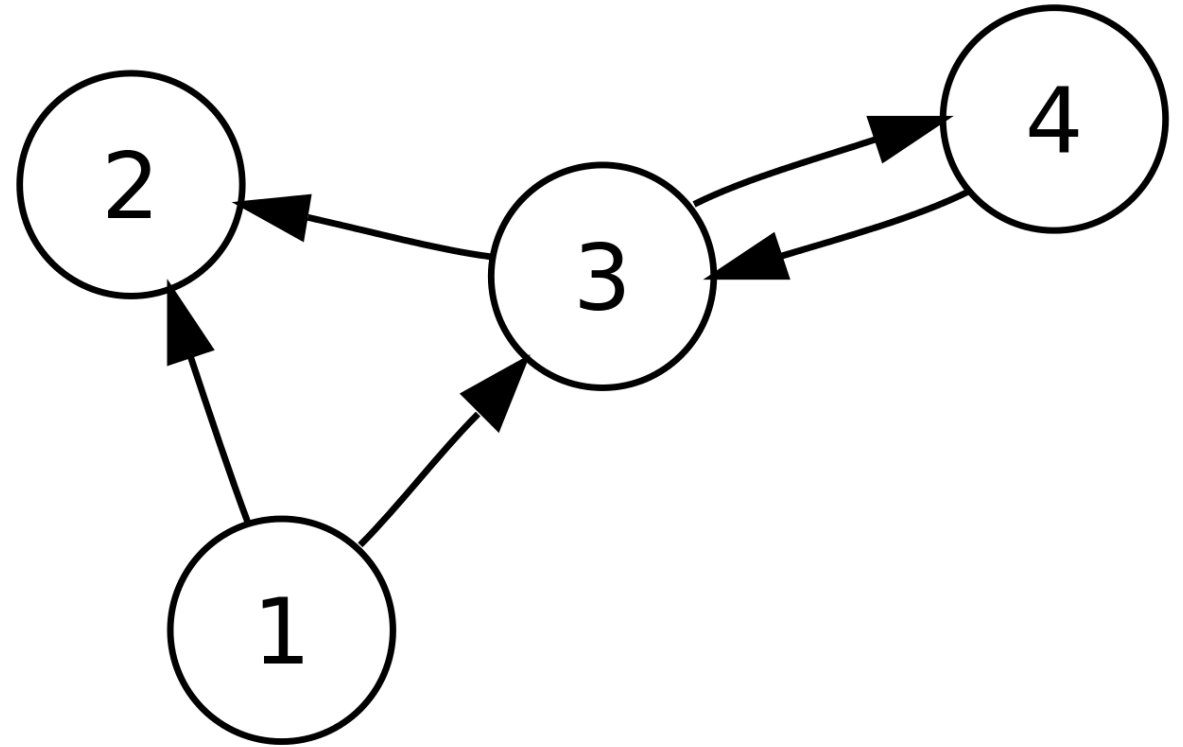
# Answers



	1	2	3	4	5
1	0	<u>1</u>	0	0	<u>1</u>
2	<u>1</u>	0	<u>1</u>	0	<u>1</u>
3	0	<u>1</u>	0	<u>1</u>	0
4	0	0	<u>1</u>	0	0
5	<u>1</u>	<u>1</u>	0	0	0

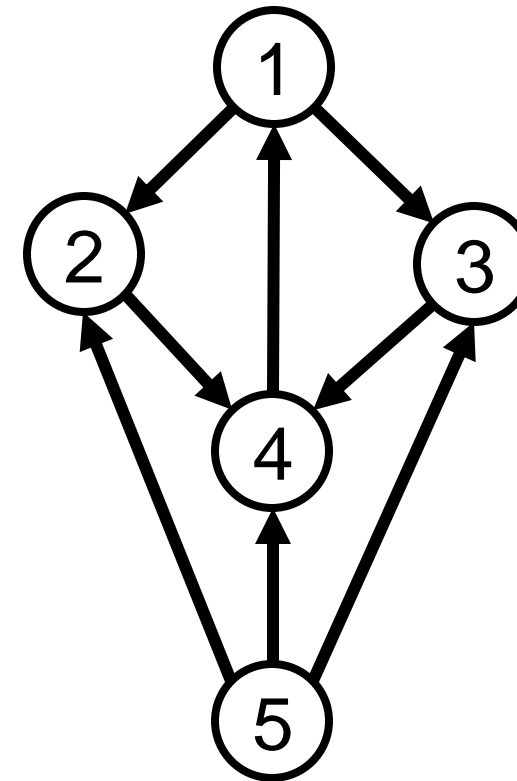
# Answers

	1	2	3	4
1	0	<u>1</u>	<u>1</u>	0
2	0	0	0	0
3	0	<u>1</u>	0	<u>1</u>
4	0	0	<u>1</u>	0



# Exercise: Draw the Network Corresponding to This Neighbor Table. Is It Directed or Undirected?

	1	2	3	4	5
1	0	<u>1</u>	<u>1</u>	0	0
2	0	0	0	<u>1</u>	0
3	0	0	0	<u>1</u>	0
4	<u>1</u>	0	0	0	0
5	0	<u>1</u>	<u>1</u>	<u>1</u>	0



- It's directed

# Converting Lists of Links from Input

- Because lists of links is the most straightforward way to represent networks, this is the most common raw data format
- Need to first convert to neighbor table yourself before doing any processing, to enable fast neighbor checking
- Shouldn't be too hard

# Practical Tip

- Most problems use 1-indexed nodes, but Python and C++ are 0-indexed
- Can keep using 1-index but use slightly more space
  - Make storage for  $n + 1$  nodes instead of  $n$
  - Have an unused node 0
- Or decrement labels before storing and doing processing
  - Just remember you did this decrement while debugging/printing
  - Code below does this



# Converting List of Links (from Standard Input) into Neighbor Table (in Python)

```
n, m = [int(x) for x in input().split()]
is_neighbor = [
    [False for j in range(n)]
    for i in range(n)
]
for i in range(m):
    u, v = [int(x) for x in input().split()]
    is_neighbor[u - 1][v - 1] = True
    is_neighbor[v - 1][u - 1] = True # for undirected
```

# Converting List of Links (from Standard Input) into Neighbor Table (in C++)

```
int n, m;
cin >> n >> m;
vector<vector<bool>> is_neighbor(n, vector<bool>(n));
for(int i = 0; i < m; i++) {
    int u, v;
    cin >> u >> v;
    is_neighbor[u - 1][v - 1] = true;
    is_neighbor[v - 1][u - 1] = true; // for undirected
}
```

# Think: How to do Each One, and How Fast/Slow?

- Determine if two nodes are neighbors
  - Easy  $\Theta(1)$  table lookup
- Find all neighbors of a node  $v$ 
  - Go through row  $v$ 
    - $\Theta(N)$ , where  $N$  is the number of nodes in the network
- Find how many neighbors  $v$  has
  - Also need to go through row  $v$ 
    - Also  $\Theta(N)$

# Neighbor Finding/Counting Also Improved

- For most networks,  $\Theta(N)$  is much better than  $\Theta(L)$ . Why?
- Assuming at most one link between any two nodes, in the worst case,  $L = \Theta(N^2)$
- $\Theta(N^2)$  distinct pairs  $(u, v)$  where  $u$  and  $v$  are integers from 1 to  $N$
- This is true for both directed and undirected networks

# Neighbor Finding/Counting Also Improved, But Still Not Quite the Best

- In the worst case, a node has  $\Theta(N)$  neighbors, so this running time is unavoidable
- However, for nodes with only few neighbors,  $\Theta(N)$  is not ideal

# The Real Dealbreaker for Neighbor Tables: Wastes Too Much Space

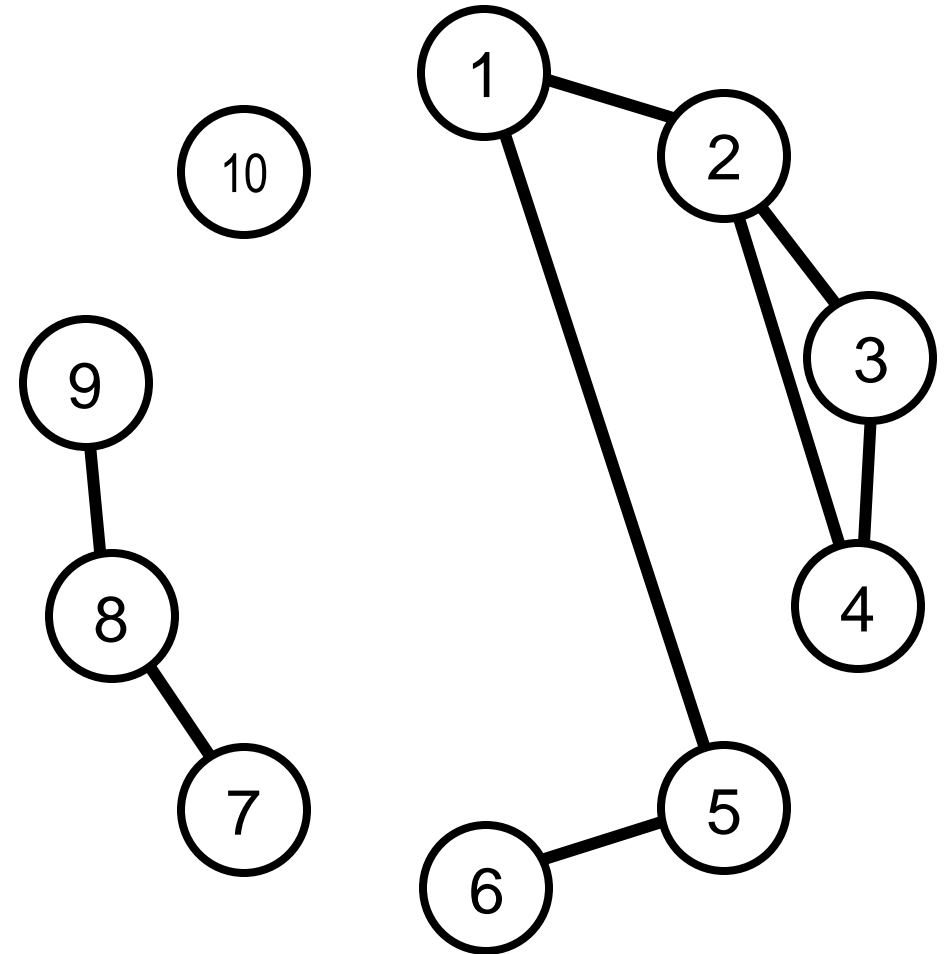
- Always requires  $\Theta(N^2)$  space
- This is fine if the network is **dense**
  - Dense: lots of links, close to  $\Theta(N^2)$
  - How much space (in terms of  $N$ ) does the list of links use in this case?
- However, if the network is **sparse**, this may be unacceptable
  - Sparse: relatively few links, closer to  $\Theta(N)$
  - How much space (in terms of  $N$ ) does the list of links use in this case?

# The Real Dealbreaker for Neighbor Tables: Wastes Too Much Space

- In lots of applications, networks are sparse
- Incidentally, another way to criticize  $\Theta(N)$  neighbor finding: it's bad for sparse networks

# Look at All Those Zeros!

	1	2	3	4	5	6	7	8	9	10
1	<u>0</u>		<u>0</u>	<u>0</u>		<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>
2		<u>0</u>			<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>
3	<u>0</u>		<u>0</u>		<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>
4	<u>0</u>			<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>
5		<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>		<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>
6	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>		<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>
7	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>		<u>0</u>	<u>0</u>
8	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>		<u>0</u>		<u>0</u>
9	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>		<u>0</u>	<u>0</u>
10	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>





# What Makes Them Good/Bad?

## List of Links

- Hard to search for a link because there is no order
- No unnecessary info
- What if we sorted?

## Neighbor Tables

- Easy to search for a link, because tables are organized
- Too many zeros
- What if we got rid of the zeros?

# How to Fix/Get the Advantages of the Other?

## List of Links

- Sort the links so that checking if two nodes are neighbors can be done with binary search
  - For an undirected network, store both  $(u, v)$  and  $(v, u)$  so that the search can work regardless of the order of the query pair
- Save a few steps of the search:
  - Keep all links  $(1, x)$  in one list, all links  $(2, x)$  in another, etc.
  - Put all lists in one list – have a *list of lists of pairs* – and in the right indices, so that we can get the list containing the links connected to one node in  $\Theta(1)$
  - The first member of each pair is redundant information – index of inner list within outer list already says what the first member is

# How to Fix/Get the Advantages of the Other?

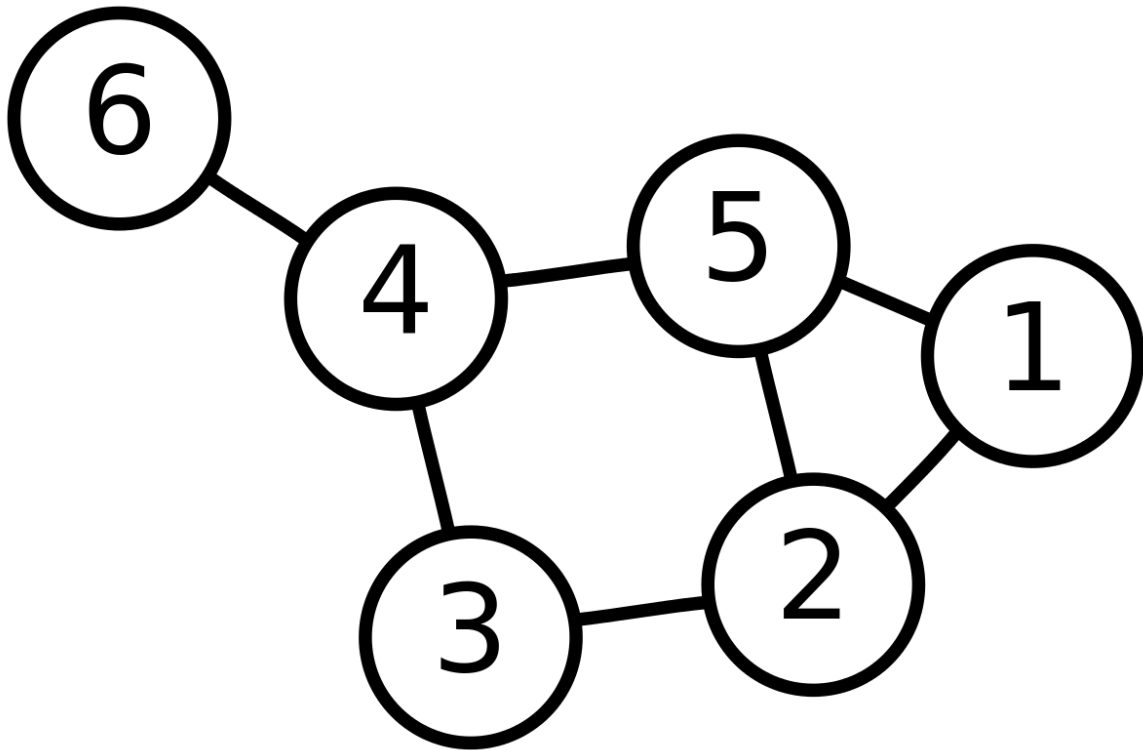
## Neighbor Tables

- Instead of storing a list of zeros and ones, just store the indices of the ones

# Leads to the Same Idea: Neighbor Lists

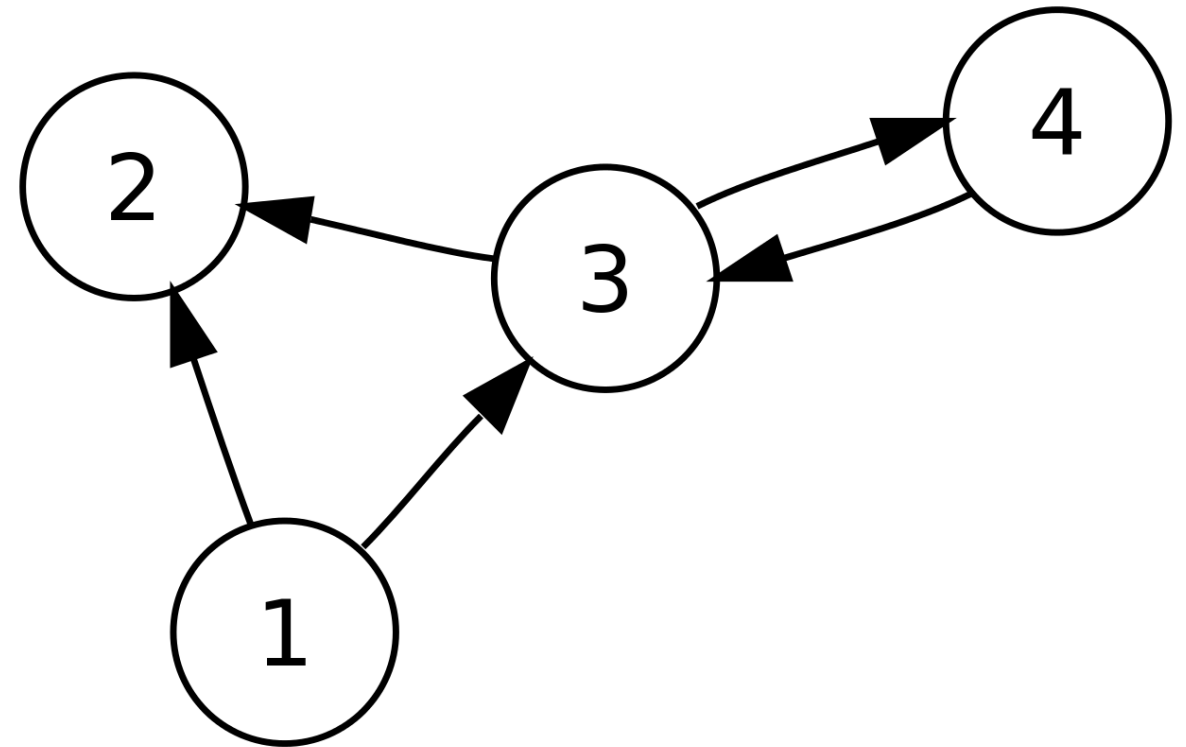
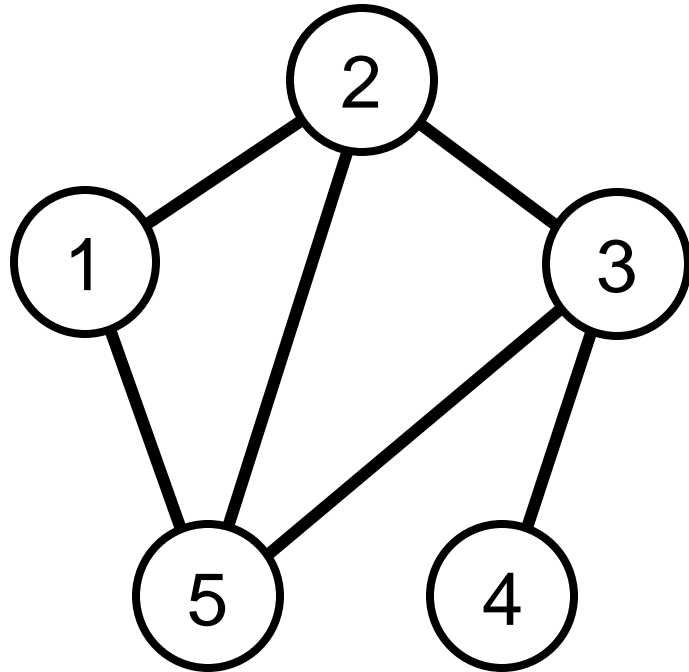
- For each node, store a list of its neighbors
  - Then, store all these lists in an outer list, properly indexed
  - The whole structure will be a list of lists
- If the network has  $n$  nodes, then we need a list of  $n$  lists
- $u$ th list contains  $v$  if  $u$  and  $v$  are directly connected
- Note that for undirected networks, every link corresponds to two entries in the neighbor lists:  $u$ th list contains  $v$  and  $v$ th list contains  $u$
- Sorting the neighbors is optional: depends if required to quickly repeatedly determine if nodes are neighbors

# Neighbor Lists

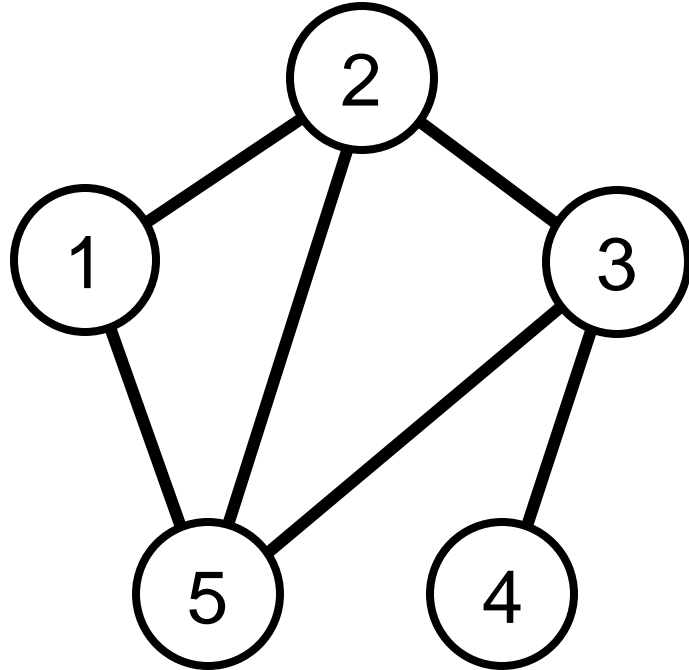


- $1 \rightarrow \{2, 5\}$
- $2 \rightarrow \{1, 3, 5\}$
- $3 \rightarrow \{2, 4\}$
- $4 \rightarrow \{3, 5, 6\}$
- $5 \rightarrow \{1, 2, 4\}$
- $6 \rightarrow \{4\}$

# Exercise: Write the Neighbor Lists of the Following Networks



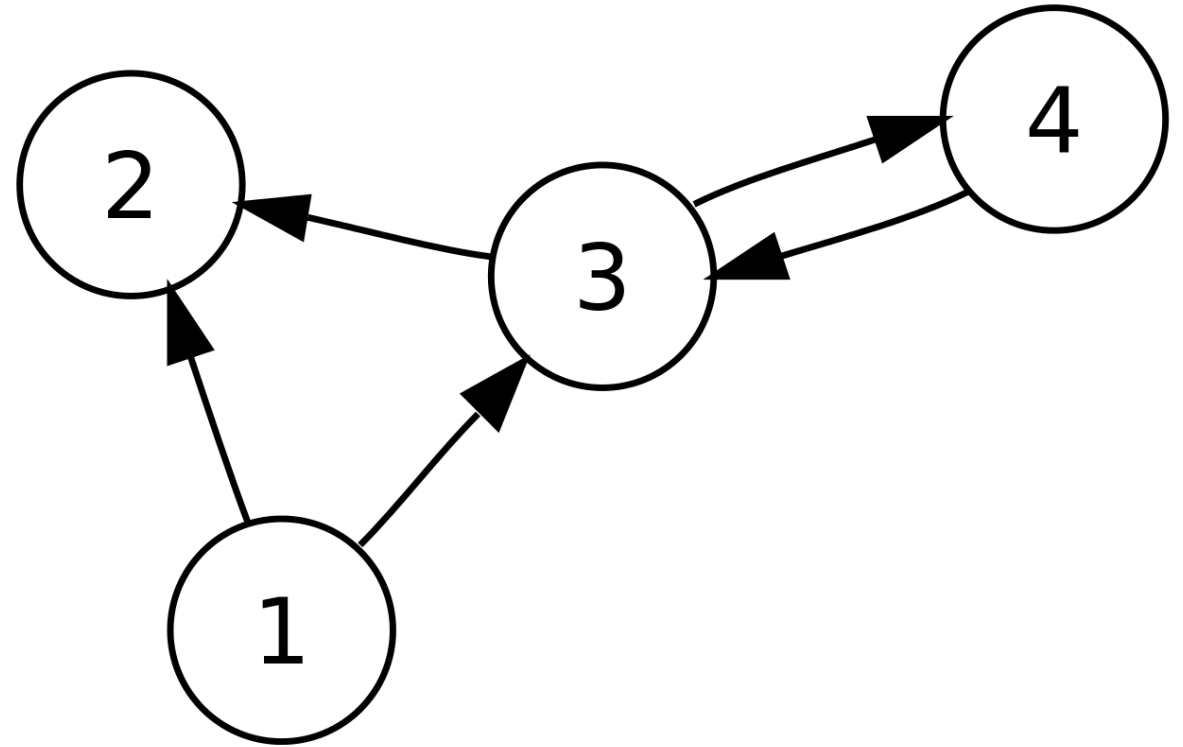
# Answers



- $1 \rightarrow \{2, 5\}$
- $2 \rightarrow \{1, 3, 5\}$
- $3 \rightarrow \{2, 4, 5\}$
- $4 \rightarrow \{3\}$
- $5 \rightarrow \{1, 2, 3\}$

# Answers

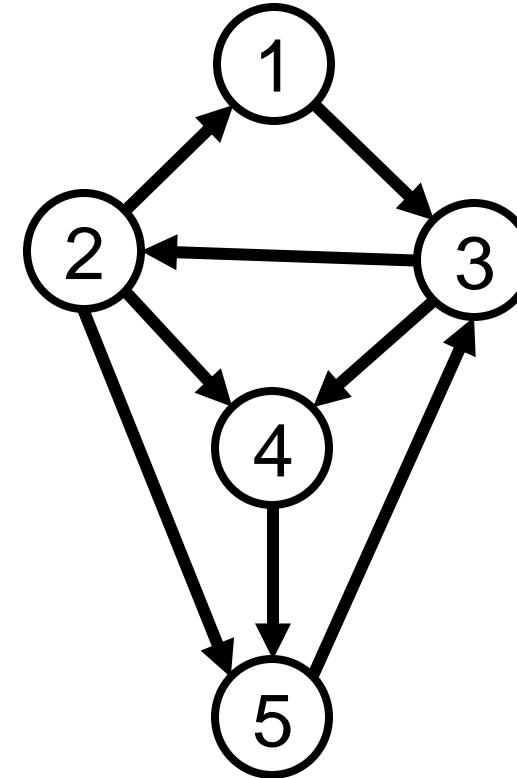
- $1 \rightarrow \{2, 3\}$
- $2 \rightarrow \{ \}$
- $3 \rightarrow \{2, 4\}$
- $4 \rightarrow \{3\}$





# Exercise: Draw the Network Corresponding to This Neighbor Lists. Is It Directed or Undirected?

- $1 \rightarrow \{3\}$
- $2 \rightarrow \{1, 4, 5\}$
- $3 \rightarrow \{2, 4\}$
- $4 \rightarrow \{5\}$
- $5 \rightarrow \{3\}$



- It's directed

# Converting List of Links (from Standard Input) into Neighbor Lists (in Python)

```
n, m = [int(x) for x in input().split()]
neighbors = [[] for i in range(n)]
for i in range(m):
    u, v = [int(x) for x in input().split()]
    neighbors[u - 1].append(v - 1)
    neighbors[v - 1].append(u - 1) # for undirected
```

# Converting List of Links (from Standard Input) into Neighbor Lists (in C++)

```
int n, m;
cin >> n >> m;
vector<vector<int>> neighbors(n + 1);
for(int i = 0; i < m; i++) {
    int u, v;
    cin >> u >> v;
    neighbors[u - 1].push_back(v - 1);
    neighbors[v - 1].push_back(u - 1); // for undirected
}
```

# Exercise

- How to do and how much time does it take to do these with neighbor lists?
  - Find how many neighbors a node has
  - Find all neighbors of a node
  - Determine if two nodes are neighbors of each other
- How much space does the neighbor lists require?

# Neighbor Lists Analysis

- Finding neighbor count (let's call this  $\text{neigh}(u)$ ):  $\Theta(1)$ 
  - Just take the length of the appropriate list
  - Obviously better than alternative representations
- Finding all neighbors of node  $u$  takes  $\Theta(\text{neigh}(u))$ 
  - Better than list of links'  $\Theta(L)$
  - For sparse networks usually better than neighbor table's  $\Theta(N)$

# Neighbor Lists Analysis

- Determining if two nodes are neighbors
  - If the lists are unsorted, still quite slow at  $\Theta(\text{neigh}(u))$ , not as good as neighbor table but way better than list of links
  - If the lists are sorted,  $\Theta(\log \text{neigh}(u))$ , not as good as neighbor table (but quite close) but way better than list of links
- Amount of space needed is  $\Theta(L)$ 
  - Directed: each link  $u \rightarrow v$  appears exactly once in neighbor list of  $u$
  - Undirected: each link  $u \leftrightarrow v$  appears exactly twice
    - Once in neighbor list of  $u$ , once in neighbor list of  $v$
- For these reasons, this is usually the best representation

# Summary of Performance Characteristics

## (Underlined Means “Winner”)

	List of Links	Neighbor Table	Neighbor Lists
Neighbor Count	$\Theta(L)$	$\Theta(N)$	<u><math>\Theta(1)</math></u>
Neighbor Enumerate	$\Theta(L)$	$\Theta(N)$	<u><math>\Theta(\text{neigh})</math></u>
Neighbor Determination	$\Theta(L)$	<u><math>\Theta(1)</math></u>	$\Theta(\text{neigh})$ or $\Theta(\log \text{neigh})$
Space	<u><math>\Theta(L)</math></u>	$\Theta(N^2)$	<u><math>\Theta(L)</math></u>

# Scarier But Official Terms

- network = **graph**
  - nothing to do with X-Y graphs
- node = **vertex**
- link = **edge**
- number of neighbors = **degree**
- two nodes are neighbors = two vertices are **adjacent**
- list of links = **edge list**
- neighbor lists = **adjacency lists**
- neighbor table = **adjacency matrix**
- Euler invented graphs around 1736, to solve the exact bridges puzzle we started with
- Networks were called graphs because you can draw (graph) them on paper
  - The sole connection to X-Y graphs
- Euler and friends imagined graphs to be 2D drawings of 3D solids, which have vertices and edges, hence the terms



# Scarier But Official Terms

- network = **graph**
  - nothing to do with X-Y graphs
- node = **vertex**
- link = **edge**
- number of neighbors = **degree**
- two nodes are neighbors = two vertices are **adjacent**
- list of links = **edge list**
- neighbor lists = **adjacency lists**
- neighbor table = **adjacency matrix**
- We invented the terms “network,” “node,” and “link” because they’re more intuitive and better match how graphs are used in the modern world
- But you should know the traditional terms because they’re what you’ll find in existing books, articles, and videos

# Practice Problems

- <https://progvar.fun/problemsets/graphs-basics>

# Thanks!

