GTDM – 2019/20

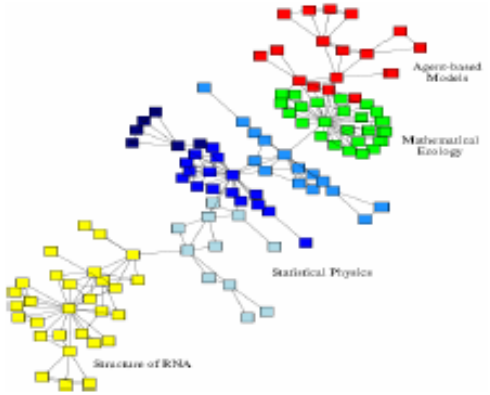
- Introduction to Graph Theory

Partially based on Lecture Notes of Keijo Ruohonen

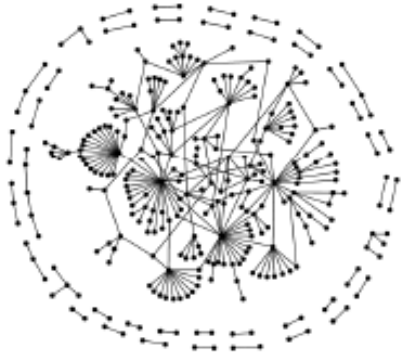
Why graphs? Graphs are a general language for describing and modeling complex systems

Many Data are Graphs

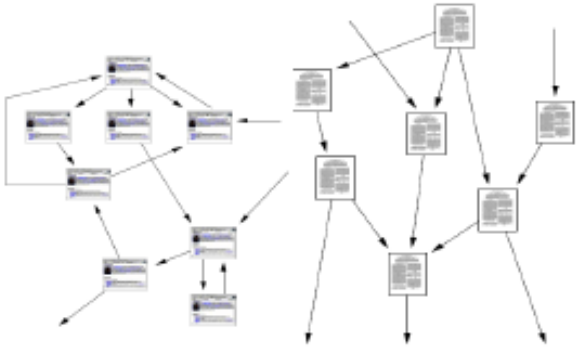
Social networks



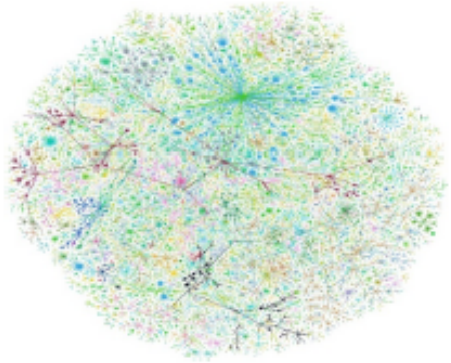
Economic networks



Biomedical networks



Information networks:
Web & citations



Internet

Networks of neurons

Graph: A representation of data

Why Graphs? Why Now?

Universal language for describing complex data

Networks/graphs from science, nature, and technology are more similar than one would expect

Shared vocabulary between fields

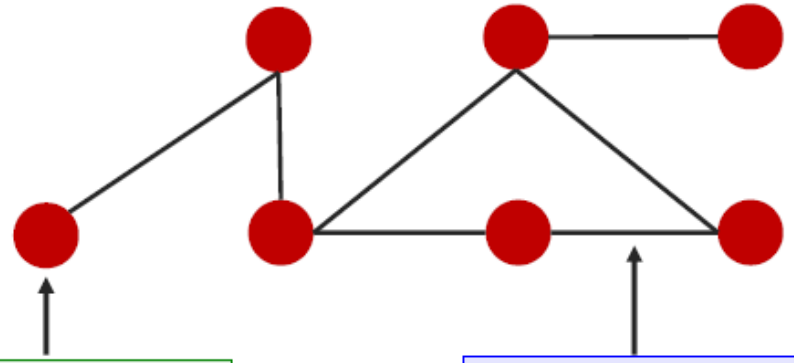
Computer Science, Social science, Physics, Economics, Statistics, Biology

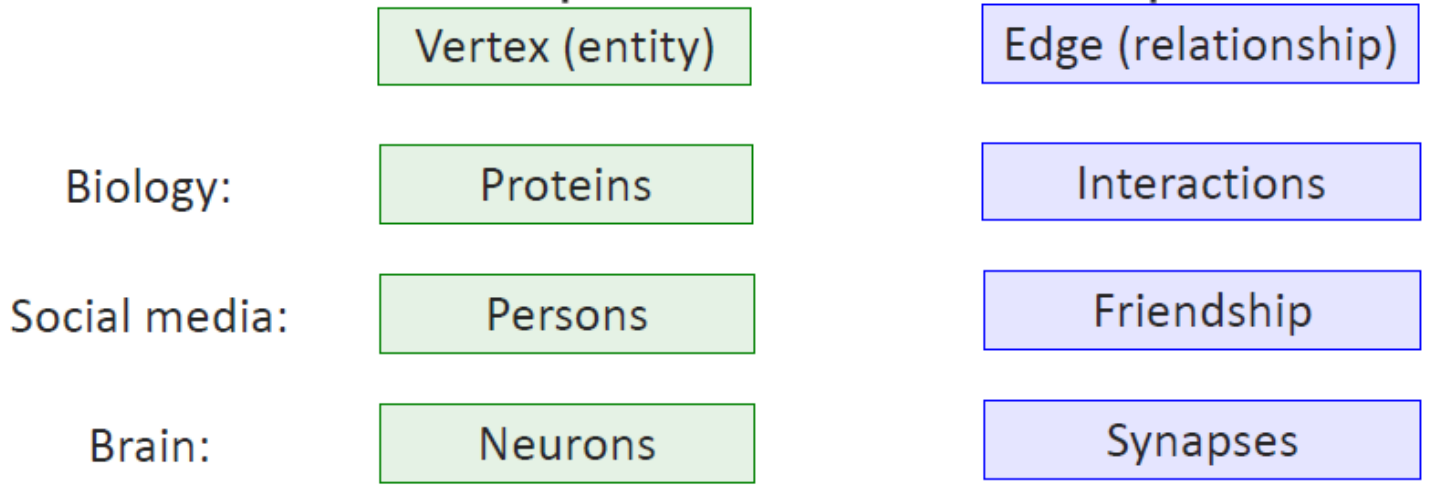
Data availability (+computational challenges)

Web/mobile, bio, health, and medical

Impact!

Social networking, Social media, Drug design

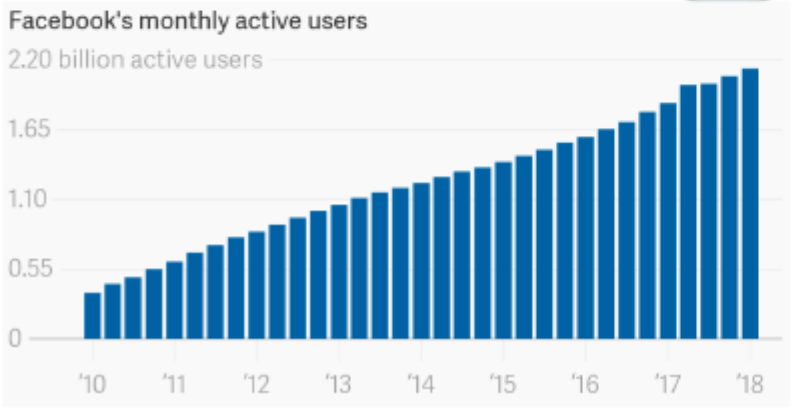




Social networks

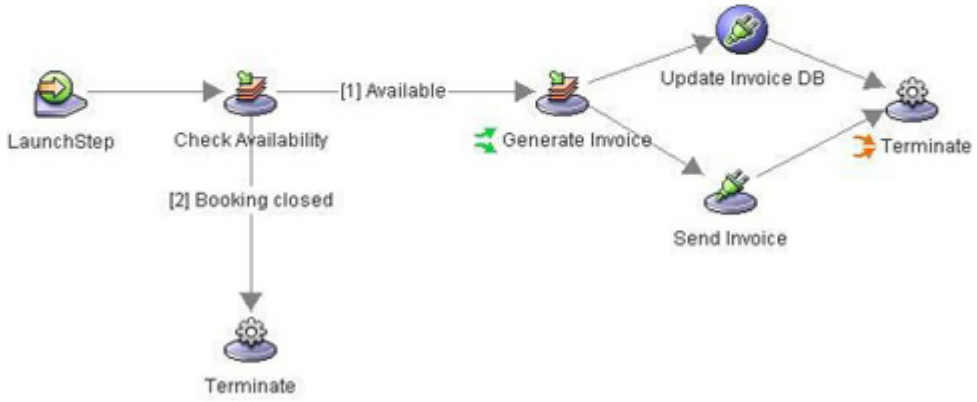
Problems

- Information spread
- Recommend friend
- Advertisement



Facebook: > 2.2 billion active users

Workflow: It's sequence of processes through which a piece of work passes from initiation to completion. Can be also represented as directed graph.



Google Maps:

Hyperlink network

Problems

- Page ranking
- Advertisement
- Reachability

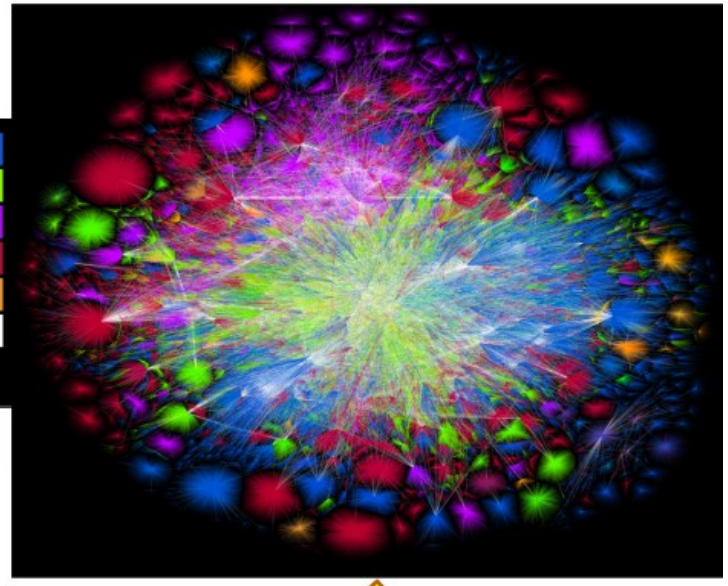
Google: > 100 trillion indexed pages

The Internet

North America (ARIN)
Europe (RIPE)
Latin America (LACNIC)
Asia Pacific (APNIC)
Africa (AFRINIC)
"Backbone" (highly connected networks)

Date: July 11 2015

http://www.opte.org/the-internet/

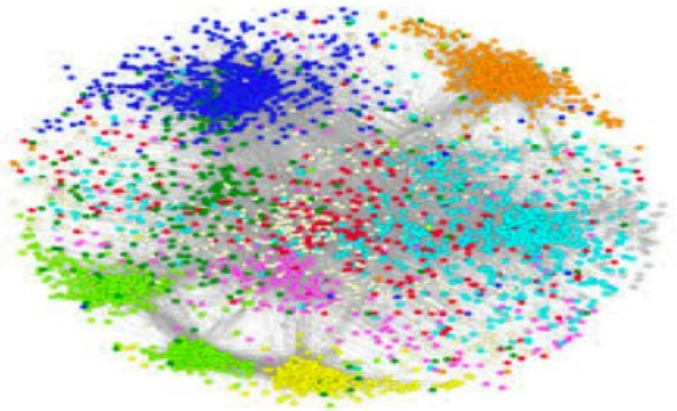


Problems
- Information propagation
- Fake news
- Anomalies

The Web: >50B webpages

Biological network

- New protein family
- Gene/protein functions

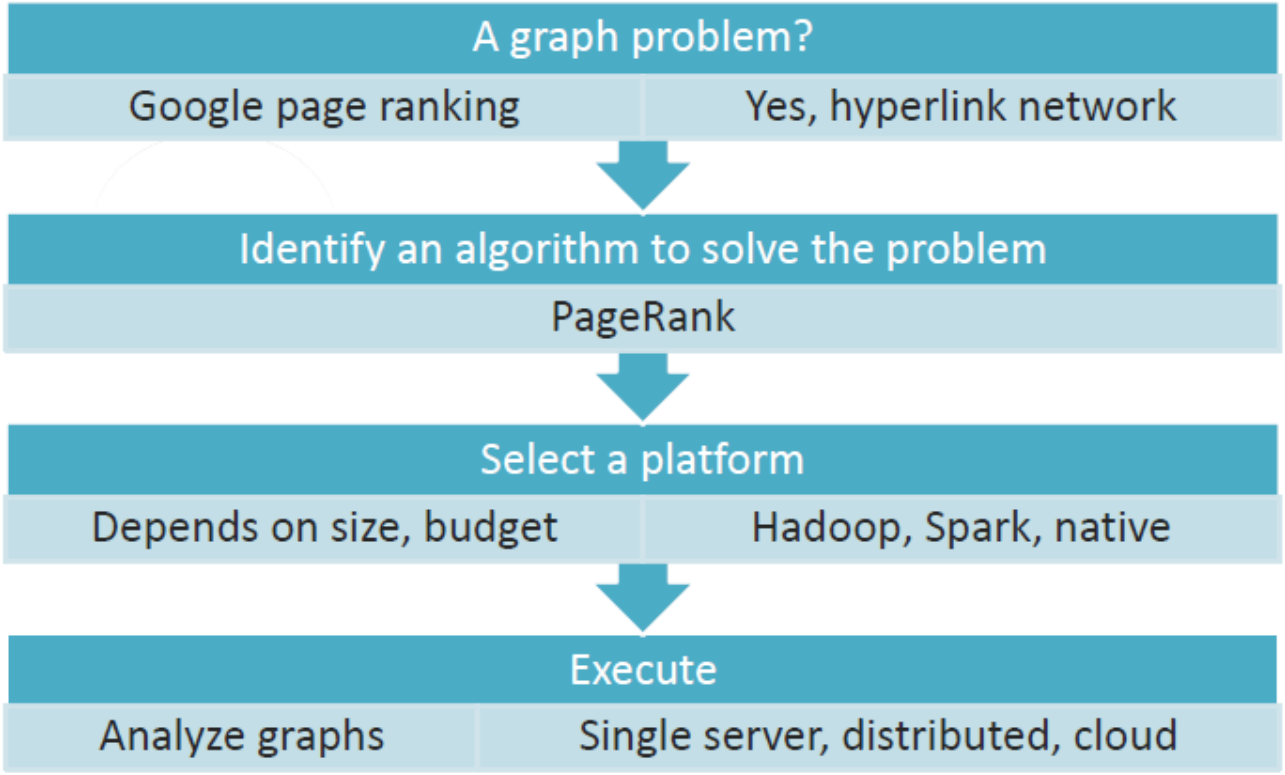


Metagenomic data: > 50 trillion edges among 50B proteins (IMG database)

Graph analytics

- Graph analytics has three components
  - **Data model**: Leverage graph structures to model and understand entities and their relationships in data
    - Not every dataset can be mapped to graphs
  - **Graph algorithms**: Use graph theory and algorithms to solve problems in graphs
    - Finding suitable algorithms is important
  - **Analytics platforms**: Use a suitable platform to analyze graphs with appropriate graph algorithms.
    - Depends on the scale of the problem and available resources
    - Underlying hardware needs to be considered

Graph analytics: how to solve a problem?



Graphs

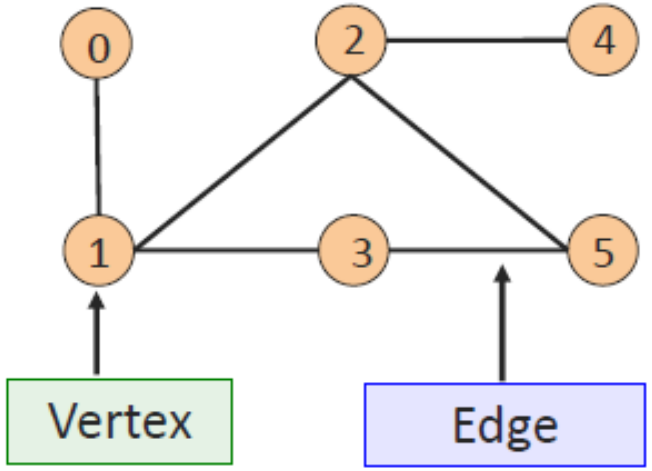
A graph is a formalism for representing relationships among items

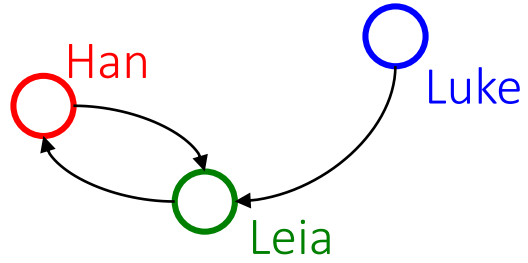
- Very general definition
- Very general concept

A graph is a pair: G = (V, E)

- A set of vertices, also known as nodes: $V = \{v_1, v_2, \ldots, v_n\}$
- A set of edges $E = \{e_1, e_2, \ldots, e_m\}$
  - Each edge e_i is a pair of vertices (v_j, v_k)
  - An edge "connects" the vertices

Graphs can be *directed* or *undirected*



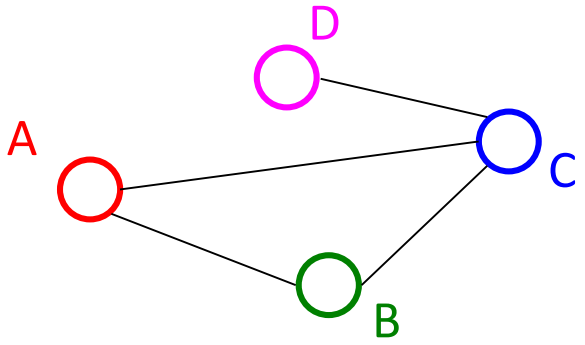


V = {Han,Leia,Luke}
E = {(Luke,Leia),
(Han,Leia),
(Leia,Han)}

Undirected Graphs

In undirected graphs, edges have no specific direction

- Edges are always "two-way"



Thus, $(u, v) \in E$ implies $(v, u) \in E$.

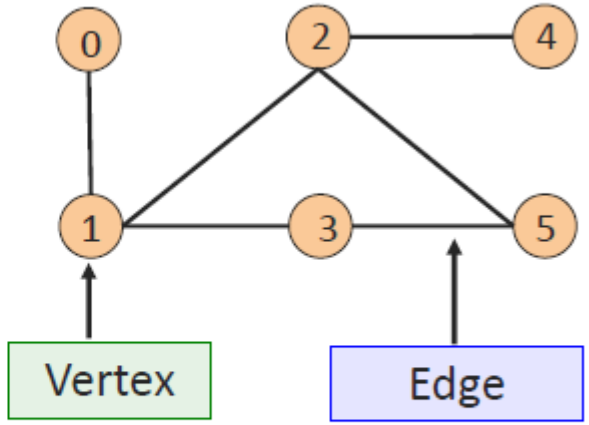
- Only one of these edges needs to be in the set
- The other is implicit, so normalize how you check for it

Degree of a vertex: number of edges containing that vertex

- Put another way: the number of adjacent vertices

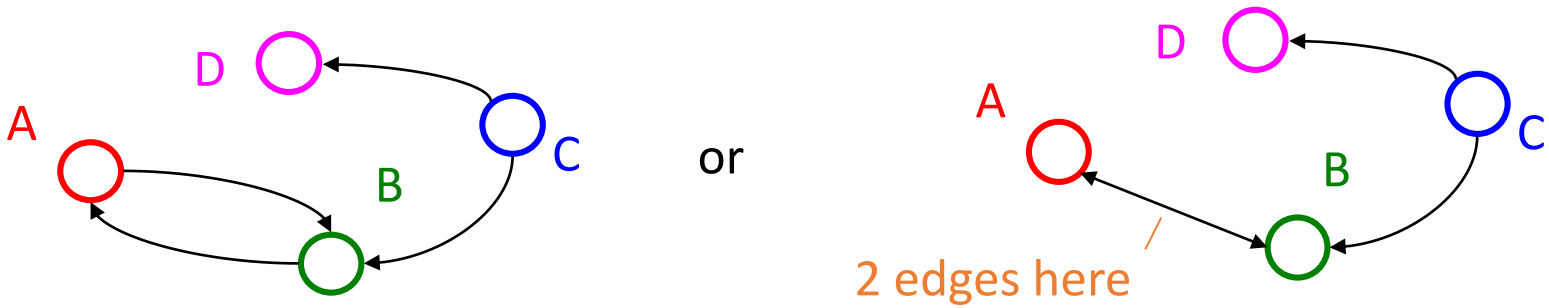
❑ Degree of a vertex: number of vertices adjacent to v
  – d(1) = 3

❑ In directed networks we define an in-degree and out-degree.



Directed Graphs

In directed graphs (or digraphs), edges have direction



Thus, $(u, v) \in E$ does not imply $(v, u) \in E$.

Let $(u, v) \in E$ mean $u \rightarrow v$

- Call u the source and v the destination
- In-Degree of a vertex: number of in-bound edges (edges where the vertex is the destination)
- Out-Degree of a vertex: number of out-bound edges (edges where the vertex is the source)

Self-Edges

A self-edge a.k.a. a loop edge is of the form (u, u)

- The use/algorithm usually dictates if a graph has:
  - No self edges
  - Some self edges
  - All self edges

A node can have a(n) degree / in-degree / out-degree of zero

Theorem 1.1. *The graph $G = (V, E)$, where $V = \{v_1, \ldots, v_n\}$ and $E = \{e_1, \ldots, e_m\}$, satisfies*

$$\sum_{i=1}^{n} d(v_i) = 2m.$$

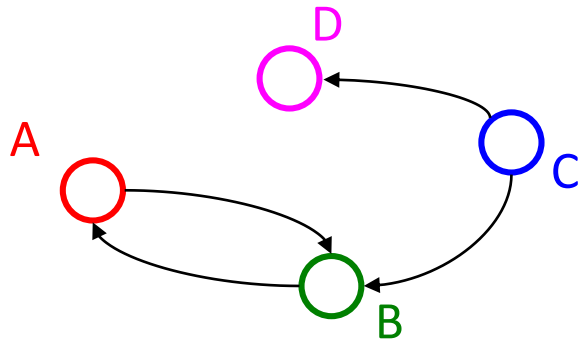
Corollary. *Every graph has an even number of vertices of odd degree.*

Proof. If the vertices $v_1, \ldots, v_k$ have odd degrees and the vertices $v_{k+1}, \ldots, v_n$ have even degrees, then (Theorem 1.1)

$$d(v_1) + \cdots + d(v_k) = 2m - d(v_{k+1}) - \cdots - d(v_n)$$

is even. Therefore, k is even. □

More Notation



$V = \{A, B, C, D\}$
$E = \{(C, B), (A, B), (B, A), (C, D)\}$

For a graph $G = (V, E)$:

- $|V|$ is the number of vertices
- $|E|$ is the number of edges
  - Minimum? **0**
  - Maximum for undirected? **$|V|\ (|V|-1)/2$**
  - Maximum for directed? **$|V|\ (|V|-1)$**

If $(u, v) \in E$, then v is a neighbor of u (i.e., v is adjacent to u)

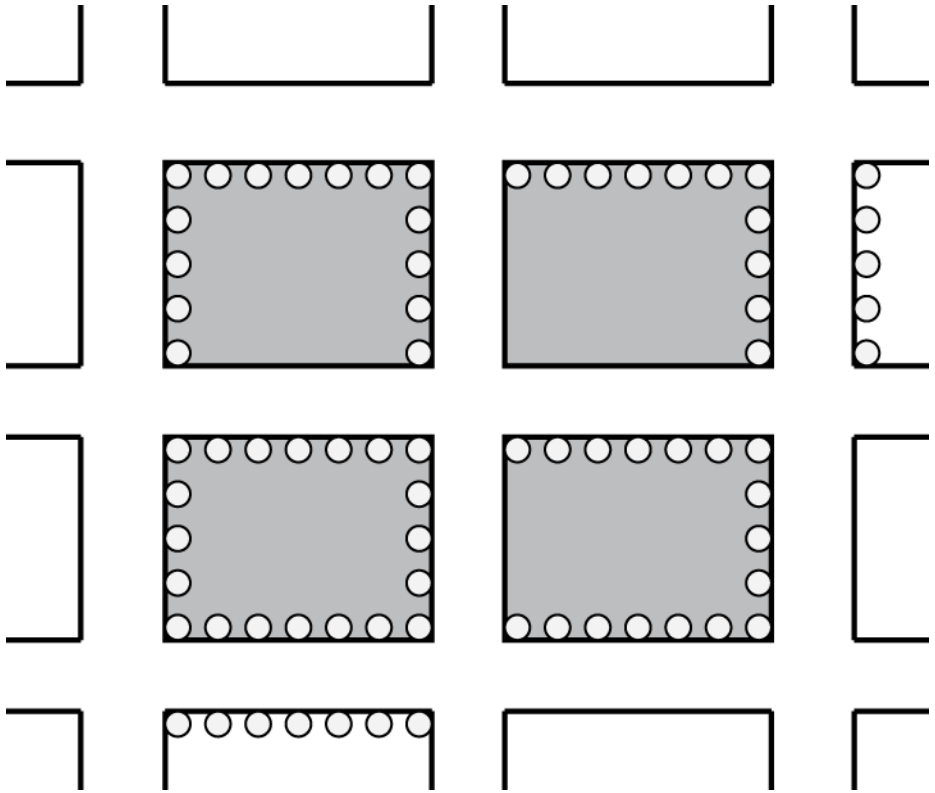
- Order matters for directed edges:
  u is not adjacent to v unless $(v, u) \in E$

Parking Meters

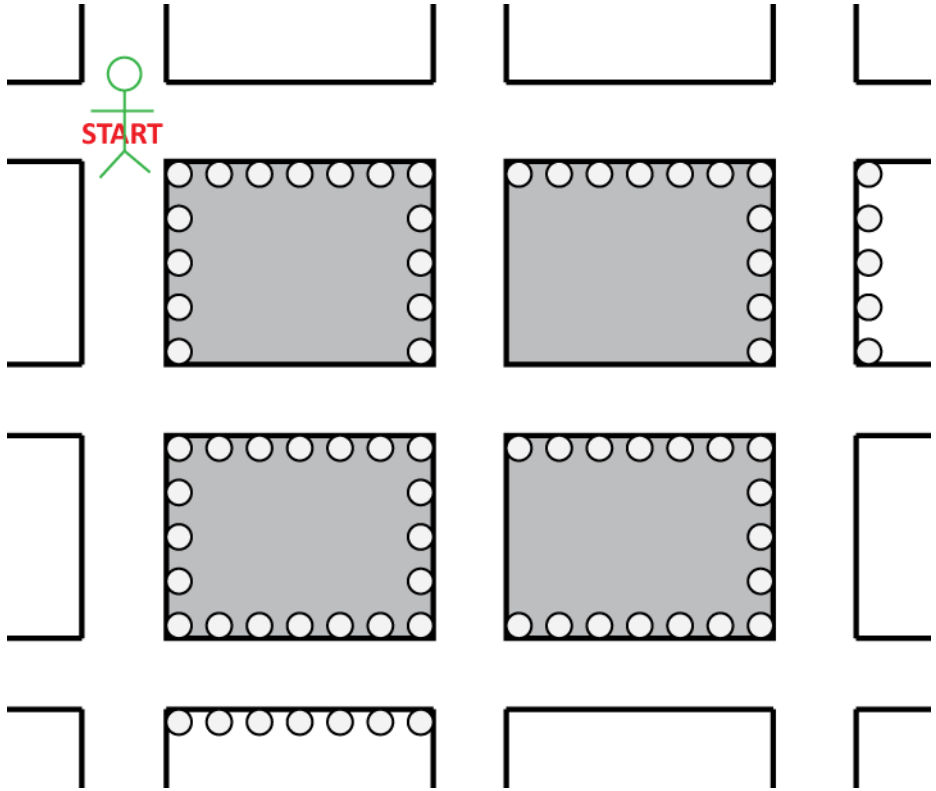
- Here is a map of the parking meters in a small neighborhood

- Our goal is to start at an intersection, check the meters, and return to our starting point…

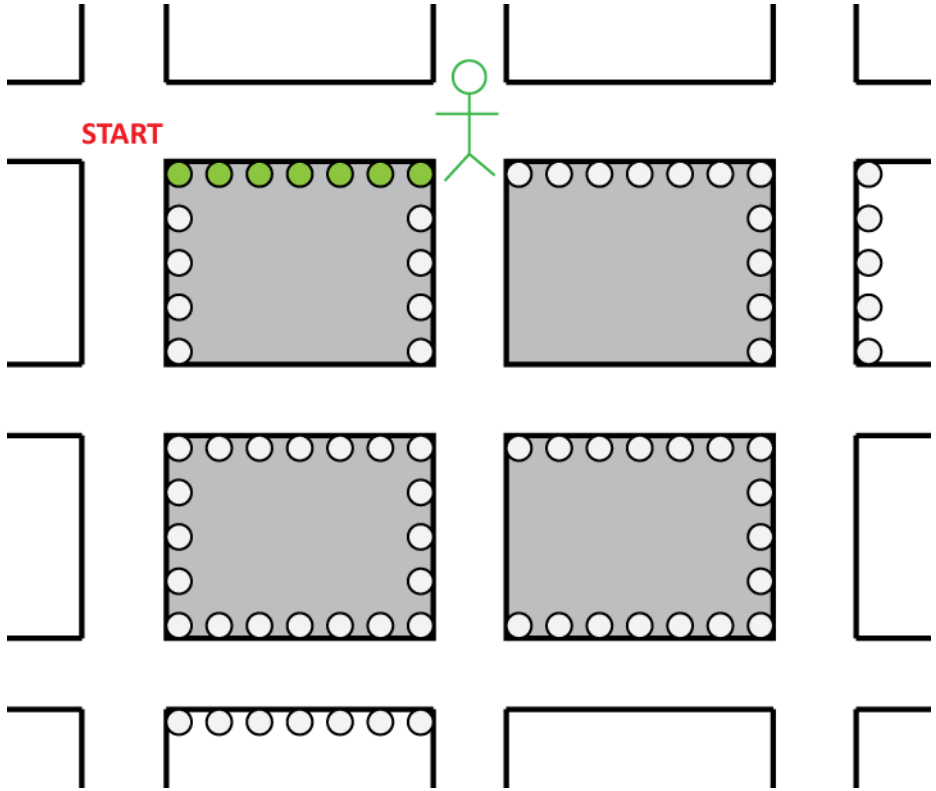
- …without retracing our steps!



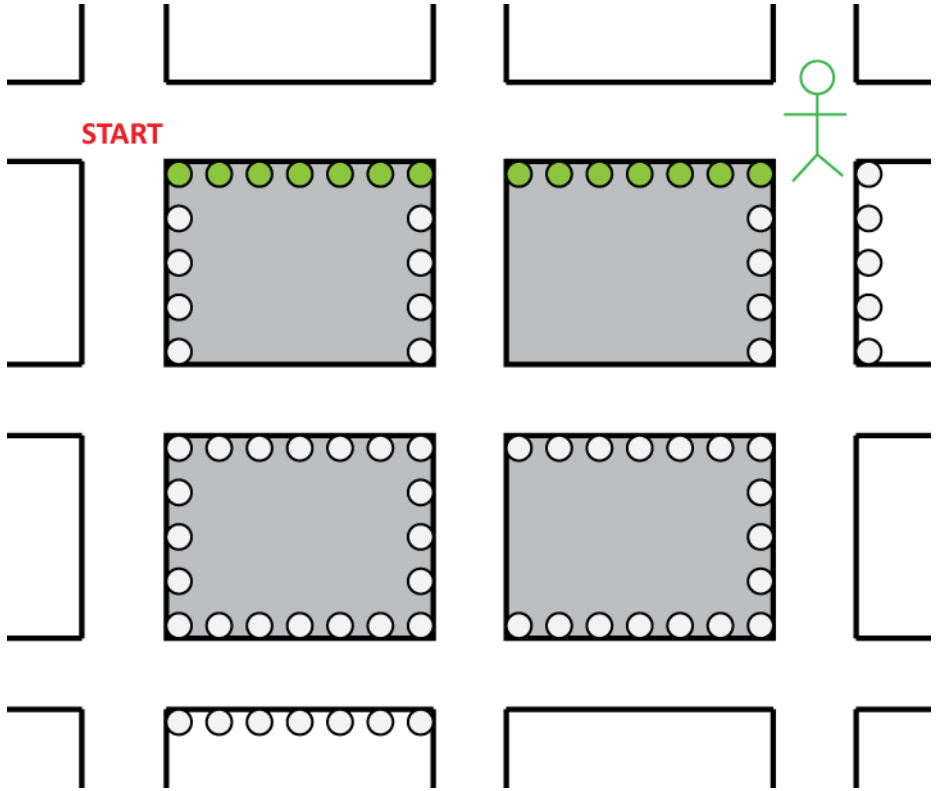
Solution Attempt #1



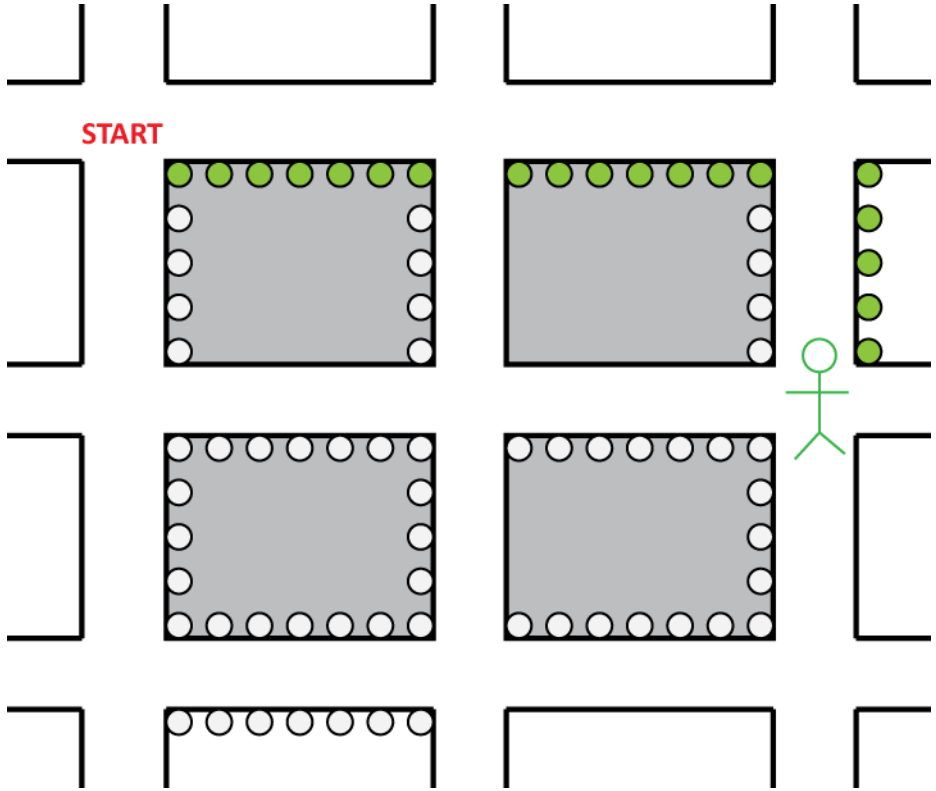
Solution Attempt #1



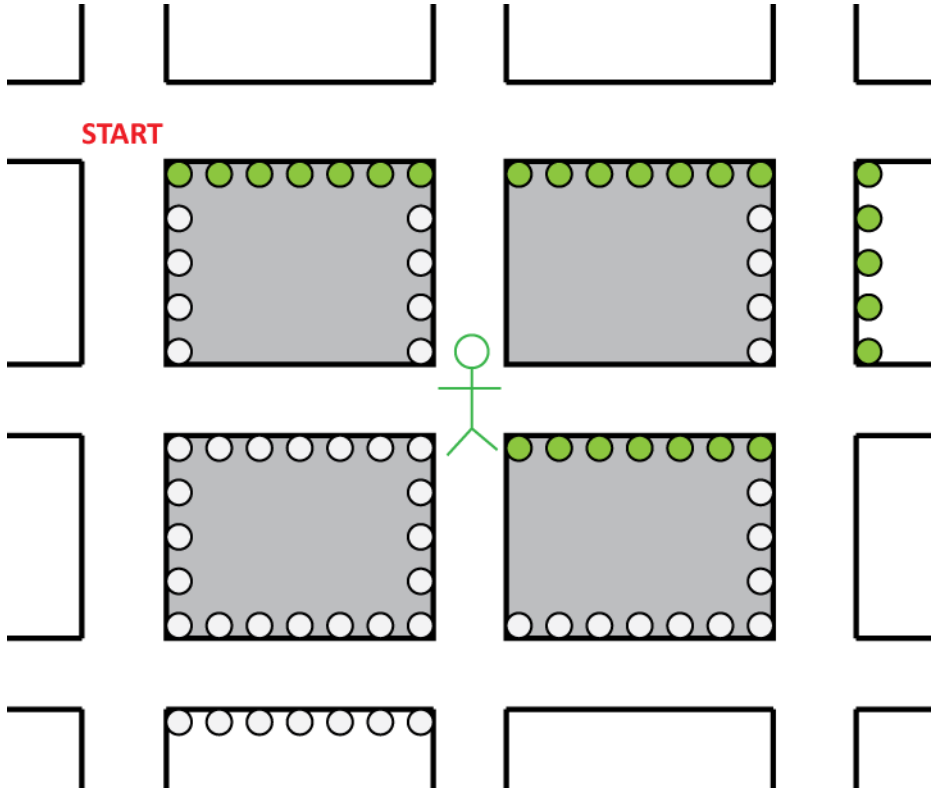
Solution Attempt #1



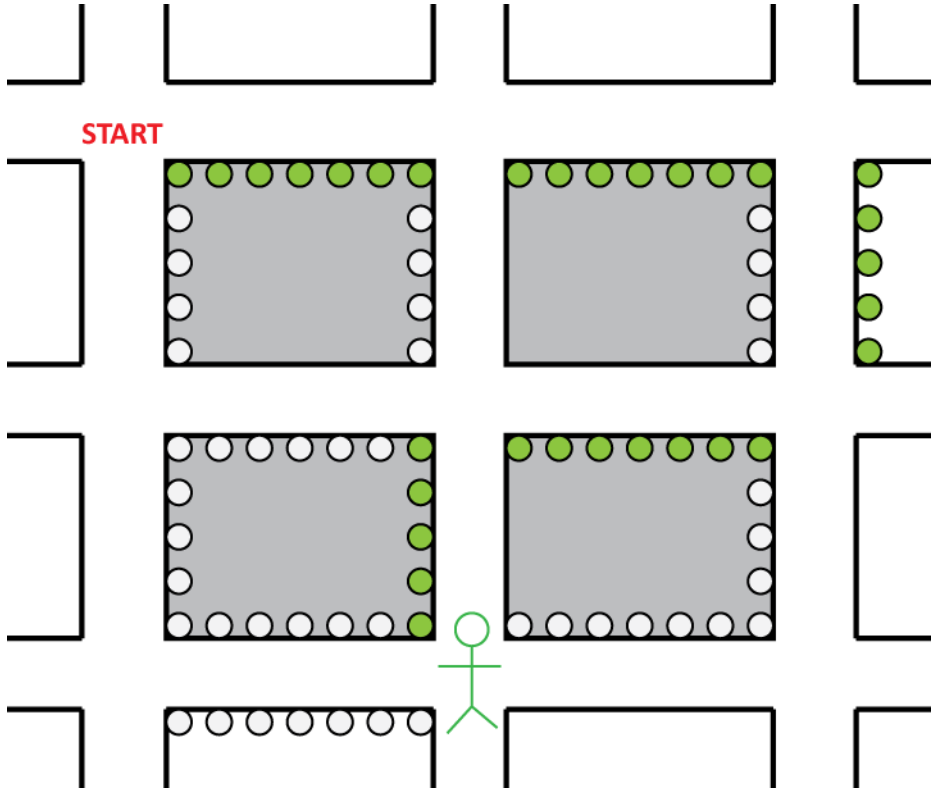
Solution Attempt #1



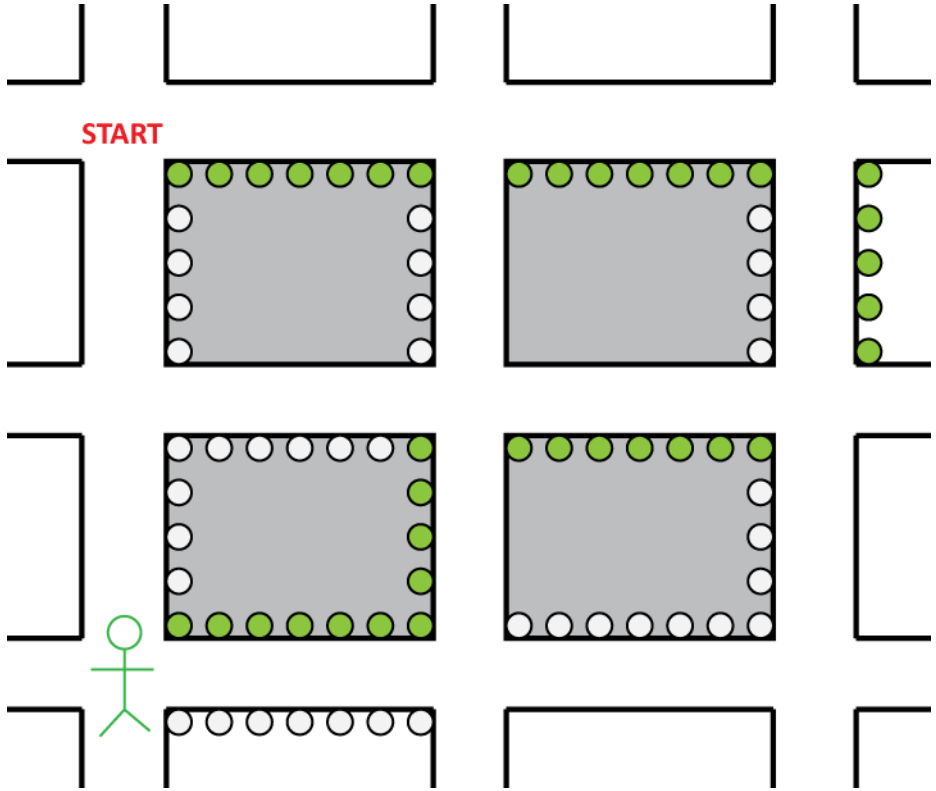
Solution Attempt #1



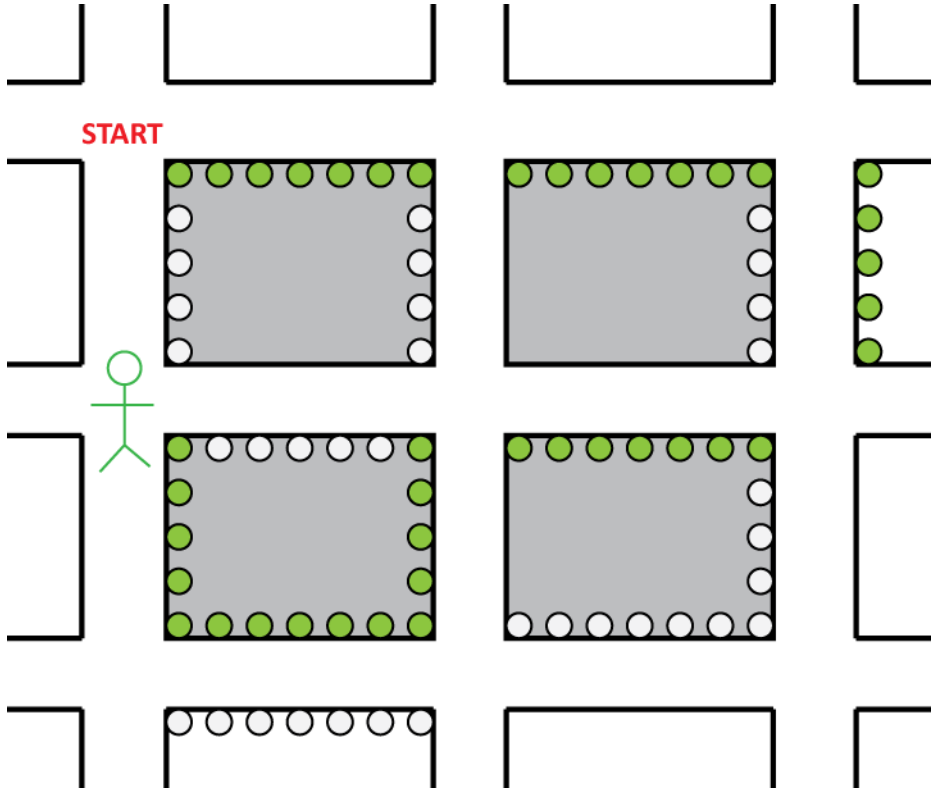
Solution Attempt #1



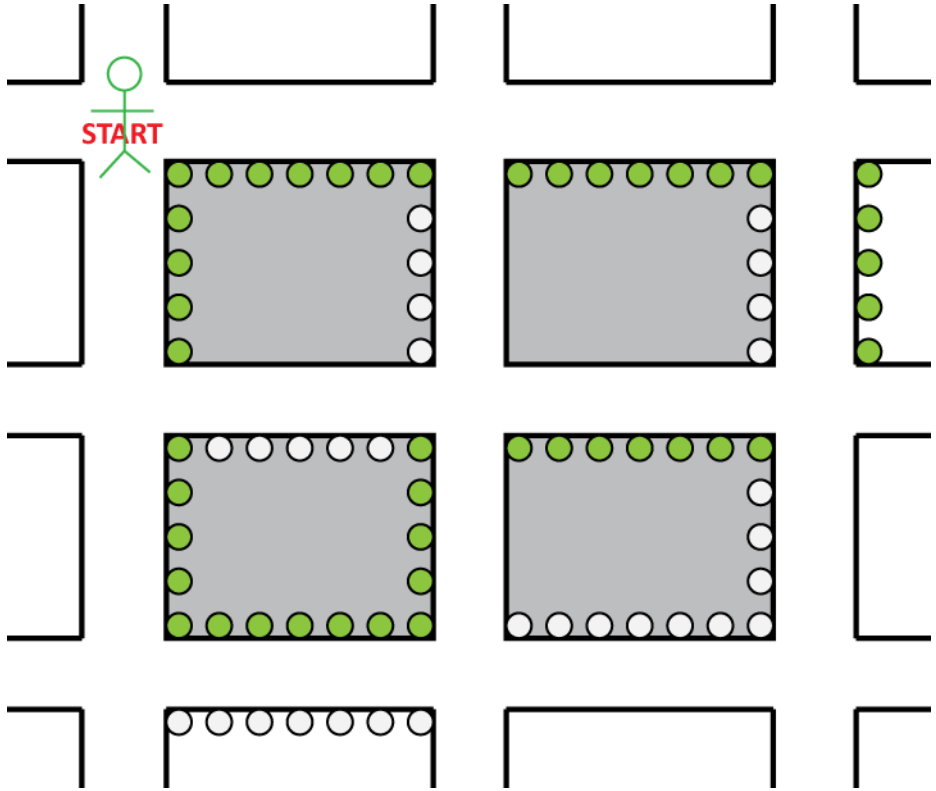
Solution Attempt #1



Solution Attempt #1

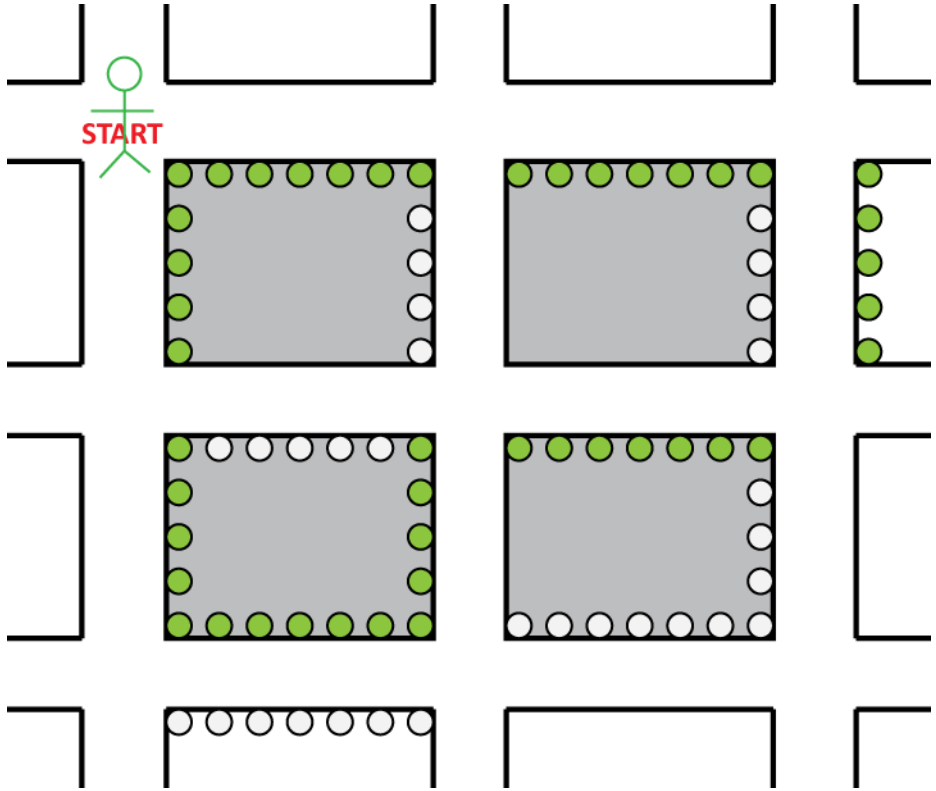


Solution Attempt #1



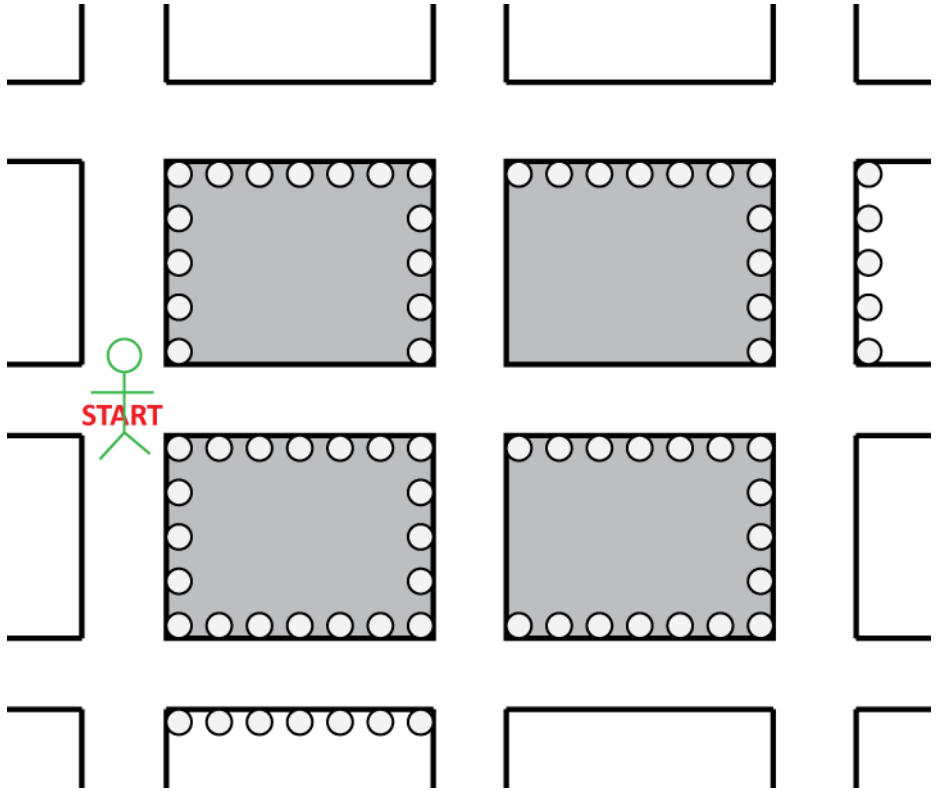
Solution Attempt #1

Now we're stuck, so we know this can't be a solution to the problem!

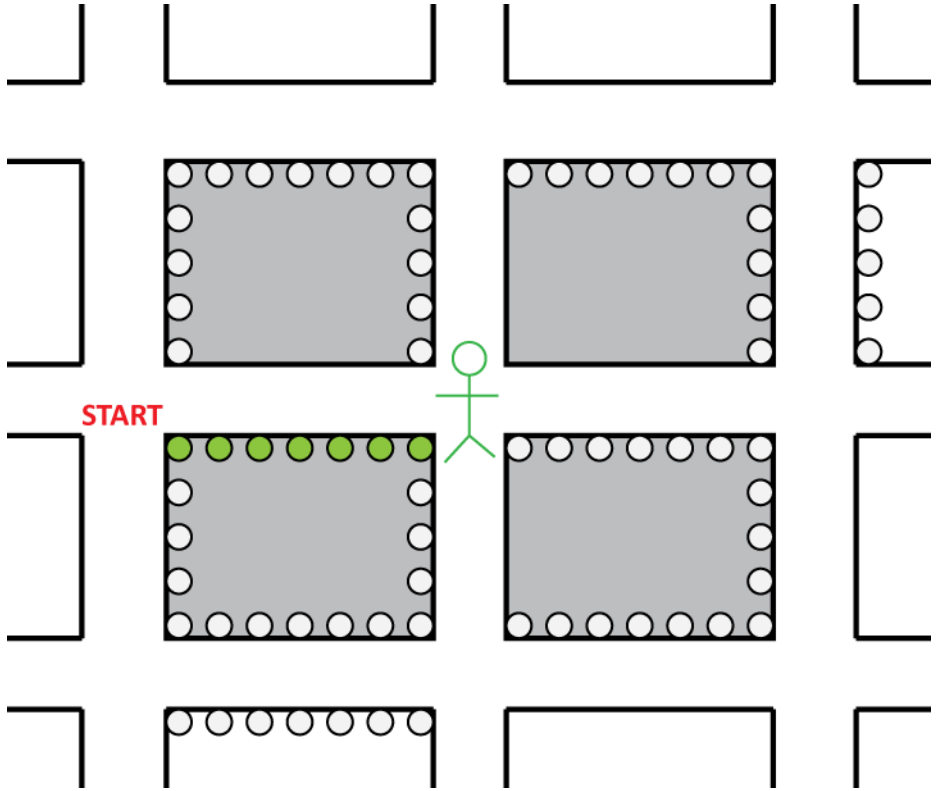


Solution Attempt #2

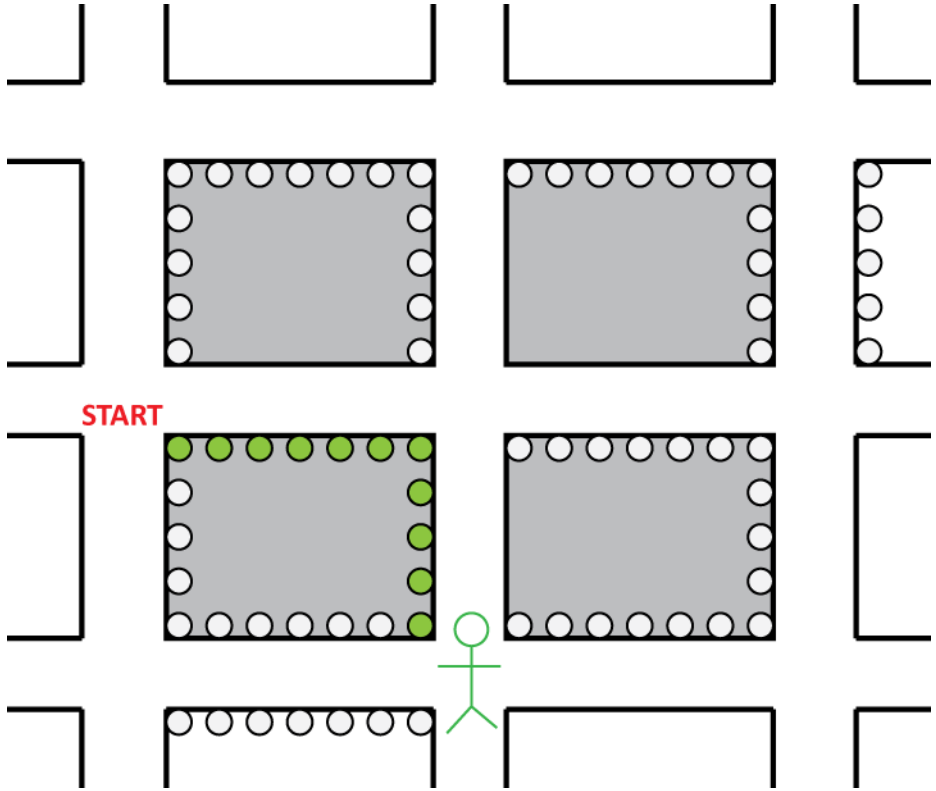
Let’s try again, this time with a different starting point…



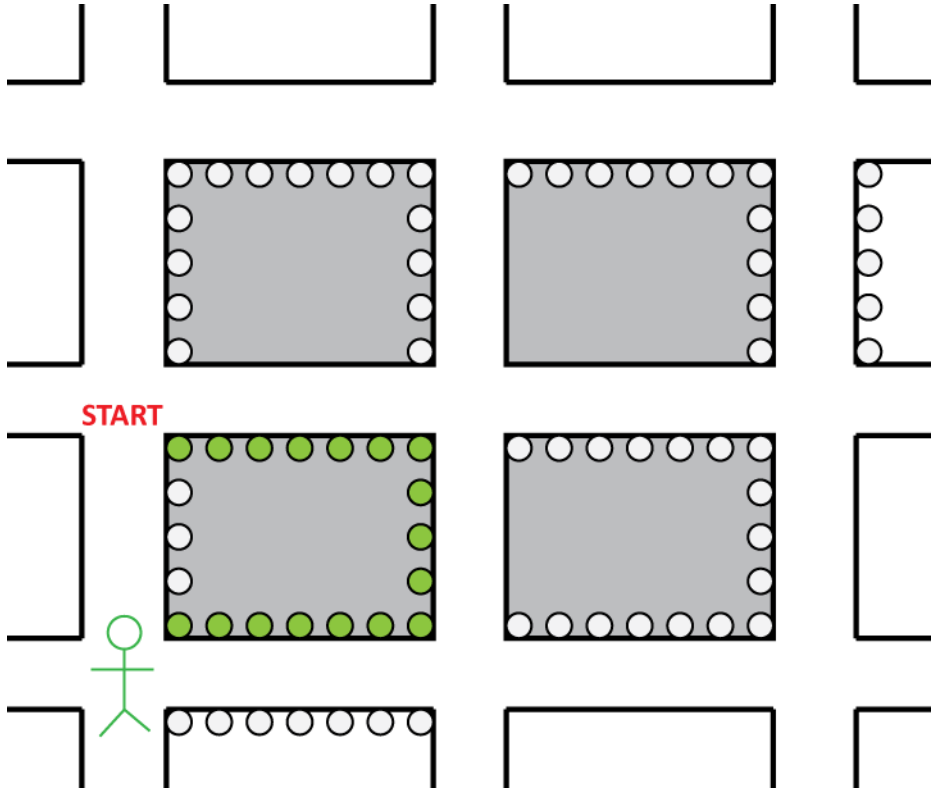
Solution Attempt #2



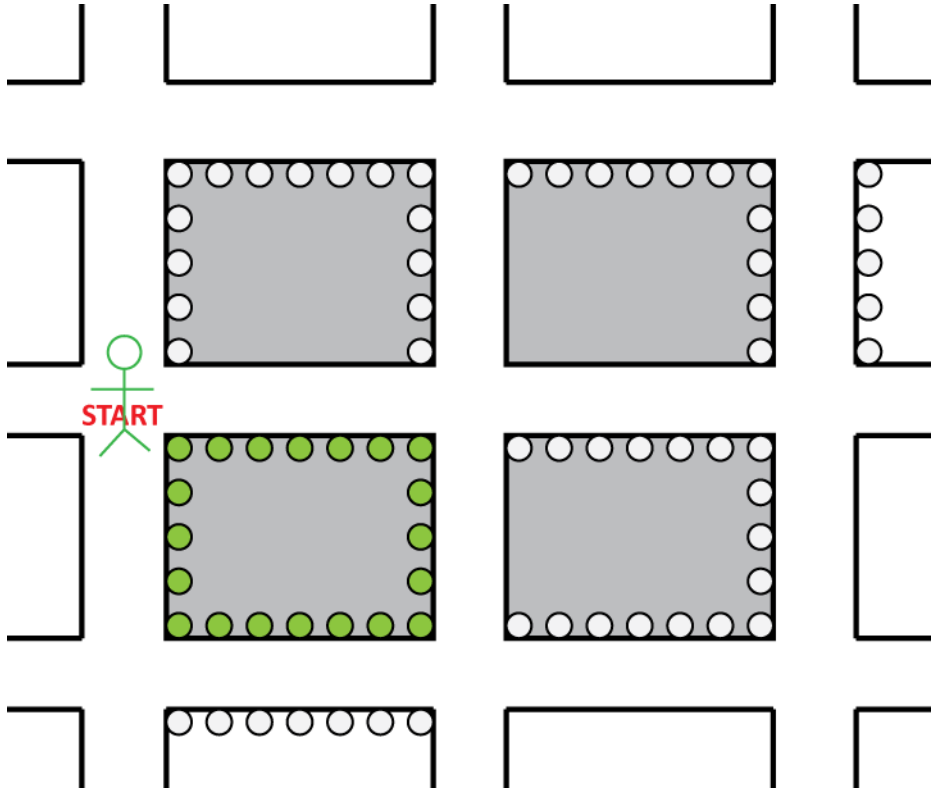
Solution Attempt #2



Solution Attempt #2

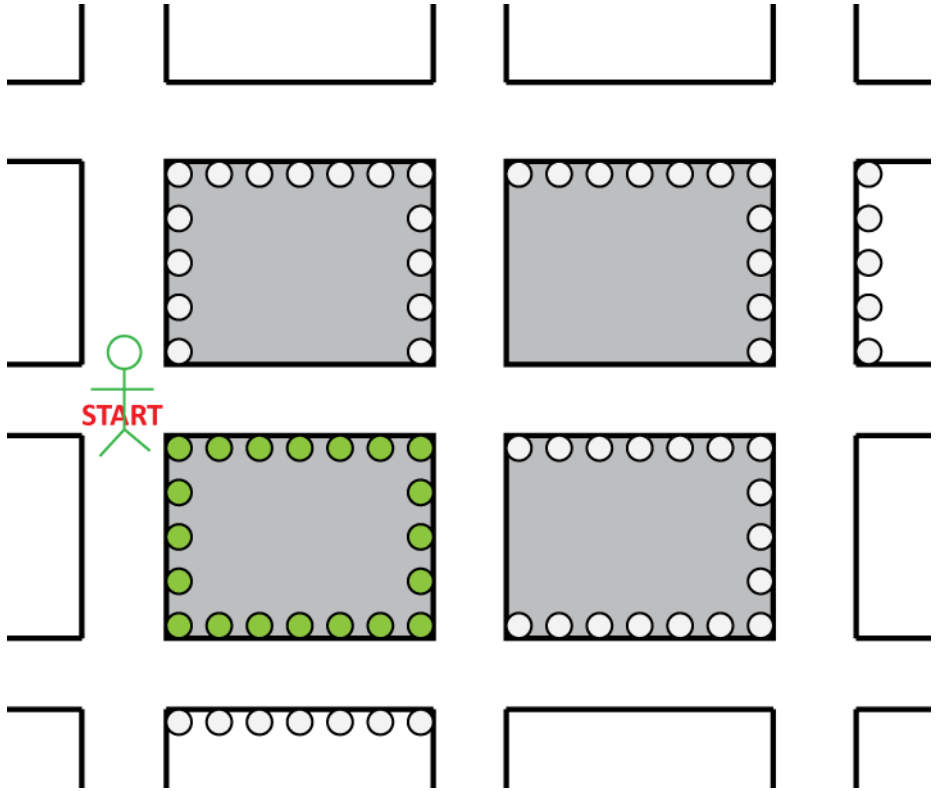


Solution Attempt #2

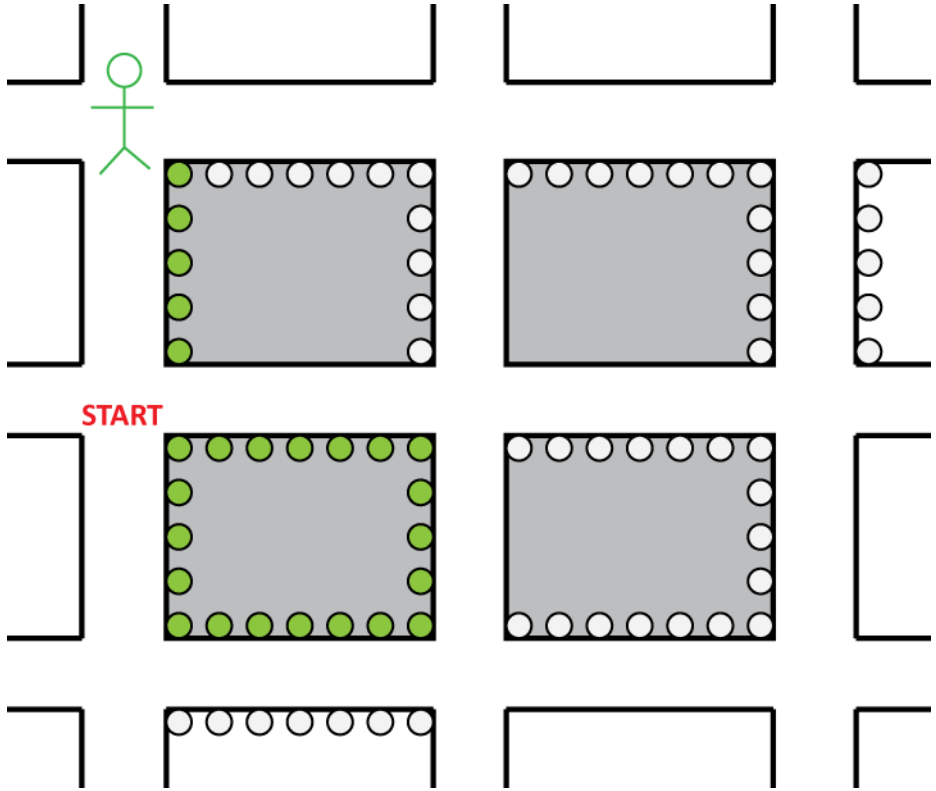


Solution Attempt #2

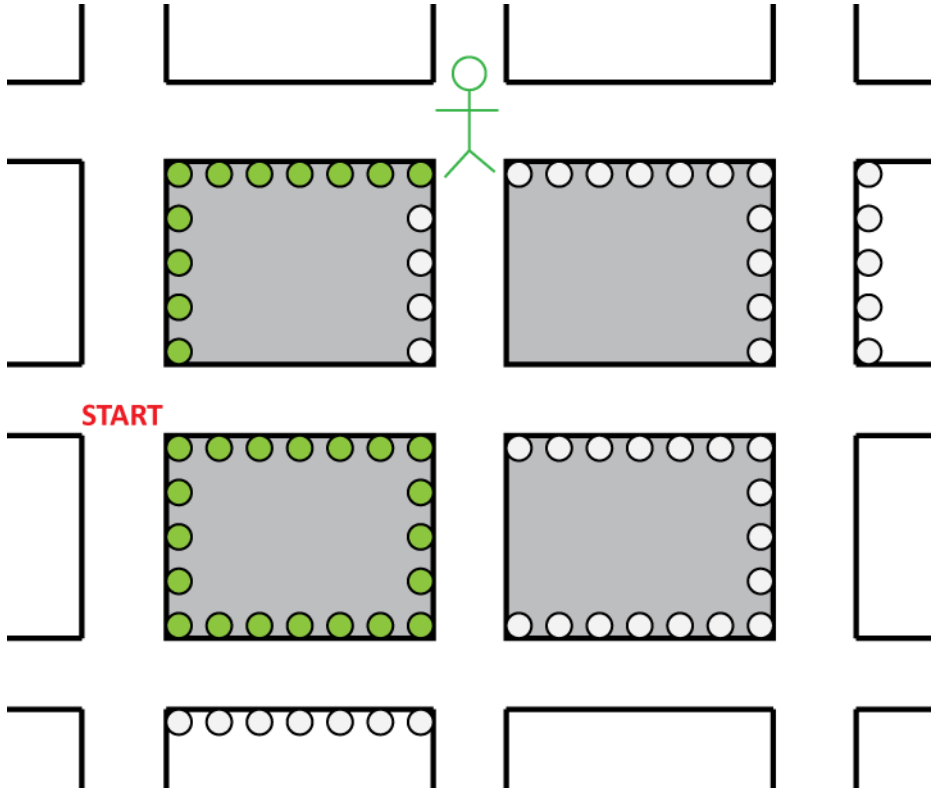
Notice that it's OK for us to return to the start since there is still a way to leave this spot without retracing our steps.



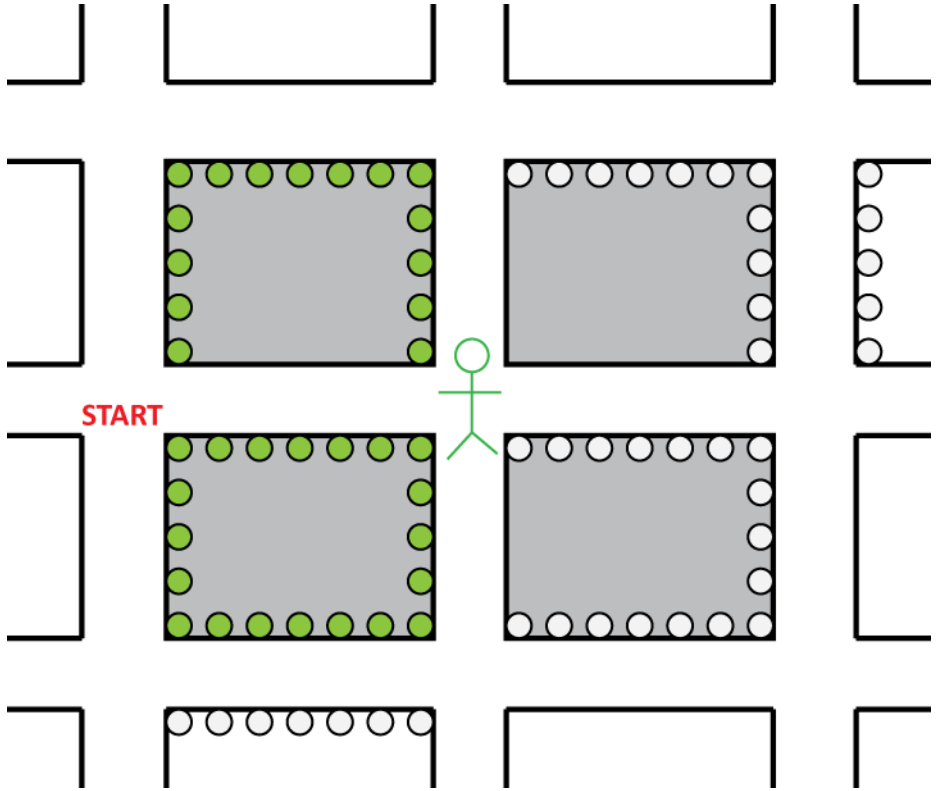
Solution Attempt #2



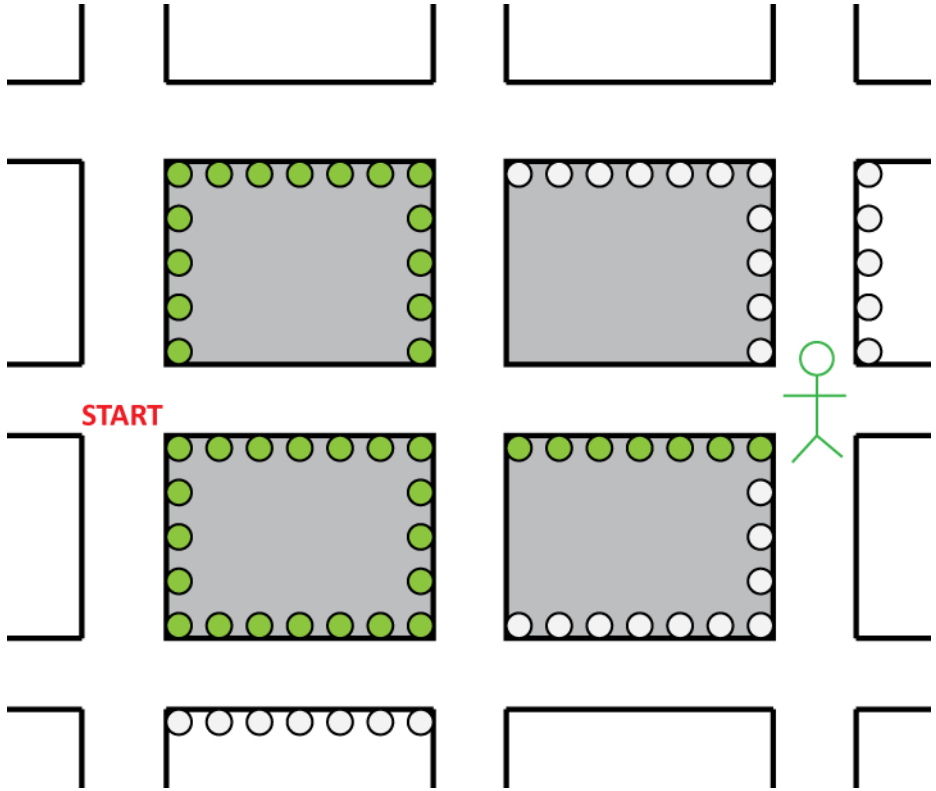
Solution Attempt #2



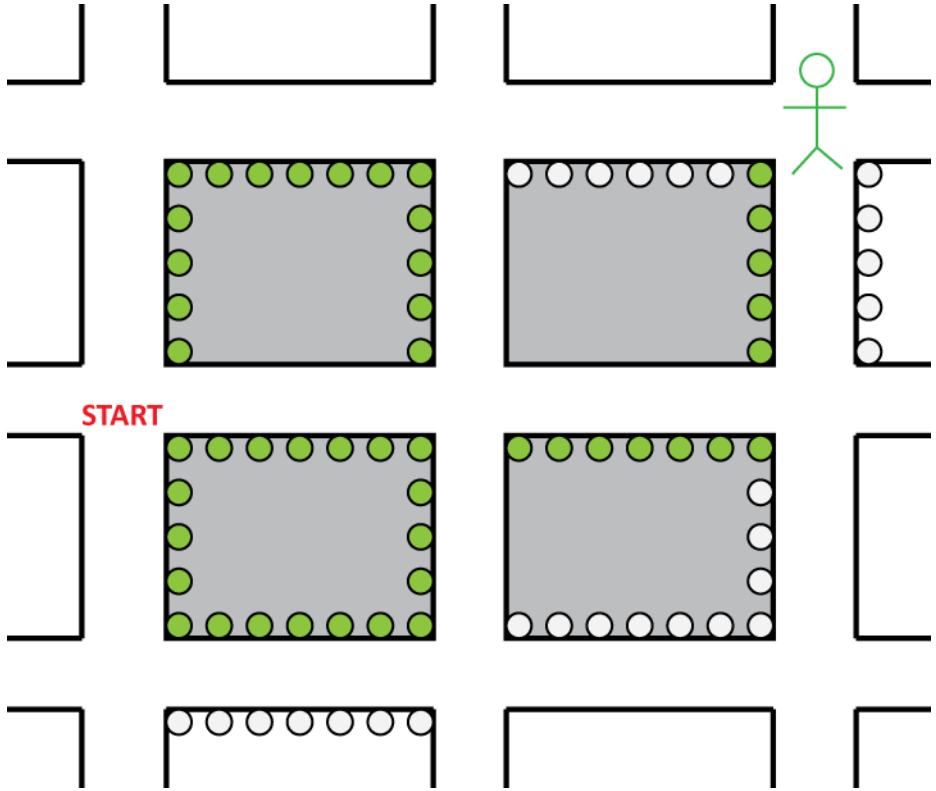
Solution Attempt #2



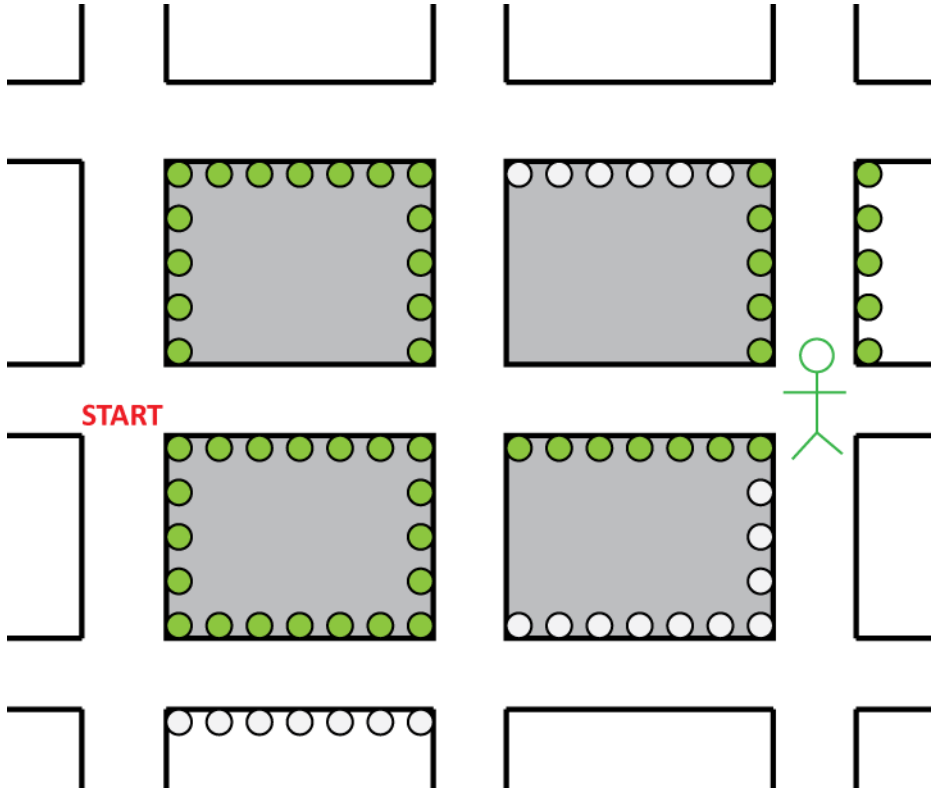
Solution Attempt #2



Solution Attempt #2

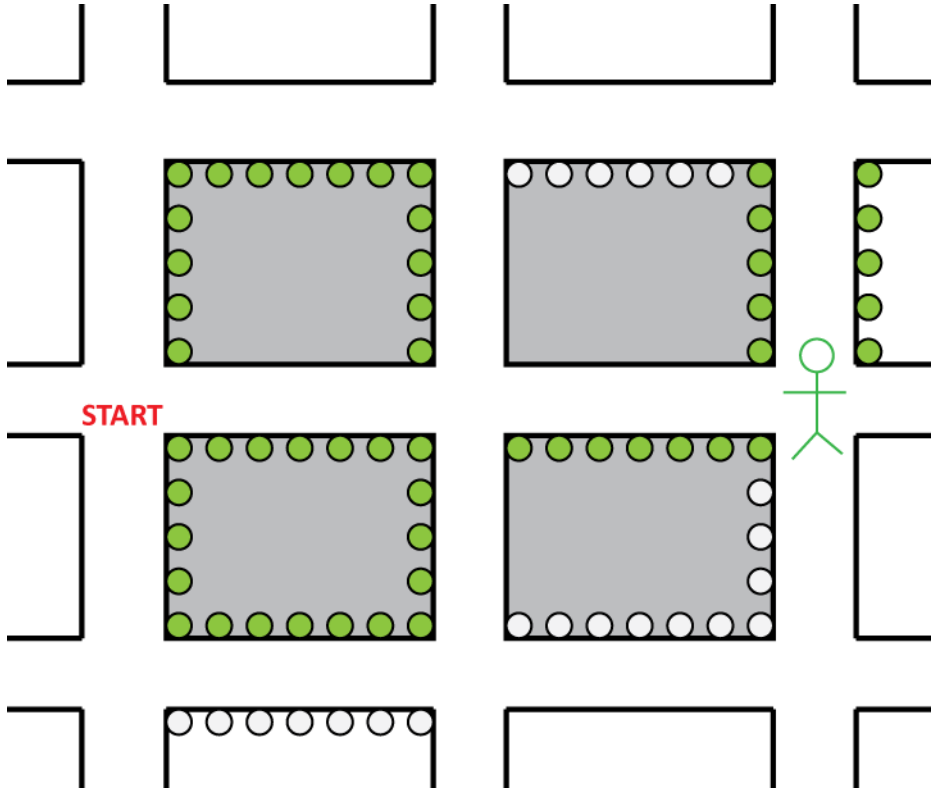


Solution Attempt #2

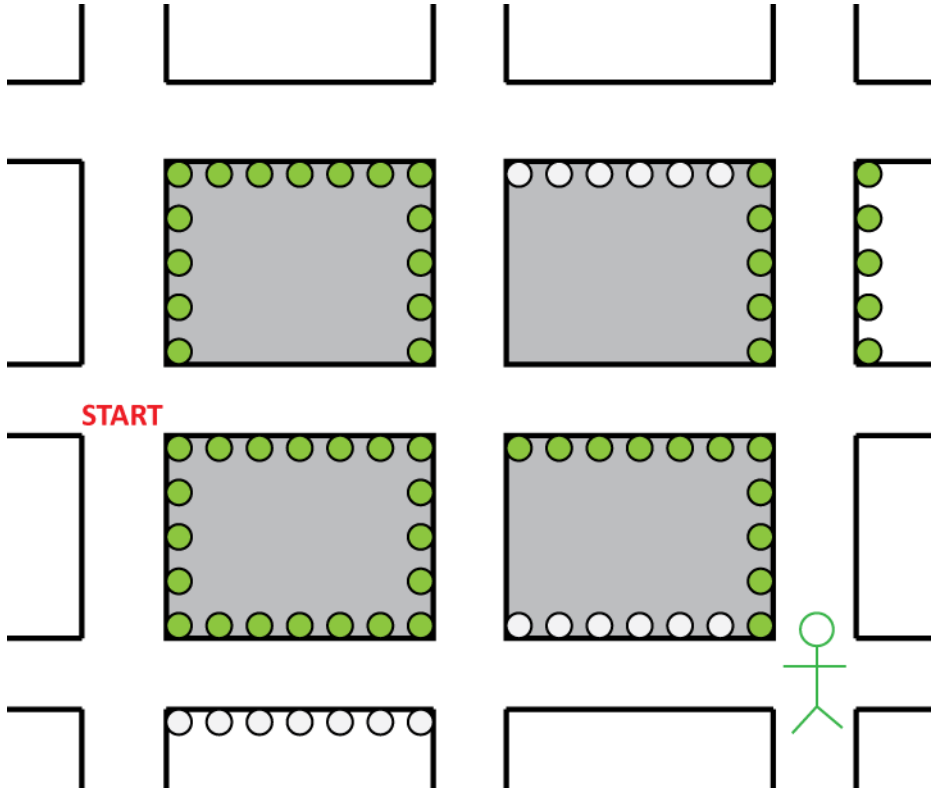


Solution Attempt #2

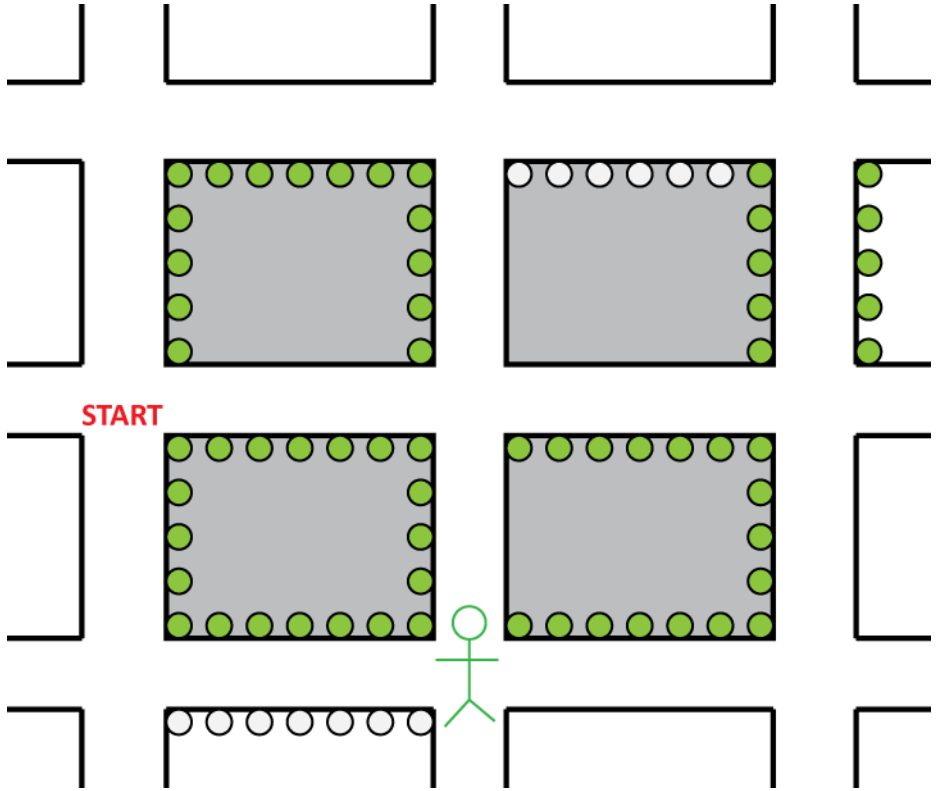
Notice how we just went back the way we came. This is not a “retrace” because there are meters on both sides of the street.



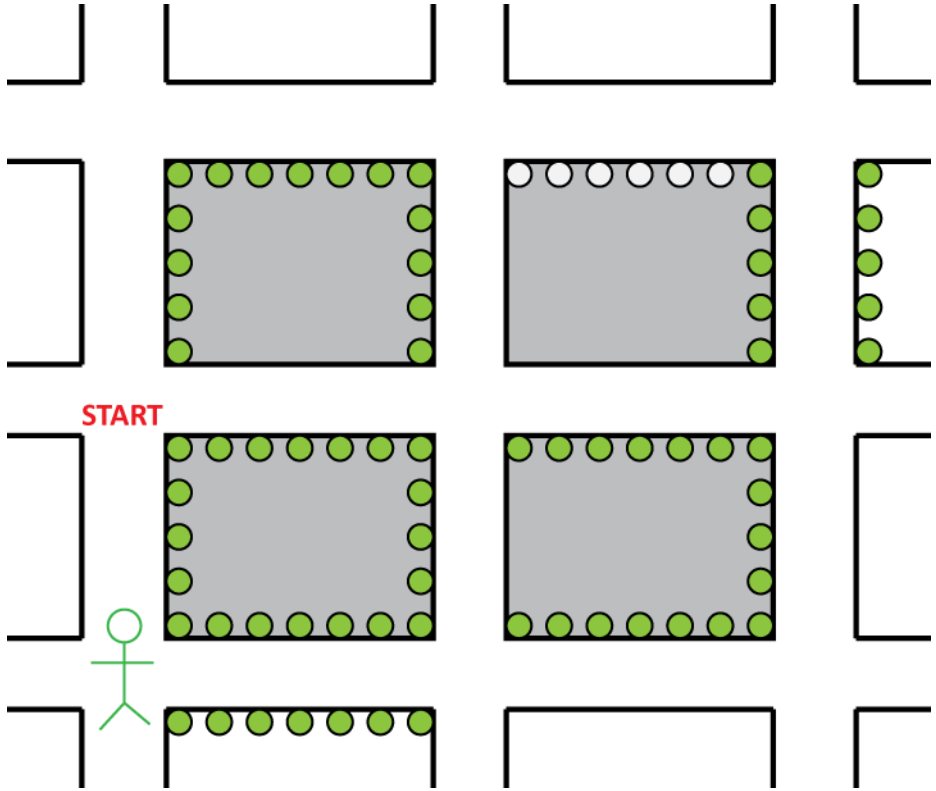
Solution Attempt #2



Solution Attempt #2

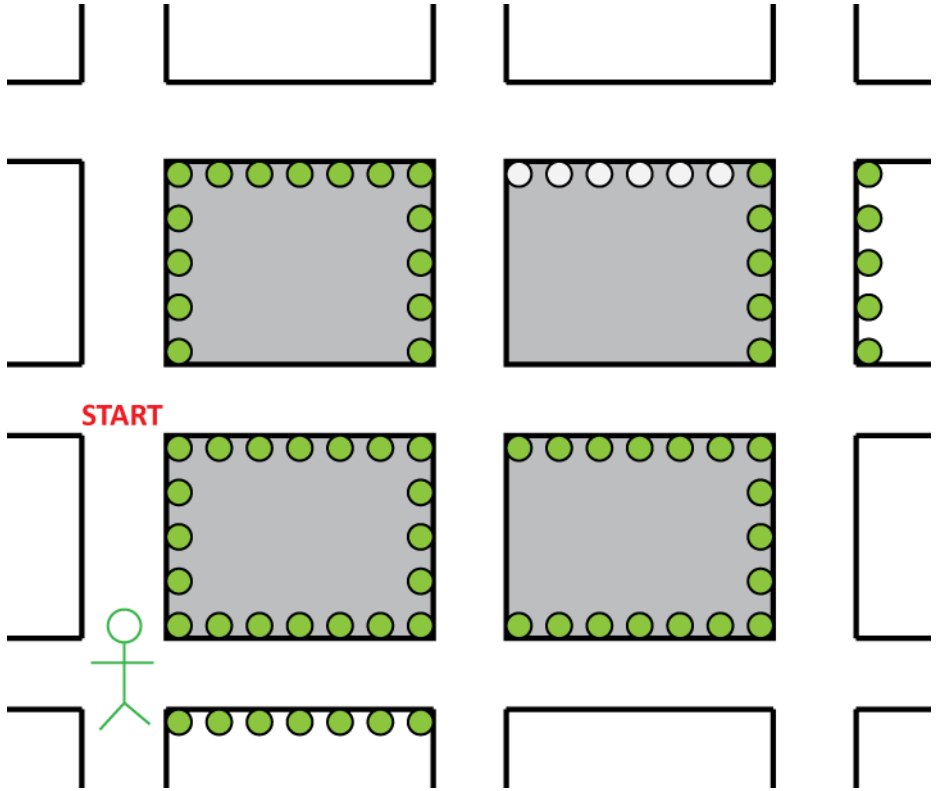


Solution Attempt #2



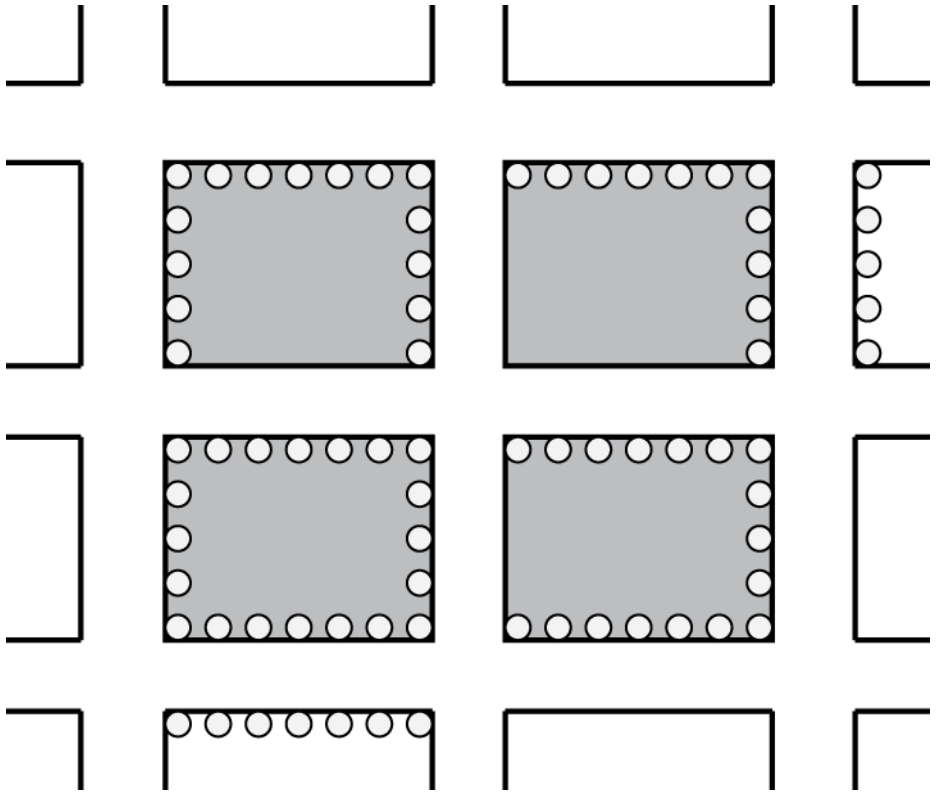
Solution Attempt #2

And we're stuck again, so we know this can't be a solution to the problem either!



You Try It

- We haven't had any luck so far… now it's your turn to give it a try
- Can you choose a starting point, visit all the parking meters, and return to the start without retracing?



Now What?

- We could continue to use trial and error to see if we can find a solution
- We might eventually become convinced that there is no solution
- We're going to use a *model* to represent this problem to make it easier to study

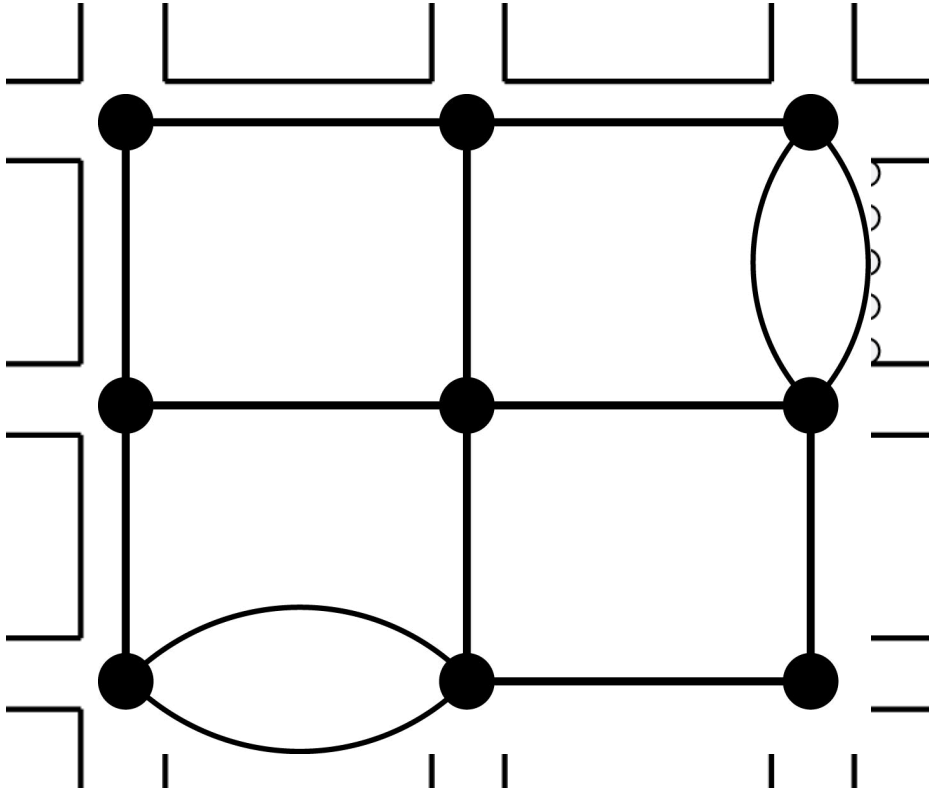
Models and Math

- A *model* is a mathematical tool that simplifies a problem, but keeps all of the important information

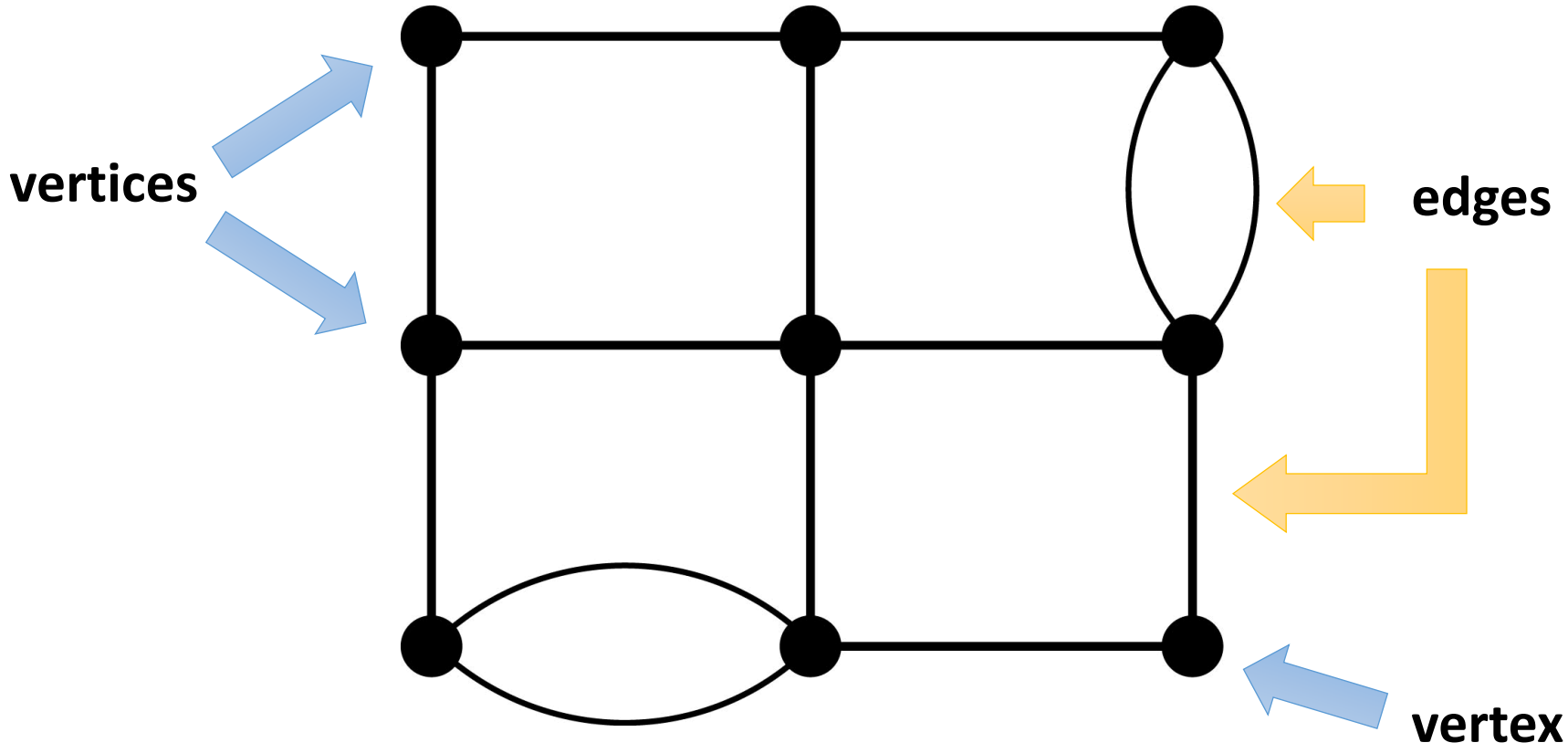
- For example, if you ever had to read a word problem and create an equation that you used to solve it, then you have created a mathematical model

Our Model: A “Graph”

- In our model, we will represent each intersection by a big black dot called a “**vertex**”
- Each row of parking meters is connected by a line called an “**edge**”



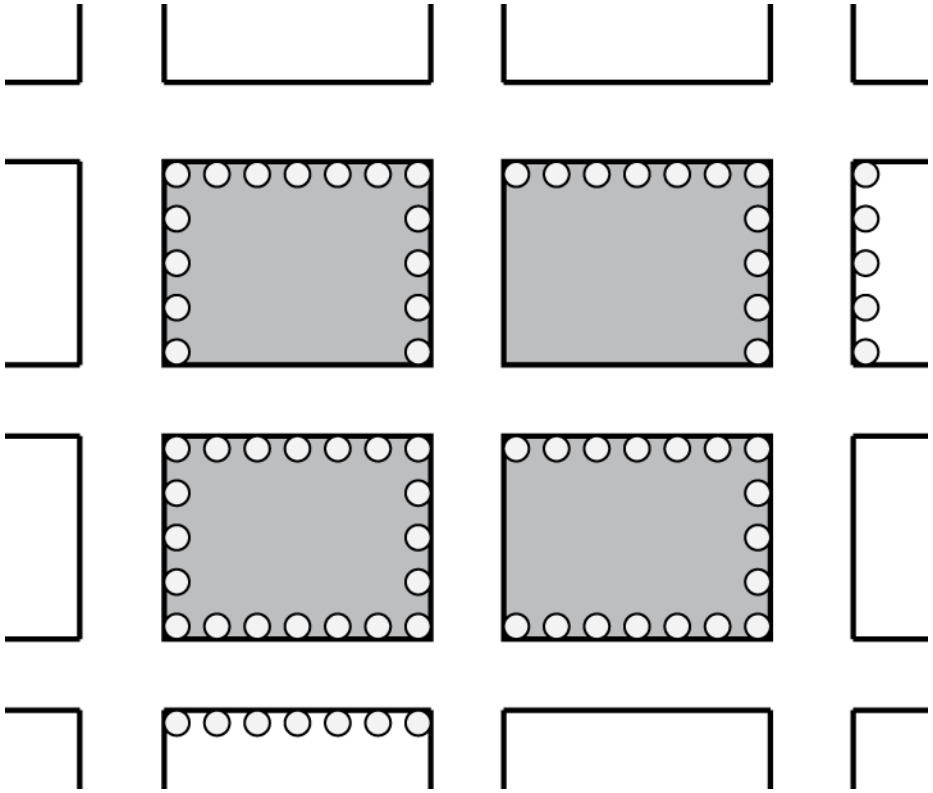
Graph

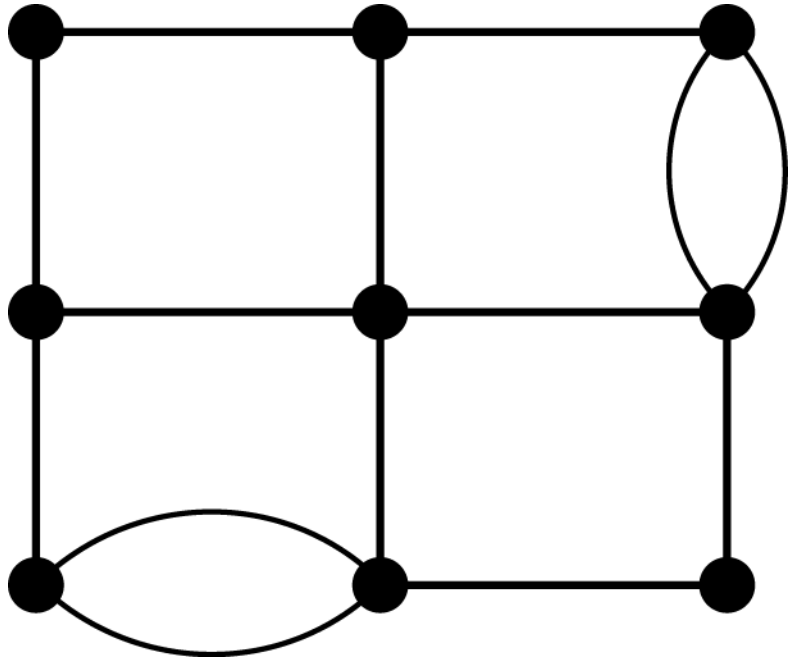


Using the Graph

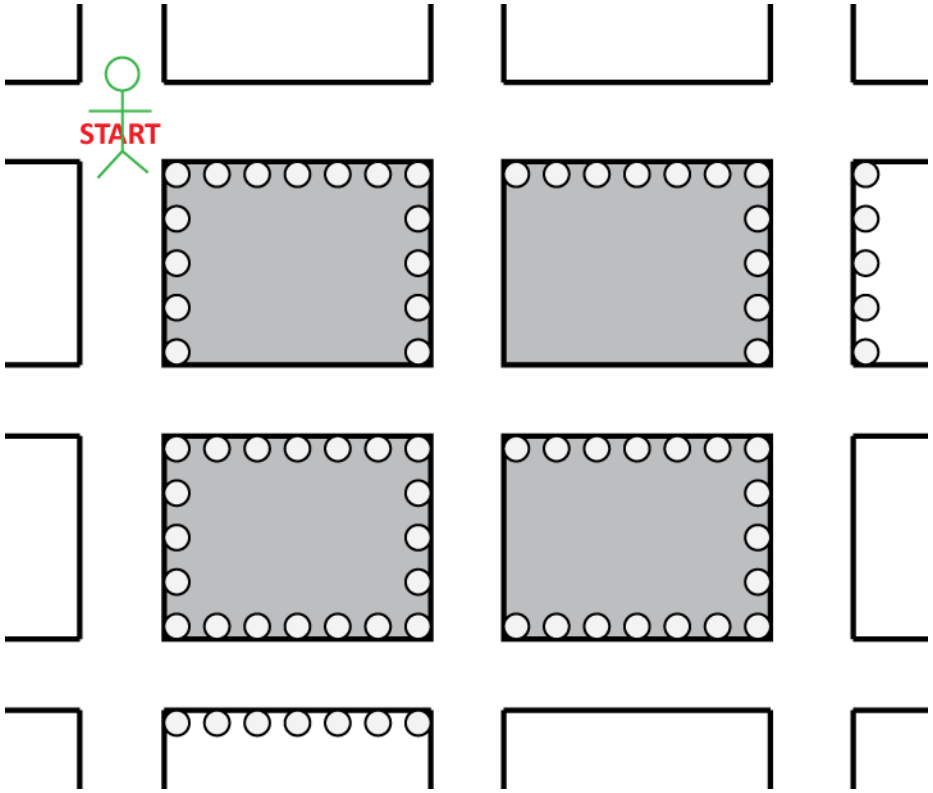
- The graph contains all of the information we need to solve our parking meter problem
- We don’t need to know the names of the streets or how long they are
- All we need to know is which streets have parking meters, and how they intersect each other

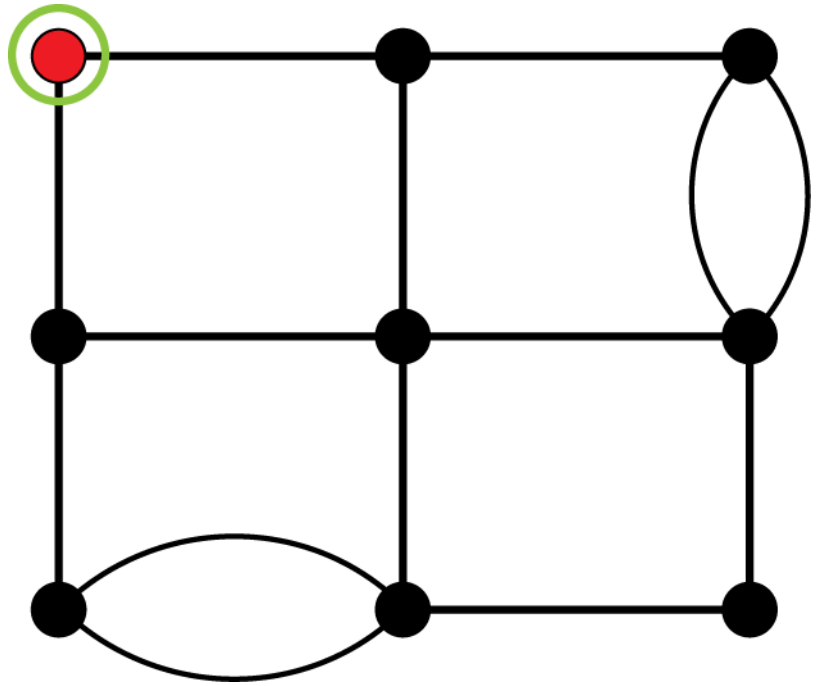
Two Paths: Map vs. Graph



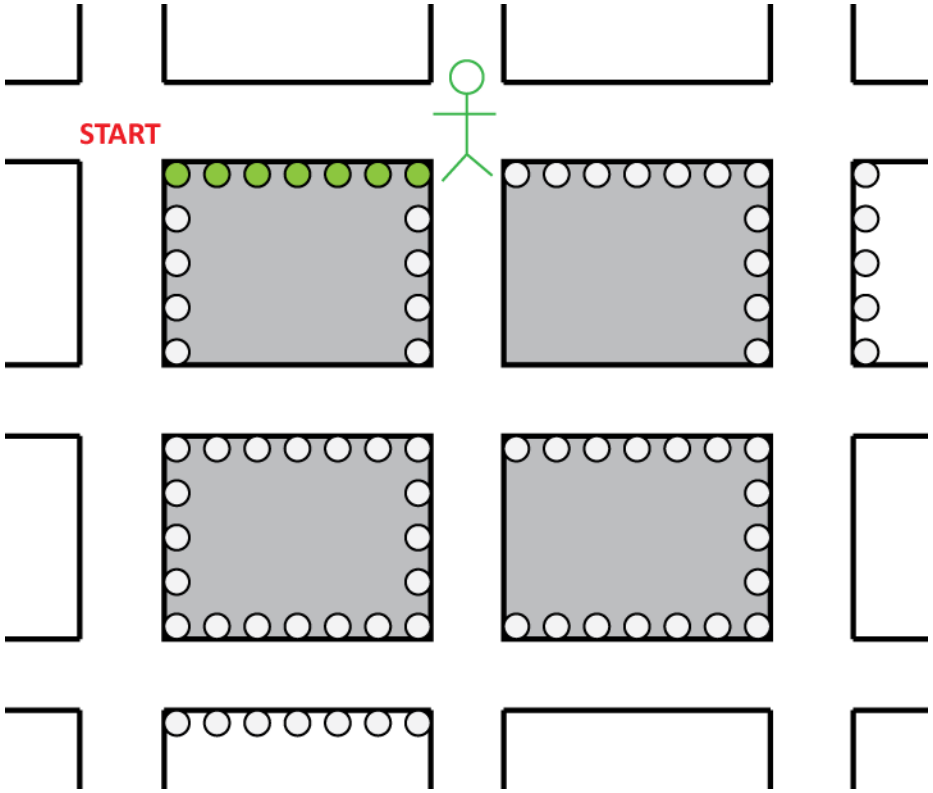


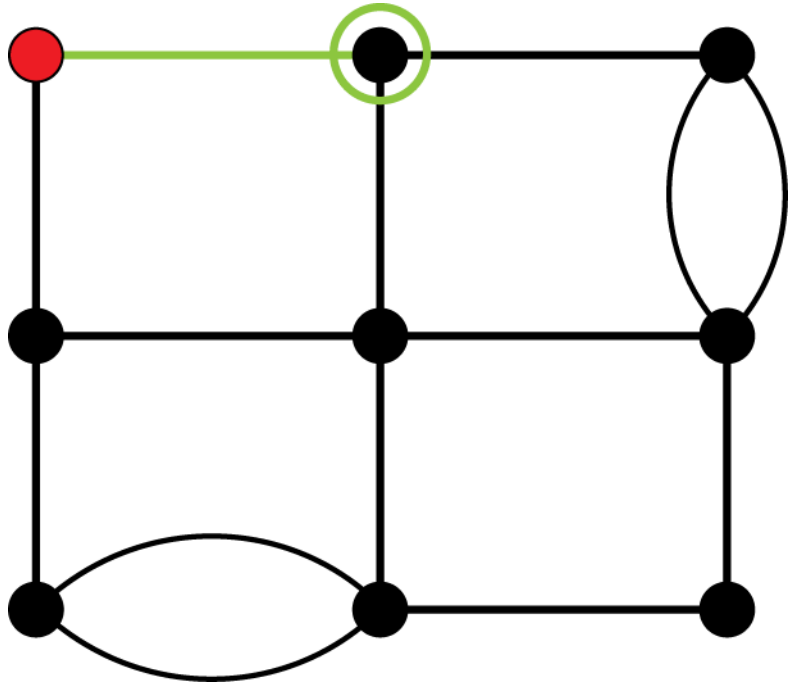
Two Paths: Map vs. Graph



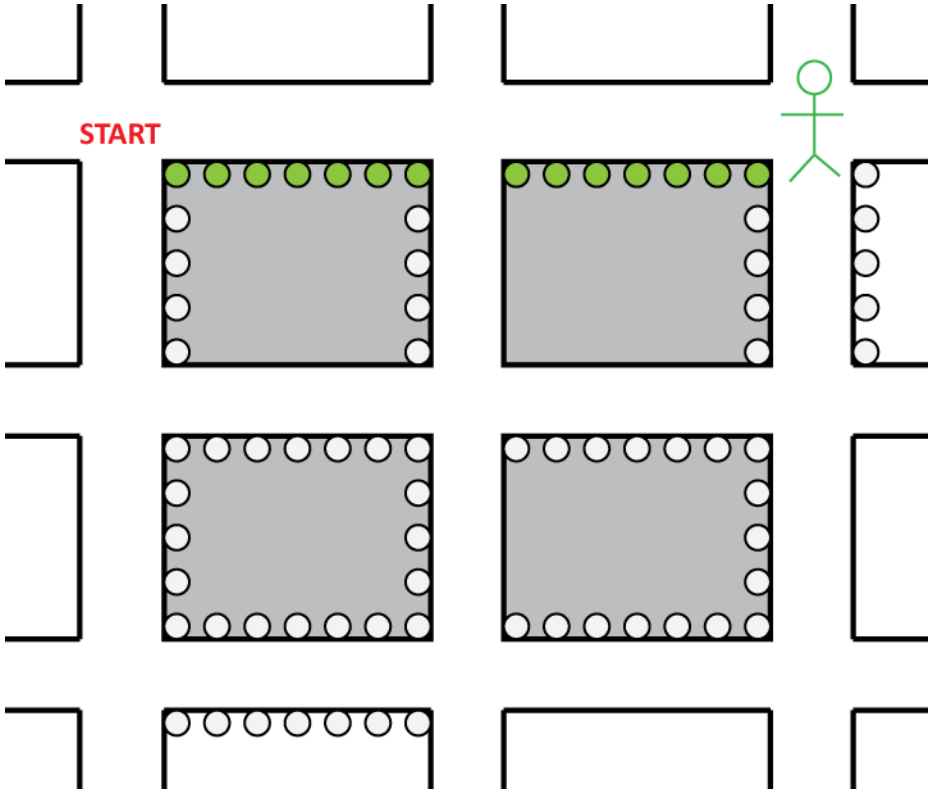


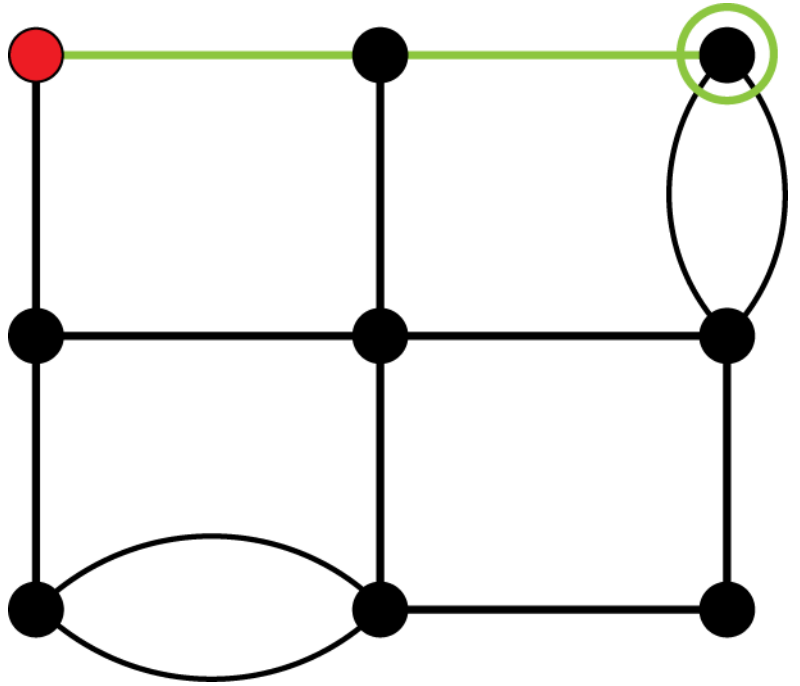
Two Paths: Map vs. Graph



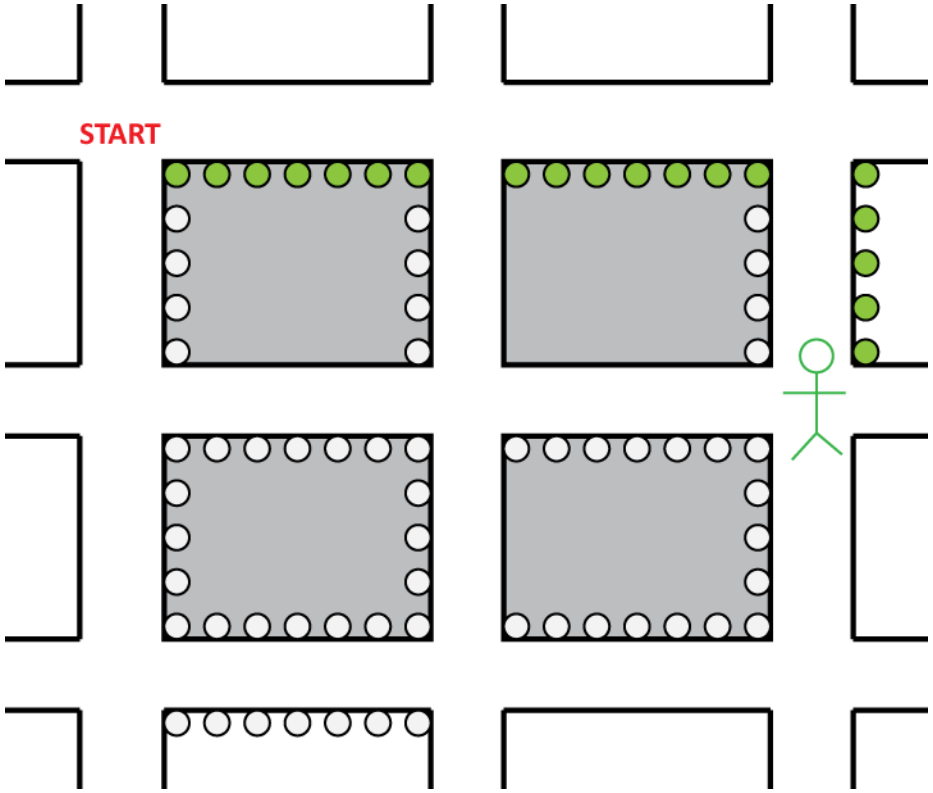


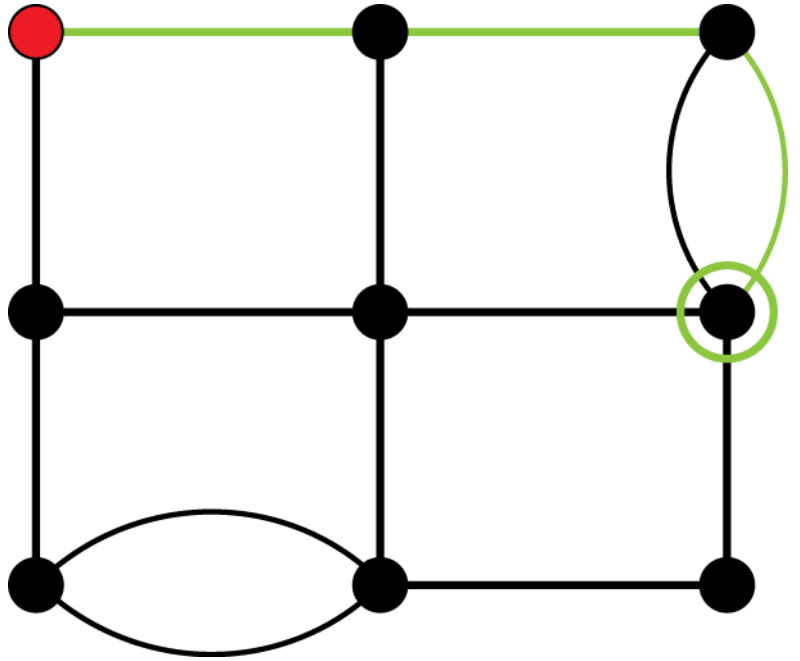
Two Paths: Map vs. Graph



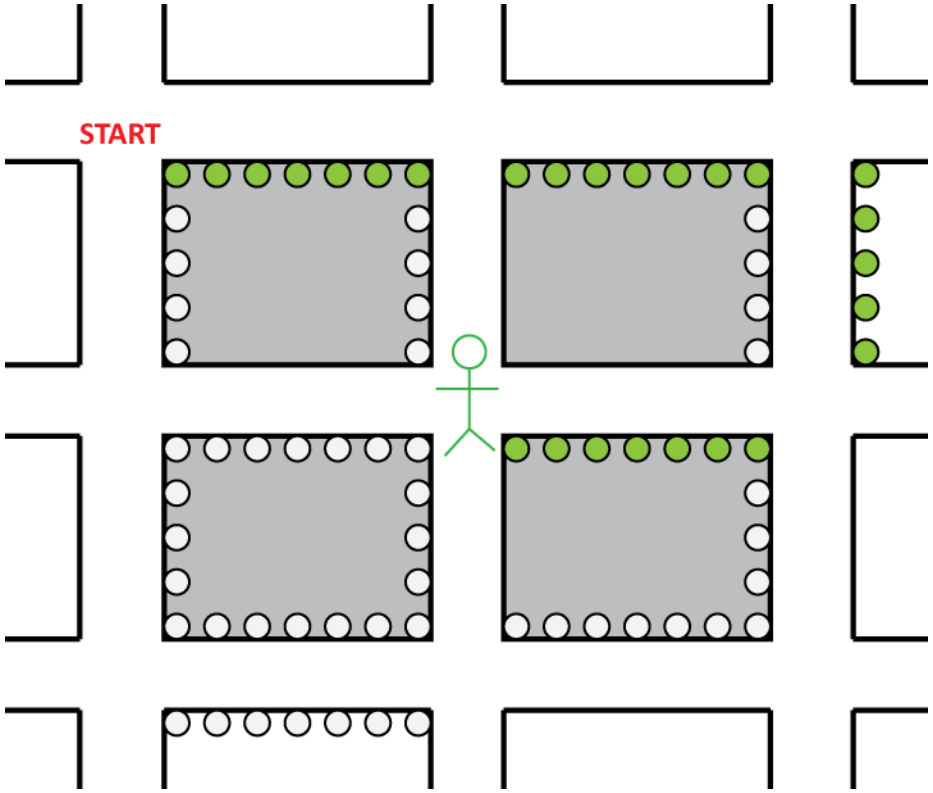


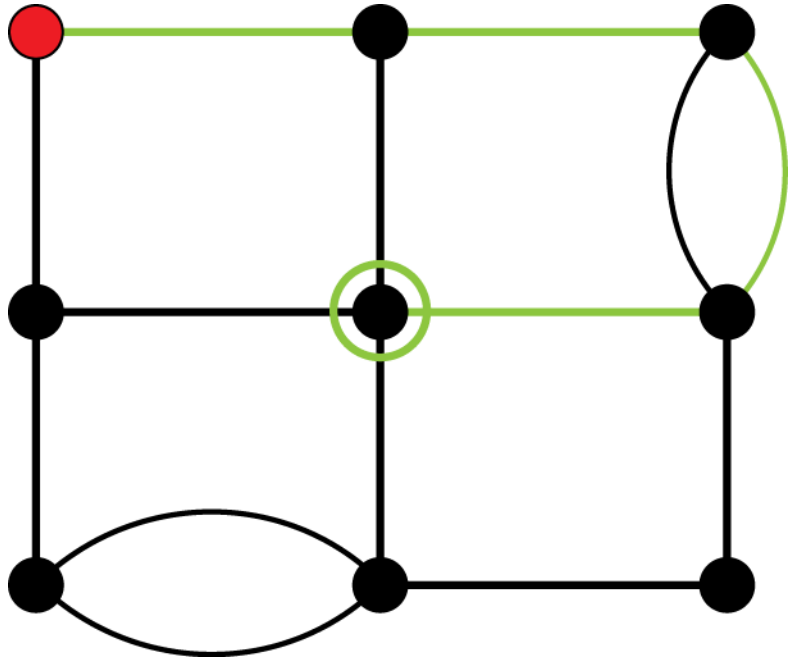
Two Paths: Map vs. Graph



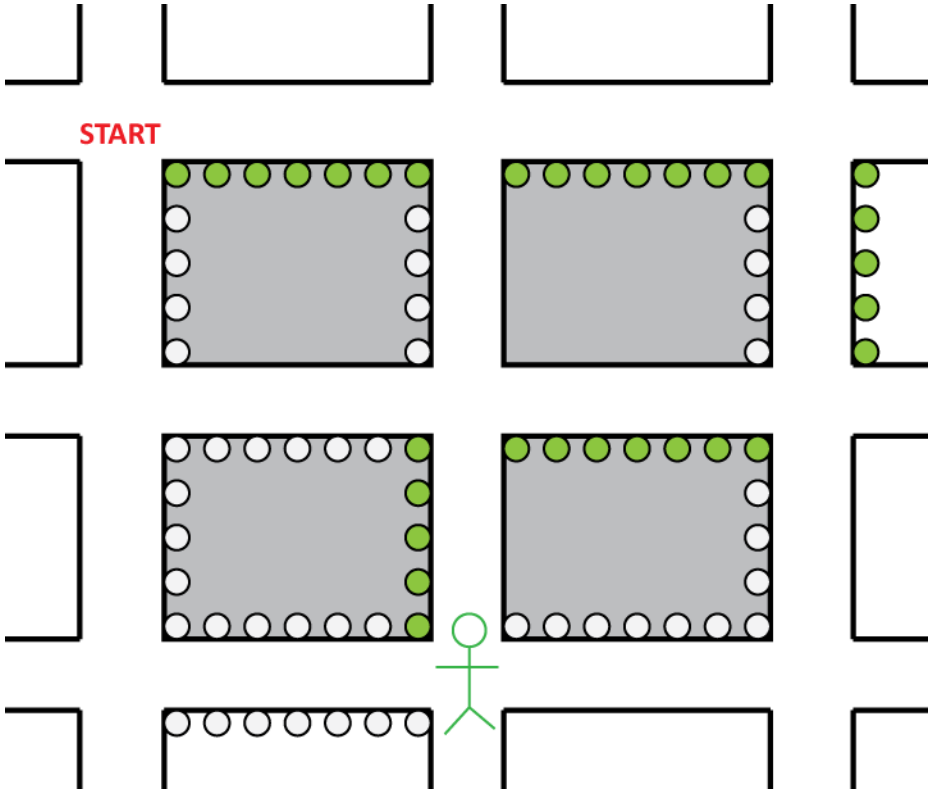


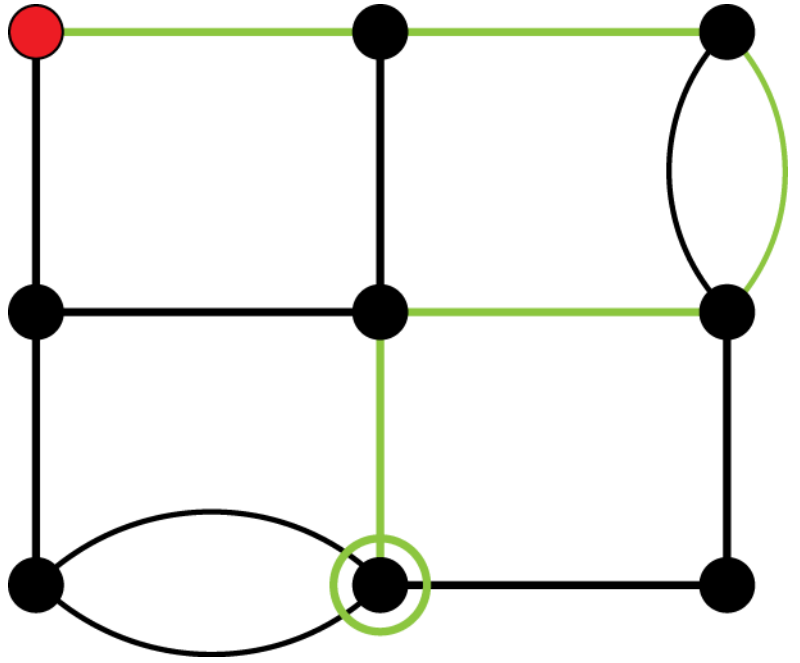
Two Paths: Map vs. Graph



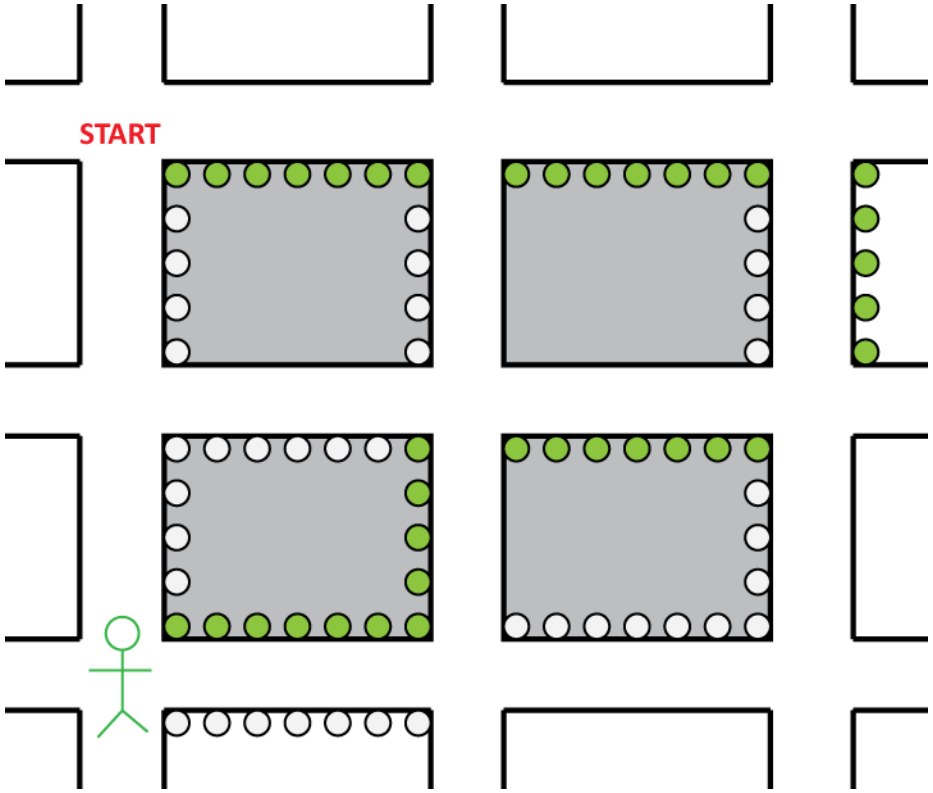


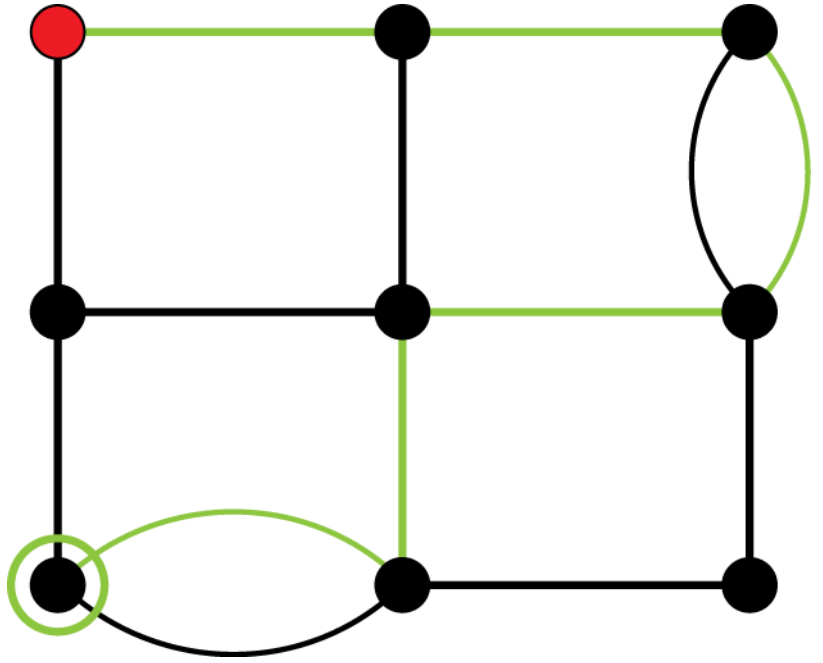
Two Paths: Map vs. Graph



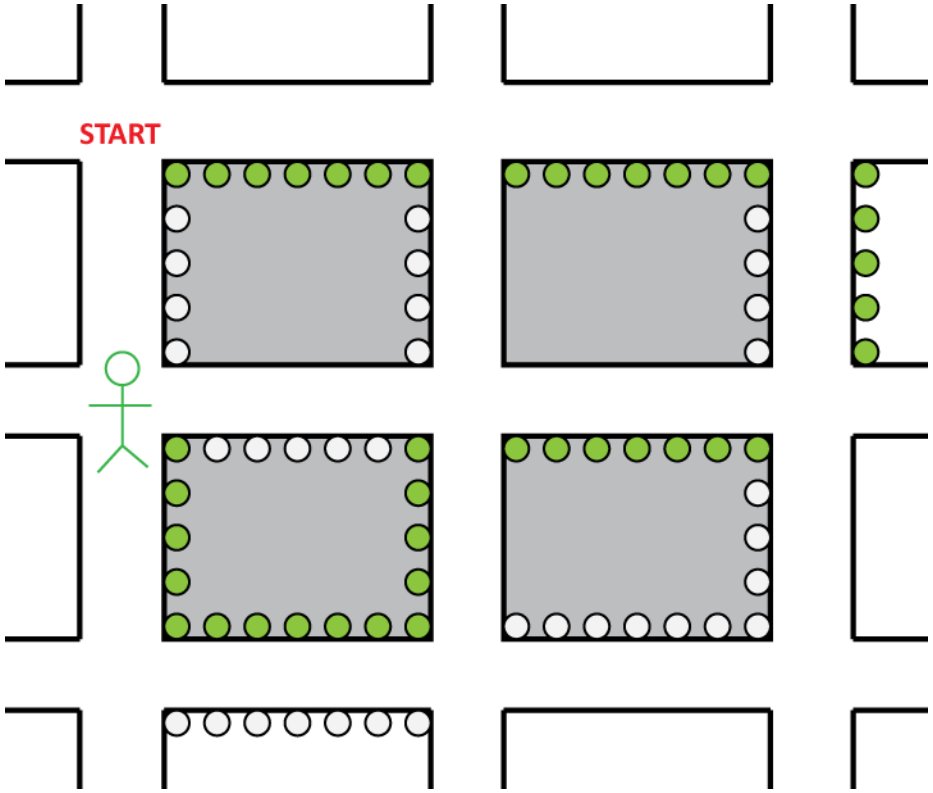


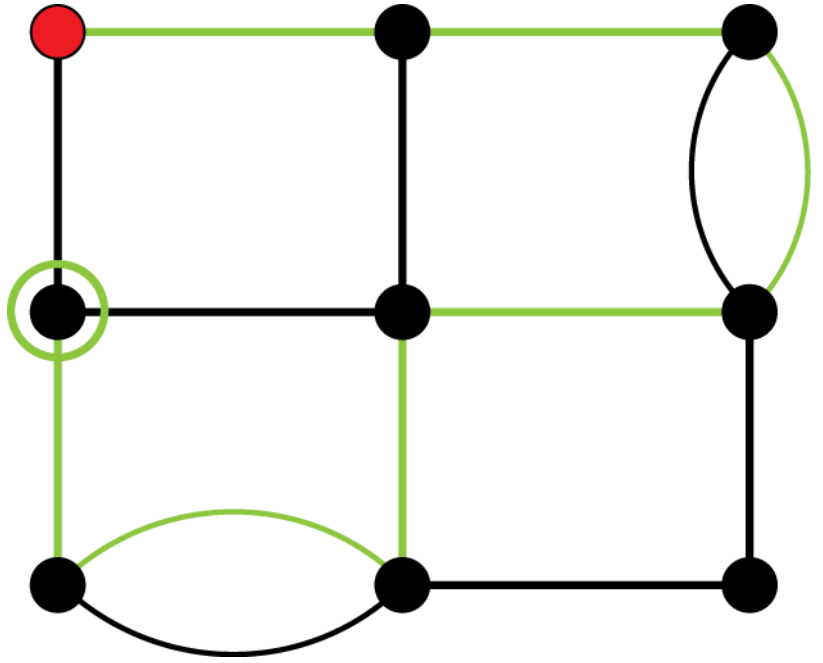
Two Paths: Map vs. Graph



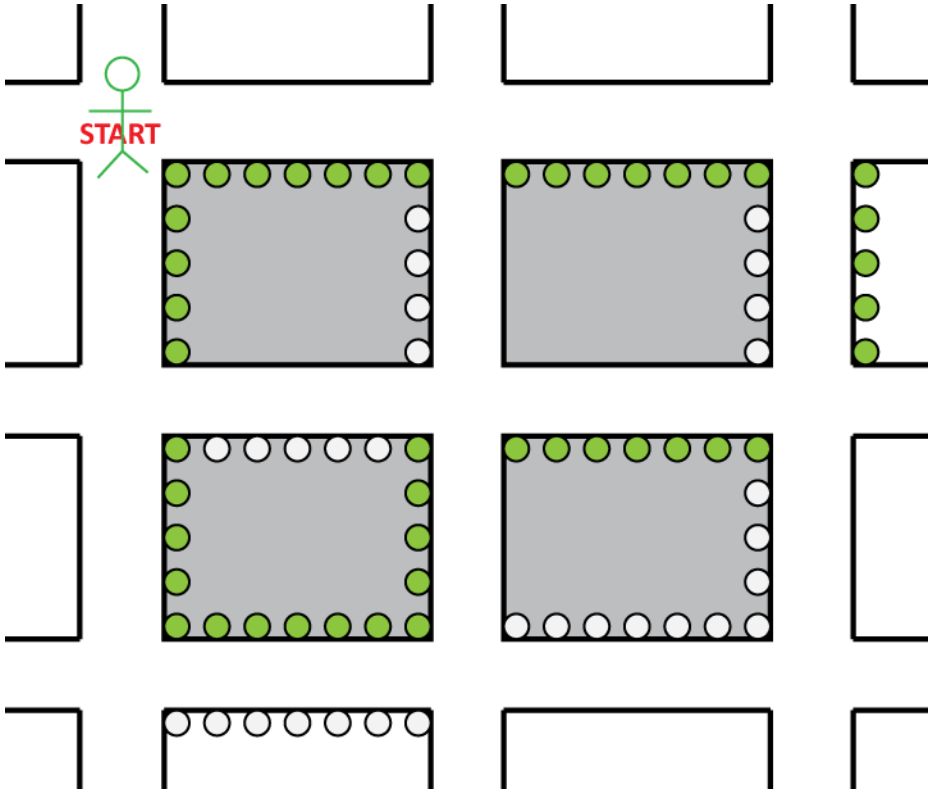


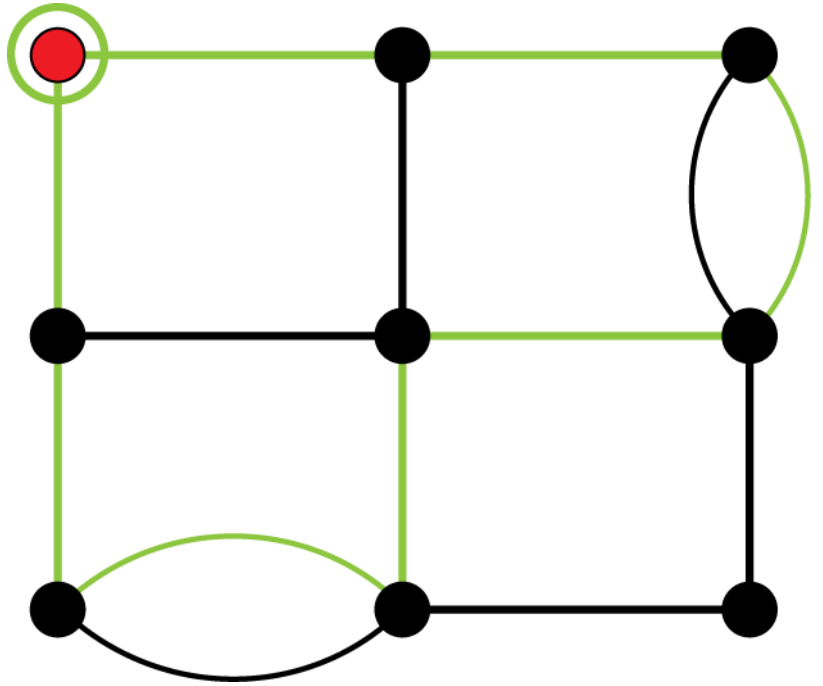
Two Paths: Map vs. Graph





Two Paths: Map vs. Graph





From Now On: Graphs Only

- From now on, we will only use graphs to study these kinds of problems
- We will eventually learn how to tell which graphs have solutions to the parking meter problem and which do not
- For now, we'll establish some vocabulary and practice creating graphs

Definitions

- A **path** is a sequence of vertices that are connected by edges. A path can repeat the same edge or vertex multiple times, and doesn't have to end at the same place it begins.

- A **circuit** is a path that ends at the same place it begins.

Definitions

- An **Euler circuit** is a circuit that includes every edge of the graph exactly once.

- This Euler circuit (pronounced “oiler”) is exactly what we are looking for in our parking meter problem

- It turns out there are many other kinds of problems for which we want Euler circuits

Applications of Euler circuits

- checking parking meters
- street cleaning
- snow plowing
- mail delivery
- garbage collection
- bridge inspection
- etc.

Applications of Euler Circuits

- Here is a map of a portion of the DC Metro system
- This map already looks like a graph

Applications of Euler Circuits

- Each station is a vertex
- The connections between stations are edges

Applications of Euler Circuits

- A maintenance worker might need to inspect every tunnel in the system and return to his/her starting point and minimize retracing

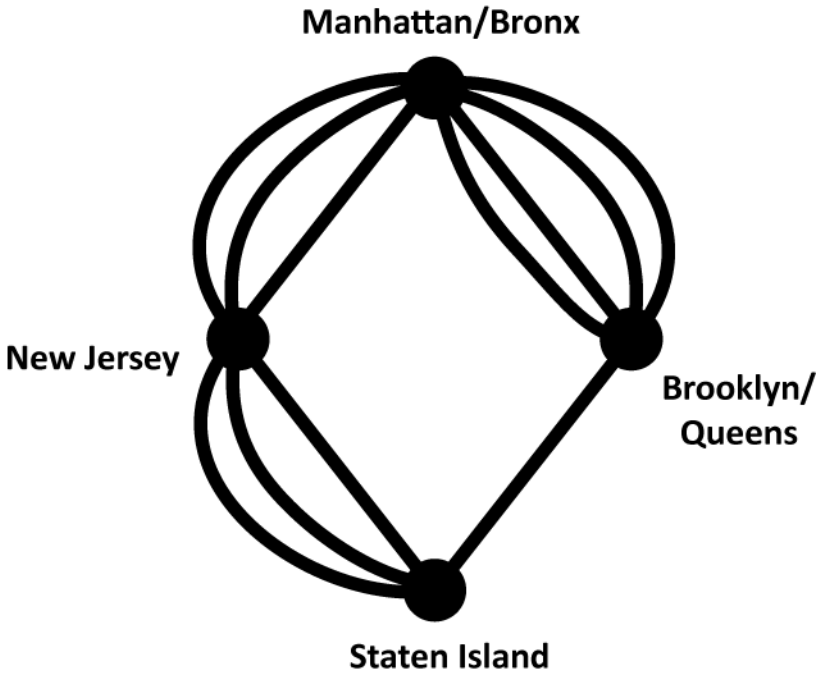
Applications of Euler Circuits

- Here is a map of some of the bridges in New York City

Applications of Euler Circuits

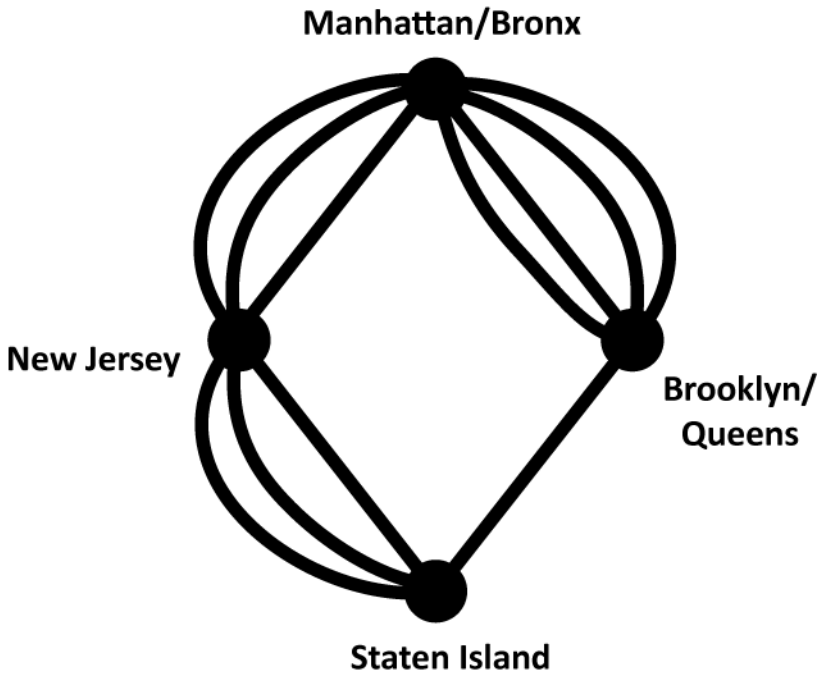
- We can represent this system with a graph
- Each bridge is represented by an edge
- Each location is represented by a vertex

Applications of Euler Circuits



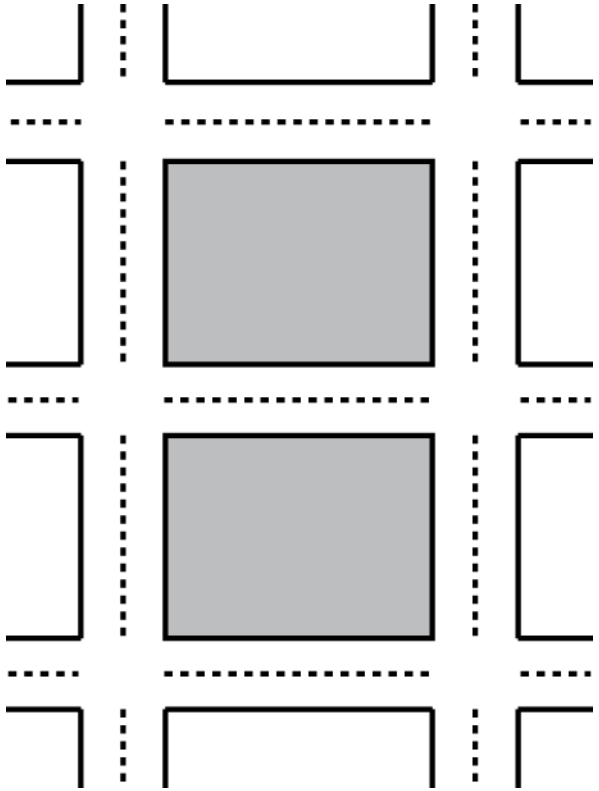
The graph on the left represents the map on the right.

Applications of Euler Circuits



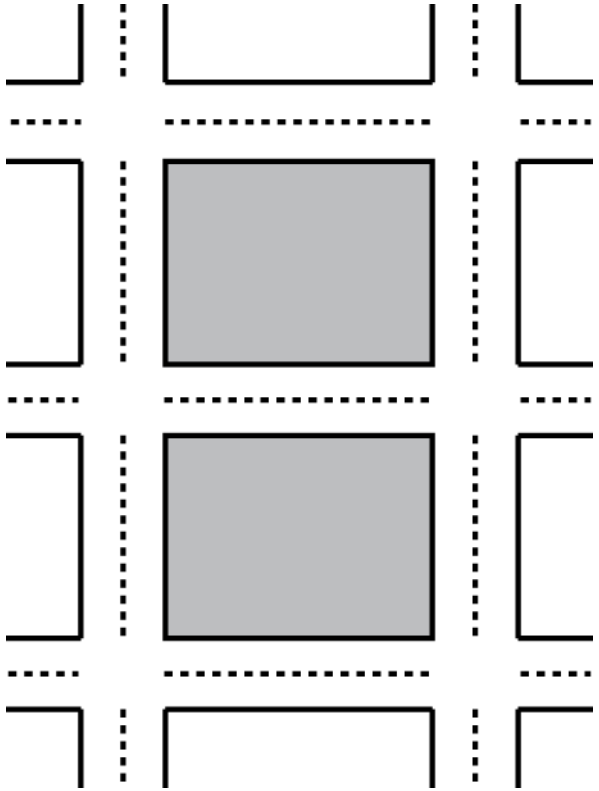
- Using this graph, we can more easily solve problems that involve traveling over these bridges
- For example, a bridge inspector might want to find an Euler circuit for this graph

Applications of Euler Circuits



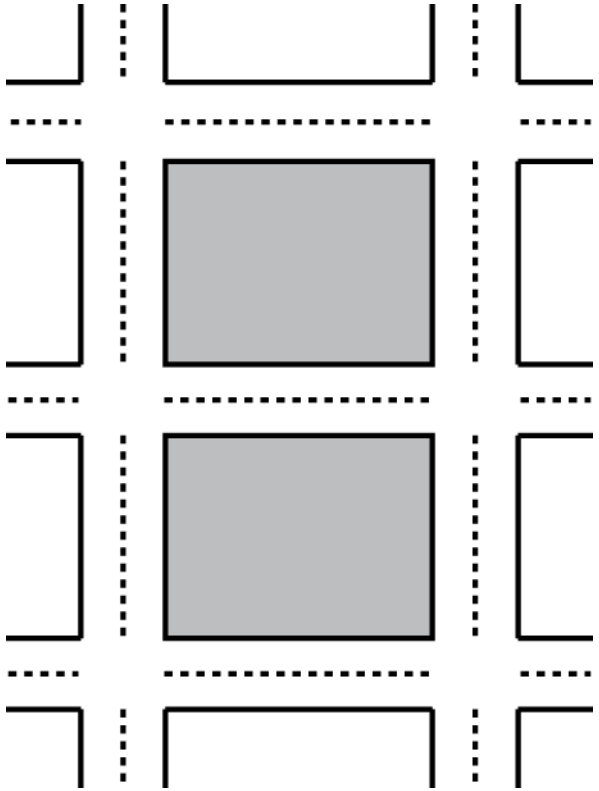
- Now consider this map of a small neighborhood
- Suppose it is your job to drive a snow plow and clean the streets of this neighborhood after a snowstorm

Applications of Euler Circuits



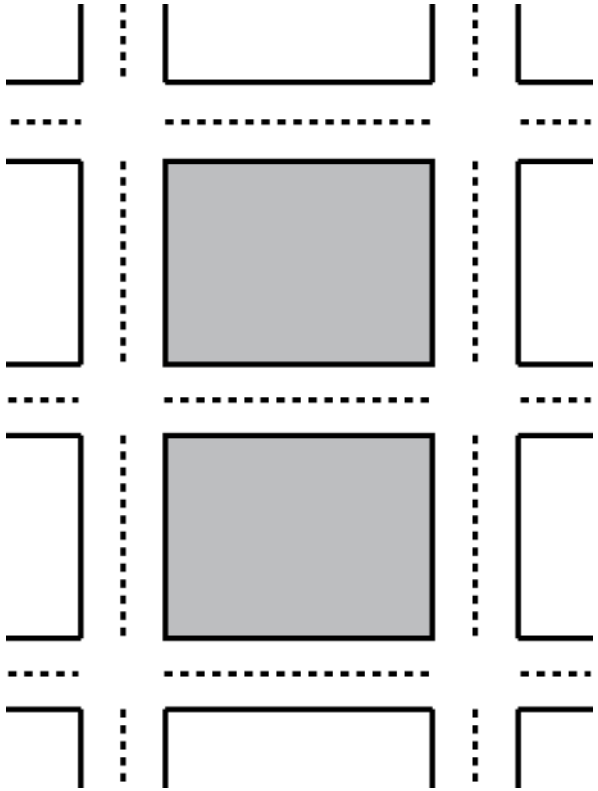
- You want to make sure to plow every lane (notice each street has 2 lanes, so you'll have to drive down each road twice)
- You don't want to retrace your steps unless you have to
- You want to return to your starting point

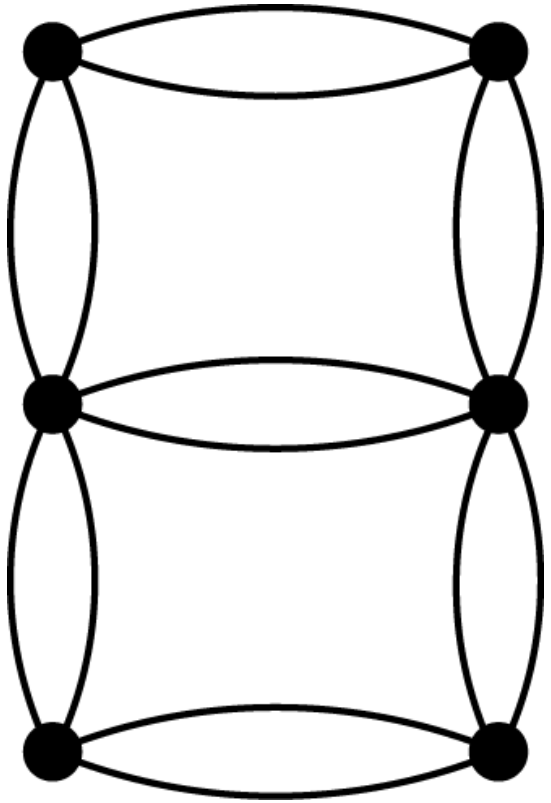
Applications of Euler Circuits



- Let’s draw the graph for this problem
- Each intersection will be represented by a vertex
- Each lane will be represented by an edge

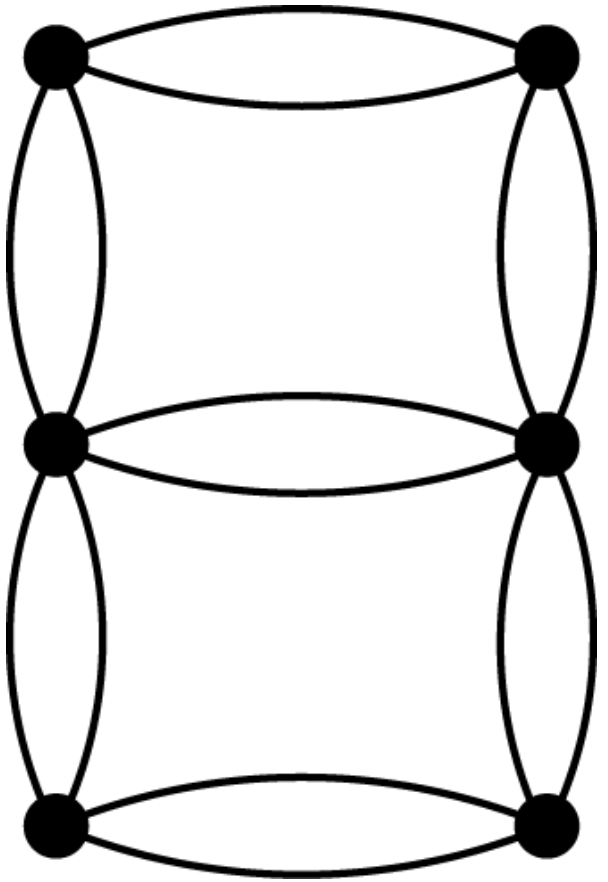
Applications of Euler Circuits





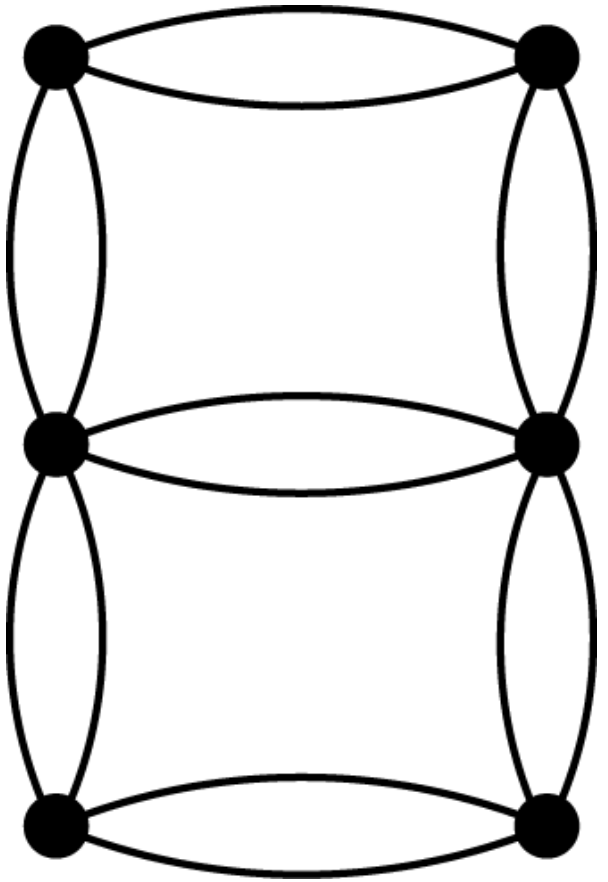
Applications of Euler Circuits

- Can you solve the problem?
- Pick a starting point and try to find an Euler circuit for this graph

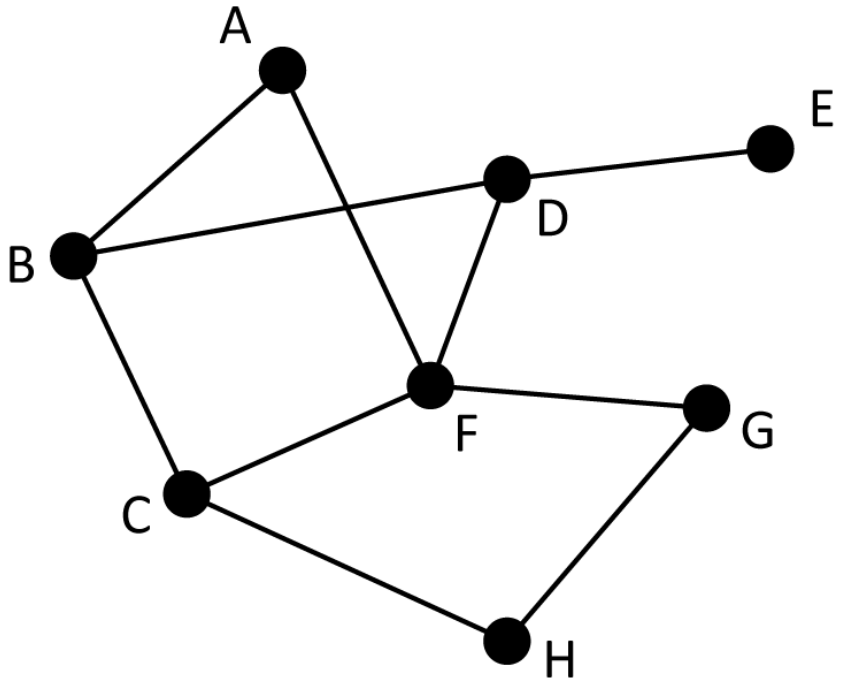


Applications of Euler Circuits

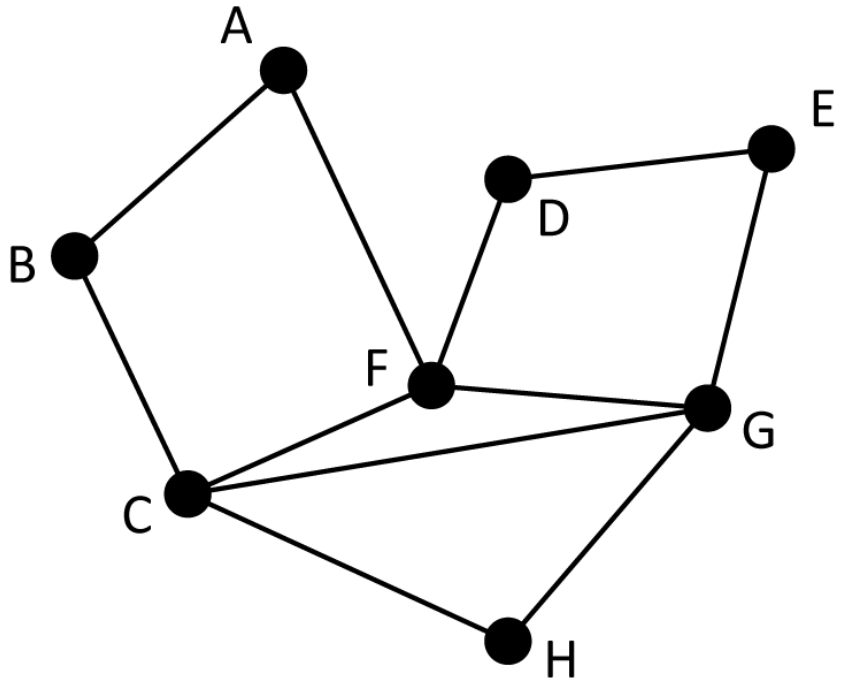
- Can you solve the problem?
- Pick a starting point and try to find an Euler circuit for this graph
- It turns out we *can* find an Euler circuit for this graph (there are lots!)



When does a graph have an Euler circuit?



This graph **does not** have an Euler circuit.

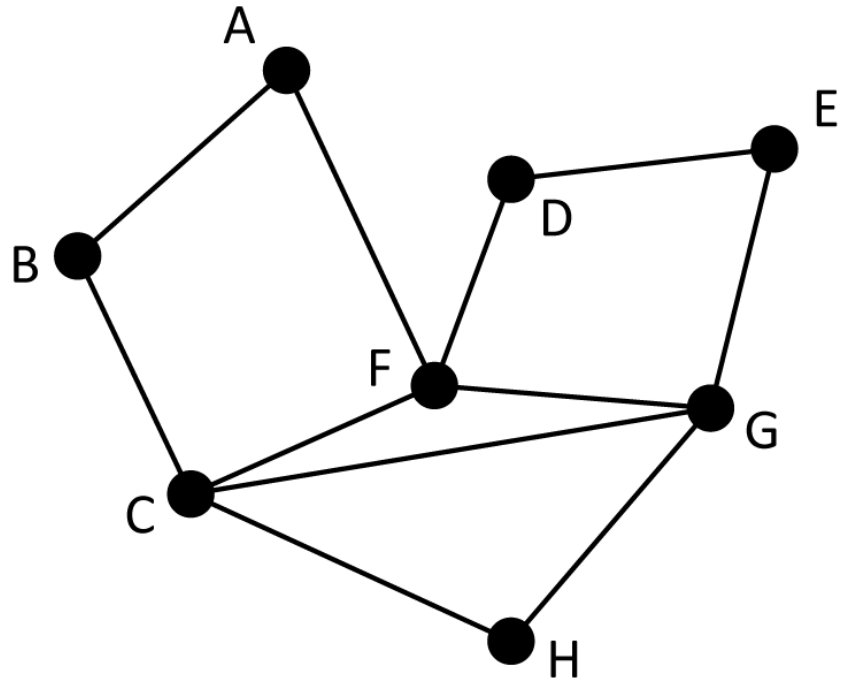


This graph **does** have an Euler circuit.

When does a graph have an Euler circuit?

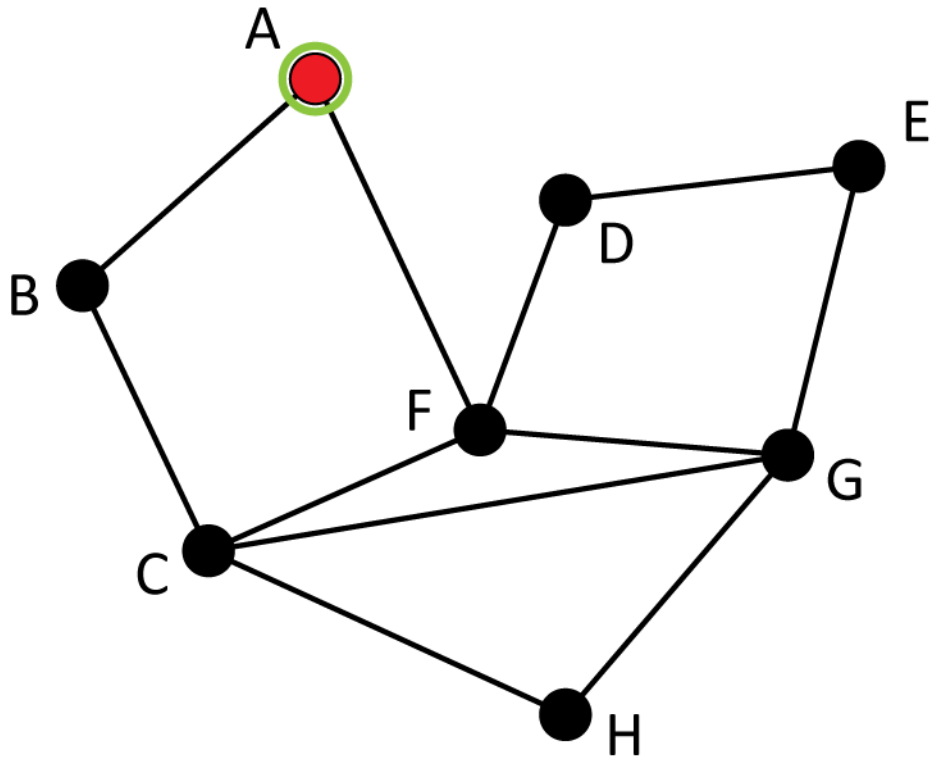
- How could I convince you that this graph has an Euler circuit?

- I can show it to you!

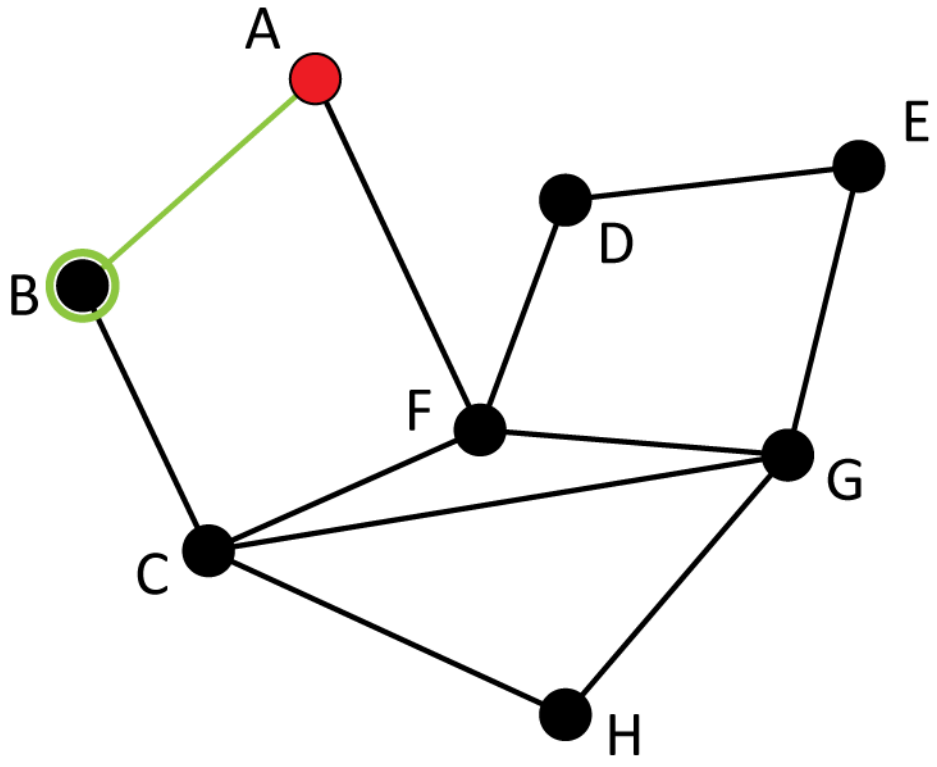


This graph **does** have an Euler circuit.

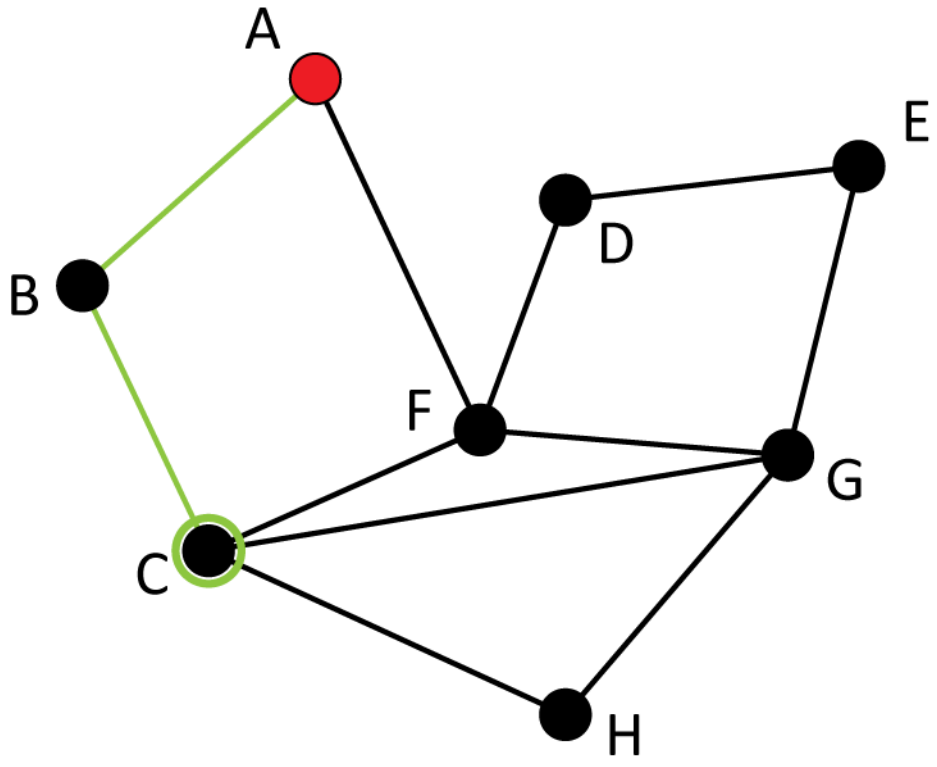
Finding the Euler Circuit



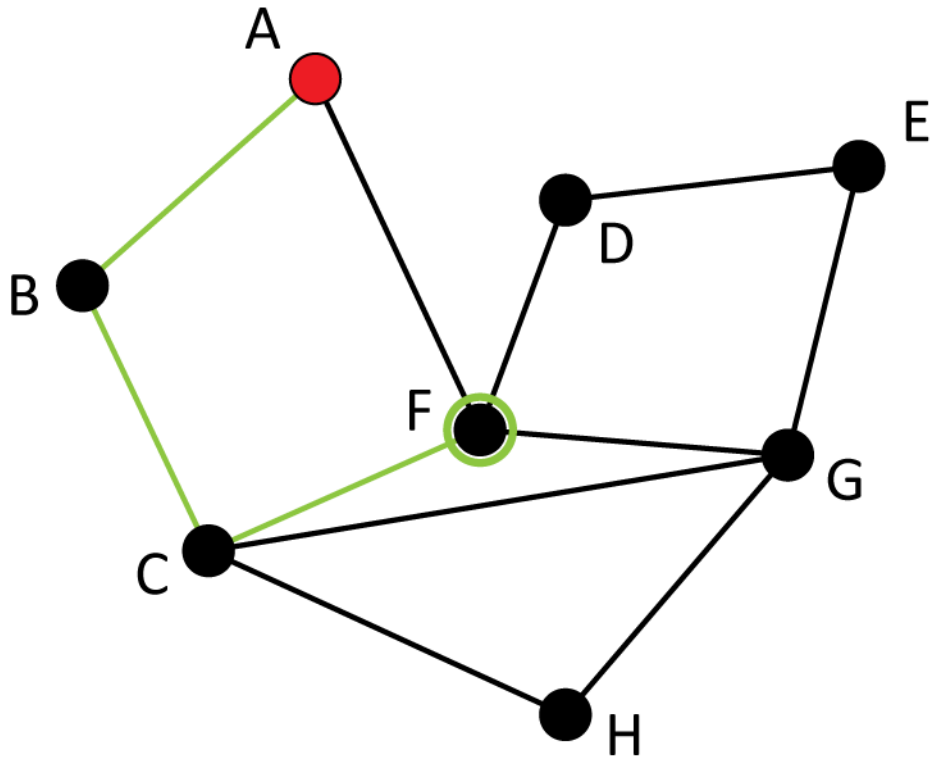
Finding the Euler Circuit



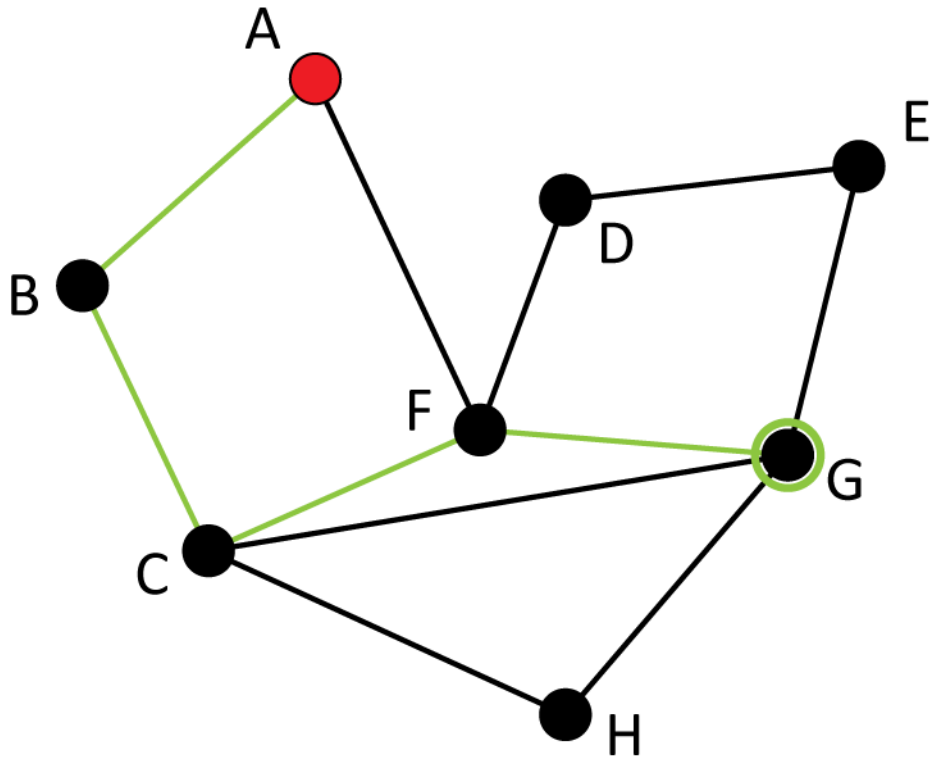
Finding the Euler Circuit



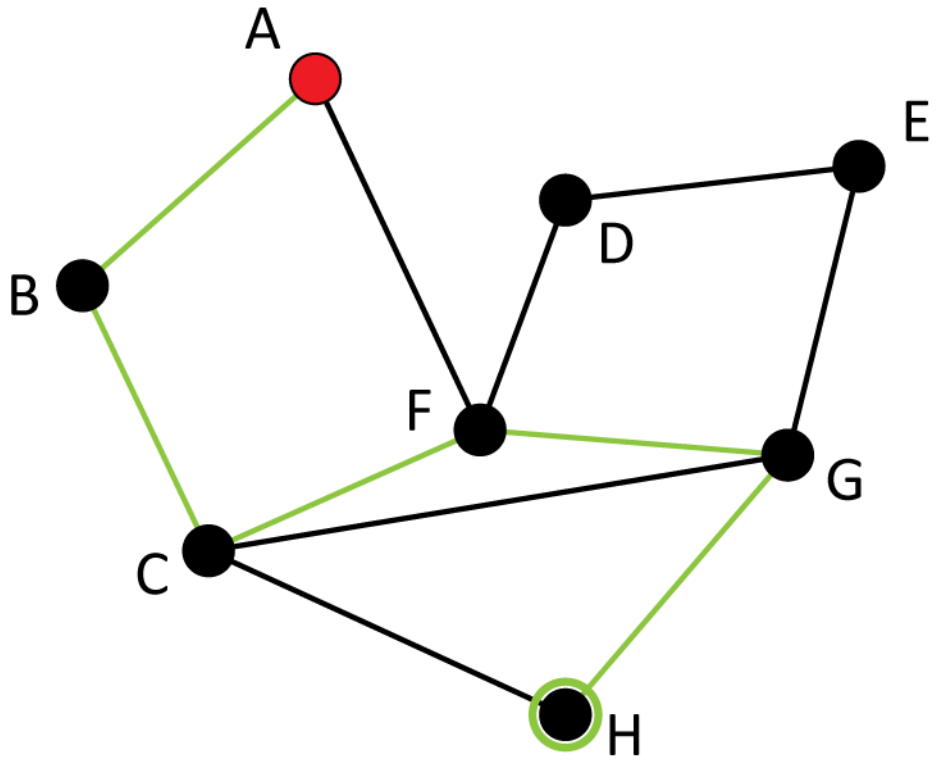
Finding the Euler Circuit



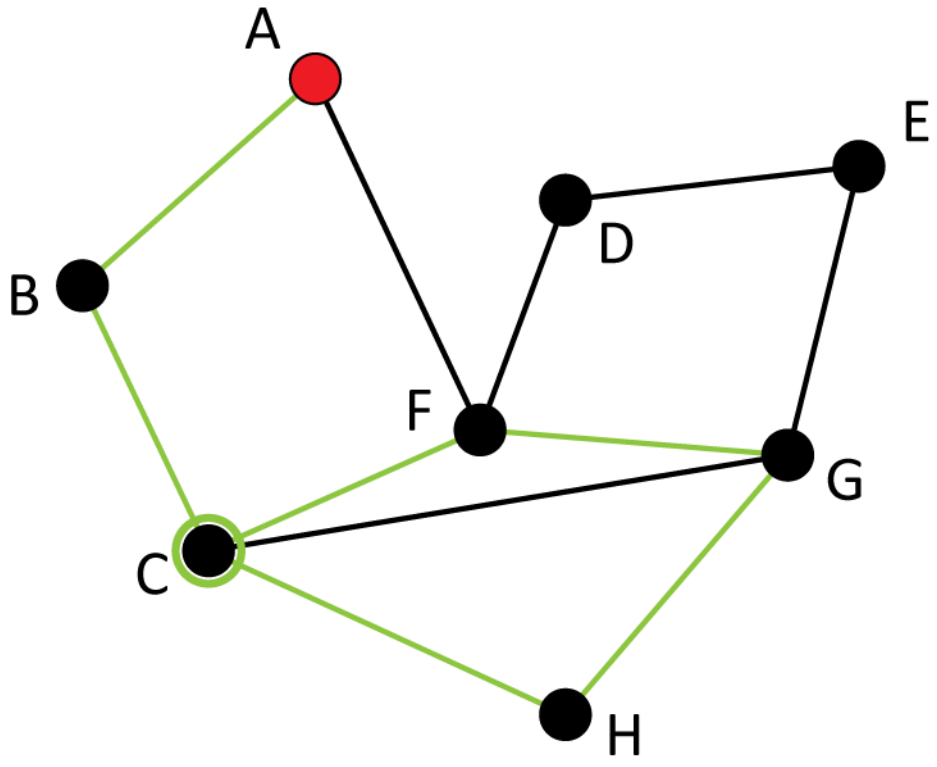
Finding the Euler Circuit



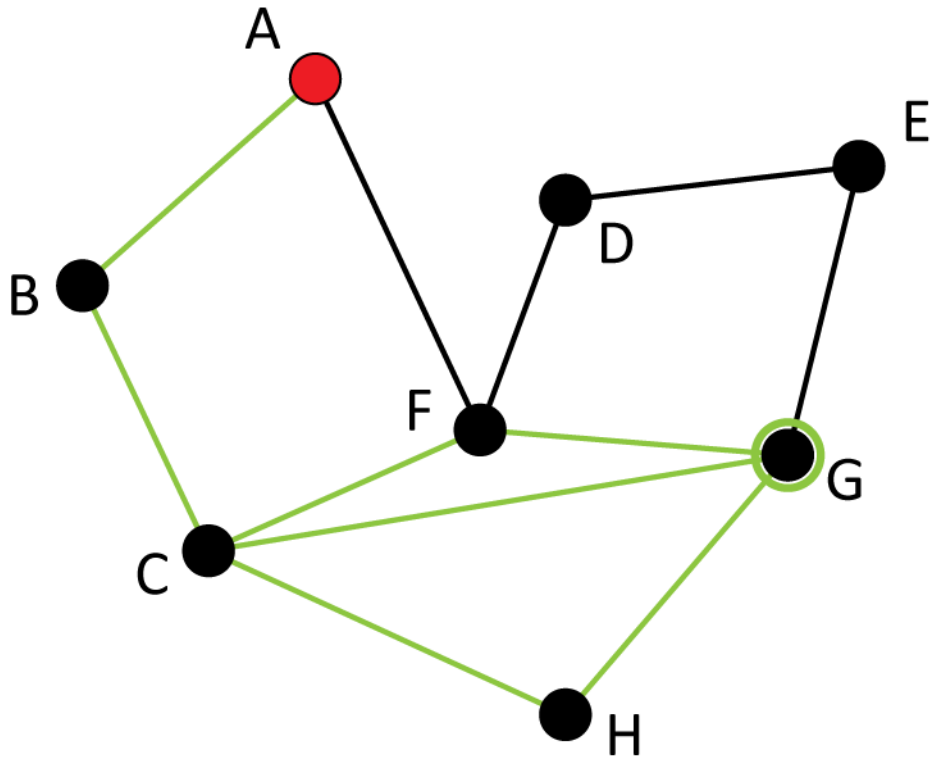
Finding the Euler Circuit



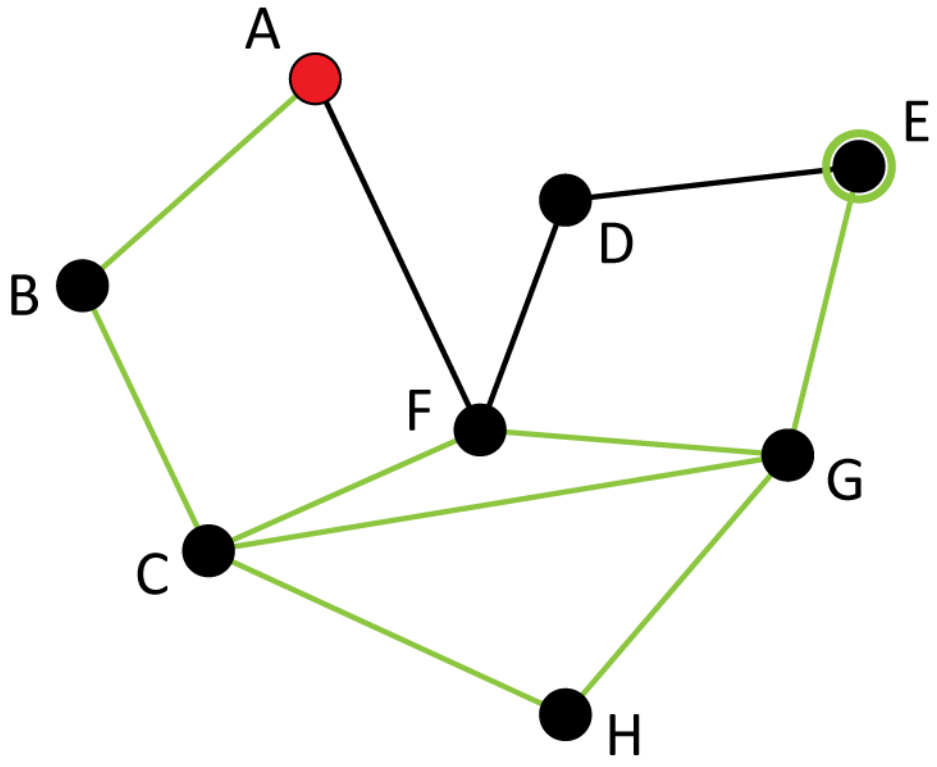
Finding the Euler Circuit



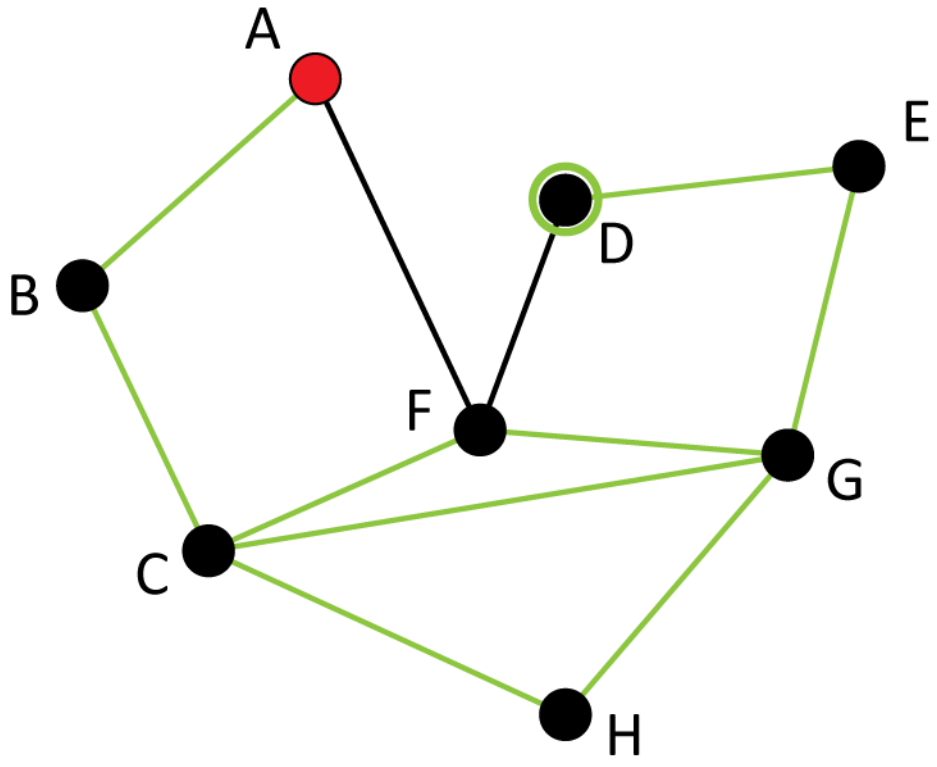
Finding the Euler Circuit



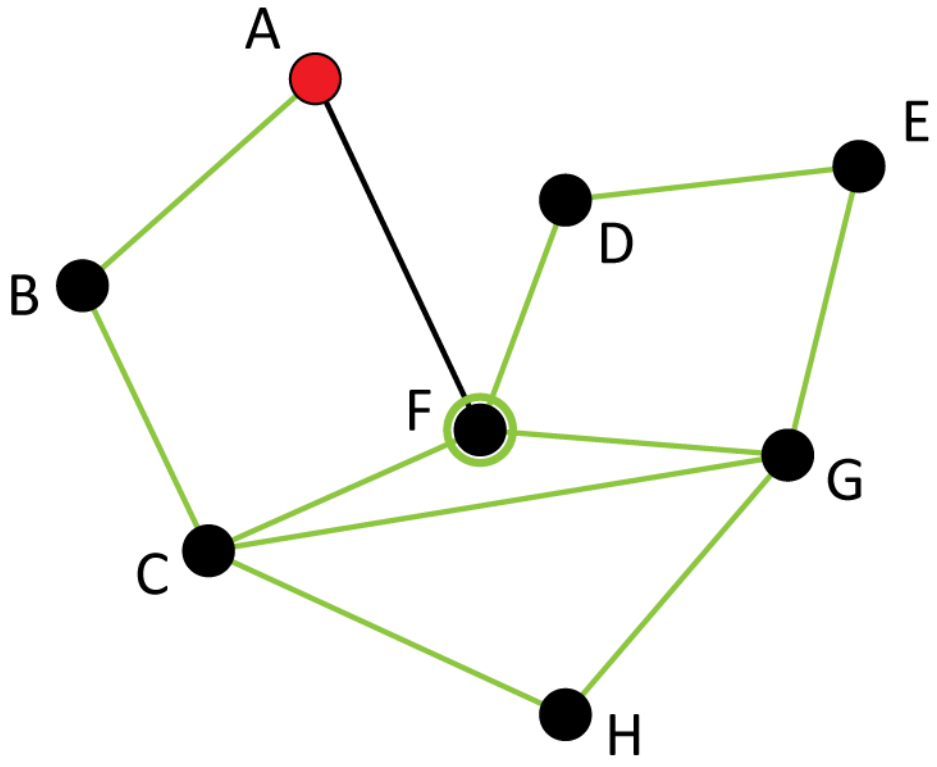
Finding the Euler Circuit



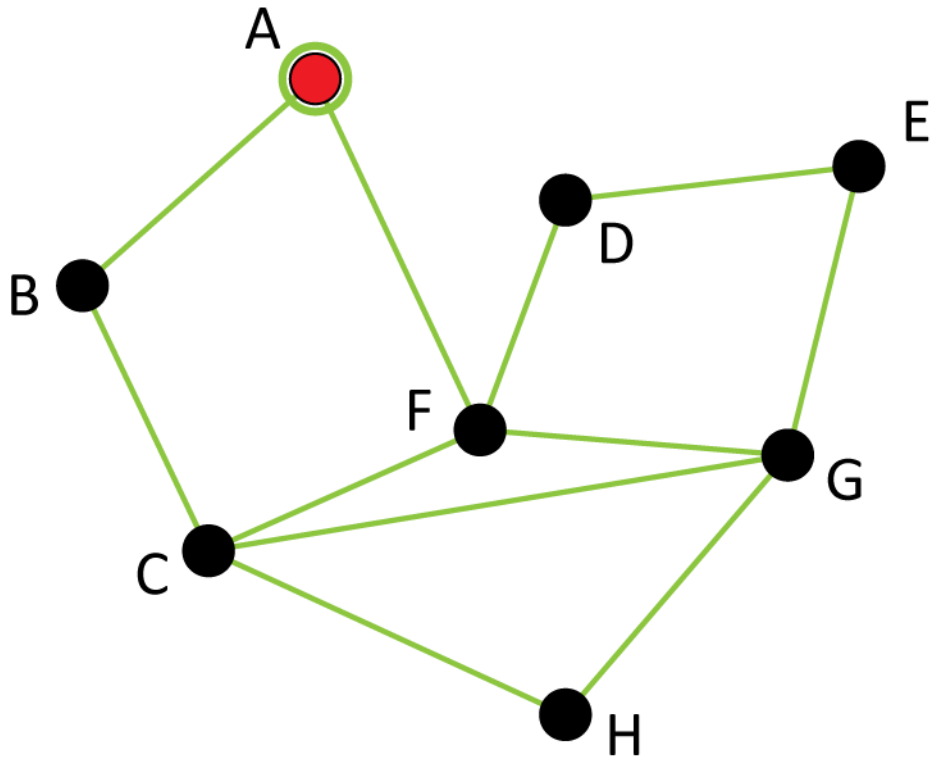
Finding the Euler Circuit



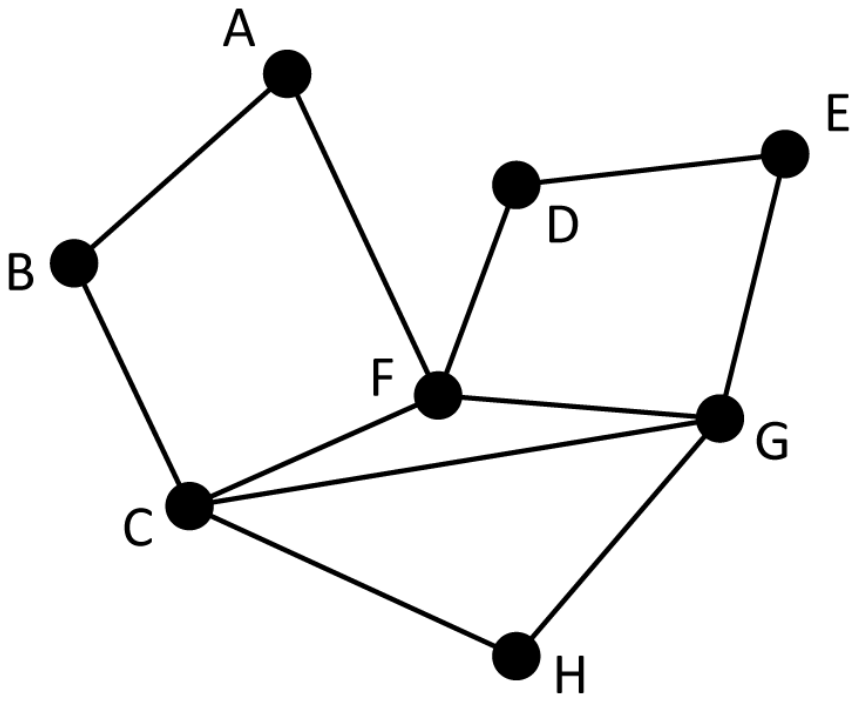
Finding the Euler Circuit



Finding the Euler Circuit

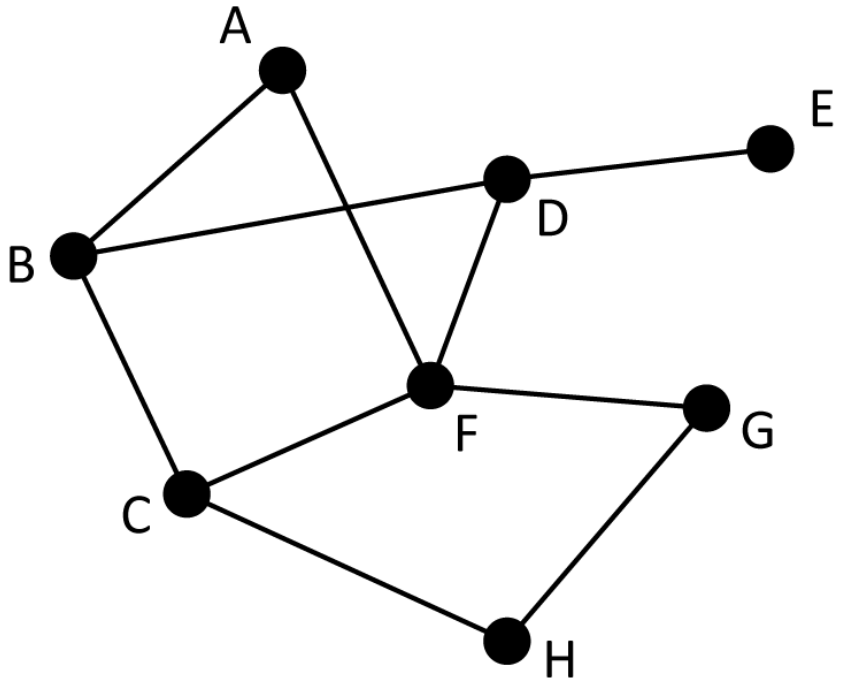


There are lots of Euler circuits…

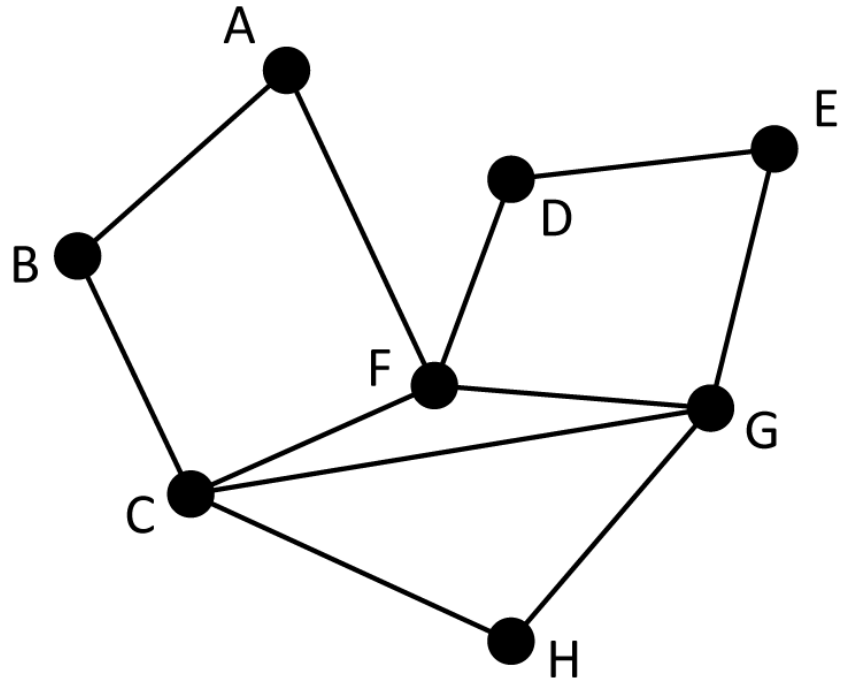


- That was just one of the many Euler circuits this graph has
- Take a moment and try to find one on your own
- You don’t have to start at A!

Why does a graph have an Euler circuit (or not)?



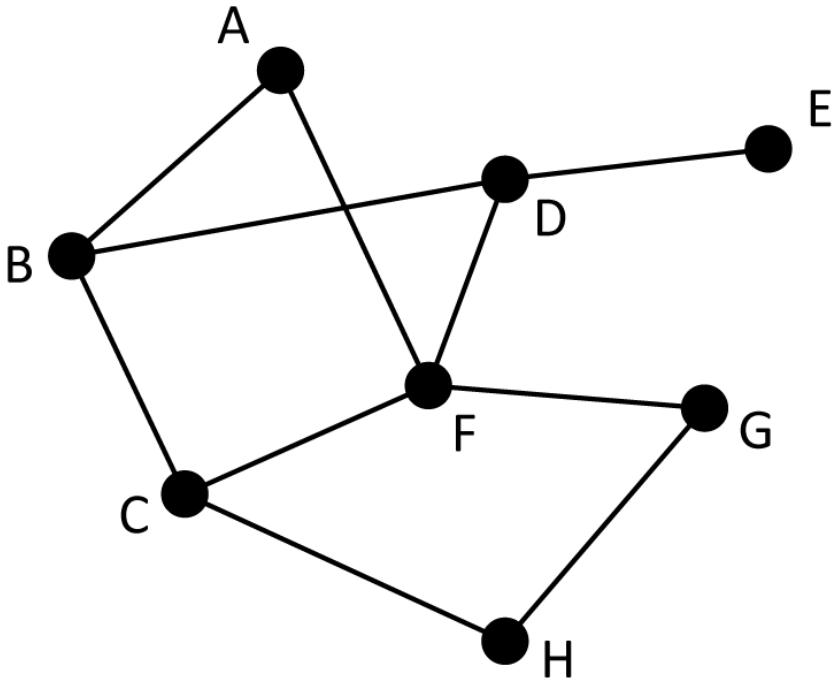
Why does this graph not have an Euler circuit?

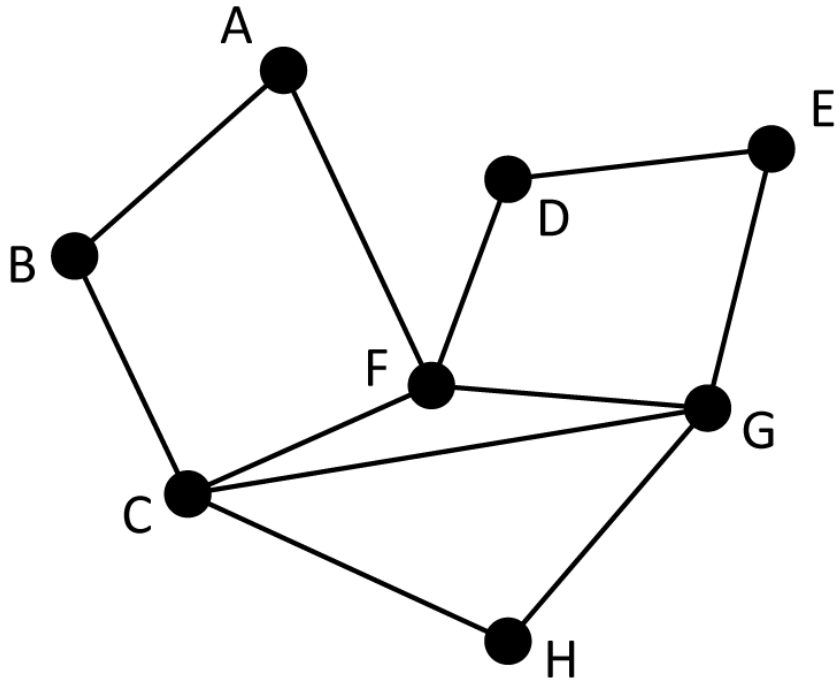


Why does this graph have an Euler circuit?

Degree

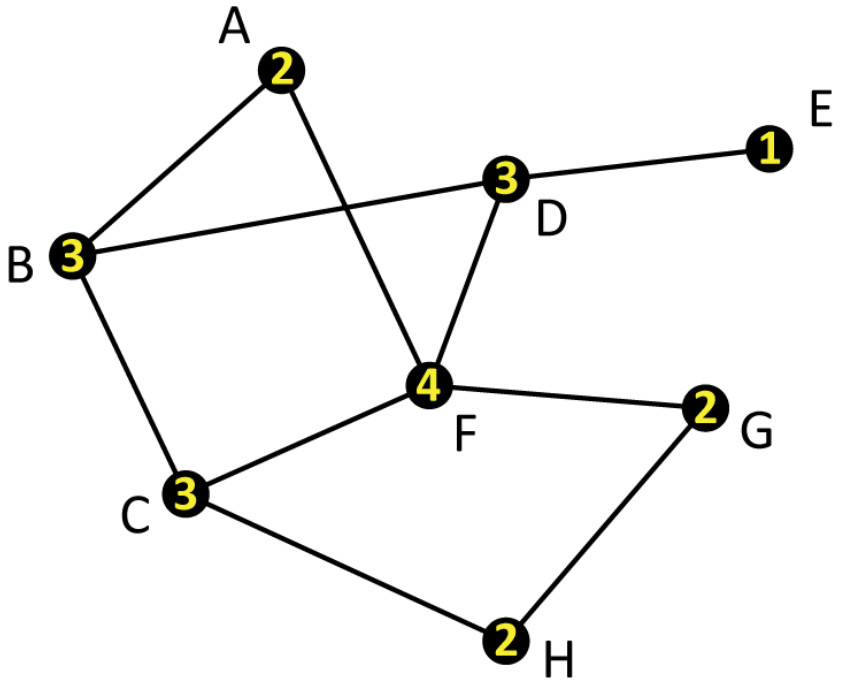
- The **degree** of a vertex is the number of edges that meet at that vertex

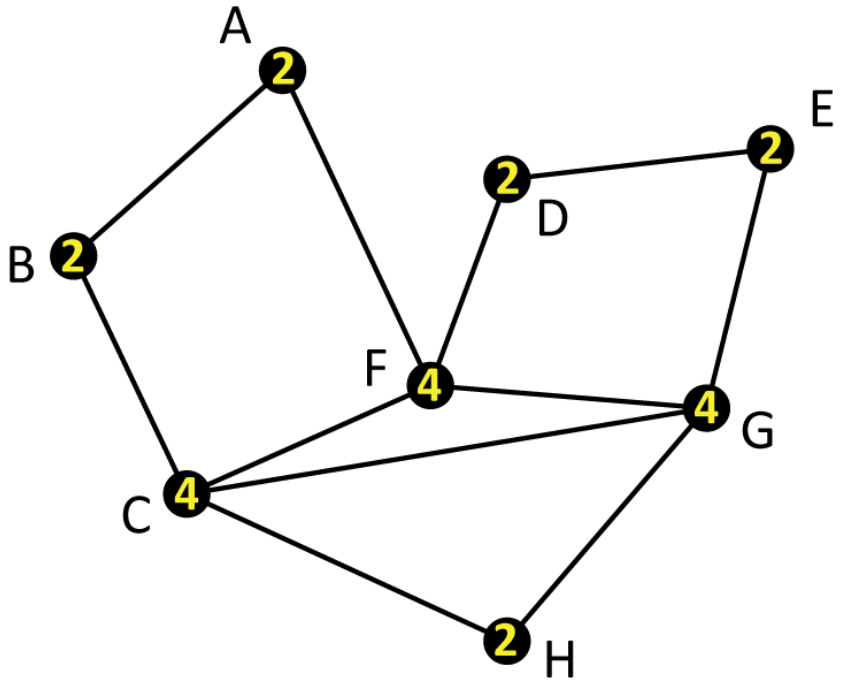




Degree

- The **degree** of a vertex is the number of edges that meet at that vertex

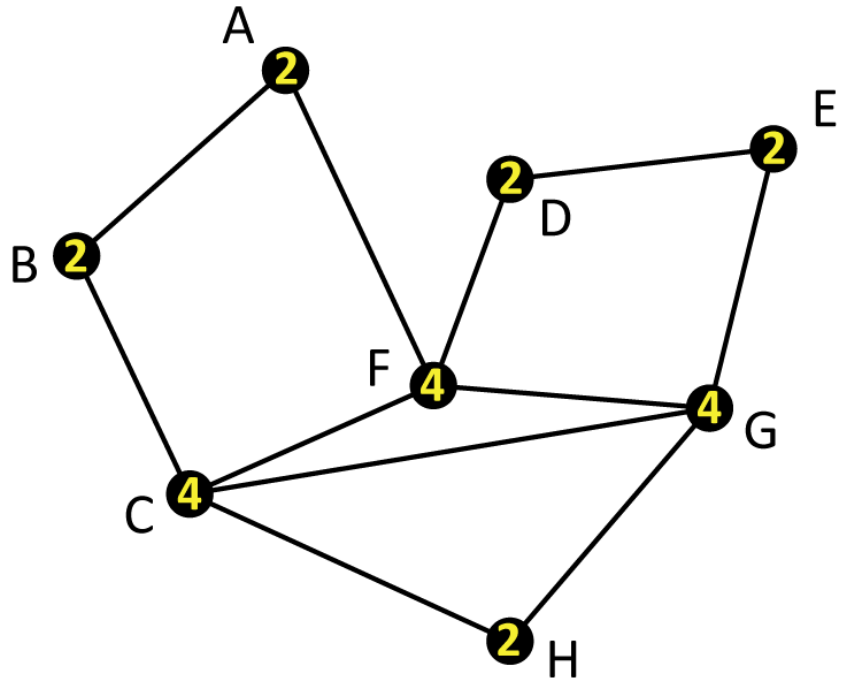




Degree

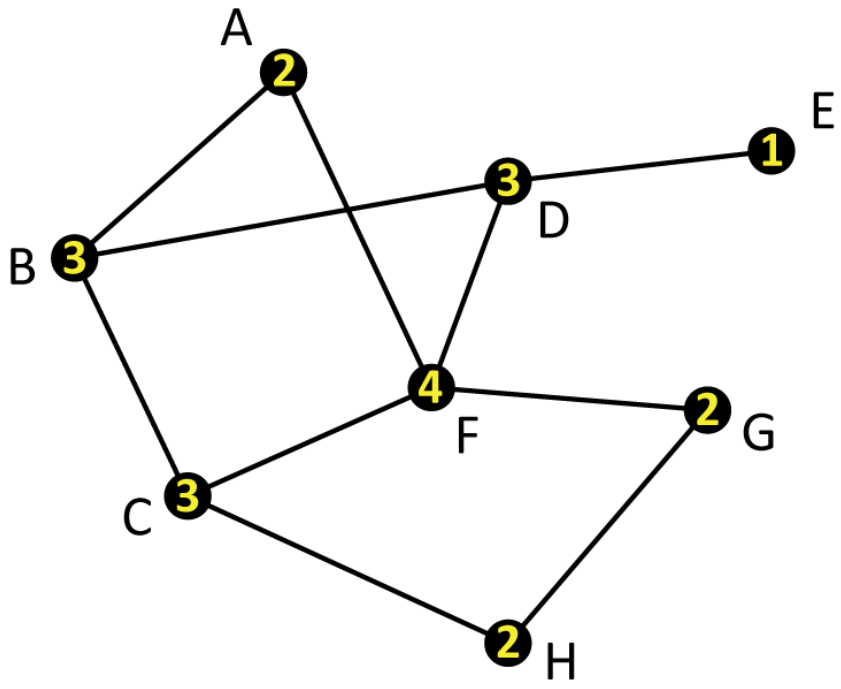
- The **degree** of a vertex is the number of edges that meet at that vertex

- For example, in this graph, the degree of C is **4** because there are four edges (to B, to F, to G, and to H) that meet there



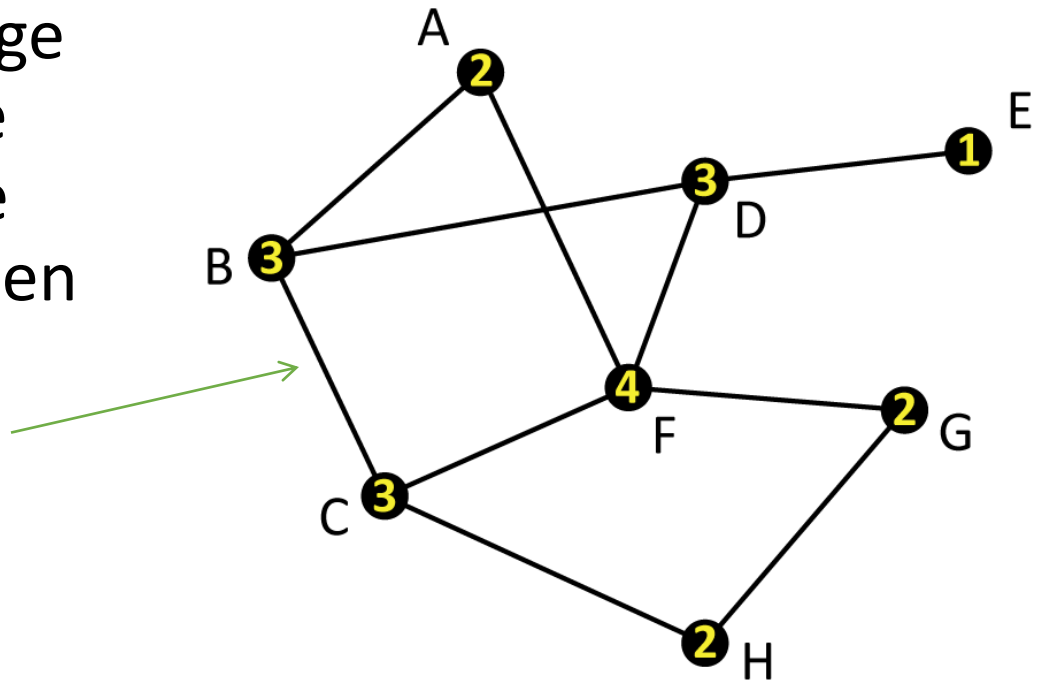
Fact About Degrees

- Add up all the degrees in this graph
- 2+3+3+3+1+4+2+2 = 20
- We have counted each edge twice



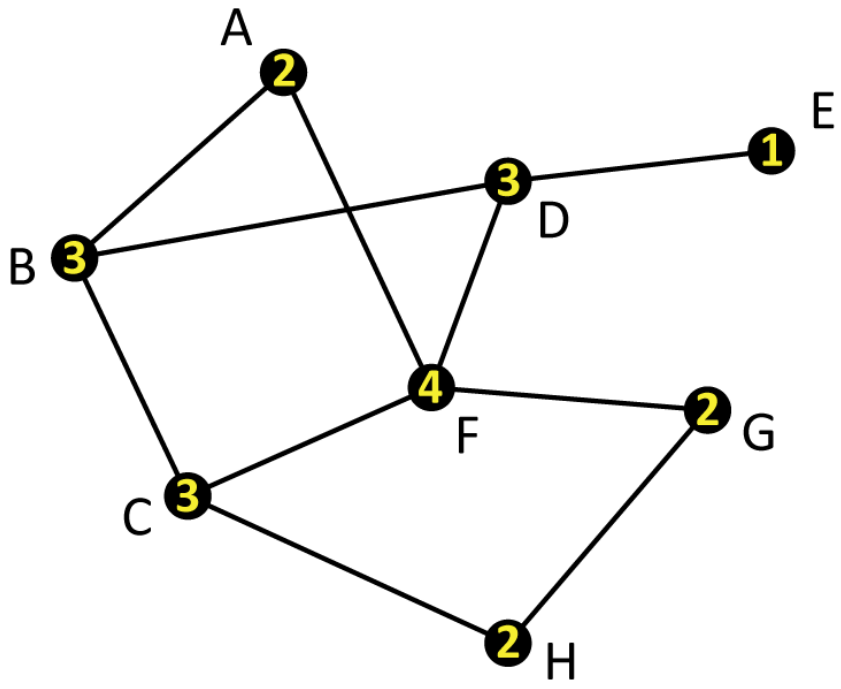
Fact About Degrees

- We counted this edge when we added the degree of B, and we counted it again when we counted the degree of C



Fact About Degrees

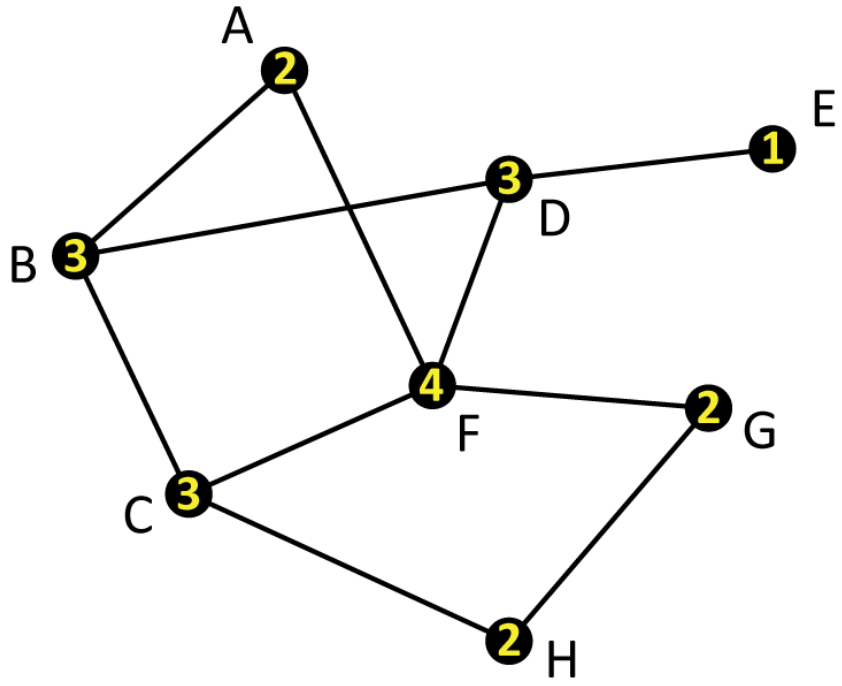
The sum of all the degrees in a graph equals two times the total number of edges.



What does degree have to do with Euler circuits?

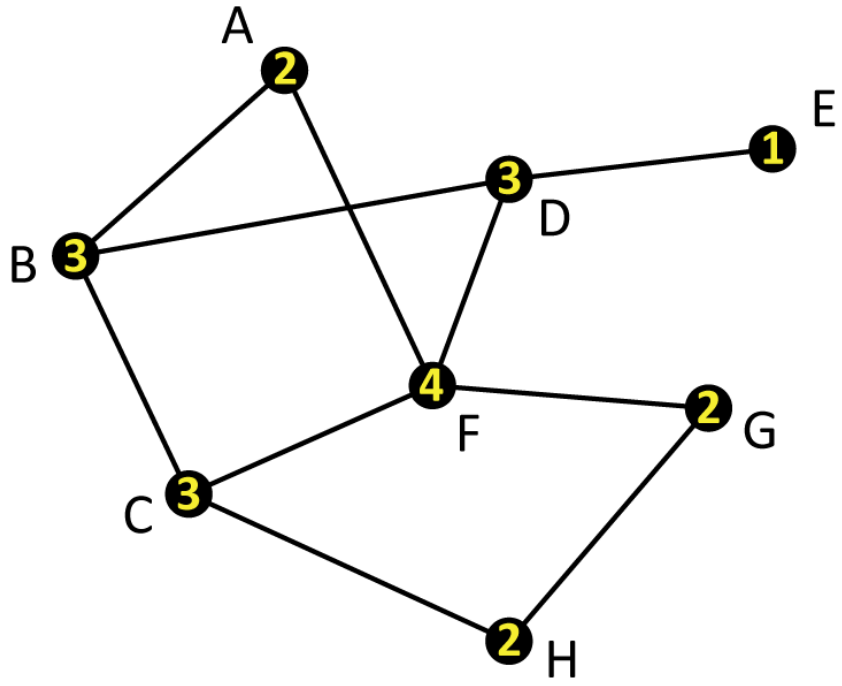
- You might be able to tell right away that this graph can't possibly have an Euler circuit

- Why not?



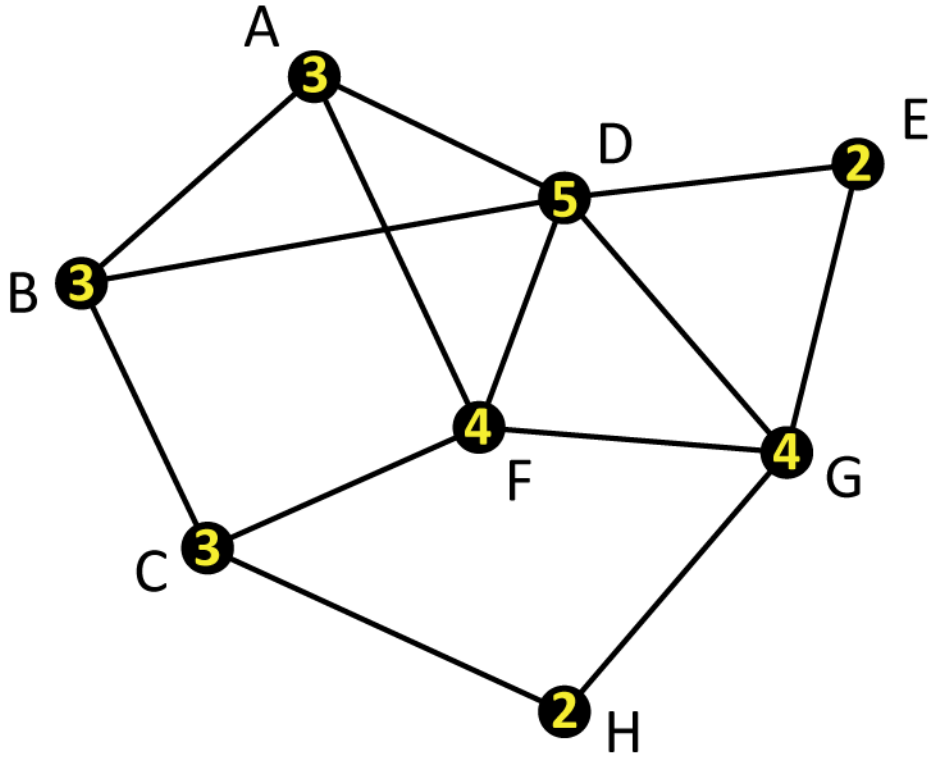
What does degree have to do with Euler circuits?

- If a graph has a vertex with degree 1, the graph cannot have an Euler circuit
  - If we start at E, we will never be able to return to E without retracing
  - If we don't start at E, we will never be able to go there, since when we leave we will have to retrace



What does degree have to do with Euler circuits?

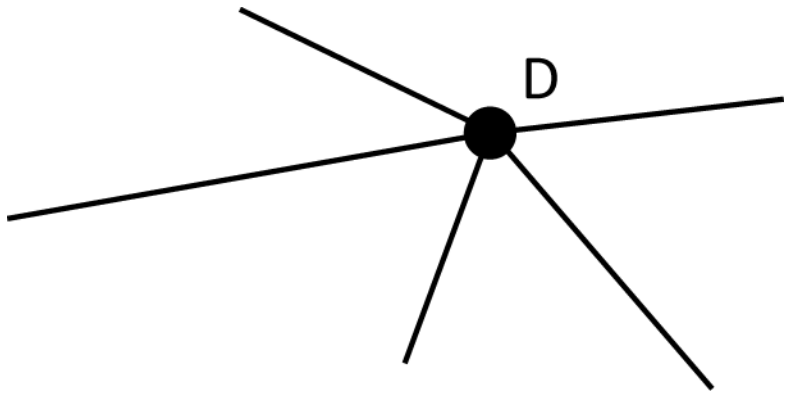
- The problem isn't just degree 1
- This graph also doesn't have an Euler circuit
- The problem is that some of the degrees are ***odd numbers***



What does degree have to do with Euler circuits?

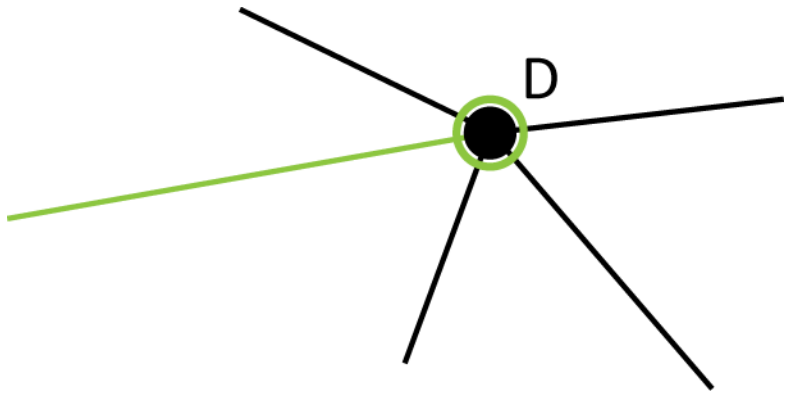
- Let’s focus on vertex D, which has degree 5

- Suppose we start elsewhere in the graph



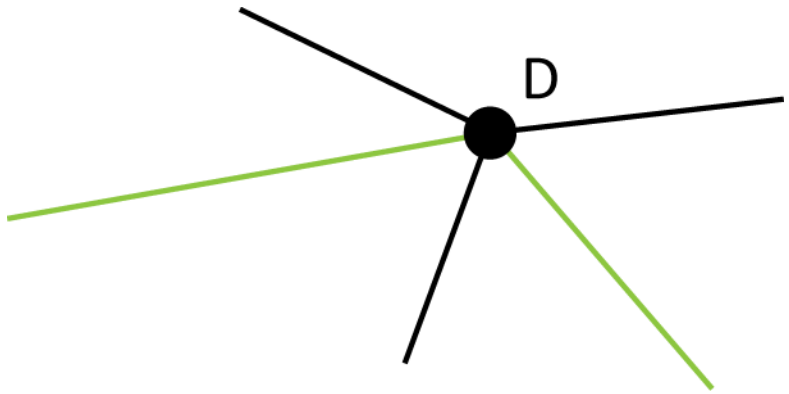
What does degree have to do with Euler circuits?

- Since we want to cover all edges, we'll have to visit D eventually



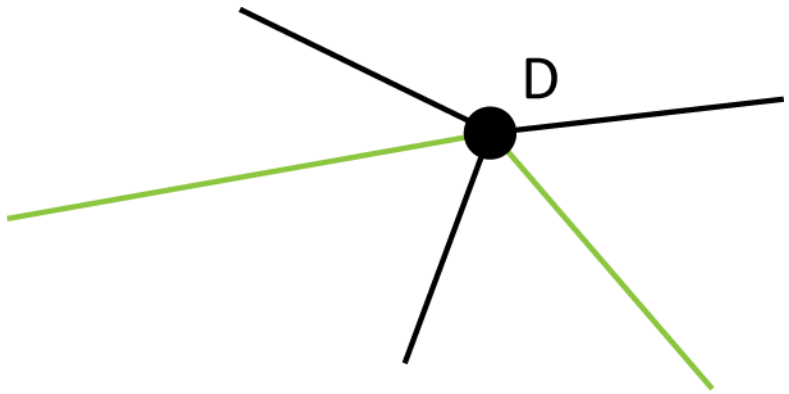
What does degree have to do with Euler circuits?

- We have several unused edges, so we need to follow one of them and leave D



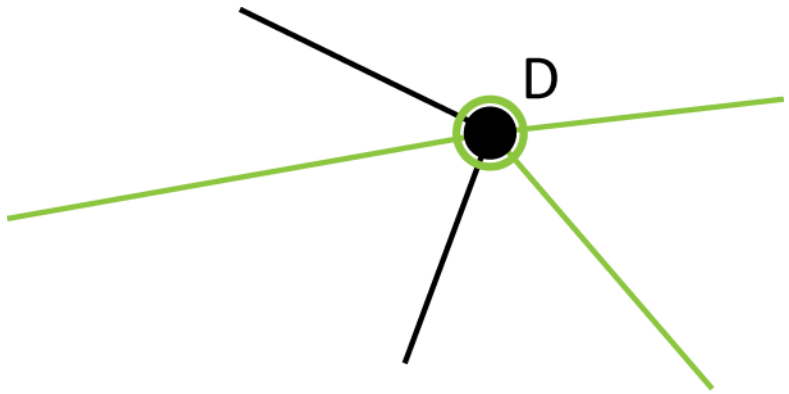
What does degree have to do with Euler circuits?

- In fact, every time we visit a vertex, we will “use up” **two** of the edges that meet at that vertex



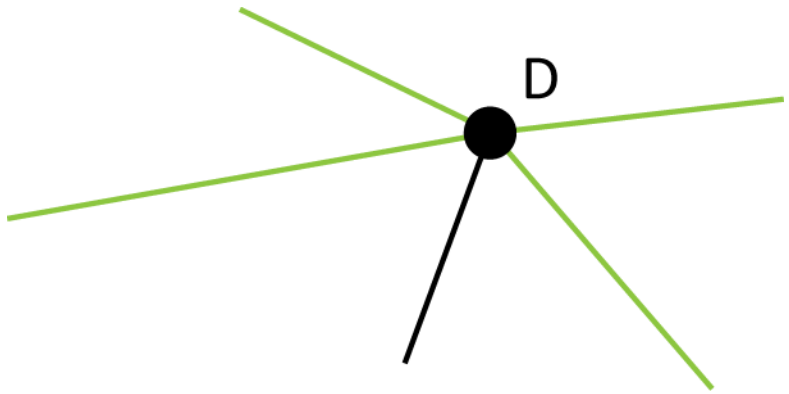
What does degree have to do with Euler circuits?

- We have unused edges, so we need to visit D again at some point…



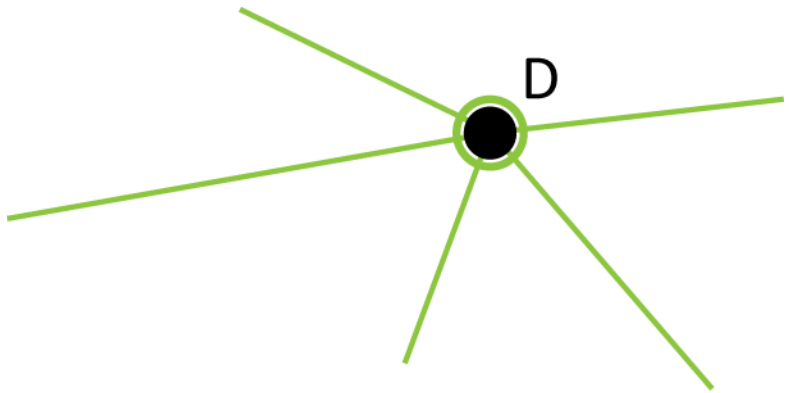
What does degree have to do with Euler circuits?

- …and then leave again…



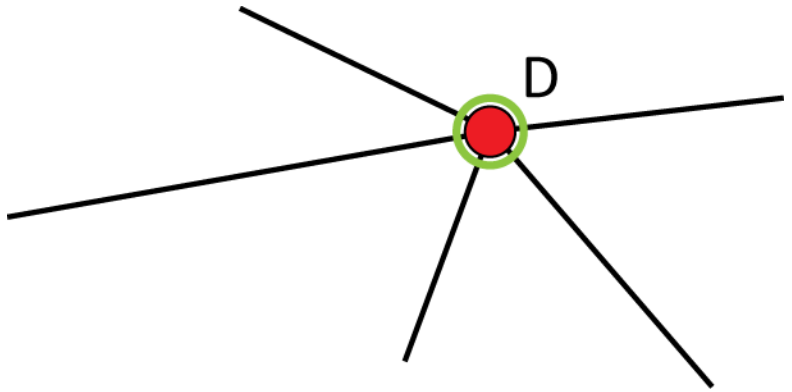
What does degree have to do with Euler circuits?

- …and then come back again.
- But now we're stuck, since we can't leave D without retracing, but D wasn't our starting point.



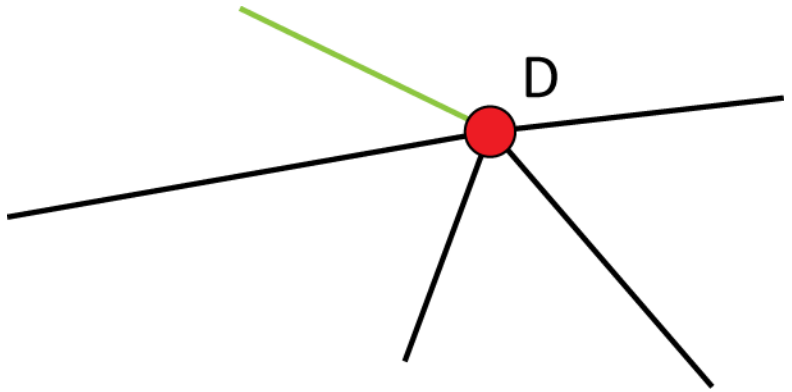
What does degree have to do with Euler circuits?

- What if we had started at D?



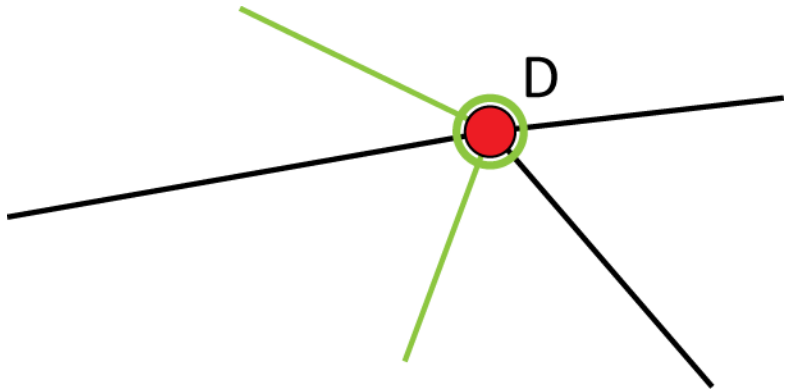
What does degree have to do with Euler circuits?

- First, we need to leave D…



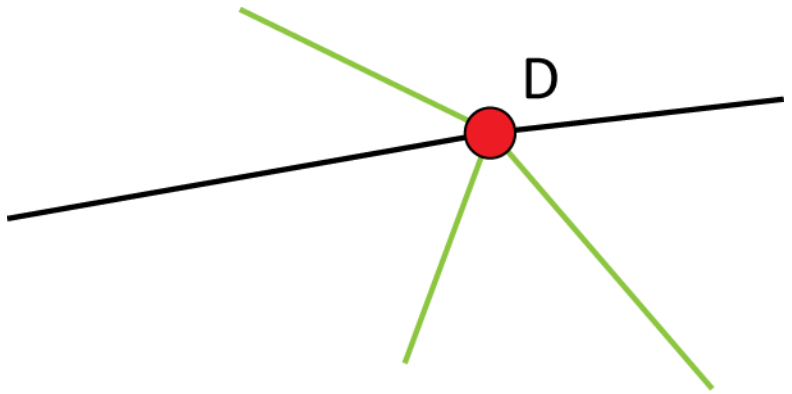
What does degree have to do with Euler circuits?

- … then sometime later, we have to come back to D…



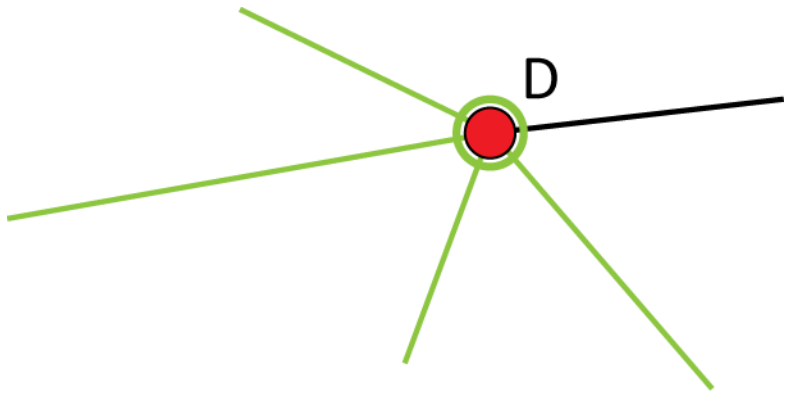
What does degree have to do with Euler circuits?

- … and then leave again …



What does degree have to do with Euler circuits?

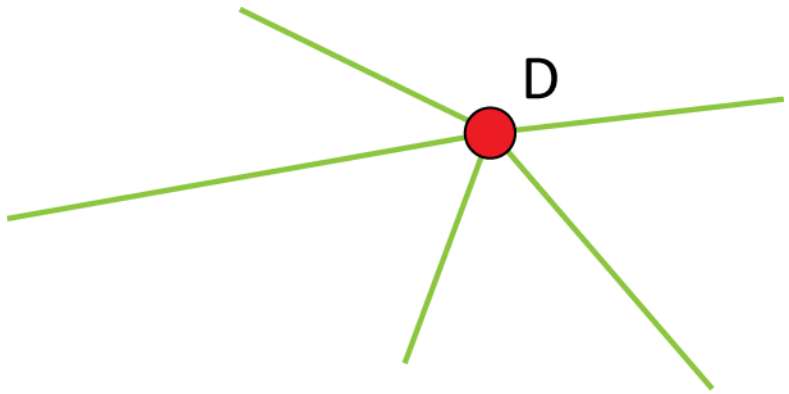
- … and then come back again …



What does degree have to do with Euler circuits?

- … and then leave again.

- But D was our starting point, and we have run out of edges to use to come back to D!



No Euler Circuits With Odd Degrees

- If a graph has *any* vertex with an odd degree, then the graph does not have an Euler circuit

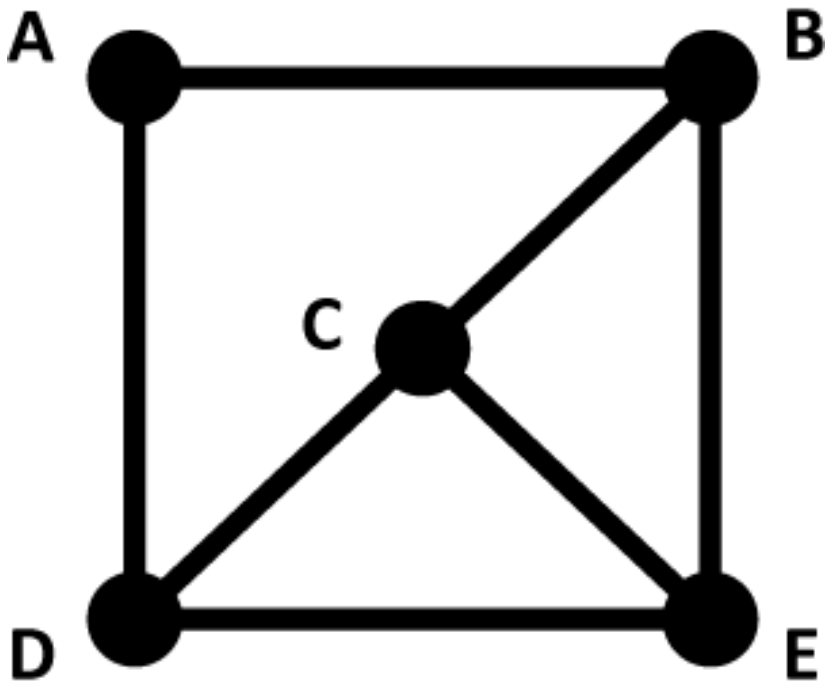
- The reverse is true as well

Euler's Theorem

- If a graph has all even degrees, then it has an Euler circuit. If a graph has any vertices with odd degree, then it does not have an Euler circuit.

Practice With Euler's Theorem

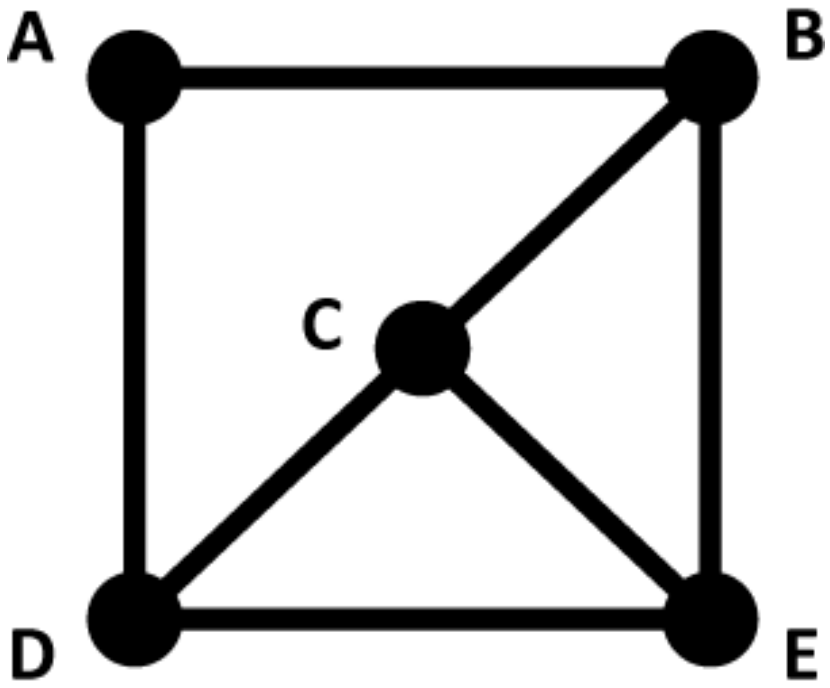
- Does this graph have an Euler circuit? If not, explain why. If so, then find one.



Practice With Euler's Theorem

- Does this graph have an Euler circuit? If not, explain why. If so, then find one.

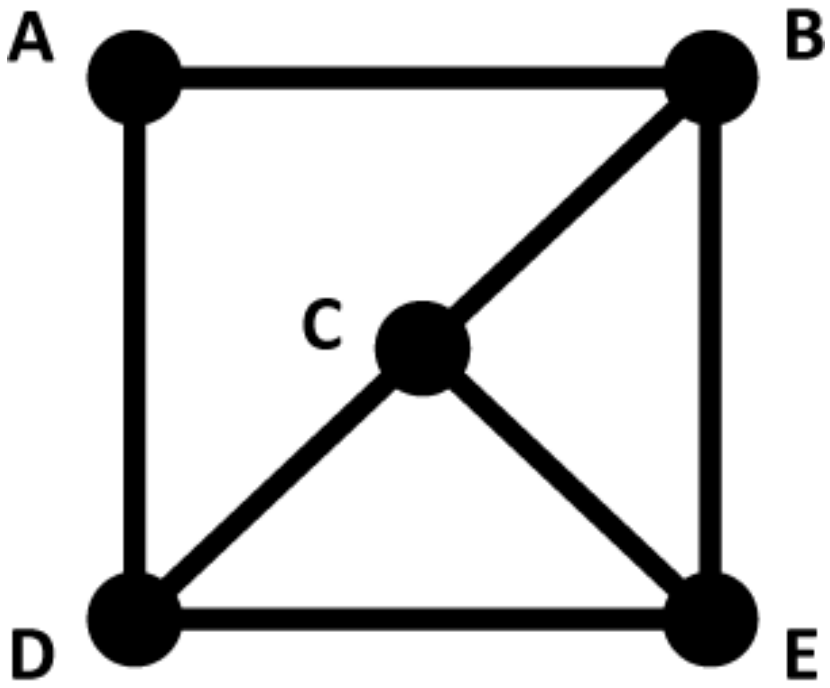
- **Answer**: No, since vertex C has an odd degree.



Practice With Euler's Theorem

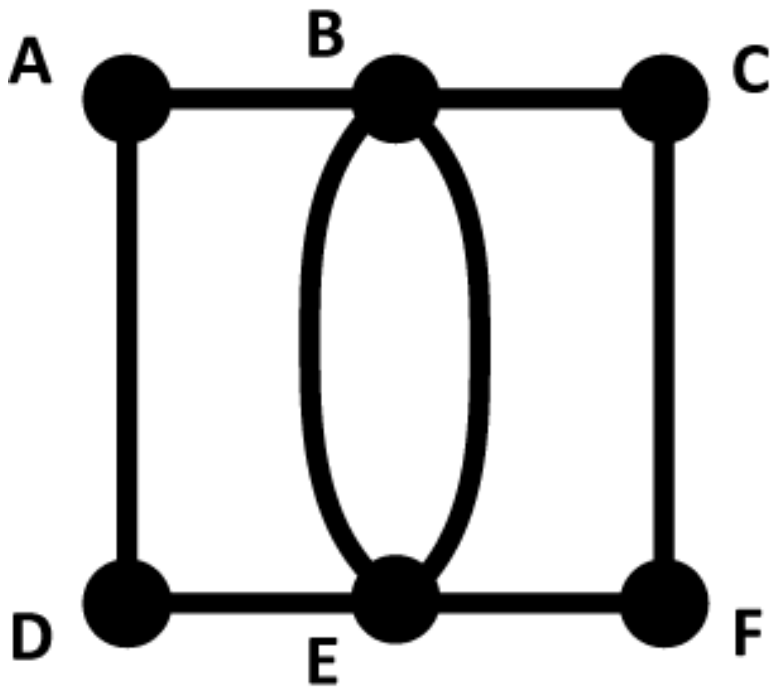
- Does this graph have an Euler circuit? If not, explain why. If so, then find one.

- Notice that D and E also have odd degree, but we only need one odd-degree vertex for Euler's Theorem



Practice With Euler's Theorem

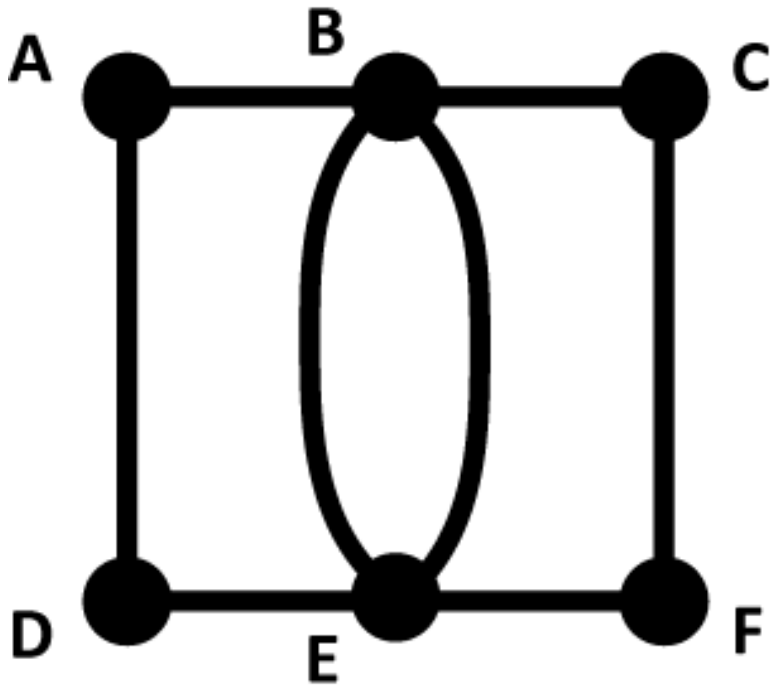
- Does this graph have an Euler circuit? If not, explain why. If so, then find one.



Practice With Euler's Theorem

- Does this graph have an Euler circuit? If not, explain why. If so, then find one.

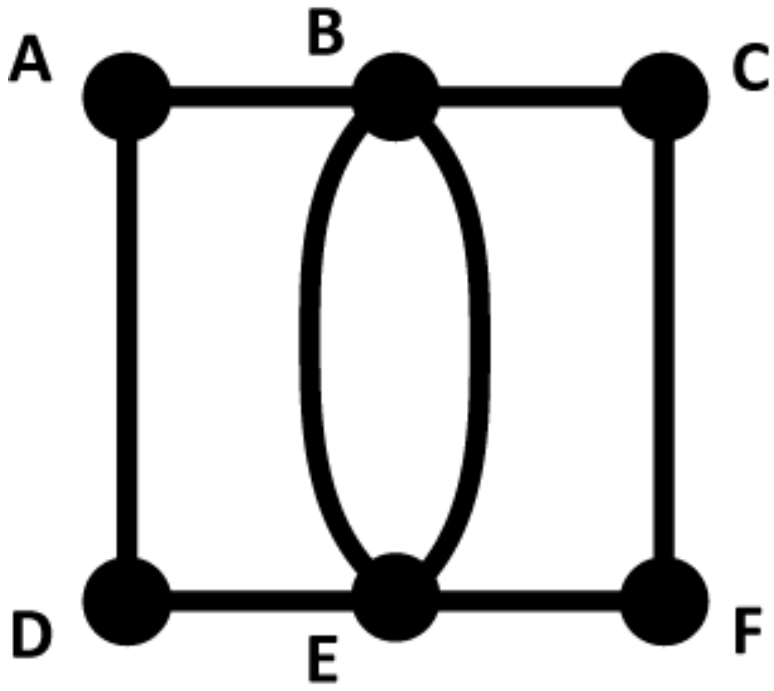
- **Answer**: All of the vertices have even degree, so there must be an Euler circuit…



Practice With Euler's Theorem

- Does this graph have an Euler circuit? If not, explain why. If so, then find one.

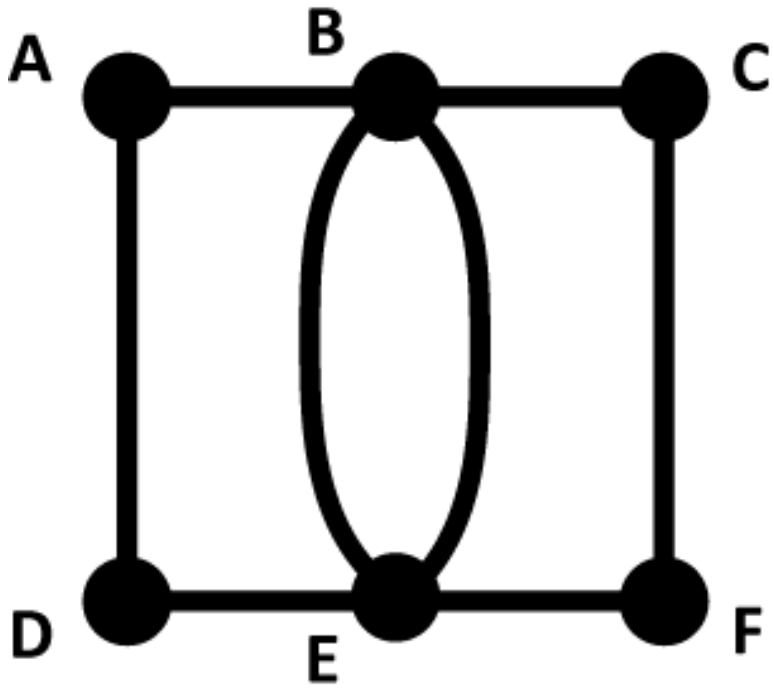
- **Answer** (continued): With some trial and error, we can find an Euler circuit: A,B,C,F,E,B,E,D,A



Practice With Euler's Theorem

- Does this graph have an Euler circuit? If not, explain why. If so, then find one.

- Note there are many different circuits we could have used



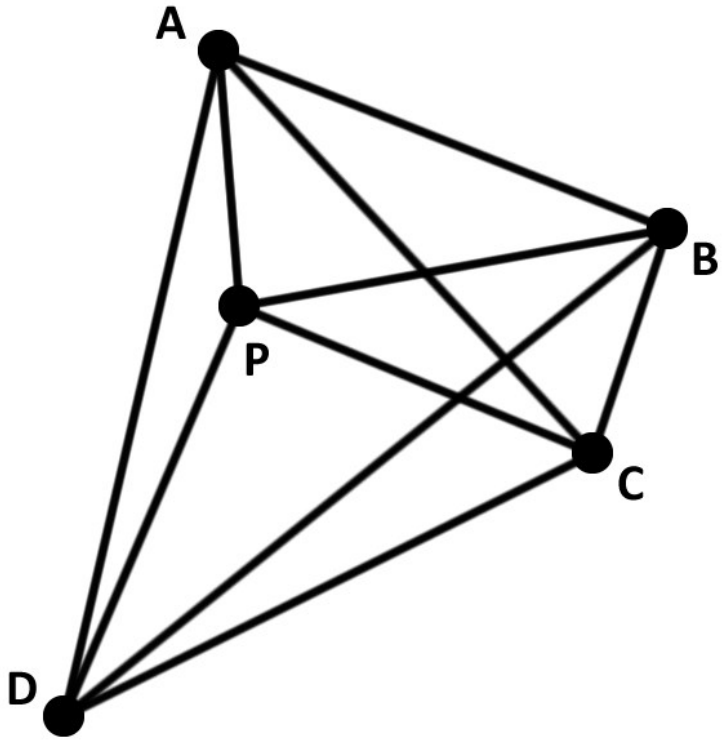
A New Kind of Problem

- A postal worker needs to take several packages from the post office, deliver them to the four locations shown on the map, and then return to the office

A New Kind of Problem

- The postal worker wants to know the best route to take to deliver the packages
- Do we want to use Euler circuits to solve this problem?

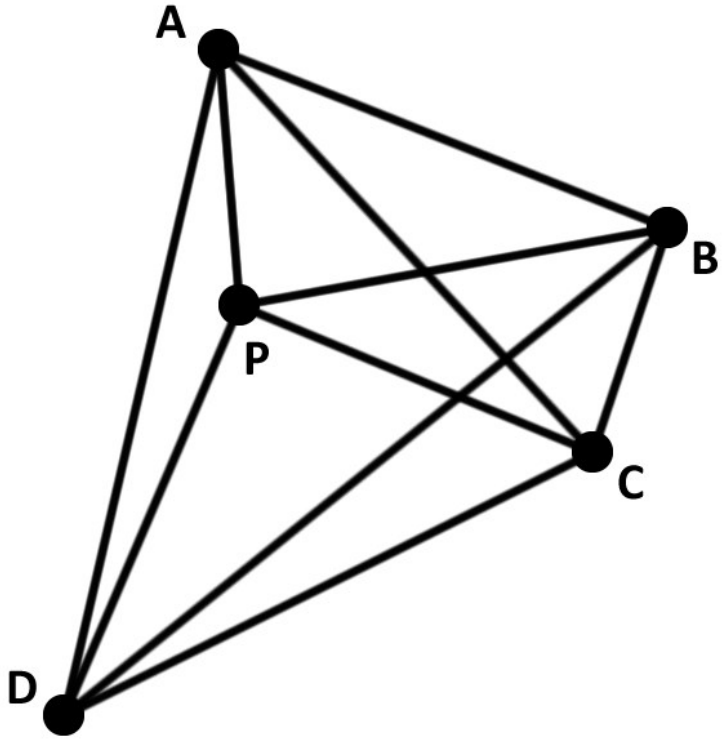
Modeling this Problem



A New Kind of Graph

- This is called a **complete graph** because every pair of vertices is connected by an edge

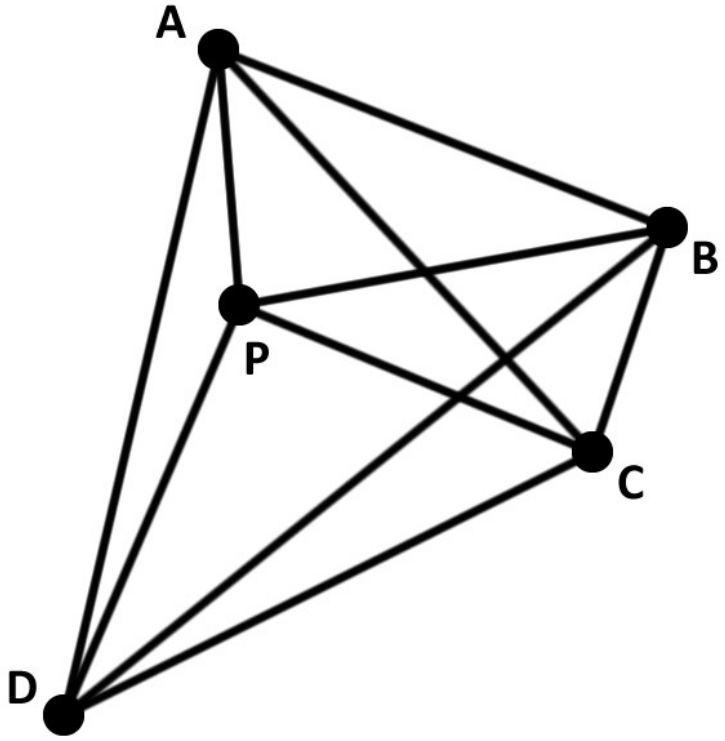
- This represents our ability to travel from any point to any other



A New Kind of Circuit

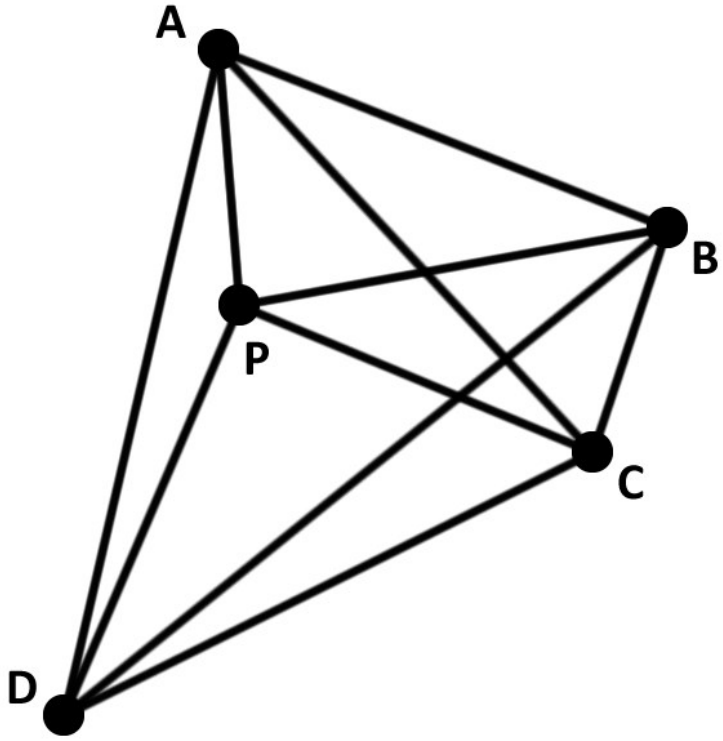
- We don’t need an Euler circuit, which would have us travel along each edge

- We just need to visit each **vertex** once and then return to our starting point

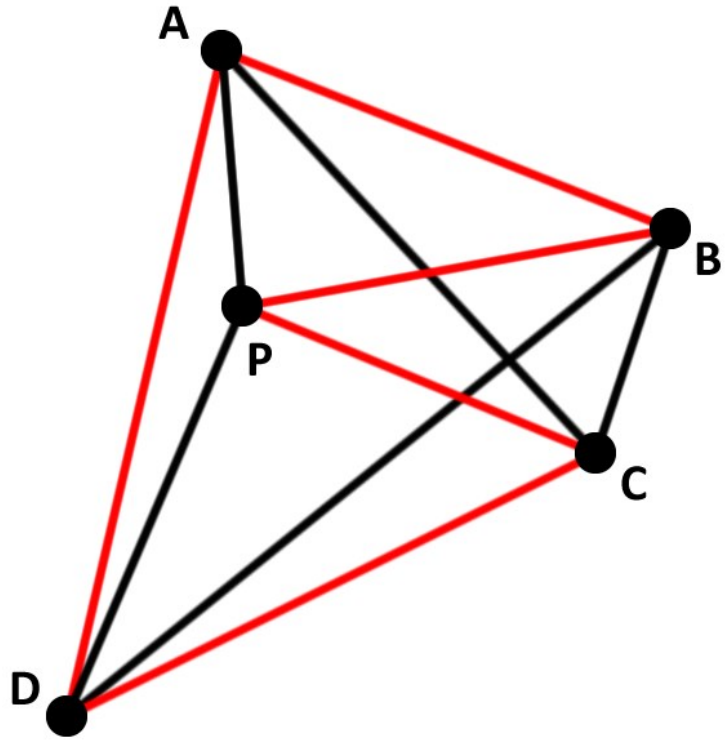


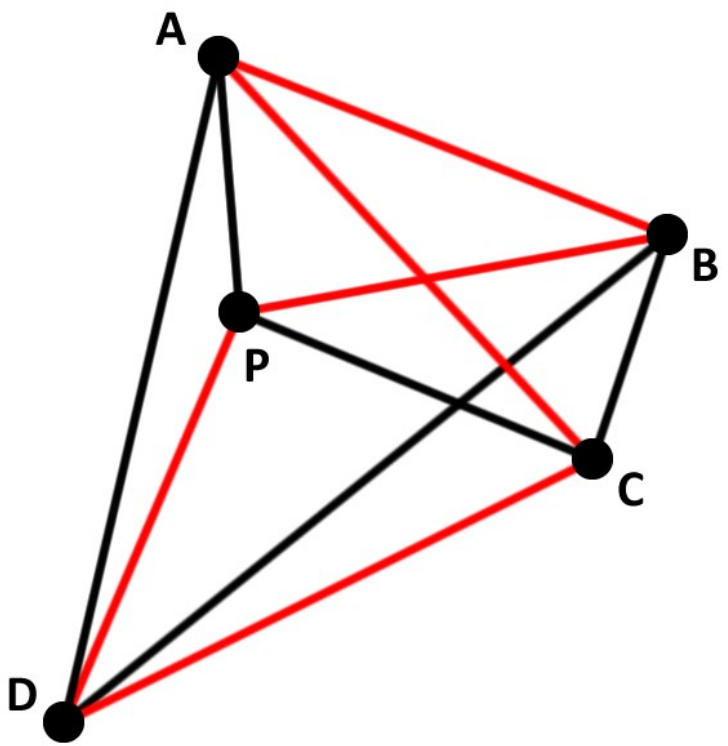
A New Kind of Circuit

- A **Hamiltonian circuit** is a circuit that visits each vertex exactly once, except for the starting vertex, which is the same as the ending vertex



Examples of Hamiltonian Circuits





Finding the “Best” Circuit

- We want to find the circuit that has the lowest total “cost”

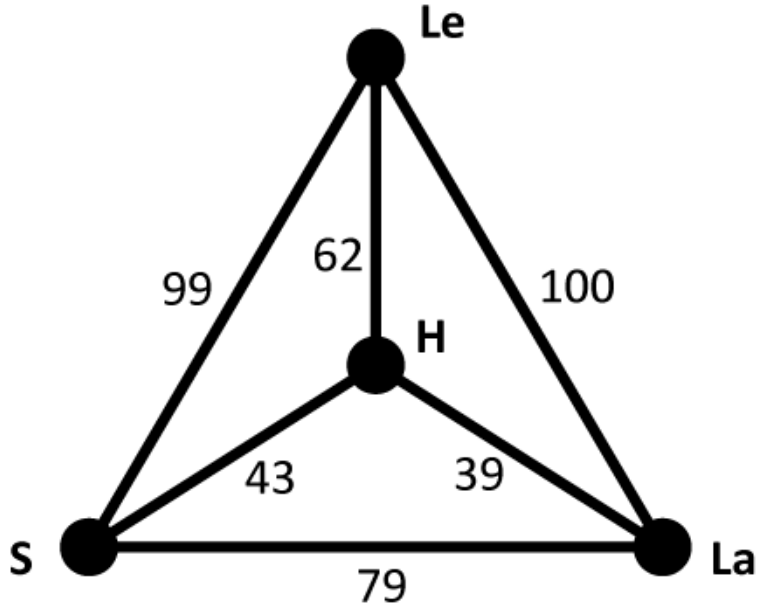
- Here “cost” might mean
  - travel time
  - distance
  - monetary cost
  - etc.

The Brute-Force Method

1. Examine all possible Hamiltonian circuits
2. Compute the total cost of all of these circuits
3. Choose the circuit with the lowest total cost

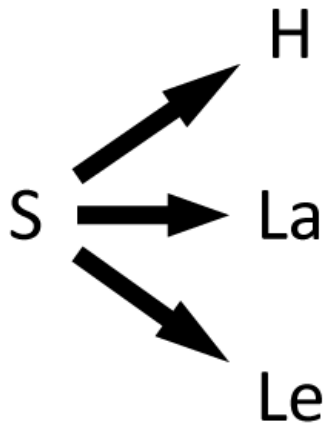
Example: Road Trip!

- Suppose you want to take a road trip for Spring Break
- You want to start from Shippensburg (S), and visit Harrisburg (H), Lancaster (La), and Lewisburg (Le) in some order before returning to Ship



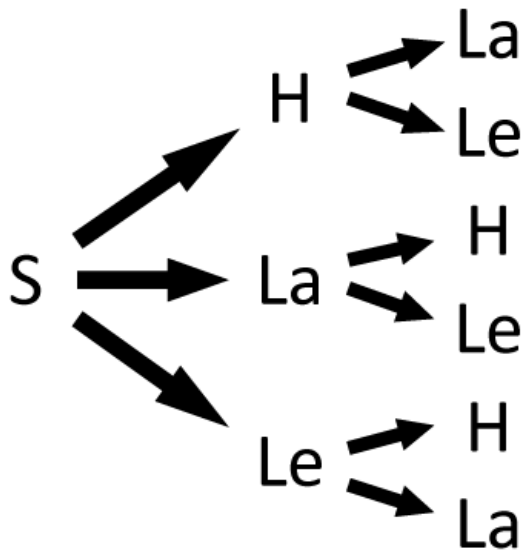
Step 1: Find All Possible Circuits

- From Shippensburg, we have three choices for our first destination



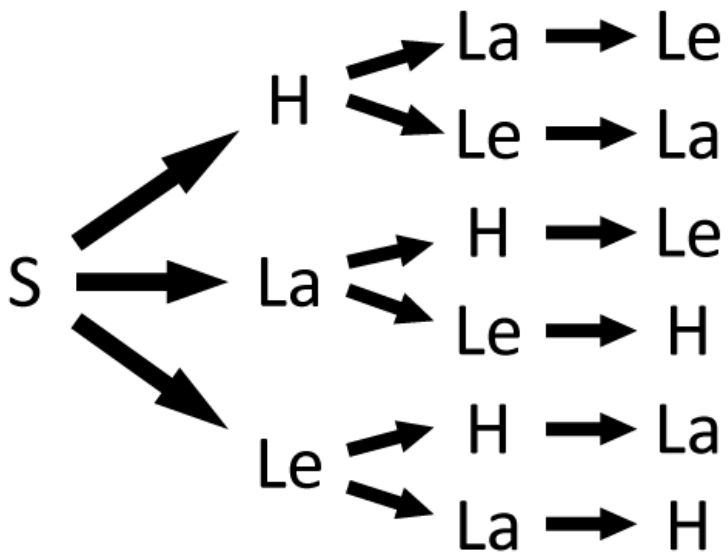
Step 1: Find All Possible Circuits

- From each of these possibilities, we now have two choices



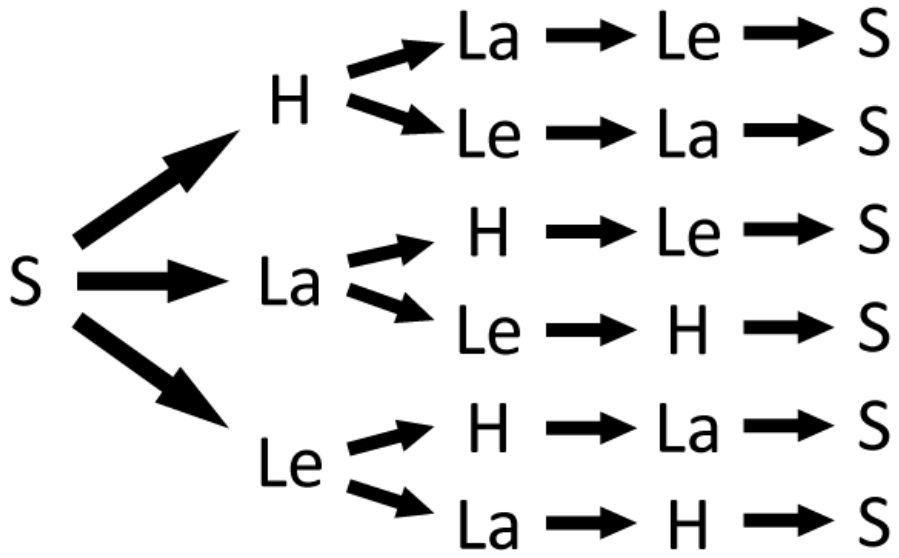
Step 1: Find All Possible Circuits

- Next, we only have one choice remaining



Step 1: Find All Possible Circuits

- And finally, we must return to S



Step 2: Find the Cost of Each Circuit

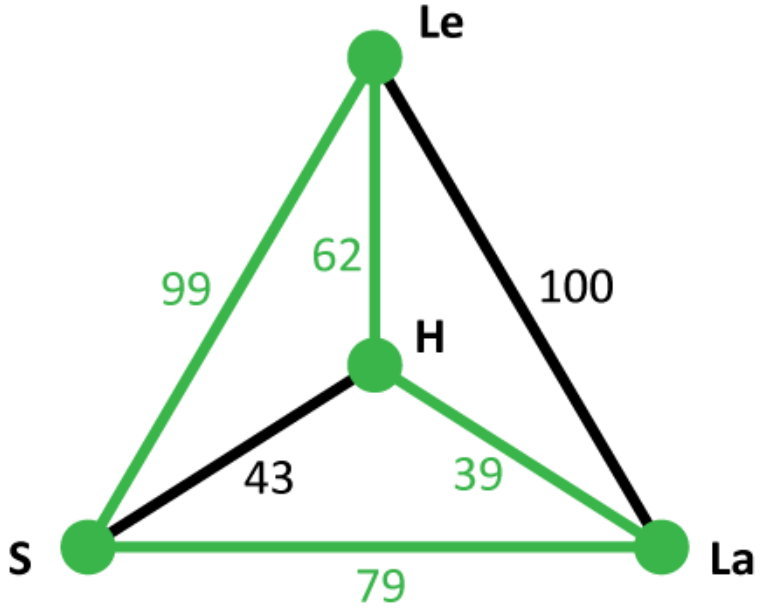
Circuit	Cost
S – H – La – Le – S	43 + 39 + 100 + 99 = 281
S – H – Le – La – S	43 + 62 + 100 + 79 = 284
S – La – H – Le – S	79 + 39 + 62 + 99 = 279
S – La – Le – H – S	79 + 100 + 62 + 43 = 284
S – Le – H – La – S	99 + 62 + 39 + 79 = 279
S – Le – La – H – S	99 + 100 + 39 + 43 = 281

Step 3: Choose the Lowest Cost Circuit

Circuit	Cost
S – H – La – Le – S	43 + 39 + 100 + 99 = 281
S – H – Le – La – S	43 + 62 + 100 + 79 = 284
S – La – H – Le – S	**79 + 39 + 62 + 99 = 279**
S – La – Le – H – S	79 + 100 + 62 + 43 = 284
S – Le – H – La – S	**99 + 62 + 39 + 79 = 279**
S – Le – La – H – S	99 + 100 + 39 + 43 = 281

It’s a tie! Or is it?

- If we draw these two circuits, we find that in fact they are the same
- One circuit is the reverse of the other, so the total costs are the same
- In fact, while it looked like there were 6 total circuits, really there were only 3



Pros and Cons

- The brute force method is good because we know for sure we find the best possible answer

- The biggest disadvantage of the brute force method is that the total number of circuits gets very large if we look at graphs with more vertices

How Many Circuits?

- In our example, we had 4 total vertices
- So from our starting point, we had 3 choices, then we had 2 choices, then 1 choices, then no choice but to go back to the start
- That gave us $3 \times 2 \times 1 = 6$ total circuits
- But really there were only half that: 3 circuits

How Many Circuits?

- What if we had 5 total vertices?
- We would have 4 choices, then 3, then 2, then 1, then back to the start
- That gives us $4 \times 3 \times 2 \times 1 = 24$ total circuits
- And again there would only really be half that: 12 circuits

How Many Circuits?

- The calculation we are doing is a common one in mathematics, called **factorial**
- The factorial of a whole number n is the product of all the whole numbers between 1 and n
- Factorial is written with an exclamation point:
  $n! = n \times (n - 1) \times \ldots \times 3 \times 2 \times 1$
- For example: $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$

How Many Circuits?

- Factorial numbers grow very quickly
- 7! = 5040
- This means that if we had tried to solve our road trip problem with 8 locations instead of 4, we would have had to consider over five thousand circuits instead of just six

Weighted Graphs

In a weighted graph, each edge has a weight or cost

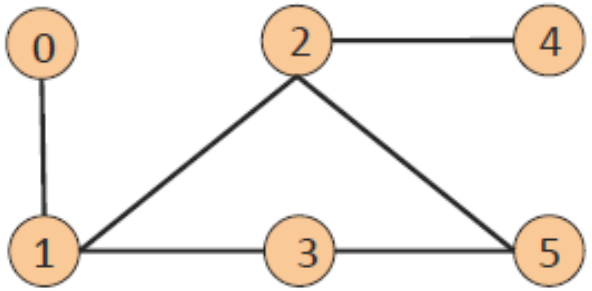
- Typically numeric (ints, decimals, doubles, etc.)
- Some graphs allow negative weights; many do not

❑ Unweighted vs weighted graph

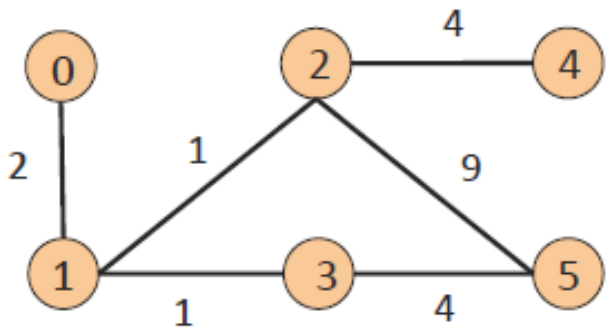
– A weighted graphs has weights on the edges

– Example:

- how many "likes" in your friend's post
- Similarity between a pair of proteins



Unweighted graph

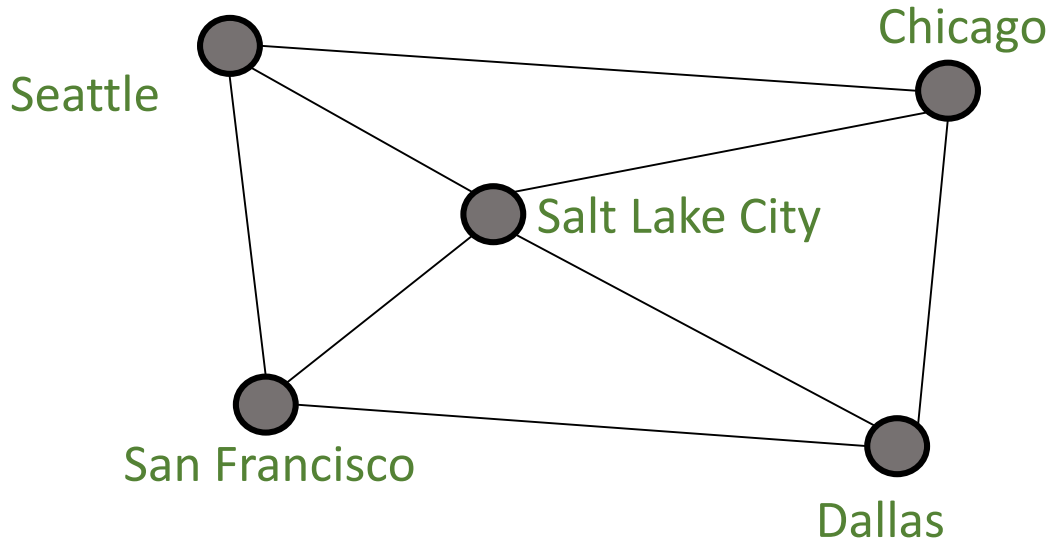


Weighted graph

Paths and Cycles

We say "a path exists from v_0 to v_n" if there is a list of vertices $[v_0, v_1, ..., v_n]$ such that $(v_i, v_{i+1}) \in E$ for all $0 \leq i < n$.

A cycle is a path that begins and ends at the same node ($v_0 == v_n$)



Example path (that also happens to be a cycle):

[Seattle, Salt Lake City, Chicago, Dallas, San Francisco, Seattle]

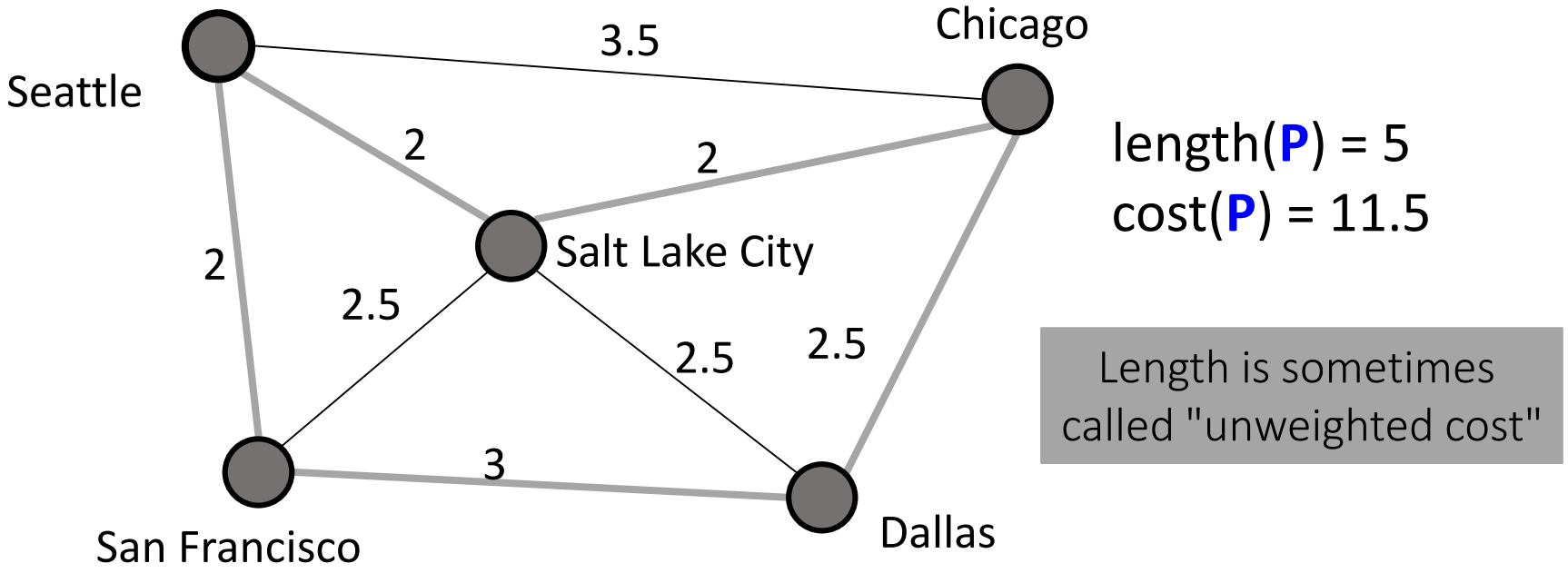
Path Length and Cost

Path length: Number of edges in a path

Path cost: Sum of the weights of each edge

Example where

P= [Seattle, Salt Lake City, Chicago, Dallas,
San Francisco, Seattle]



length(**P**) = 5

cost(**P**) = 11.5

Length is sometimes called "unweighted cost"

Simple Paths and Cycles

A simple path repeats no vertices (except the first might be the last):

[Seattle, Salt Lake City, San Francisco, Dallas]

[Seattle, Salt Lake City, San Francisco, Dallas, Seattle]

A cycle is a path that ends where it begins:

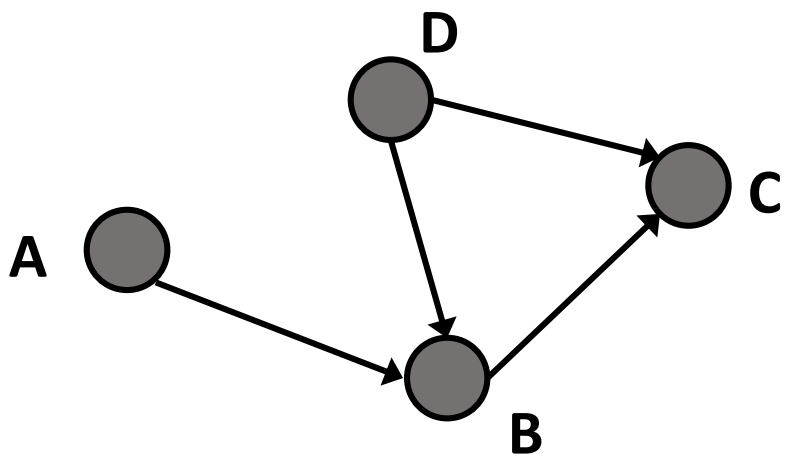
[Seattle, Salt Lake City, Seattle, Dallas, Seattle]

A simple cycle is a cycle and a simple path:

[Seattle, Salt Lake City, San Francisco, Dallas, Seattle]

Paths and Cycles in Directed Graphs

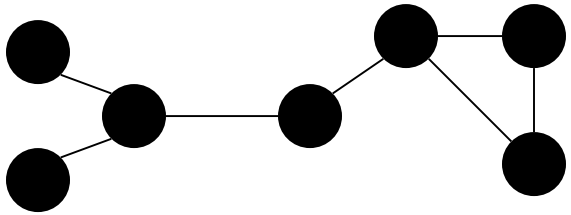
Example:



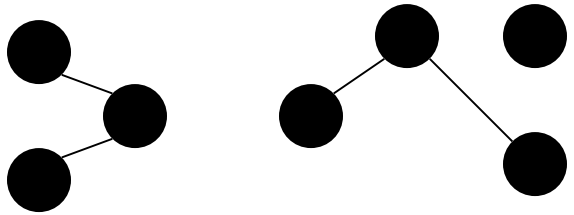
- Is there a path from A to D? No
- Does the graph contain any cycles? No

Undirected Graph Connectivity

An undirected graph is connected if for all pairs of vertices $u \neq v$, there exists a *path* from u to v

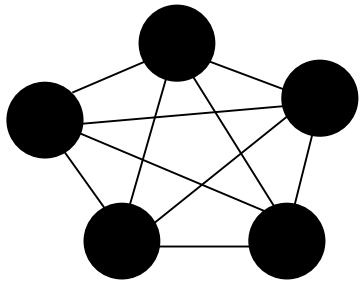


Connected graph



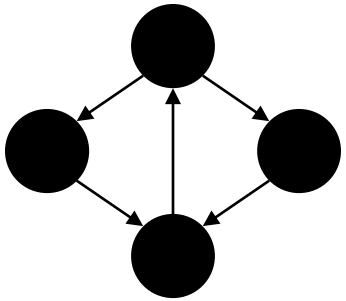
Disconnected graph

An undirected graph is complete, or fully connected, if for all pairs of vertices $u \neq v$ there exists an *edge* from u to v

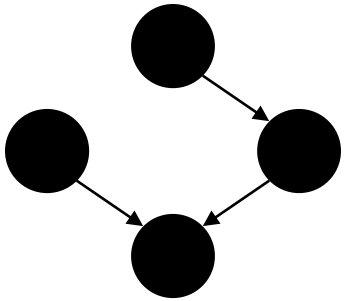


Directed Graph Connectivity

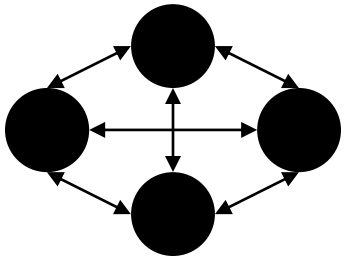
A directed graph is strongly connected if there is a path from every vertex to every other vertex



A directed graph is weakly connected if there is a path from every vertex to every other vertex *ignoring direction of edges*



A direct graph is complete or fully connected, if for all pairs of vertices $u \neq v$, there exists an *edge* from u to v

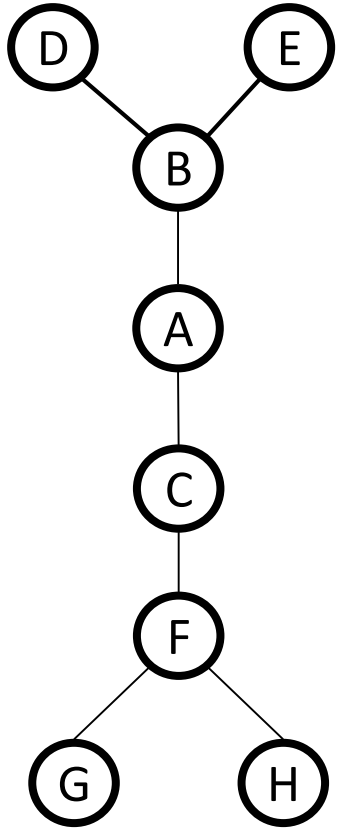


Trees as Graphs

When talking about graphs, we say a tree is a graph that is:

- undirected
- acyclic
- connected

All trees are graphs, but NOT all graphs are trees



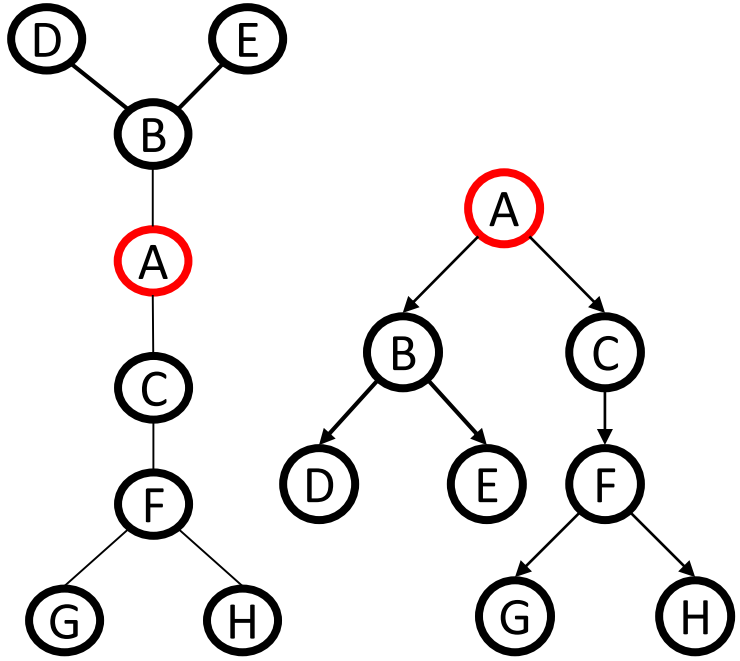
Rooted Trees

We are more accustomed to rooted trees where:

- We identify a unique root
- We think of edges as directed: parent to children

Picking a root gives a unique rooted tree

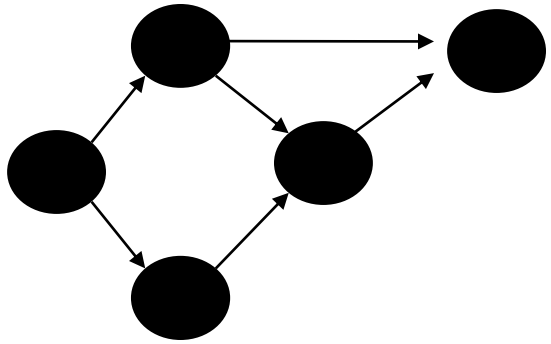
- The tree is simply drawn differently and with undirected edges



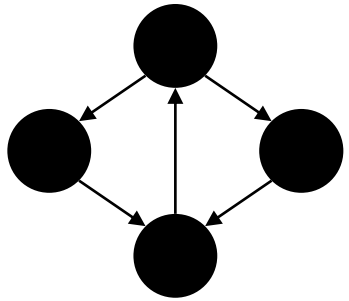
Directed Acyclic Graphs (DAGs)

A DAG is a directed graph with no directed cycles

- Every rooted directed tree is a DAG
- But not every DAG is a rooted directed tree



- Every DAG is a directed graph
- But not every directed graph is a DAG



Brute Force is Hard!

- As we have seen, the brute force method can require us to examine a very large number of circuits
- In this section we will develop *algorithms* for finding an answer much more quickly
- The downside is that we will no longer be guaranteed to have the best possible answer

Nearest-Neighbor Algorithm

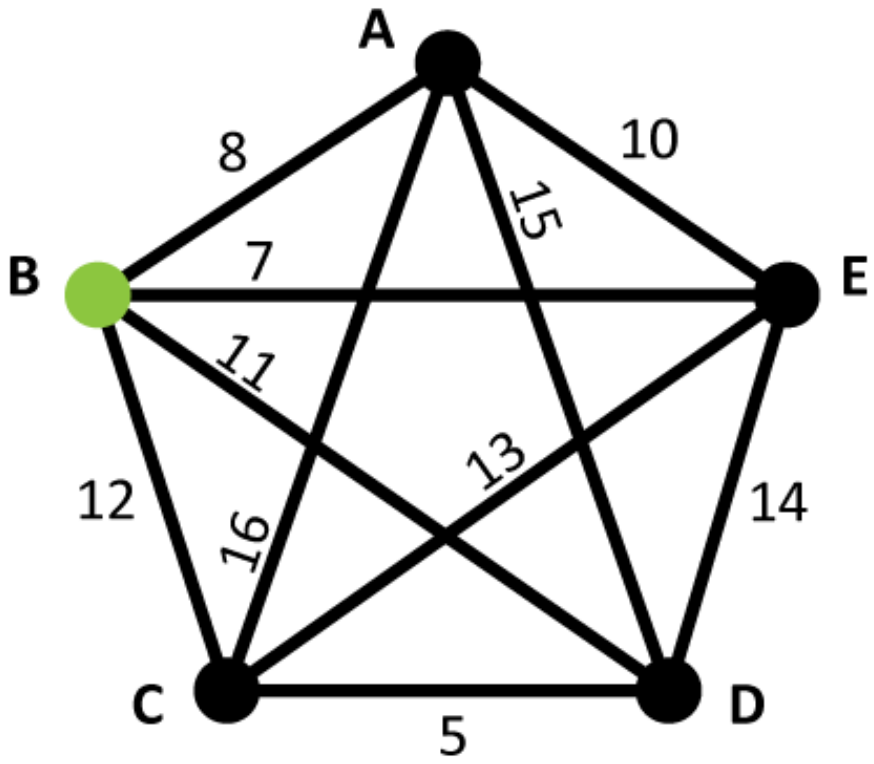
- The first algorithm we will consider is called the nearest-neighbor algorithm

- It’s based on a common sense idea: at each vertex, choose the closest vertex that you haven’t visited yet

Nearest-Neighbor Algorithm

- We have to have a starting point

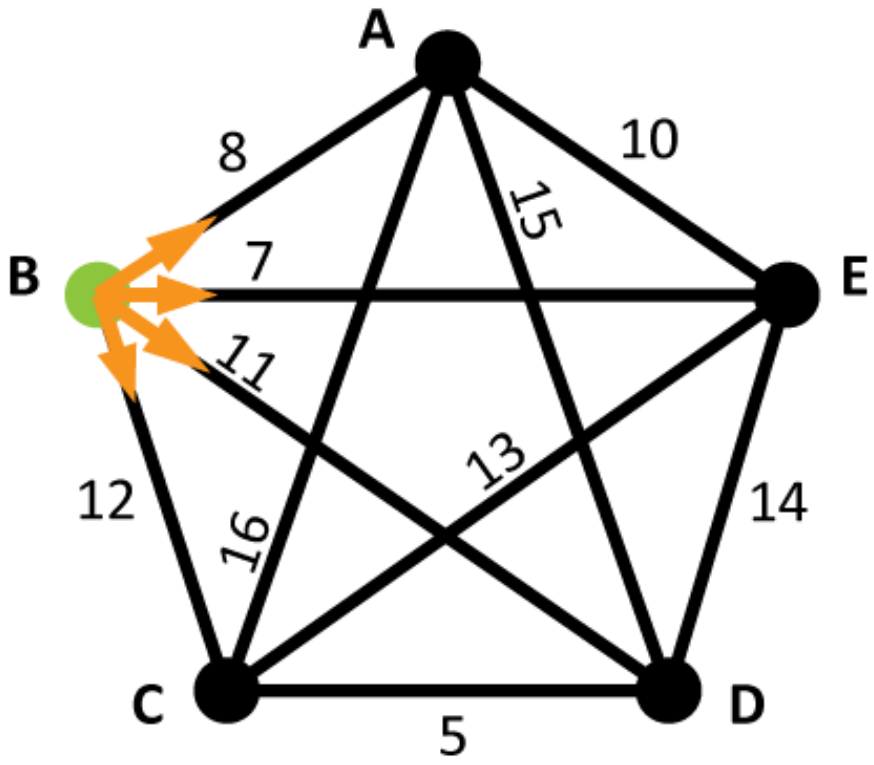
- We will choose our second vertex by finding the “nearest neighbor”



Nearest-Neighbor Algorithm

- Where do we go first?

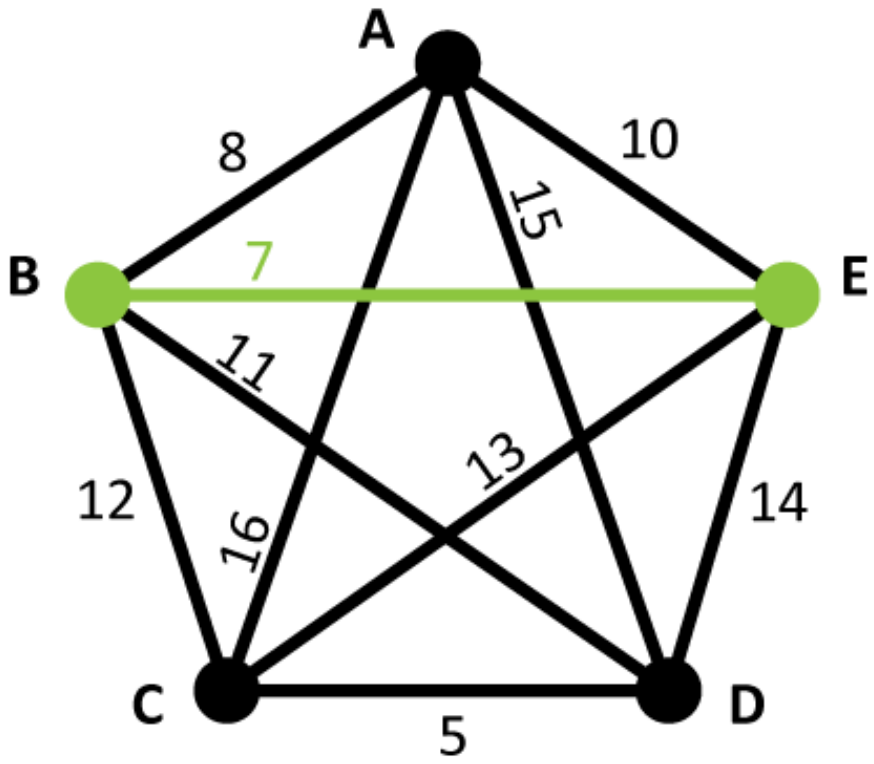
- Choose the cheapest edge



Nearest-Neighbor Algorithm

- Choose the cheapest edge

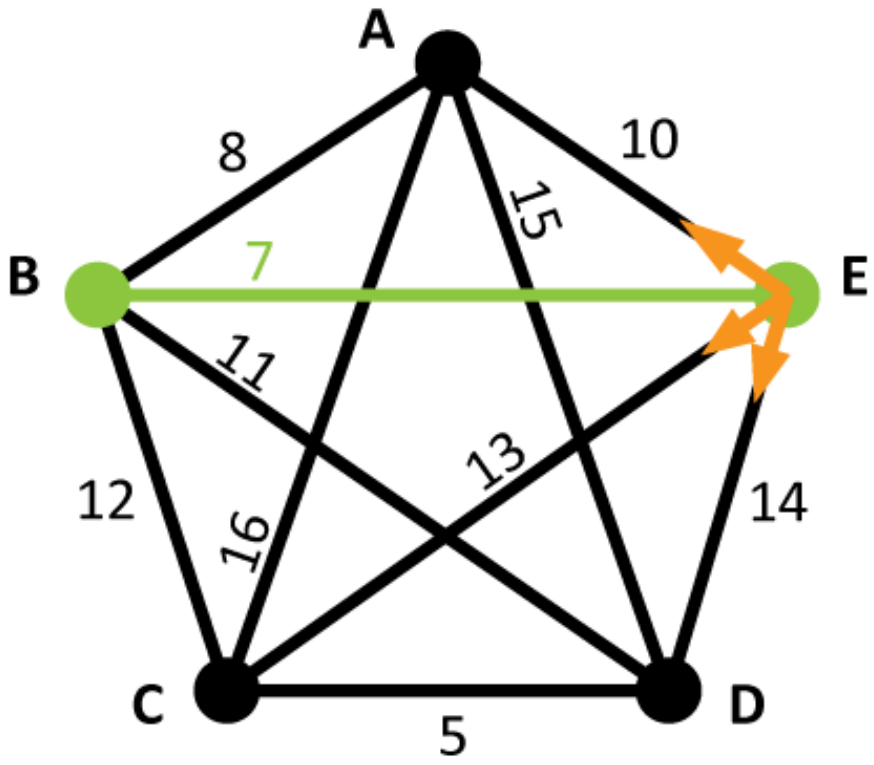
- In this case, we go from B to E (7)



Nearest-Neighbor Algorithm

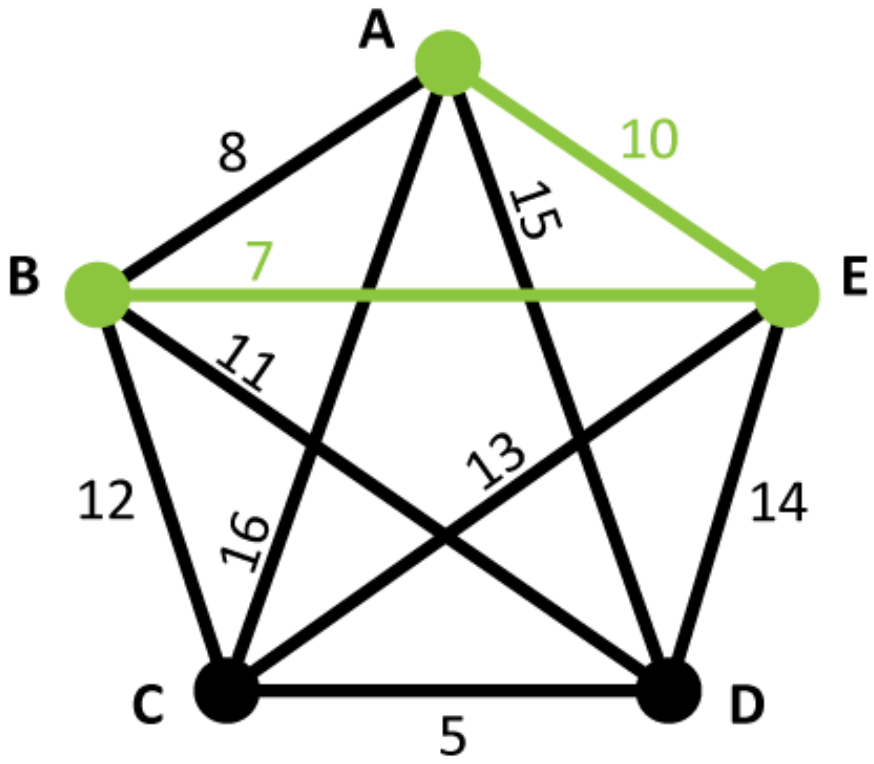
- Now where do we go?

- We can’t go back to B



Nearest-Neighbor Algorithm

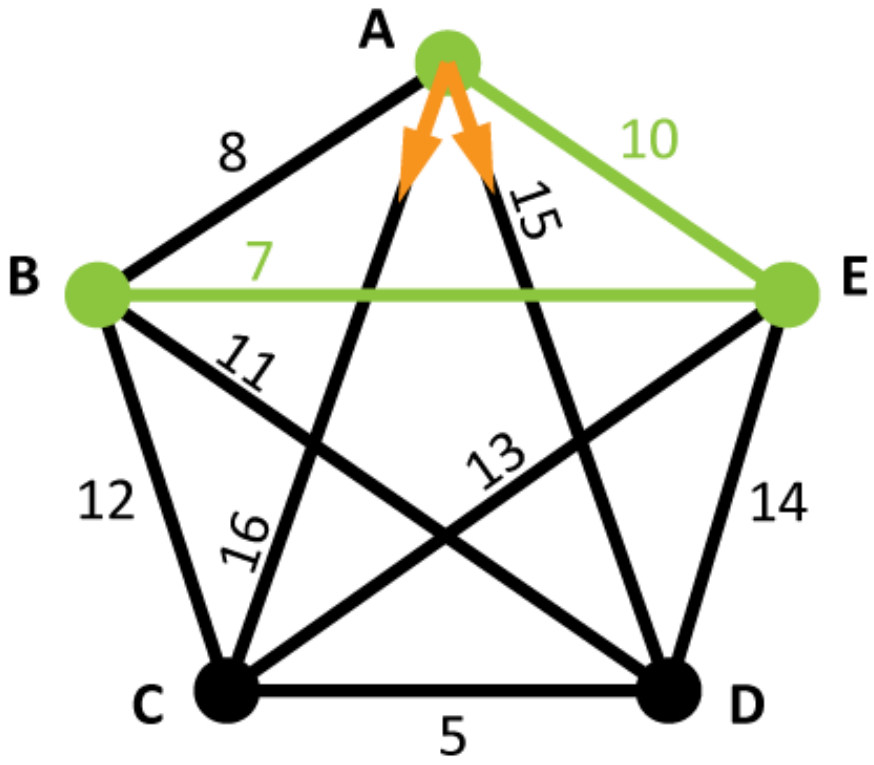
- Now where do we go?
- We can’t go back to B
- Again choose the cheapest edge



Nearest-Neighbor Algorithm

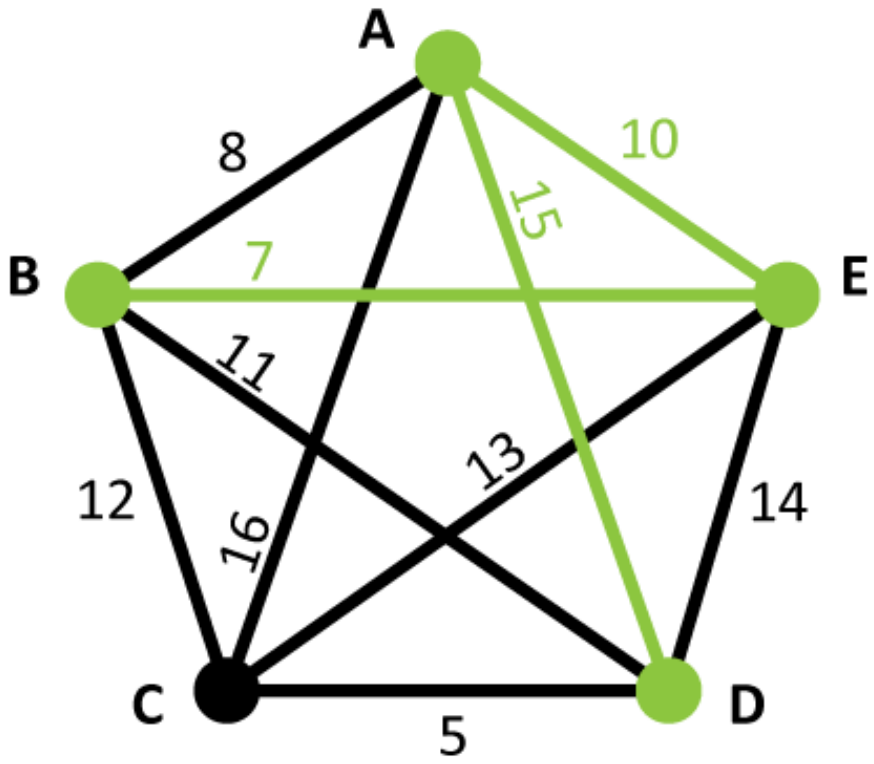
- Now where do we go?

- We can’t go back to E, but we also can’t go to B



Nearest-Neighbor Algorithm

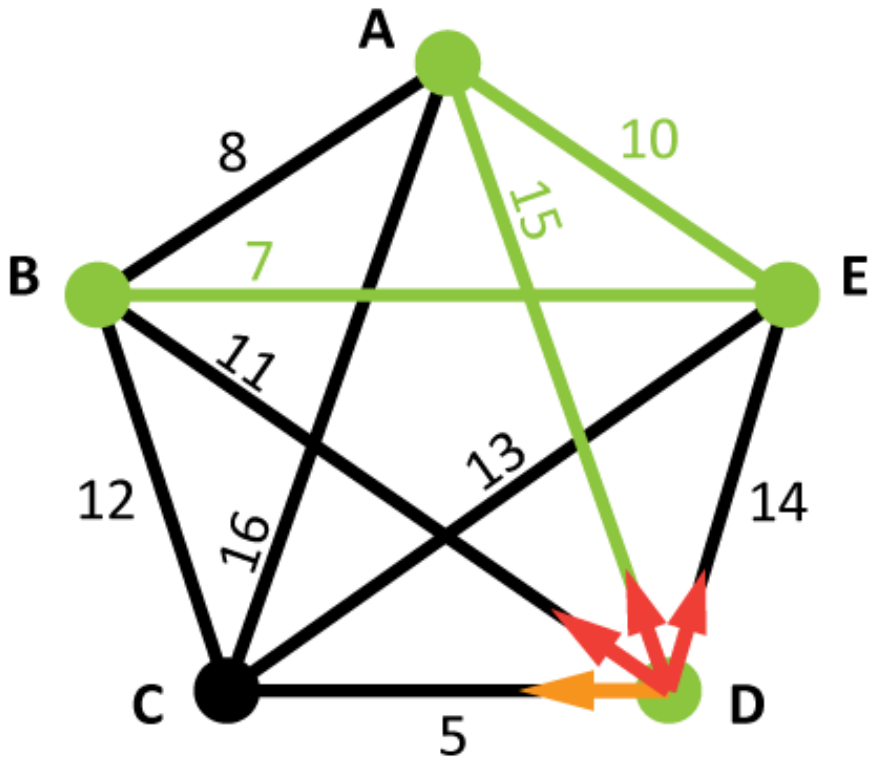
- The rule is “nearest neighbor”: always choose the lowest cost edge, unless that would take you back to a vertex you have already been to



Nearest-Neighbor Algorithm

- Now we only have one choice

- We can't go back to A or E, and we can't return to B because that would leave out C

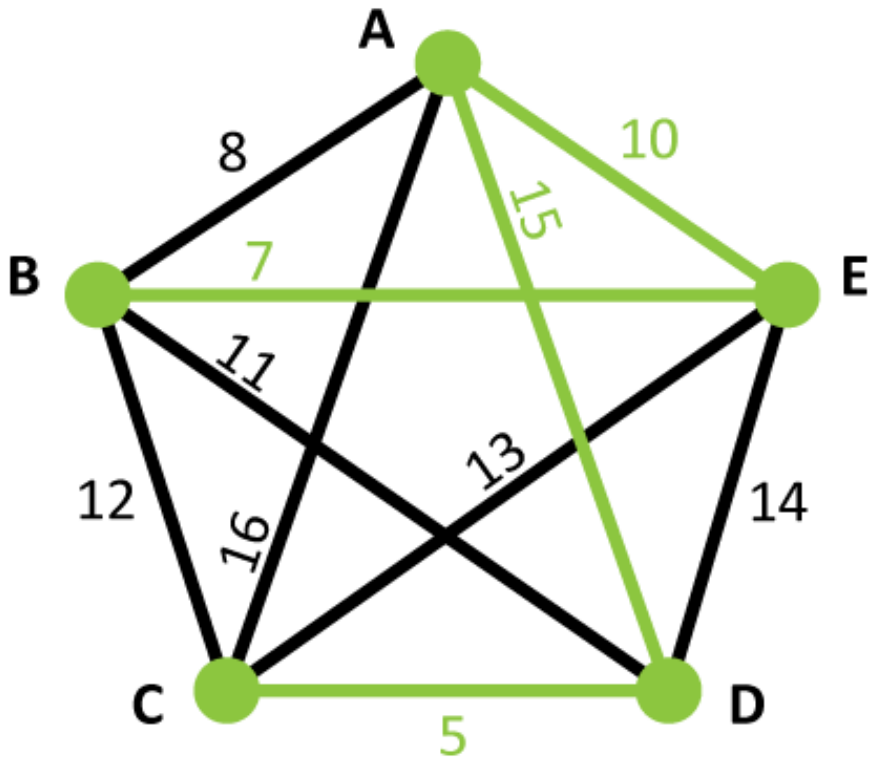


Nearest-Neighbor Algorithm

- Now we only have one choice

- We can't go back to A or E, and we can't return to B because that would leave out C

- So we must go to C

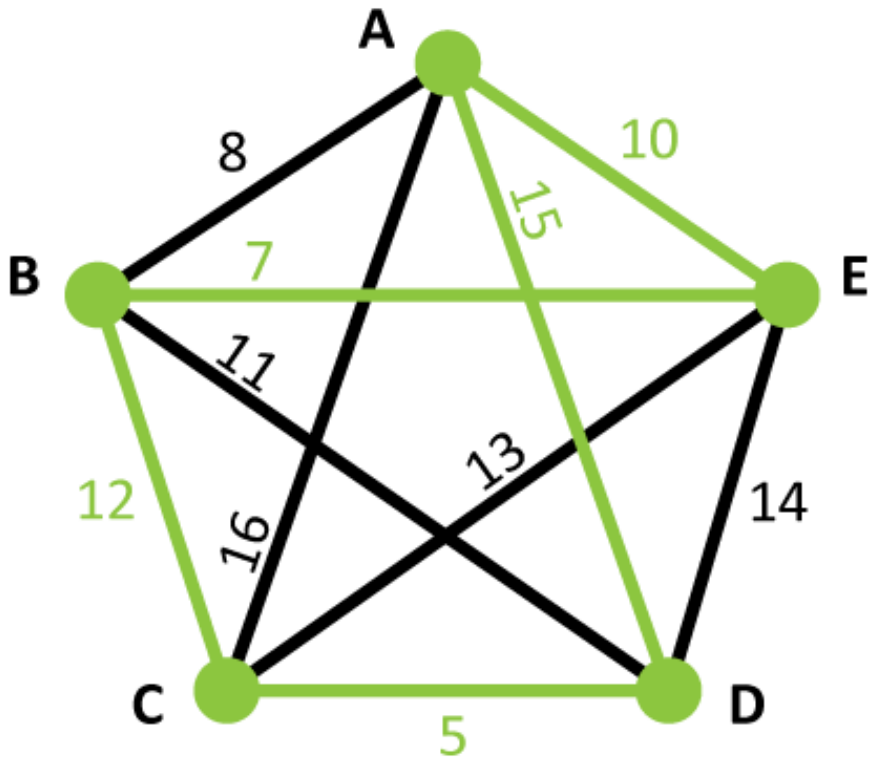


Nearest-Neighbor Algorithm

- We have now visited all of the vertices, so we finally return to B

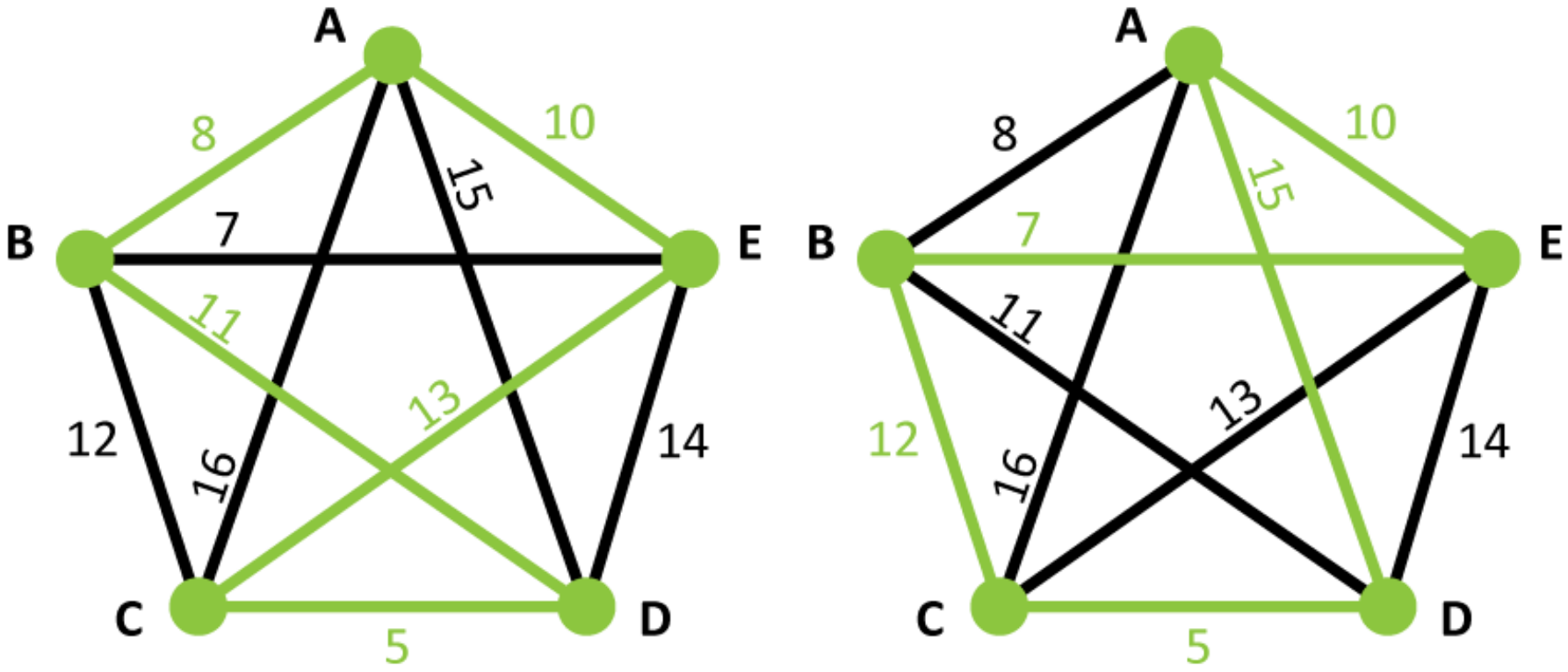
- This circuit has a total cost of 49

- Is it the best circuit?



Nearest-Neighbor Algorithm

- It is *not* the best! The solution on the left has a total cost of 47



Nearest-Neighbor Algorithm

1. From the starting vertex, choose the edge with the smallest cost and use that as the first edge in your circuit.
2. Continue in this manner, choosing among the edges that connect from the current vertex to vertices you have not yet visited.
3. When you have visited every vertex, return to the starting vertex.

Nearest-Neighbor Algorithm

- **Advantages**: easy, “heuristic,” and fast
- **Disadvantage**: doesn’t always give you the best possible answer
- “Heuristic” means that this method uses a common-sense idea

Representation

- **Incidence (Matrix):** Most useful when information about edges is more desirable than information about vertices.

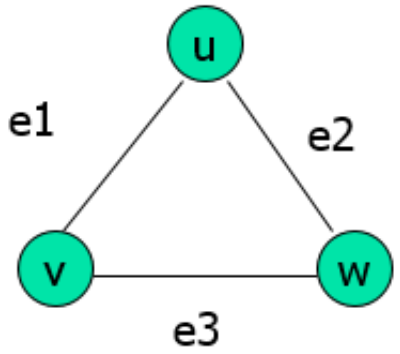
- **Adjacency (Matrix/List):** Most useful when information about the vertices is more desirable than information about the edges. These two representations are also most popular since information about the vertices is often more desirable than edges in most applications

Representing Graphs

The incidence matrix of G with respect to this listing of the vertices and edges is the n×m zero-one matrix with 1 as its (i, j) entry when edge e_j is incident with v_i, and 0 otherwise.

- In other words, for an incidence matrix $M = [m_{ij}]$,

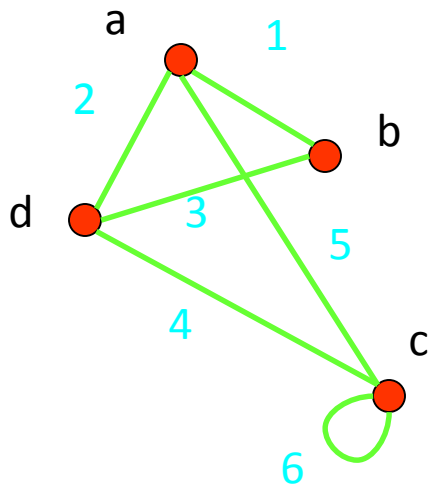
$m_{ij} = 1$ if edge e_j is incident with v_i
$m_{ij} = 0$ otherwise.



	e_1	e_2	e_3
v	1	0	1
u	1	1	0
w	0	1	1

Representing Graphs

Example: What is the incidence matrix M for the following graph G based on the order of vertices a, b, c, d and edges 1, 2, 3, 4, 5, 6?



Solution:

$$M = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

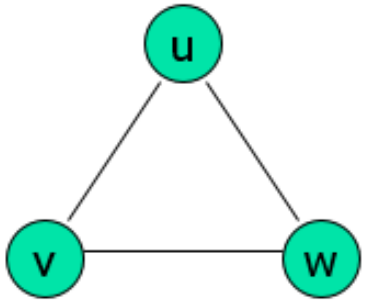
Note: Incidence matrices of undirected graphs contain two 1s per column for edges connecting two vertices and one 1 per column for loops.

Representing Graphs

The **adjacency matrix** A (or A_G) of G, with respect to this listing of the vertices, is the n×n zero-one matrix with 1 as its (i, j) entry when v_i and v_j are adjacent, and 0 otherwise.

•In other words, for an adjacency matrix $A = [a_{ij}]$,

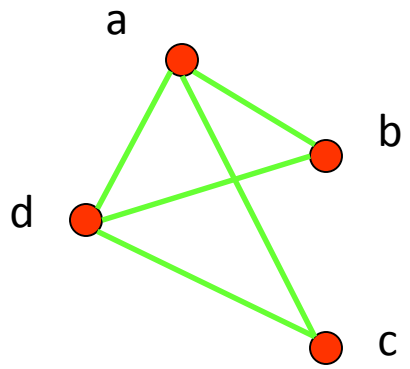
$a_{ij} = 1$ if $\{v_i, v_j\}$ is an edge of G,
$a_{ij} = 0$ otherwise.



	v	u	w
v	0	1	1
u	1	0	1
w	1	1	0

Representing Graphs

•**Example:** What is the adjacency matrix A_G for the following graph G based on the order of vertices a, b, c, d ?



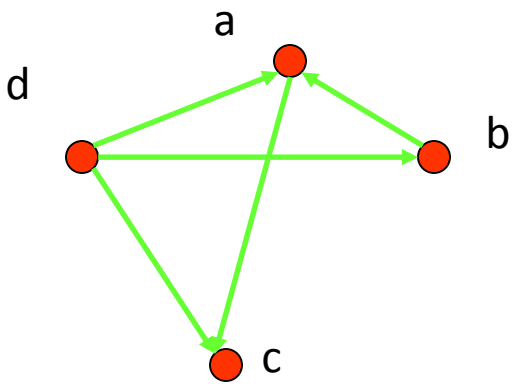
Solution:

$$A_G = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

Note: Adjacency matrices of undirected graphs are always symmetric.

Representing Directed Graphs

- **Example:** What is the adjacency matrix A_G for the following directed graph G based on the order of vertices a, b, c, d ?

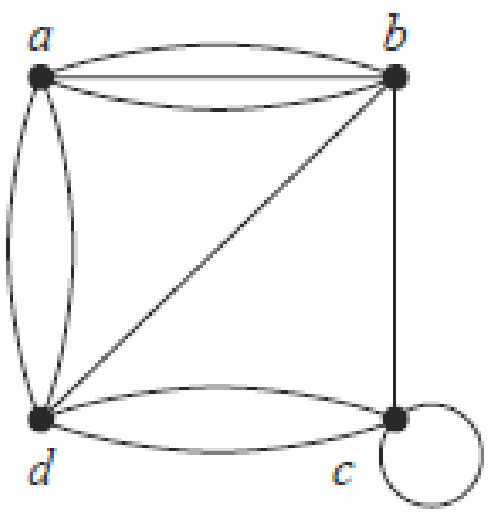


Solution:

$$A_G = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

Representing “Pseudographs”

- **Example** What is the adjacency matrix A_G for the following preudograph G based on the order of vertices a, b, c, d ?

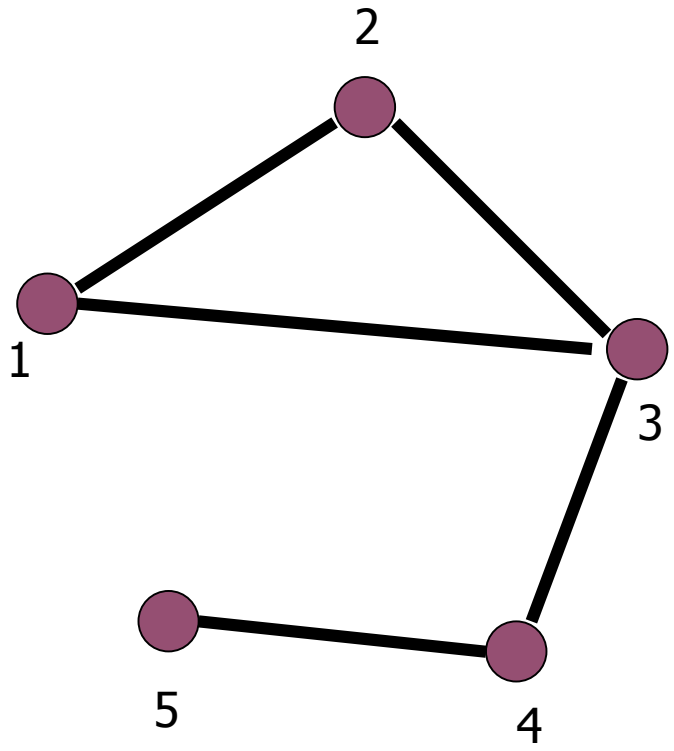


Solution:

$$\begin{bmatrix} 0 & 3 & 0 & 2 \\ 3 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \\ 2 & 1 & 2 & 0 \end{bmatrix}$$

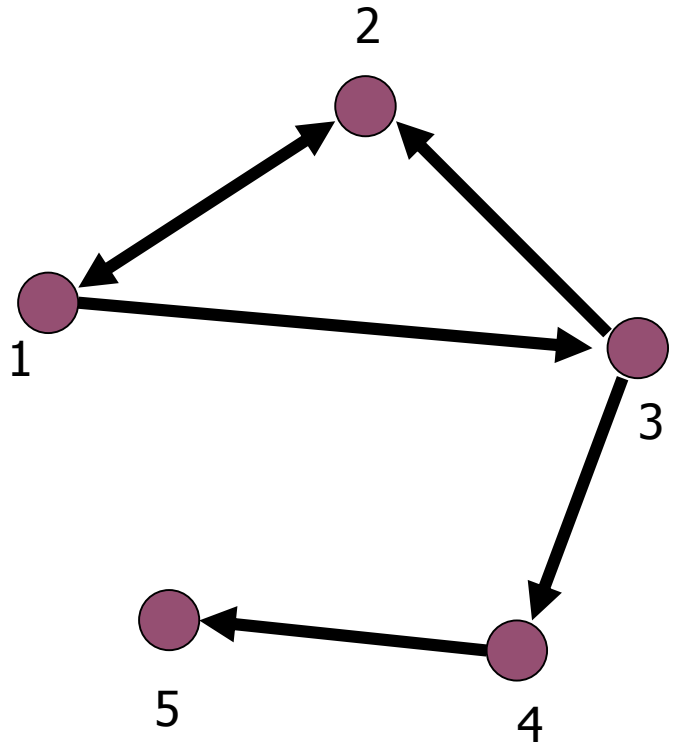
- Adjacency Matrix
  - symmetric matrix for undirected graphs

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



- Adjacency Matrix
  - unsymmetric matrix for undirected graphs

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$



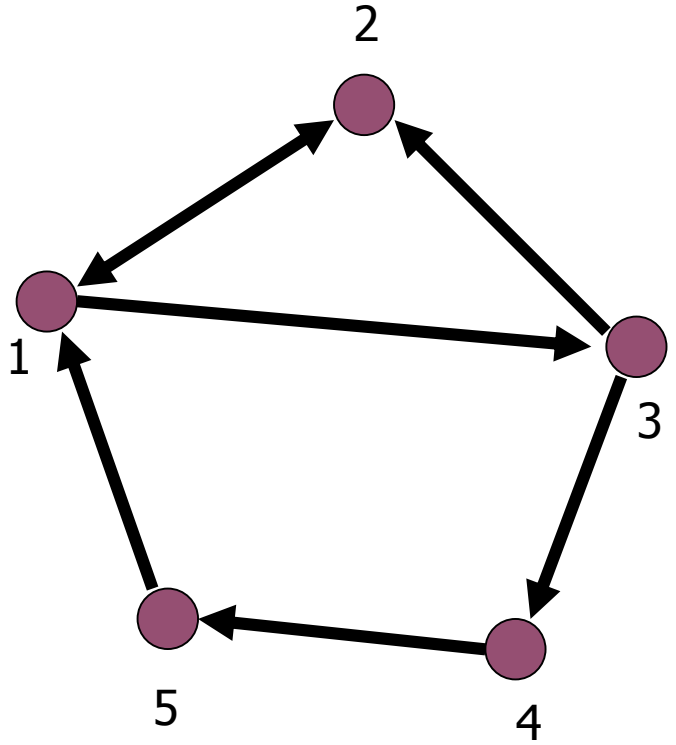
Example. Random Walks

- Start from a node, and follow links uniformly at random.
- Stationary distribution: The fraction of times that you visit node i, as the number of steps of the random walk approaches infinity
  - if the graph is strongly connected, the stationary distribution converges to a unique vector.

Random Walks

- stationary distribution: principal left eigenvector of the normalized adjacency matrix
  - x = xP
  - for undirected graphs, the degree distribution

$$P = \begin{bmatrix} 0 & 1/2 & 1/2 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$



Example. Counting Paths between Vertices

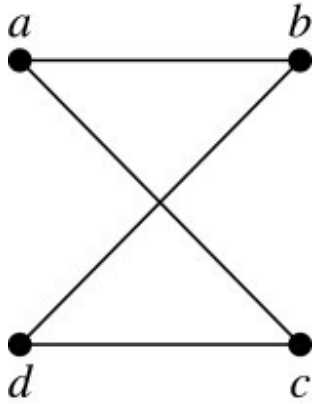
We can use the adjacency matrix of a graph to find the number of paths between two vertices in the graph.

Theorem: Let G be a graph with adjacency matrix **A** with respect to the ordering $v_1, \ldots, v_n$ of vertices (with directed or undirected edges, multiple edges and loops allowed). The number of different paths of length r from v_i to v_j equals the (i,j)th entry of $\mathbf{A}^r$, where $r > 0$ is a positive integer,

- Note: This is the standard power of matrix **A**, not a Boolean product.

Example: How many paths of length four are there from *a* to *d* in the graph G.

G



adjacency matrix A of G

$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

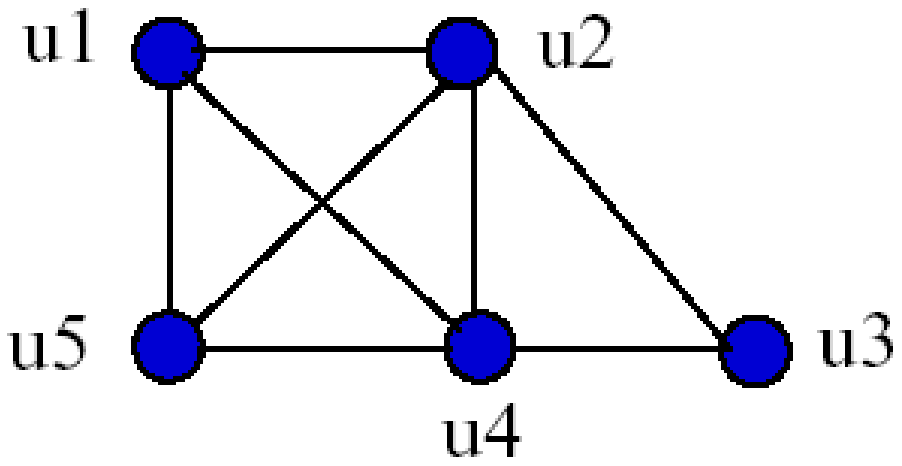
$\mathbf{A}^4 =$

$$\begin{bmatrix} 8 & 0 & 0 & 8 \\ 0 & 8 & 8 & 0 \\ 0 & 8 & 8 & 0 \\ 8 & 0 & 0 & 8 \end{bmatrix}$$

Solution: The adjacency matrix of *G* is given above. Hence the number of paths of length four from *a* to *d* is the $(1,4)$th entry of $\mathbf{A}^4$. The eight paths are as:

a, b, a, b, d *a, b, a, c, d*
a, b, d, b, d *a, b, d, c, d*
a, c, a, b, d *a, c, a, c, d*
a, c, d, b, d *a, c, d, c, d*

Example 2



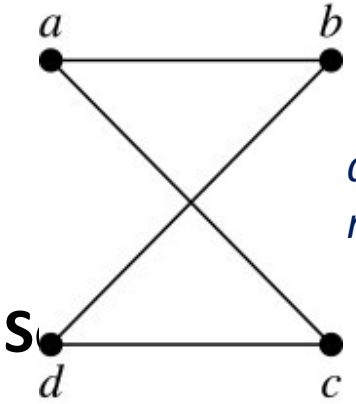
$$M = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

$$M^2 = \begin{bmatrix} 3 & 2 & 2 & 2 & 2 \\ 2 & 4 & 1 & 3 & 2 \\ 2 & 1 & 2 & 1 & 2 \\ 2 & 3 & 1 & 4 & 2 \\ 2 & 2 & 2 & 2 & 3 \end{bmatrix}$$

$$M^3 = \begin{bmatrix} 6 & 9 & 4 & 9 & 7 \\ 9 & 8 & 7 & 9 & 9 \\ 4 & 7 & 2 & 7 & 4 \\ 9 & 9 & 7 & 8 & 9 \\ 7 & 9 & 4 & 9 & 6 \end{bmatrix}$$

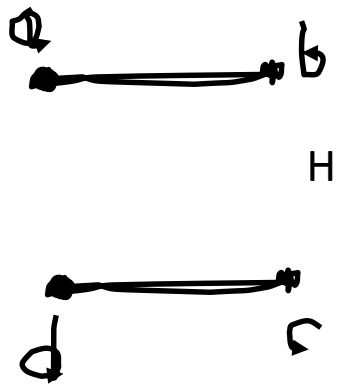
Example: Use the theorem on the previous slide to show that graph G is connected and graph H is not connected.

G



adjacency matrix A of G

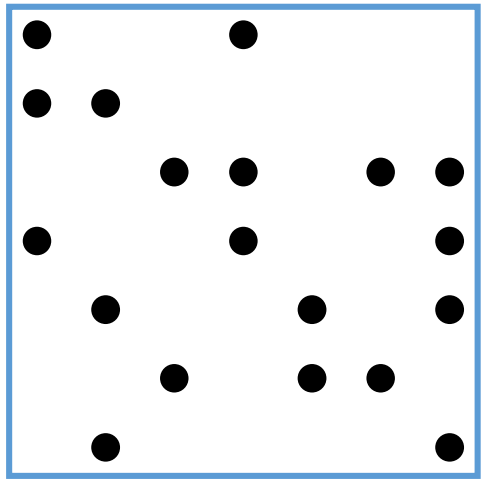
$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$



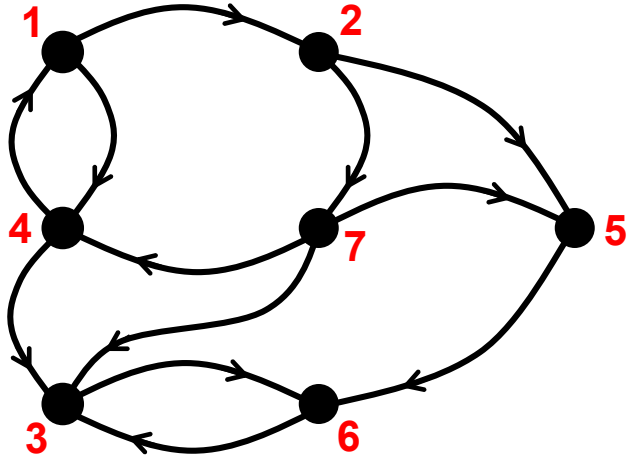
H

Graph G with n nodes is connected iff every off-diagonal entry of **$A+A^2+A^3+\cdots+A^n$ is positive, where A is the adjacency** matrix of G.

Sparse Adjacency Matrix and Graph

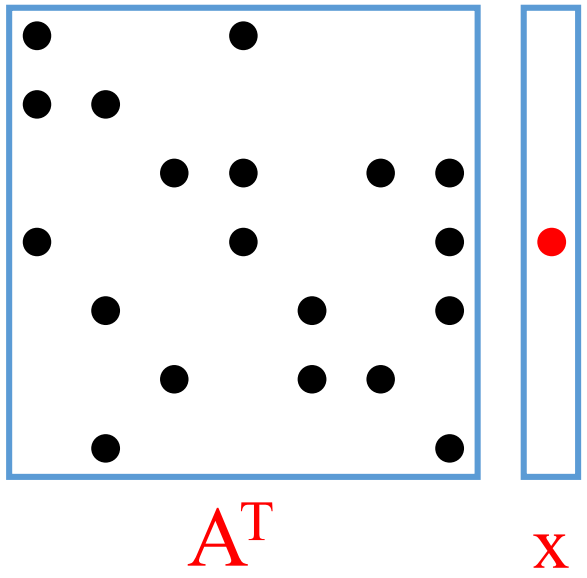


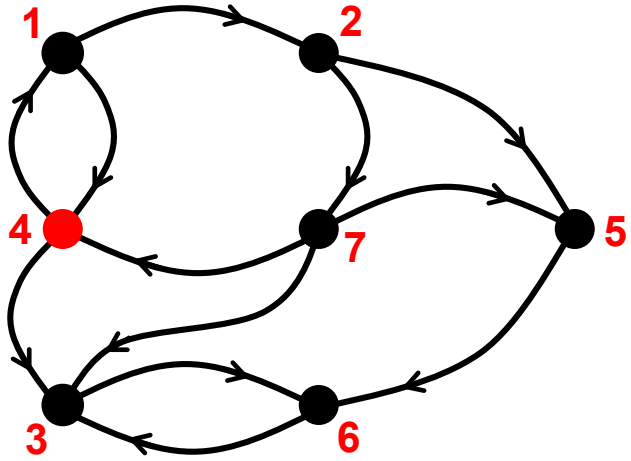
A^T



- Adjacency matrix: sparse array, few nonzeros for graph edges
- Storage-efficient implementation from sparse data structures

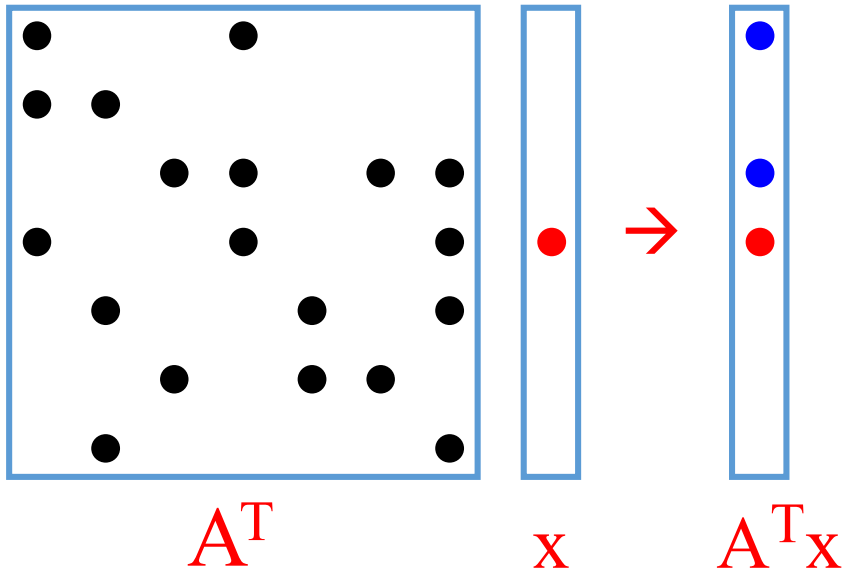
Breadth-First Search: sparse mat * vec

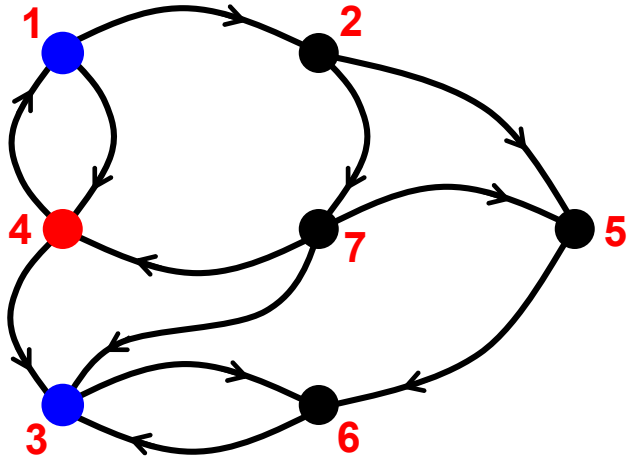




- Multiply by adjacency matrix → step to neighbor vertices
- Work-efficient implementation from sparse data structures

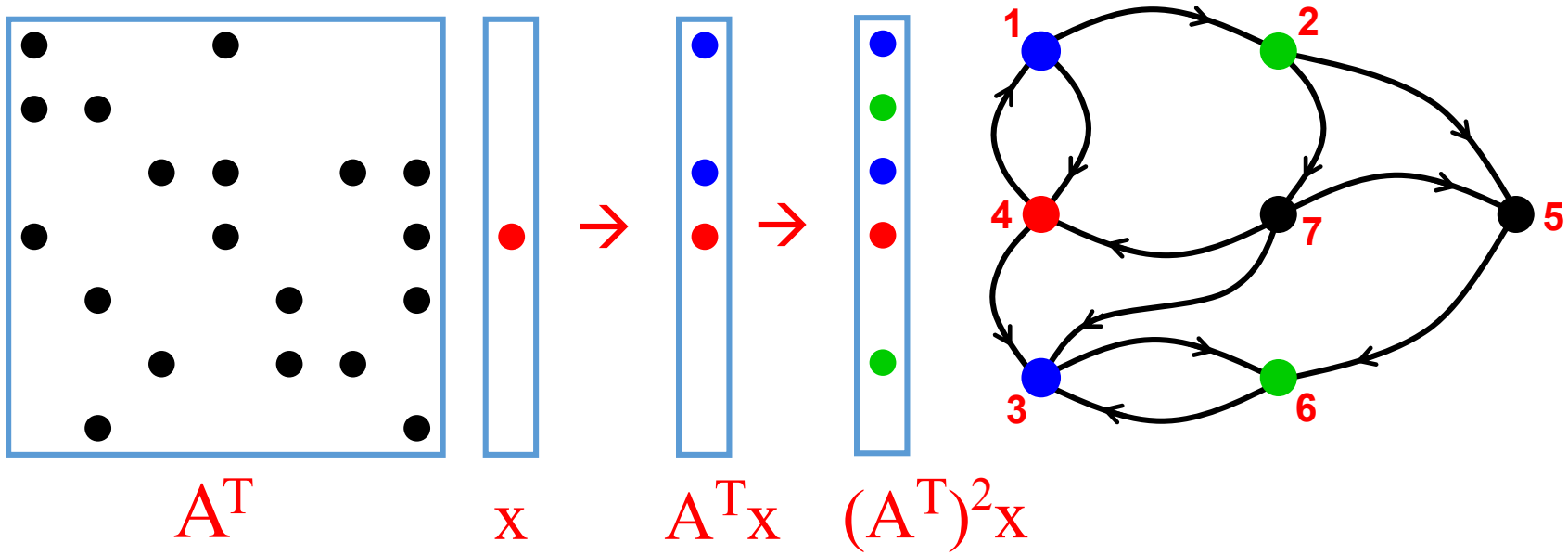
Breadth-First Search: sparse mat * vec





- Multiply by adjacency matrix → step to neighbor vertices
- Work-efficient implementation from sparse data structures

Breadth-First Search: sparse mat * vec



- Multiply by adjacency matrix → step to neighbor vertices
- Work-efficient implementation from sparse data structures