Graph Algorithms

Amrita Kaur

August 10, 2023

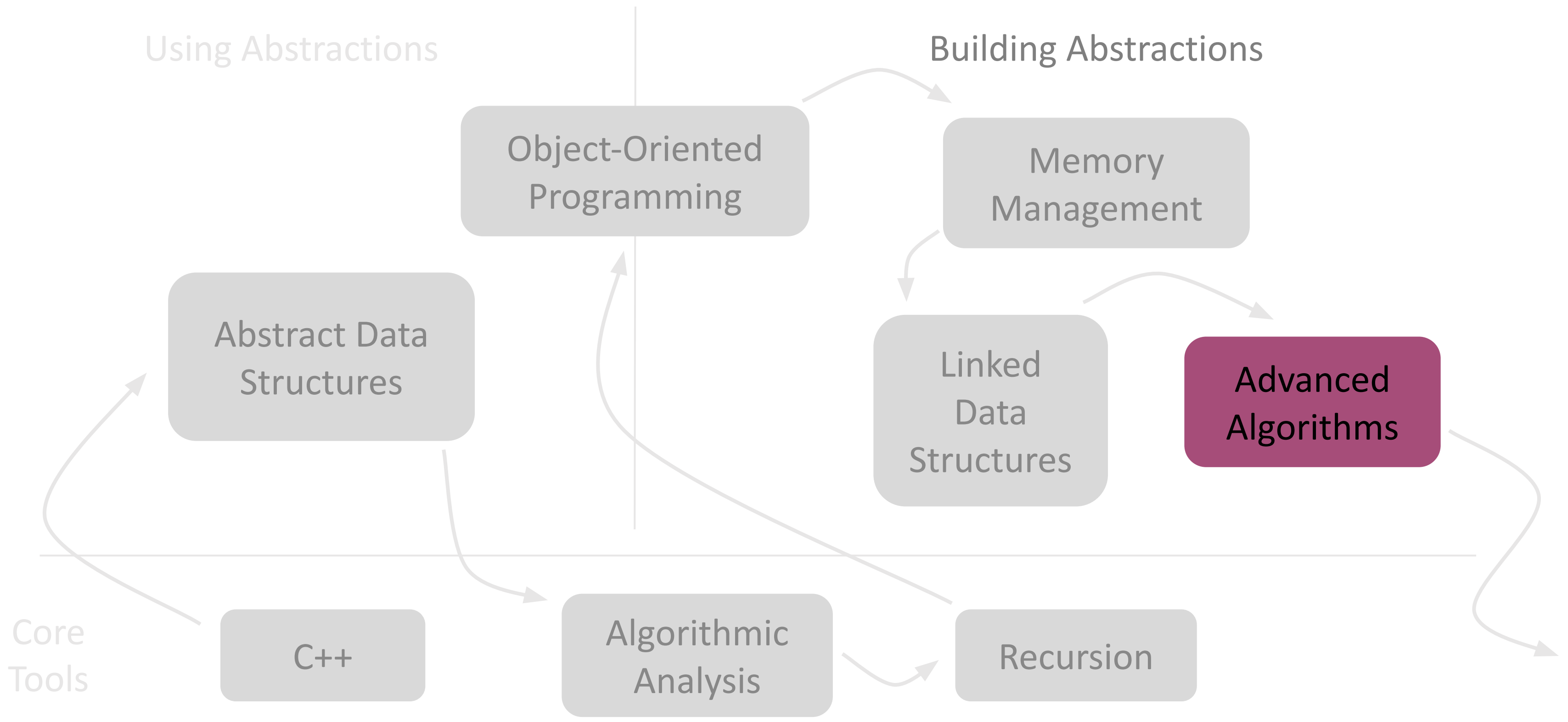
Contributions made from previous CS106B Instructors

Stanford University

Announcements

- Assignment 6 (last assignment!) has been released
  - **No late days** beyond the grace period (next Thursday 11:59pm)
  - YEAH hours on Canvas
- Assignment Retroactive Citation Form will be released tomorrow
  - Due by Friday, August 18th at 11:59pm
- No attendance tickets next week
- Slightly personal note: My last lecture for the quarter and last lecture at Stanford (ever?), so let’s have fun!

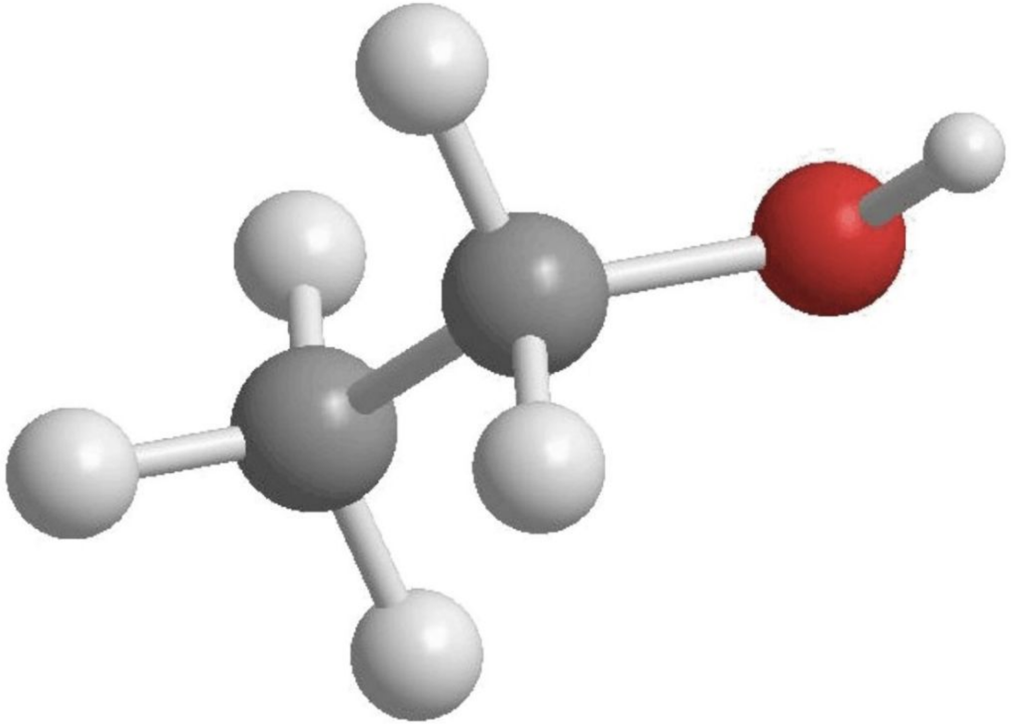
Roadmap


Using Abstractions
Building Abstractions
Object-Oriented Programming
Memory Management
Abstract Data Structures
Linked Data Structures
Advanced Algorithms
Core Tools
C++
Algorithmic Analysis
Recursion

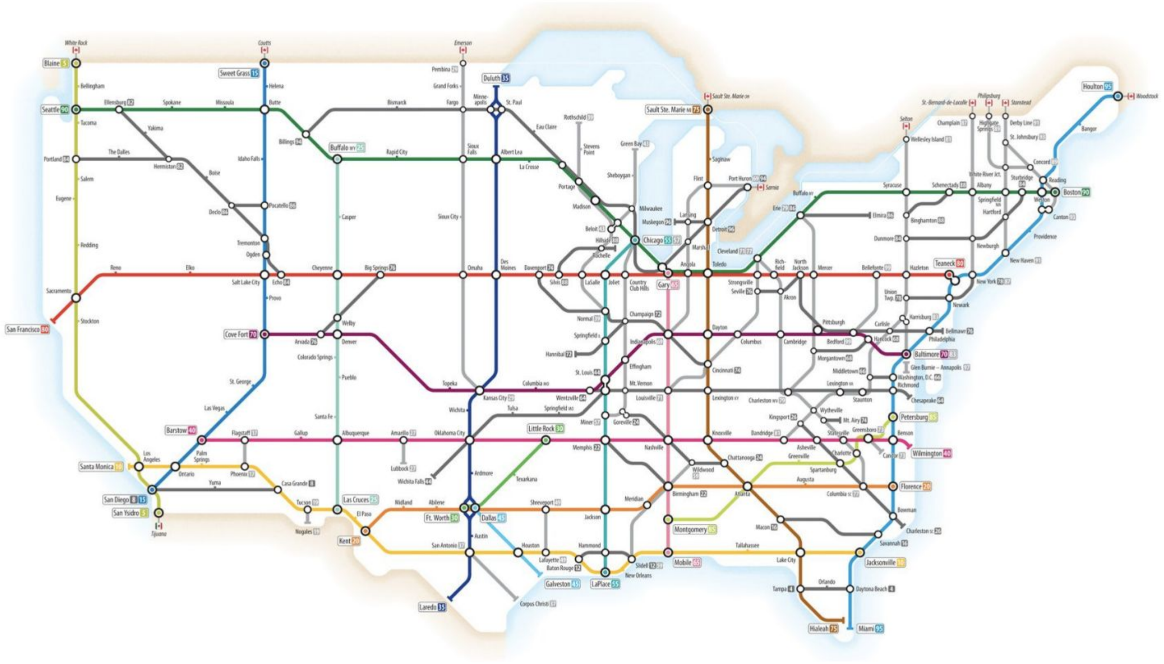
Graphs

A Social Network

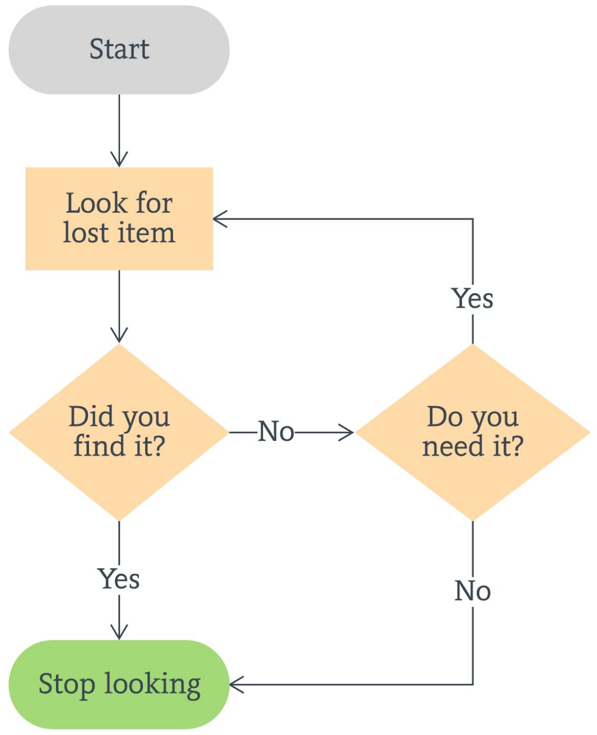
Molecules



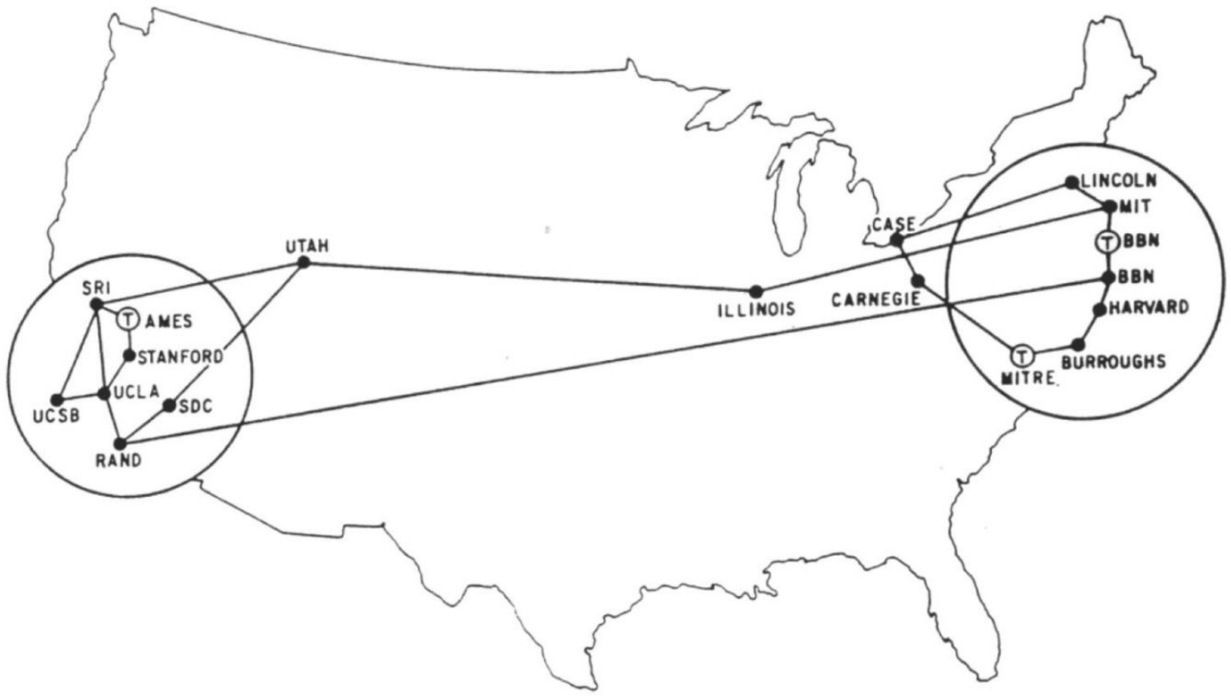
Interstate Highway System



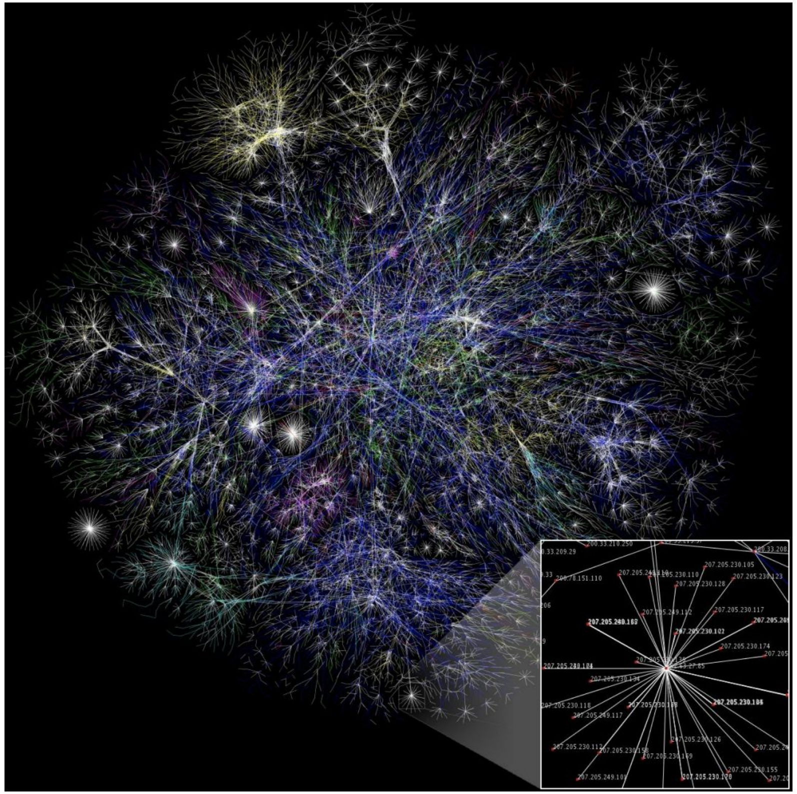
Flowcharts


Start
Look for lost item
Did you find it?
No
Do you need it?
Yes
Yes
No
Stop looking

The Internet (1971)



The Internet (2023)



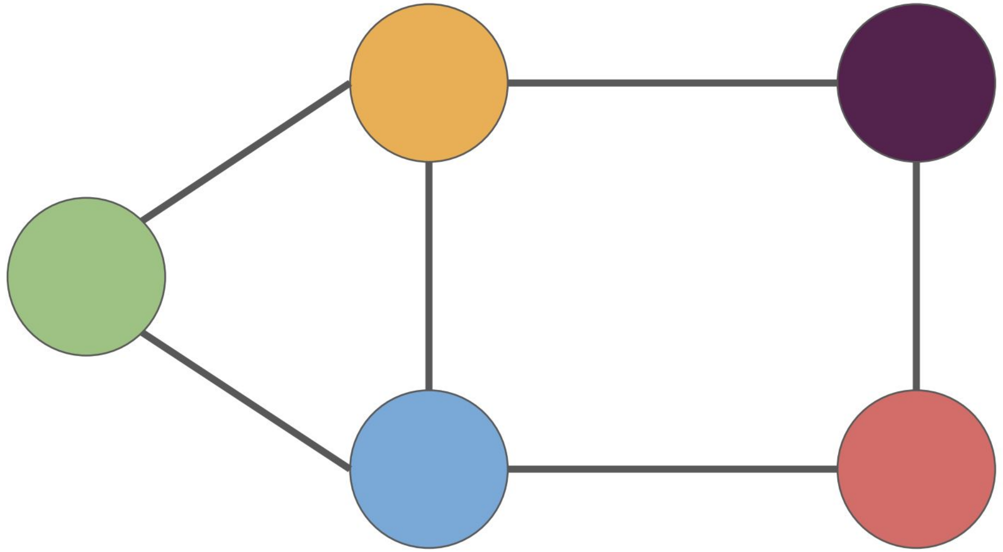
What is a graph?

graph

a structured way to represent relationships between different entities

Graph Terminology

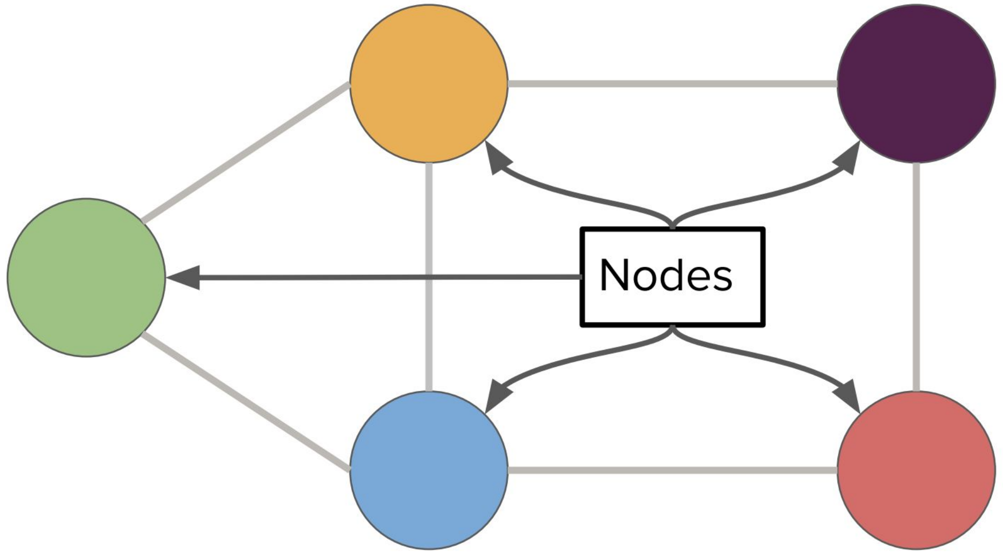
graph
a structured way to represent relationships between different entities



A graph consists of a set of **nodes** connected by **edges.**

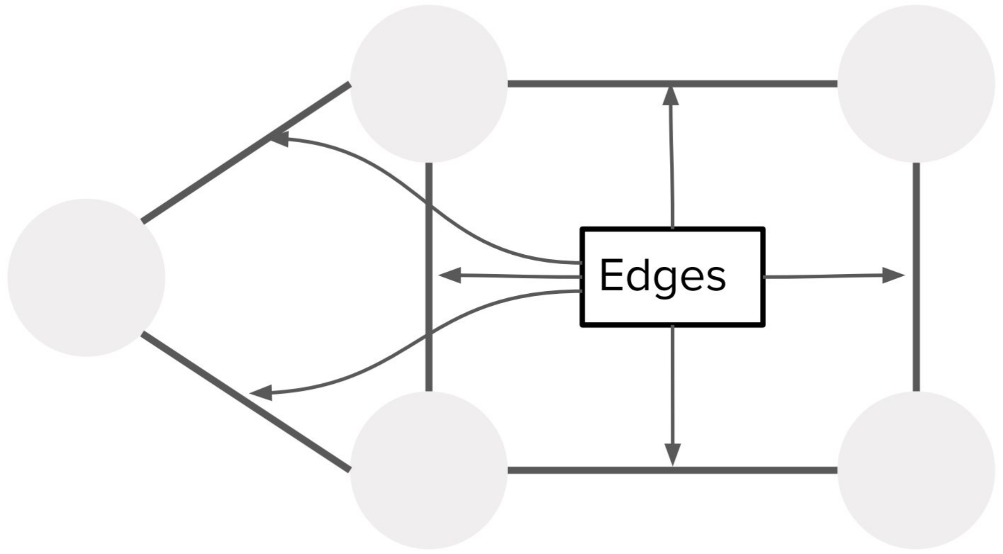
Graph Terminology

graph
a structured way to represent relationships between different **entities**



A graph consists of a set of **nodes** connected by edges.

Graph Terminology

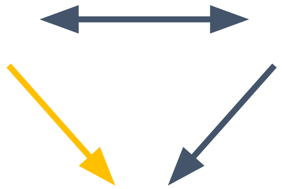


graph
a structured way to represent **relationships** between different entities

A graph consists of a set of nodes connected by **edges**.

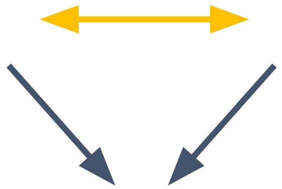
Types of Graphs

- Directed: unidirectional relationships between nodes
  - Represented by a pointed arrow
  - An action/verb that implies only one direction
  - Ex: I follow Dwayne "The Rock" Johnson on Instagram, but he doesn't follow me back



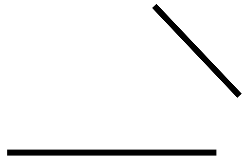
Types of Graphs

- Directed: unidirectional relationships between nodes
  - Represented by a pointed arrow
  - An action/verb that implies only one direction
  - Ex: I follow Dwayne "The Rock" Johnson on Instagram, but he doesn't follow me back



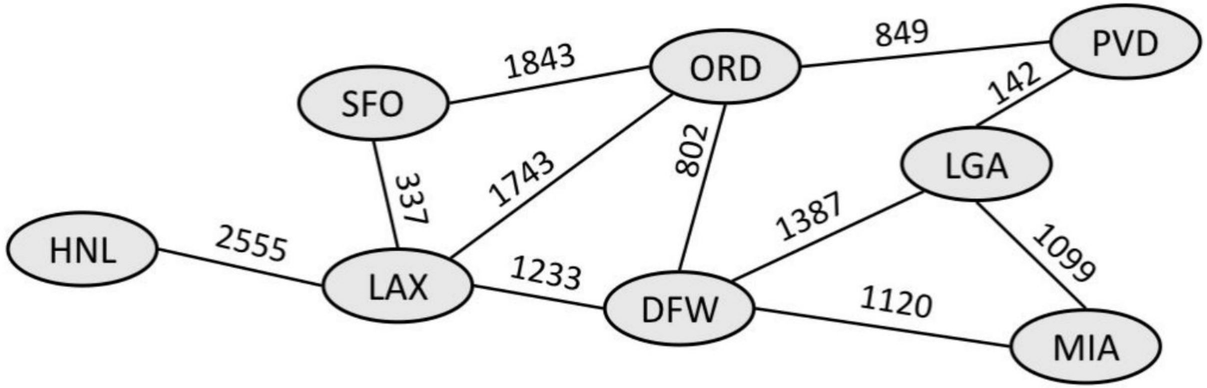
Types of Graphs

- Undirected: bidirectional relationships between nodes
  - Represented with an arrow-less line
  - An action/verb that inherently applies to both entities
  - Ex: I am related to my sister and she is related to me



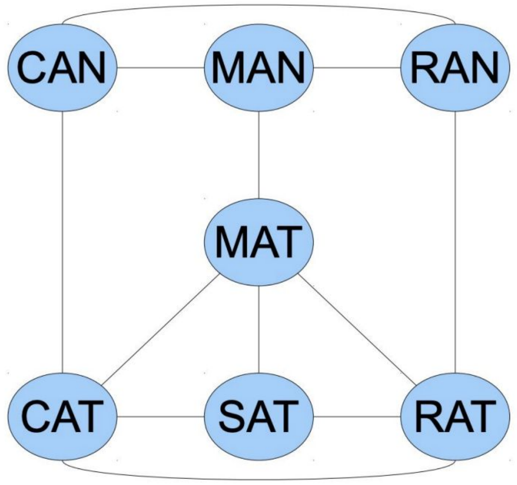
Types of Graphs

- Weighted: not all relationships between entities are equal
  - Each edge is assigned a numerical "weight" representing its relative significance/strength.
  - Ex: Different airports are different distances from each other



Types of Graphs

- Unweighted: all relationships between entities are equal
  - Each edge has equal significance and no label
  - Ex: All connected words in a word ladder are one letter apart



Social Network

Nodes: ?

Edges: ?

Directed or undirected?

Weighted or unweighted?

Social Network

Nodes: People

Edges: Friendships/Following

Directed (Instagram) or undirected (Facebook)

Unweighted

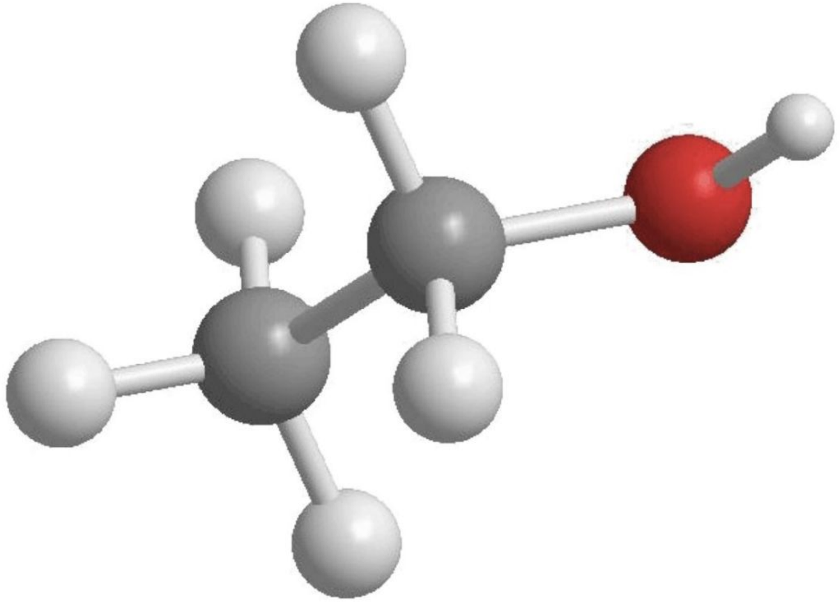
Molecules

Nodes: ?

Edges: ?

Directed or undirected?

Weighted or unweighted?



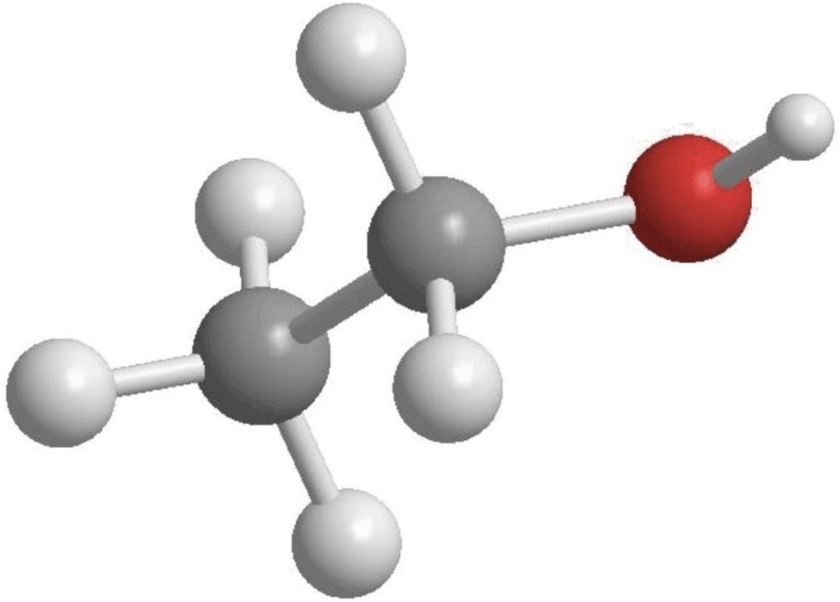
Molecules

Nodes: Atoms

Edges: Bonds

Undirected

Weighted



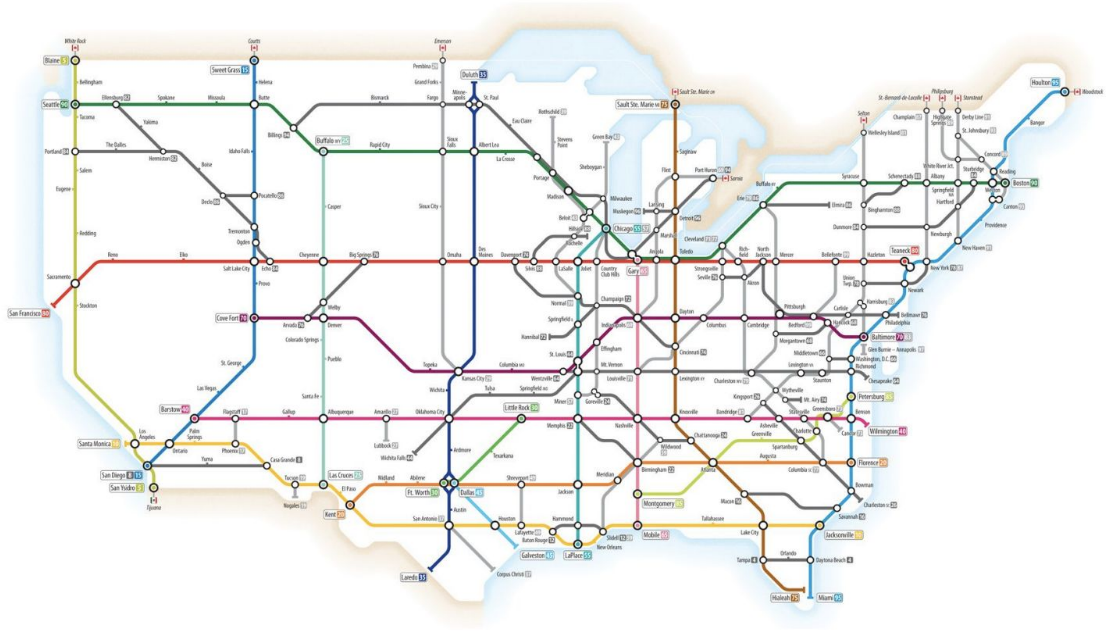
Interstate Highway System

Nodes: ?

Edges: ?

Directed or undirected?

Weighted or unweighted?



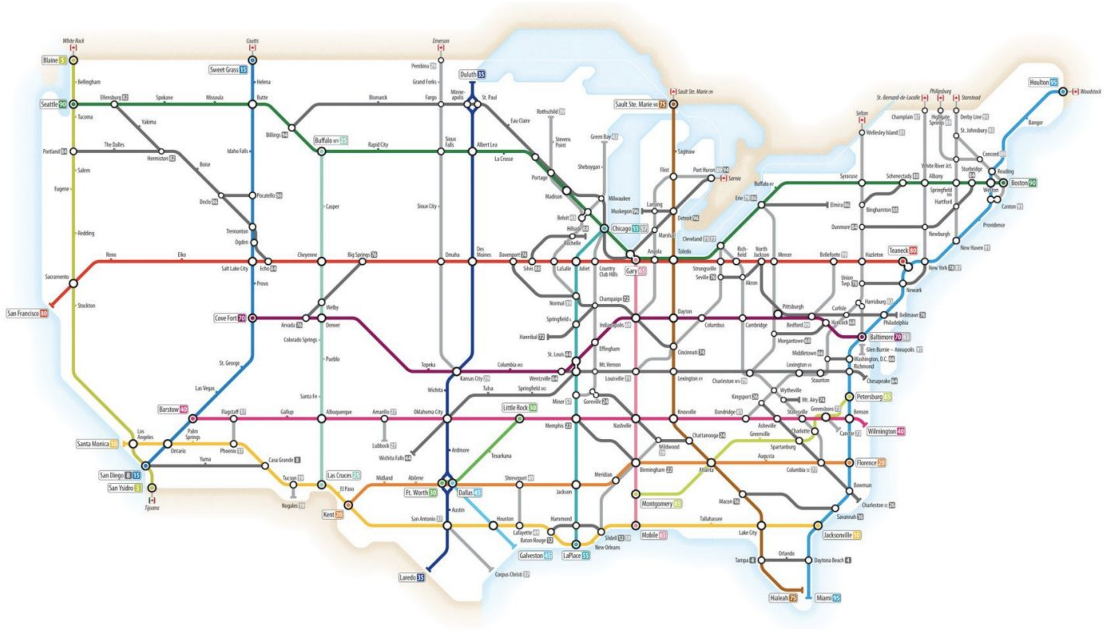
Interstate Highway System

Nodes: Cities

Edges: Roads

Undirected

Weighted



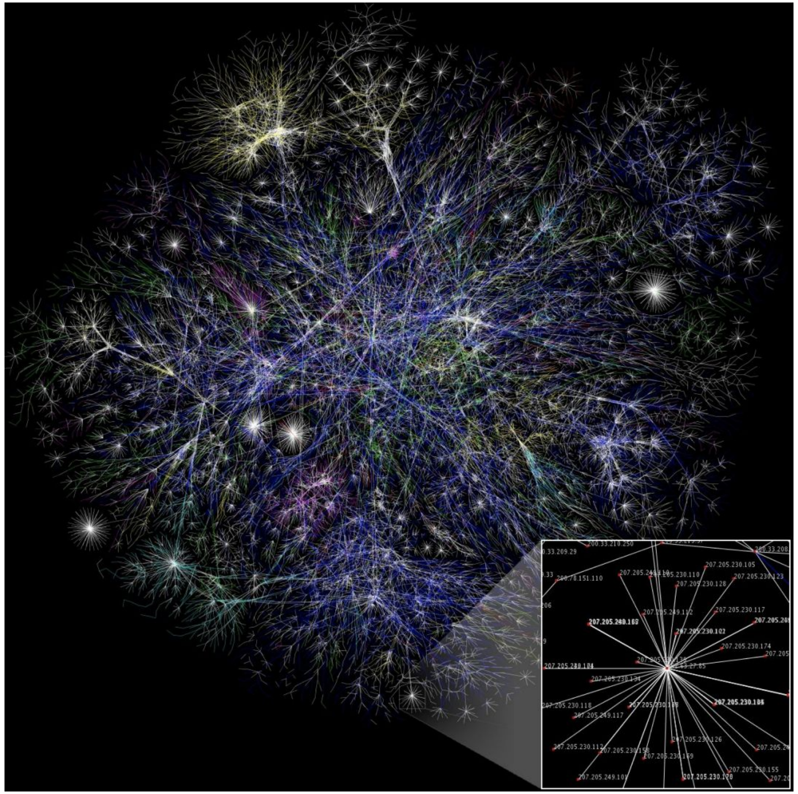
Internet

Nodes: ?

Edges: ?

Directed or undirected?

Weighted or unweighted?



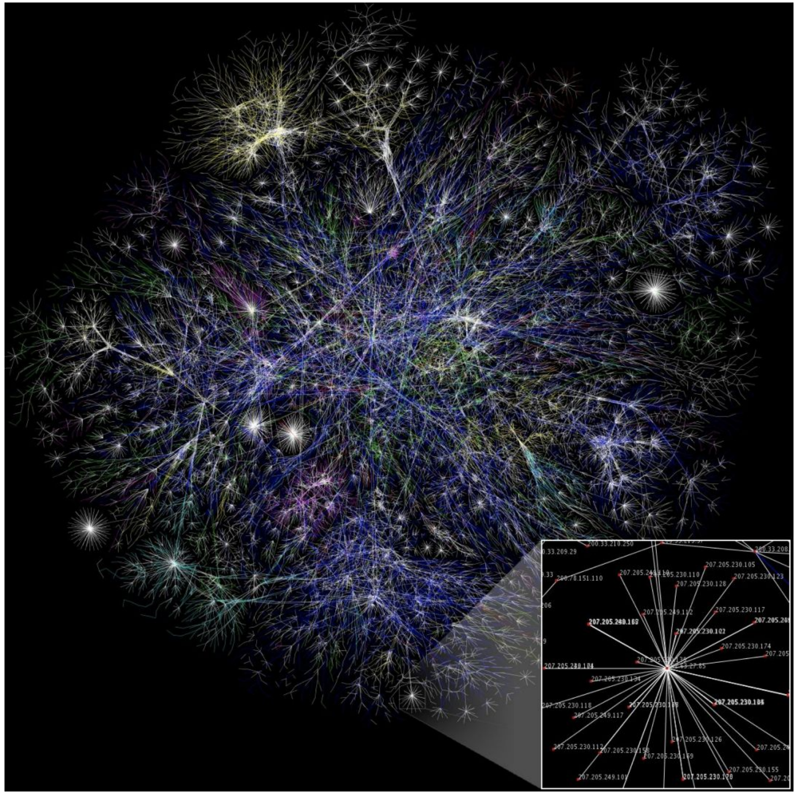
Internet

Nodes: Devices (phones, computers, etc)

Edges: Connection Pathways (Bluetooth, Wifi, Ethernet, cables)

Undirected

Unweighted

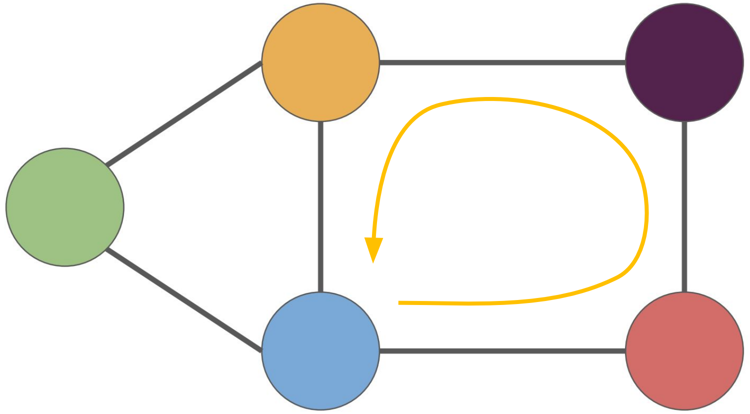


Linked Data Structures

- We've already seen nodes connected by edges before when discussing linked lists and trees
- What differentiates these linked data structures?
    - **Linked lists**: Linear structure, each node connected to at most one other node
    - **Trees**: Nodes can connect to multiple other nodes, no cycles, parent/child relationship and a single, special root node.
    - **Graphs**: No restrictions. It's the wild, wild west of the node-based world!

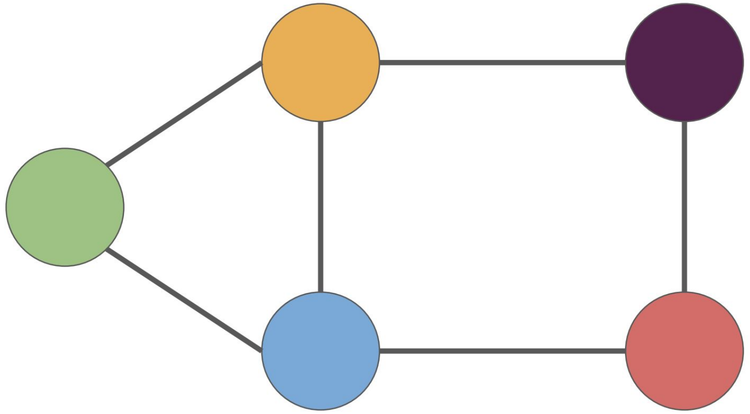
Wild World of Graphs

- Can have cycles



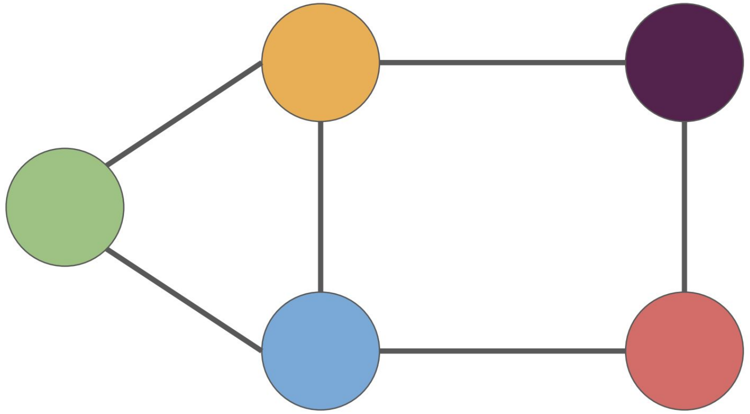
Wild World of Graphs

- Can have cycles
- No notion of a parent-child relationship between nodes



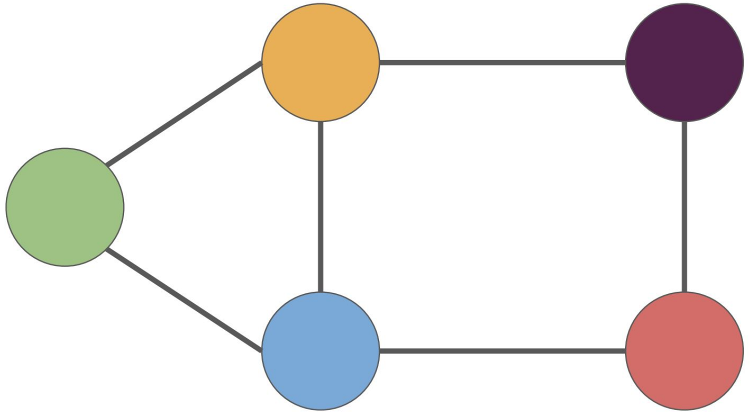
Wild World of Graphs

- Can have cycles
- No notion of a parent-child relationship between nodes
- No root node



Wild World of Graphs

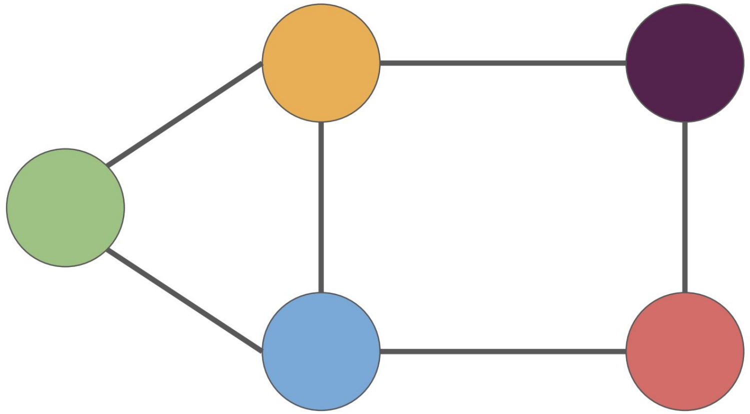
- Can have cycles
- No notion of a parent-child relationship between nodes
- No root node
- Most powerful, flexible, and expressive abstraction that we can use to model relationships between different distributed entities



Representing Graphs

Approach 1: Adjacency List

- We can represent a graph as a map from nodes to the collection of nodes that each node is adjacent to.

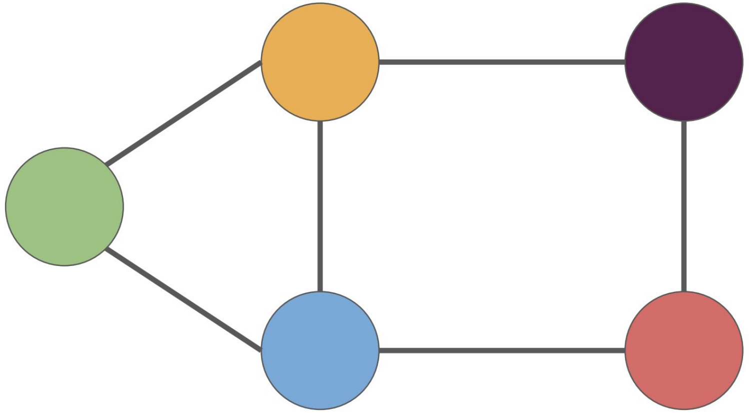


`Map<Node, Set<Node>>`

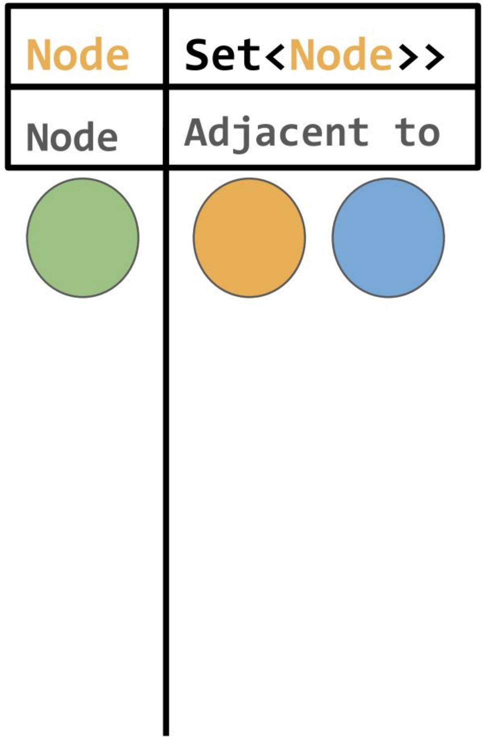
Node	Set<Node>
Node	Adjacent to

Approach 1: Adjacency List

- We can represent a graph as a map from nodes to the collection of nodes that each node is adjacent to.

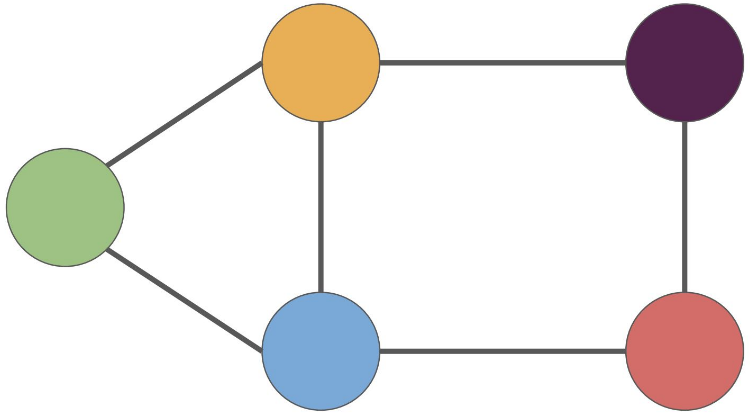


`Map<Node, Set<Node>>`

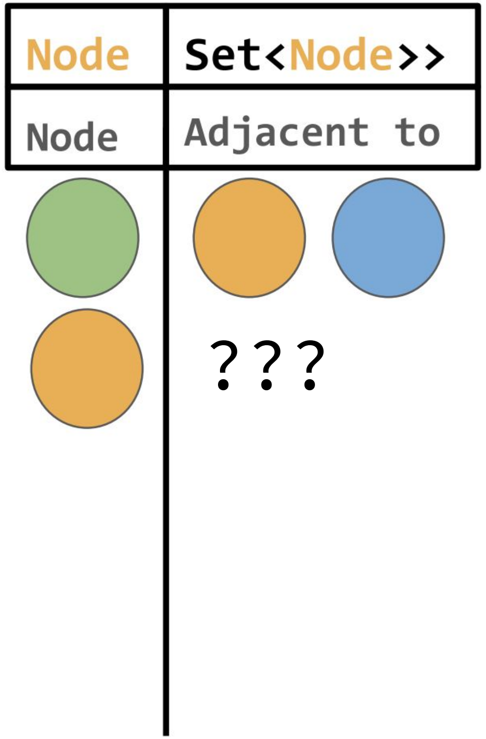


Approach 1: Adjacency List

- We can represent a graph as a map from nodes to the collection of nodes that each node is adjacent to.

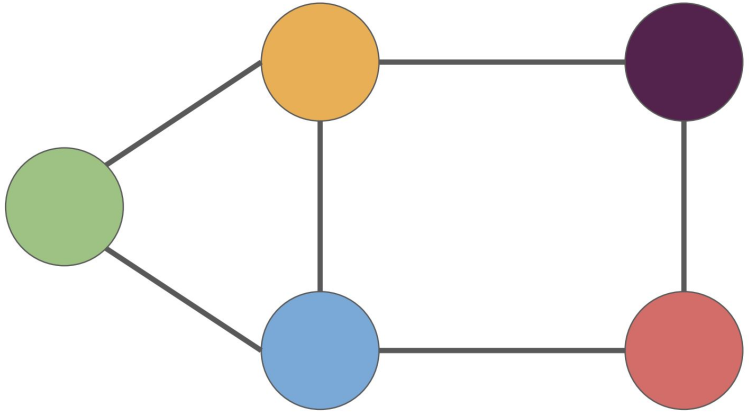


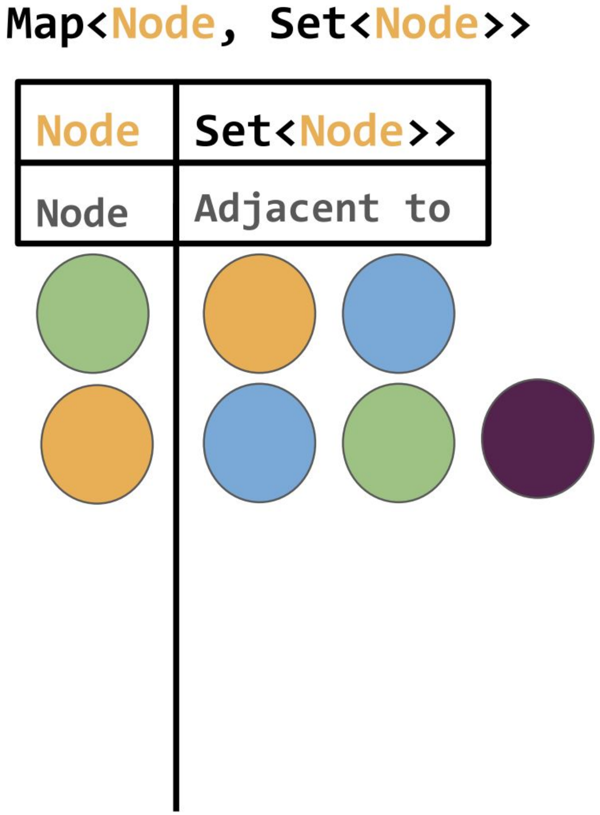
`Map<Node, Set<Node>>`



Approach 1: Adjacency List

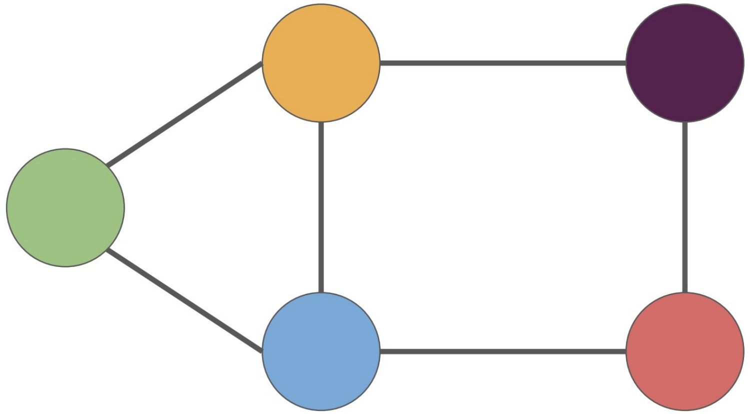
- We can represent a graph as a map from nodes to the collection of nodes that each node is adjacent to.

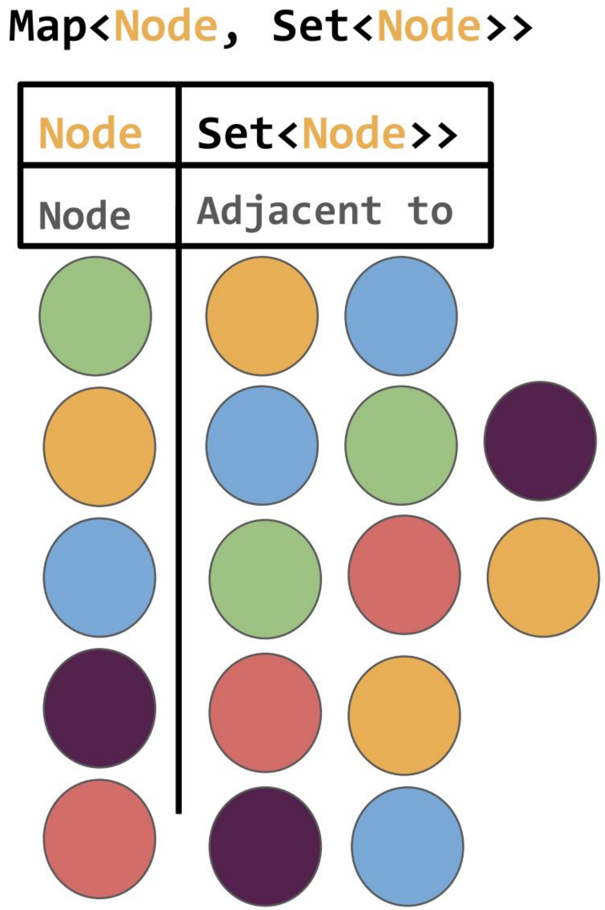




Approach 1: Adjacency List

- We can represent a graph as a map from nodes to the collection of nodes that each node is adjacent to.





Approach 1: Adjacency List

- An adjacency list can come in a number of different forms:
  - `Map<Node, Set<Node>>`
  - `Map<Node, Vector<Node>>`
  - `Vector<Vector<Node>>`

Approach 1: Adjacency List

- An adjacency list can come in a number of different forms:
  - `Map<Node, Set<Node>>`
  - `Map<Node, Vector<Node>>`
  - `Vector<Vector<Node>>`
- The core idea is that we have some kind of mapping associating each node with its outgoing edges (or neighboring nodes)

Approach 1: Adjacency List

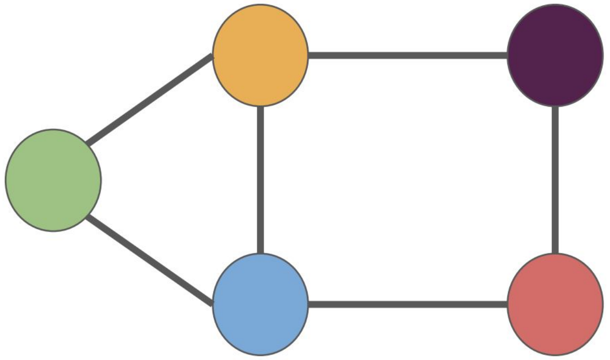
- An adjacency list can come in a number of different forms:
  - `Map<Node, Set<Node>>`
  - `Map<Node, Vector<Node>>`
  - `Vector<Vector<Node>>`
- The core idea is that we have some kind of mapping associating each node with its outgoing edges (or neighboring nodes)
- How might you incorporate weights?

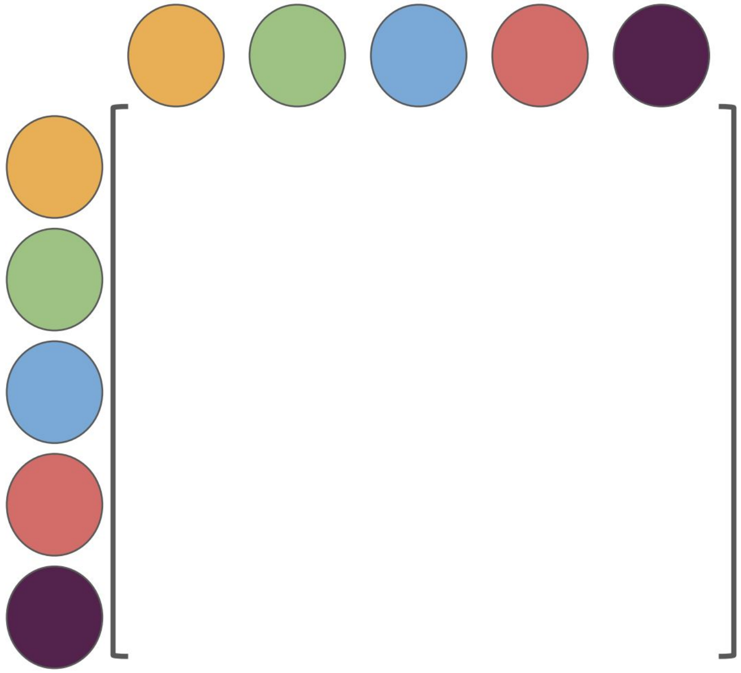
Approach 1: Adjacency List

- An adjacency list can come in a number of different forms:
  - `Map<Node, Set<`**`Edge`**`>>`
  - `Map<Node, Vector<`**`Edge`**`>>`
  - `Vector<Vector<`**`Edge`**`>>`
- The core idea is that we have some kind of mapping associating each node with its outgoing edges (or neighboring nodes)
- How might you incorporate weights?
  - **Create an Edge struct that holds both a Node and a weight!**

Approach 2: Adjacency Matrix

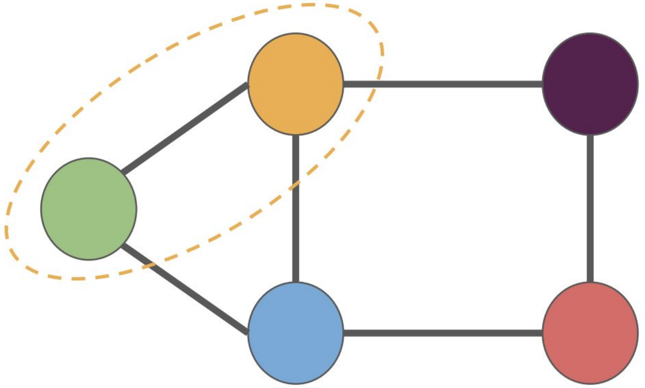
- We can also use a two-dimensional matrix to represent the relationships in a graph.

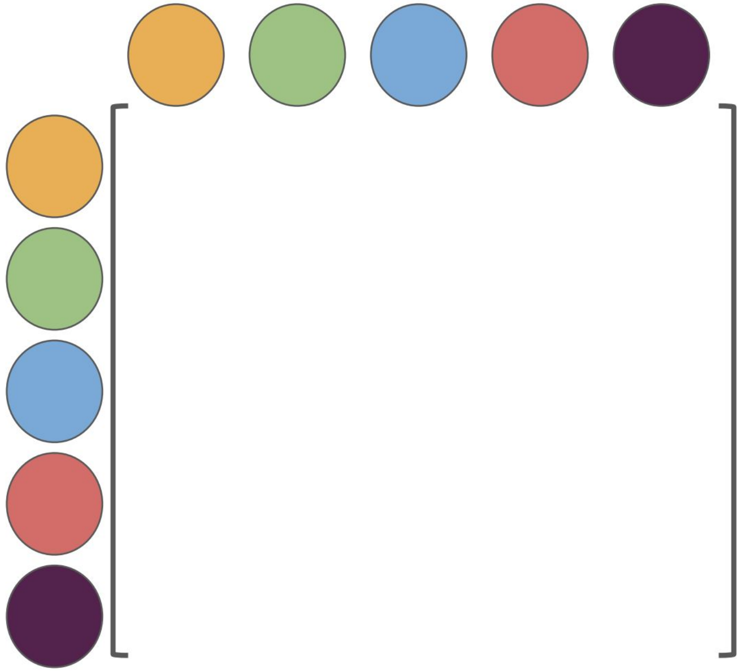




Approach 2: Adjacency Matrix

- We can also use a two-dimensional matrix to represent the relationships in a graph.

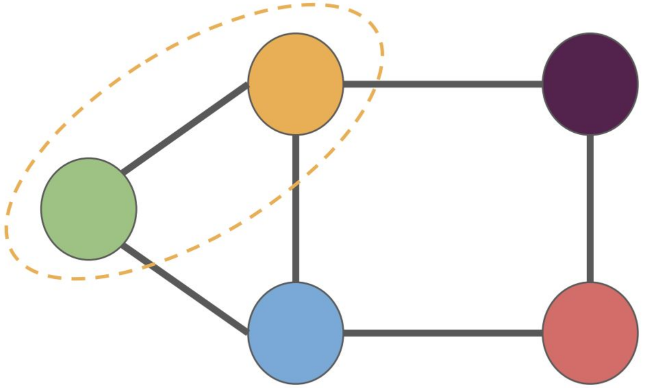


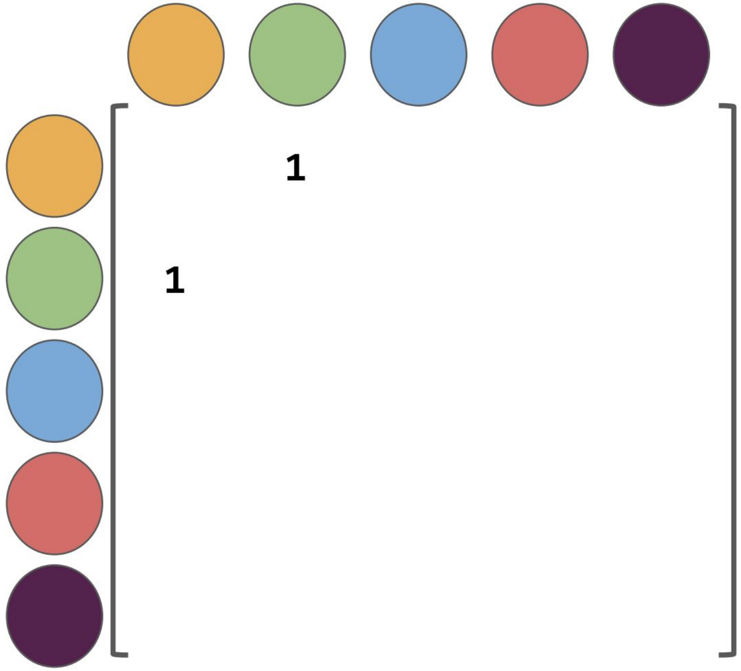


Stanford University

Approach 2: Adjacency Matrix

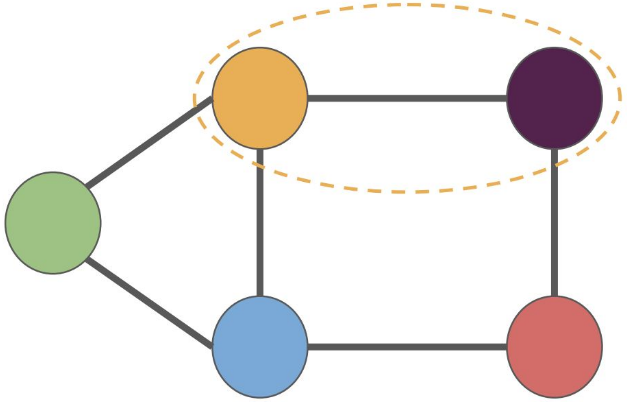
- We can also use a two-dimensional matrix to represent the relationships in a graph.

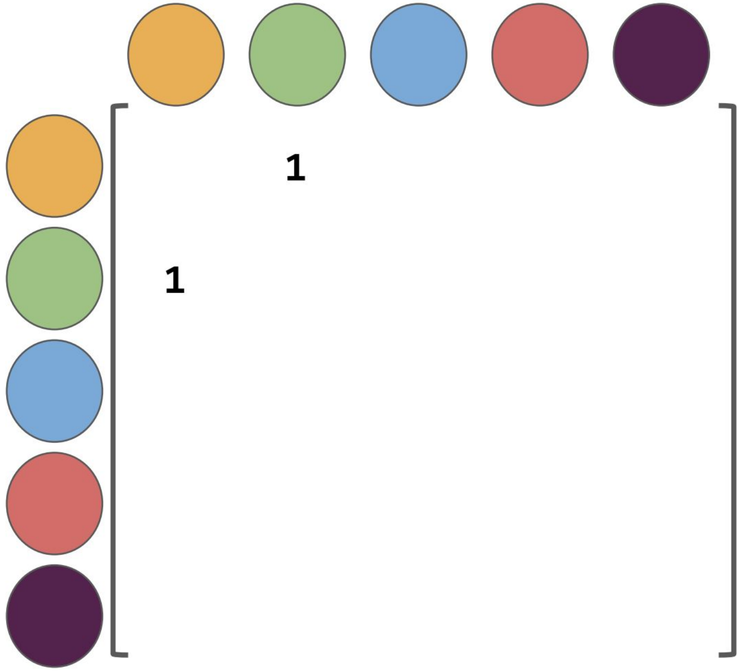




Approach 2: Adjacency Matrix

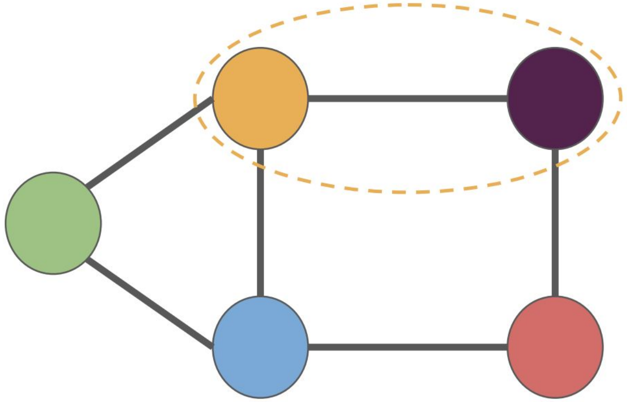
- We can also use a two-dimensional matrix to represent the relationships in a graph.





Approach 2: Adjacency Matrix

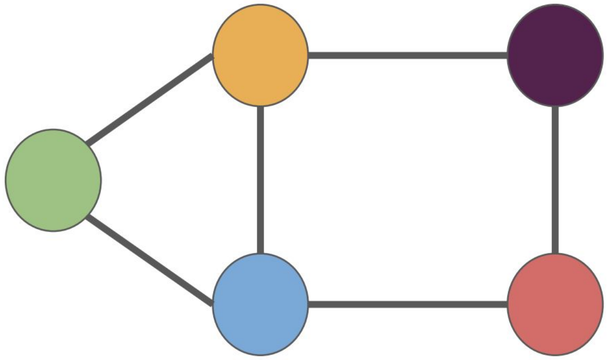
- We can also use a two-dimensional matrix to represent the relationships in a graph.



$$\begin{array}{c|ccccc} & 🟠 & 🟢 & 🔵 & 🔴 & 🟣 \\ \hline 🟠 & & 1 & & & 1 \\ 🟢 & 1 & & & & \\ 🔵 & & & & & \\ 🔴 & & & & & \\ 🟣 & 1 & & & & \end{array}$$

Approach 2: Adjacency Matrix

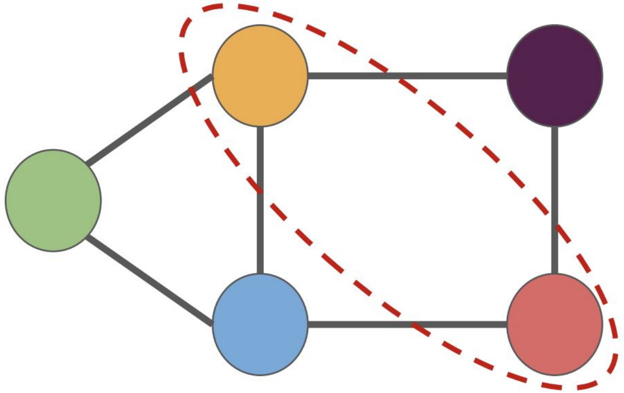
- We can also use a two-dimensional matrix to represent the relationships in a graph.



$$
\begin{array}{c|ccccc}
 & 🟠 & 🟢 & 🔵 & 🔴 & 🟣 \\
\hline
🟠 & & 1 & 1 & & 1 \\
🟢 & 1 & & 1 & & \\
🔵 & 1 & 1 & & 1 & \\
🔴 & & & 1 & & 1 \\
🟣 & 1 & & & 1 & \\
\end{array}
$$

Approach 2: Adjacency Matrix

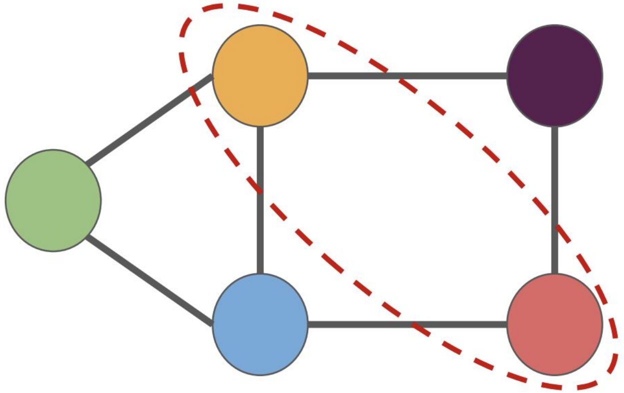
- We can also use a two-dimensional matrix to represent the relationships in a graph.



	🟠	🟢	🔵	🔴	🟣
🟠		1	1		1
🟢	1		1		
🔵	1	1		1	
🔴			1		1
🟣	1			1	

Approach 2: Adjacency Matrix

- We can also use a two-dimensional matrix to represent the relationships in a graph.



	🟠	🟢	🔵	🔴	🟣
🟠		1	1	0	1
🟢	1		1		
🔵	1	1		1	
🔴	0		1		1
🟣	1			1	

Approach 2: Adjacency Matrix

- Adjacency matrices are beneficial when our graph isn’t sparse, i.e. there aren’t a lot of 0s
    - Otherwise, storing a mostly-0s matrix is not space efficient

Approach 2: Adjacency Matrix

- Adjacency matrices are beneficial when our graph isn’t sparse, i.e. there aren’t a lot of 0s
  - Otherwise, storing a mostly-0s matrix is not space efficient
- Other benefits:
  - Grid lookup is super fast!
  - Computer hardware has been optimized for matrix math - so using a grid can help us perform complex matrix operations for data analysis

Approach 2: Adjacency Matrix

- Adjacency matrices are beneficial when our graph isn't sparse, i.e. there aren't a lot of 0s
  - Otherwise, storing a mostly-0s matrix is not space efficient
- Other benefits:
  - Grid lookup is super fast!
  - Computer hardware has been optimized for matrix math - so using a grid can help us perform complex matrix operations for data analysis
- How might you incorporate weights?

Approach 2: Adjacency Matrix

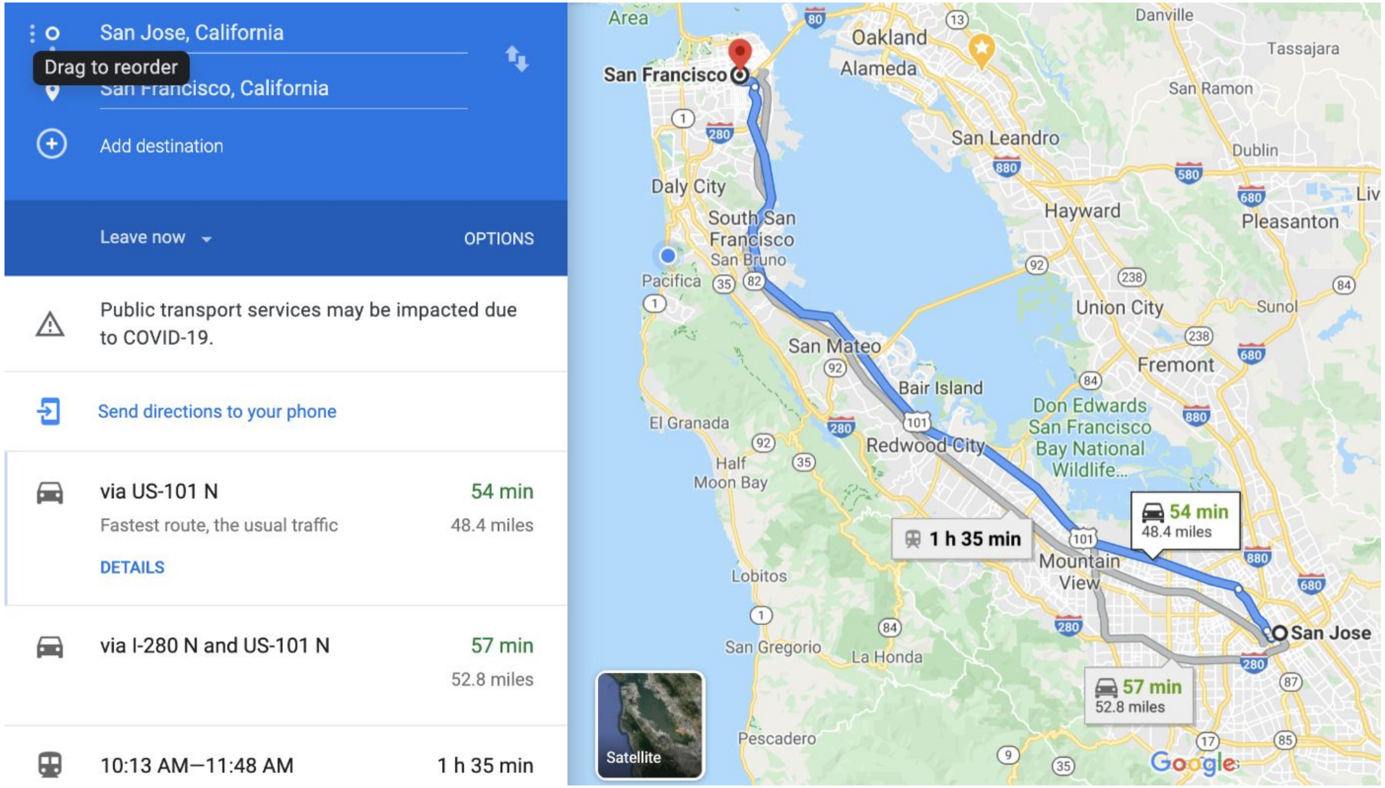
- Adjacency matrices are beneficial when our graph isn’t sparse, i.e. there aren’t a lot of 0s
  - Otherwise, storing a mostly-0s matrix is not space efficient
- Other benefits:
  - Grid lookup is super fast!
  - Computer hardware has been optimized for matrix math - so using a grid can help us perform complex matrix operations for data analysis
- How might you incorporate weights?
  - **Store other numbers besides 1 in the matrix**

Approach 2: Adjacency Matrix

- Adjacency matrices are beneficial when our graph isn't sparse, i.e. there aren't a lot of 0s
  - Otherwise, storing a mostly-0s matrix is not space efficient
- Other benefits:
  - Grid lookup is super fast!
  - Computer hardware has been optimized for matrix math - so using a grid can help us perform complex matrix operations for data analysis
  - **Storing weights is more straightforward than in the adjacency list**
- How might you incorporate weights?
  - **Store other numbers besides 1 in the matrix**

Graph Algorithms

Motivation



San Jose, California

Drag to reorder

San Francisco, California

Add destination

Leave now

OPTIONS

Public transport services may be impacted due to COVID-19.

Send directions to your phone

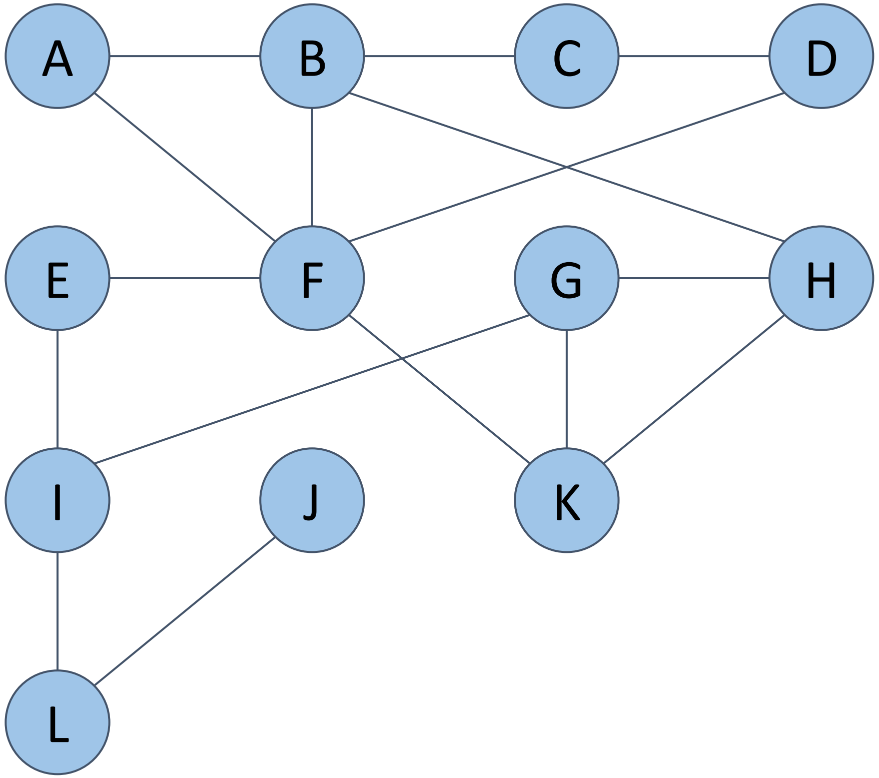
via US-101 N — 54 min
Fastest route, the usual traffic — 48.4 miles
DETAILS

via I-280 N and US-101 N — 57 min
52.8 miles

10:13 AM—11:48 AM — 1 h 35 min

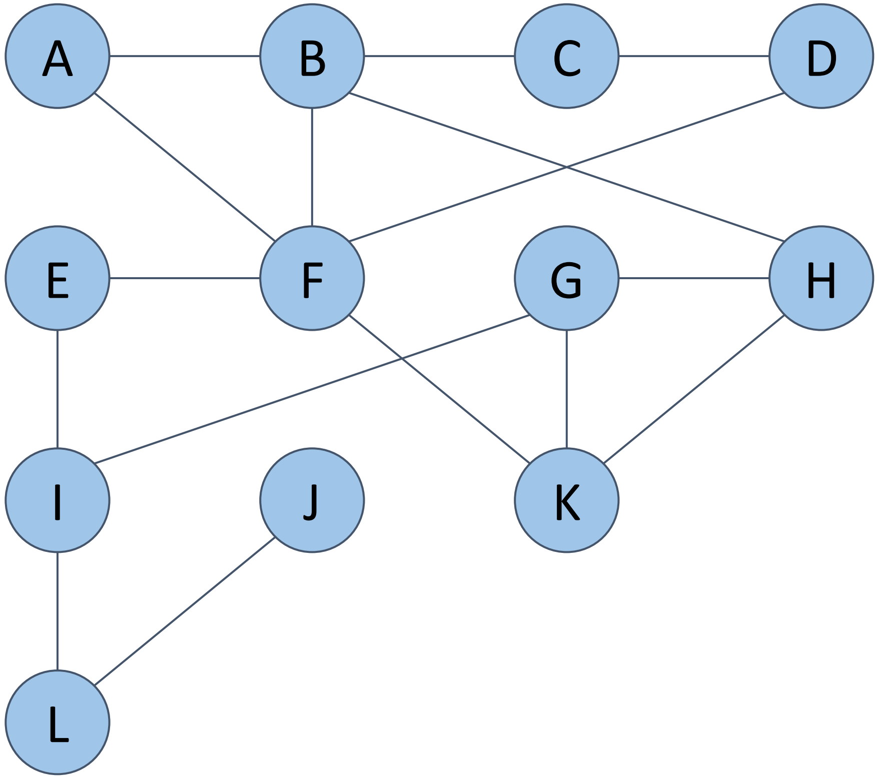
Depth-First Search

Depth-First Search



Use DFS to find a path between **F** and **G**

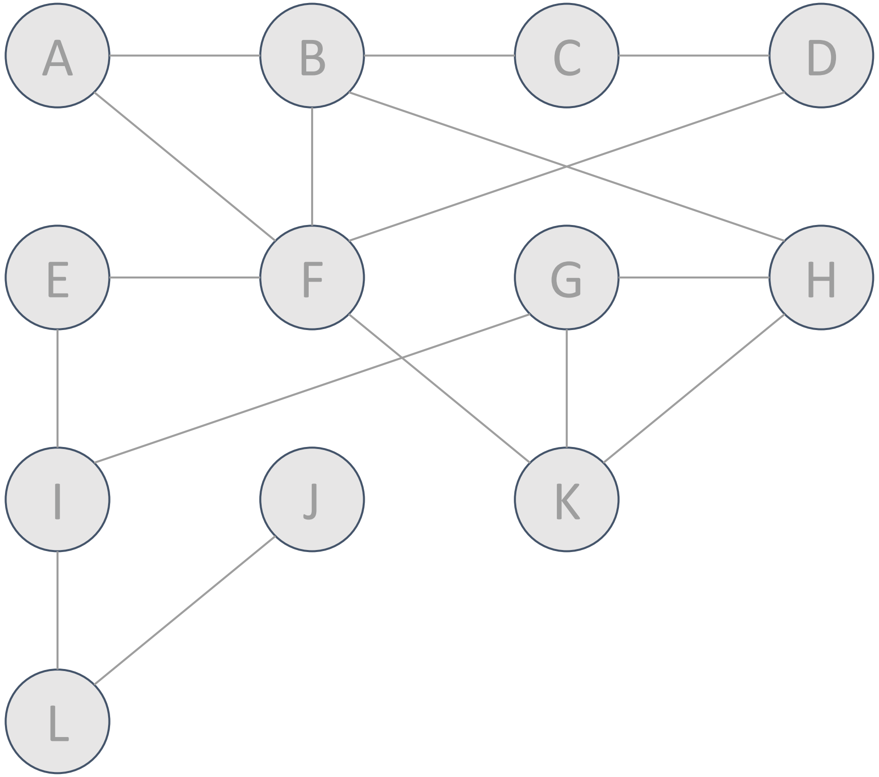
Depth-First Search



TO START:

1. Mark all nodes as unvisited

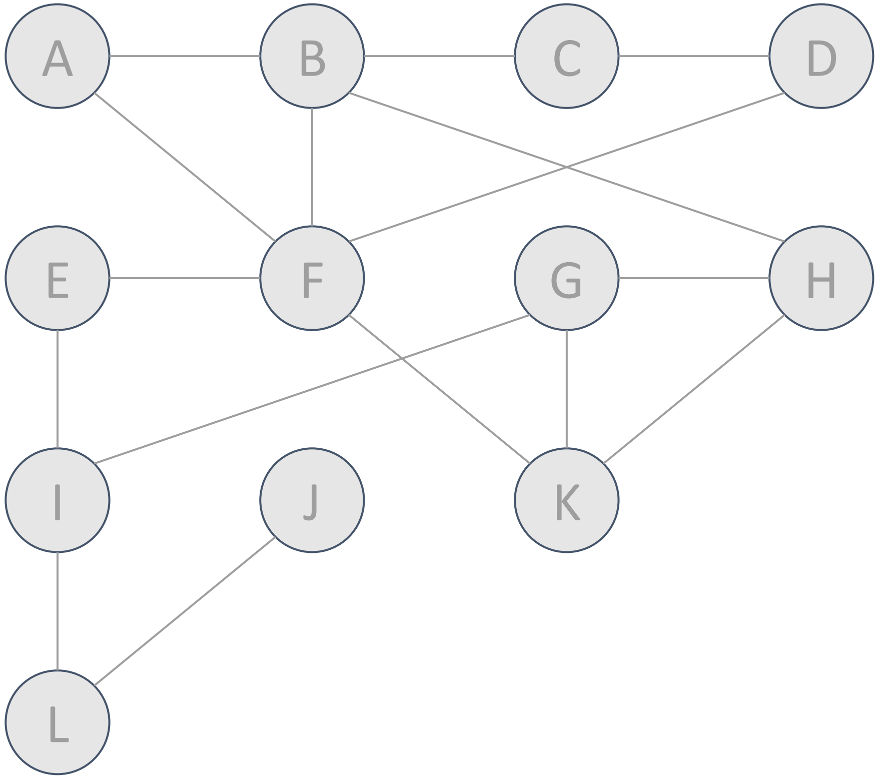
Depth-First Search



TO START:

1. Mark all nodes as unvisited

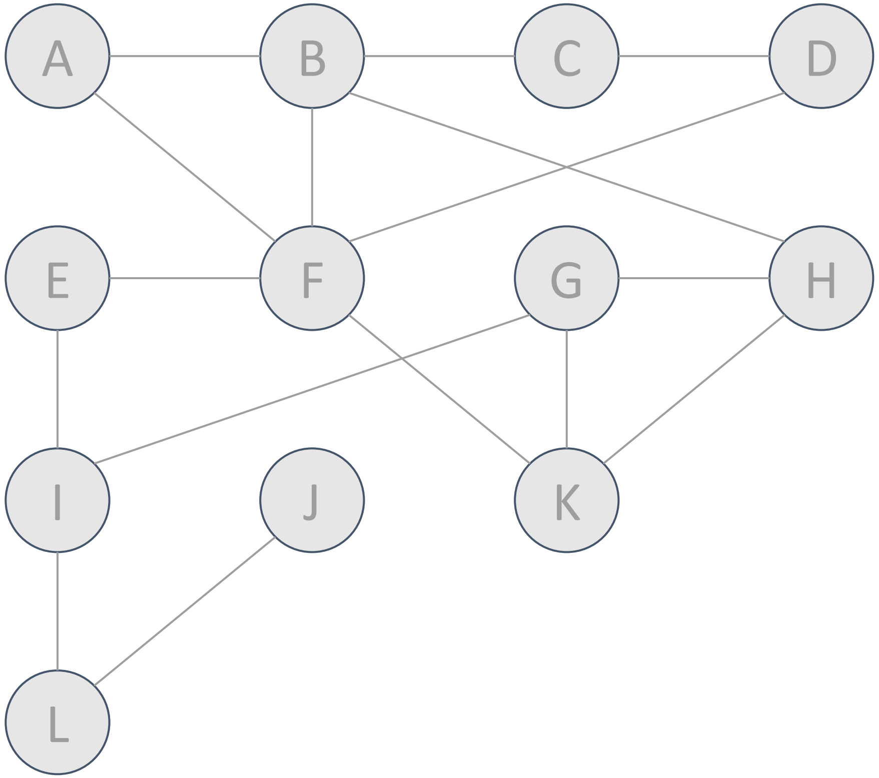
Depth-First Search



TO START:

1. Mark all nodes as unvisited
2. Make an empty stack

Depth-First Search

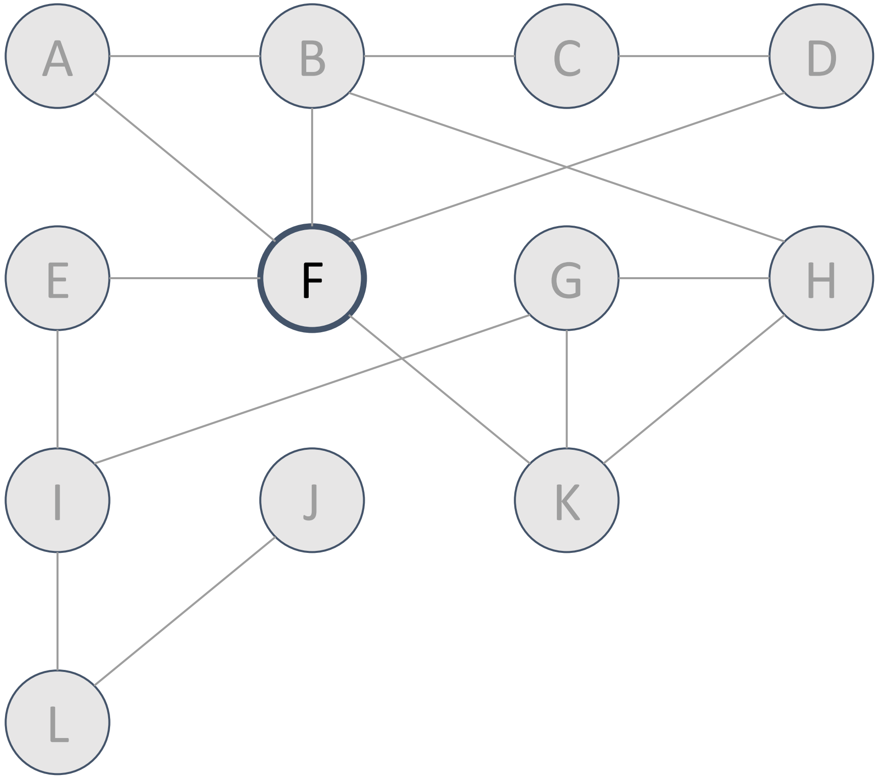


TO START:

1. Mark all nodes as unvisited
2. Make an empty stack

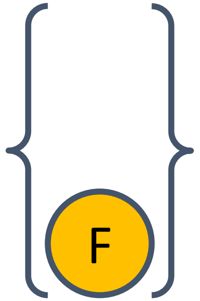
$\{ \quad \}$

Depth-First Search

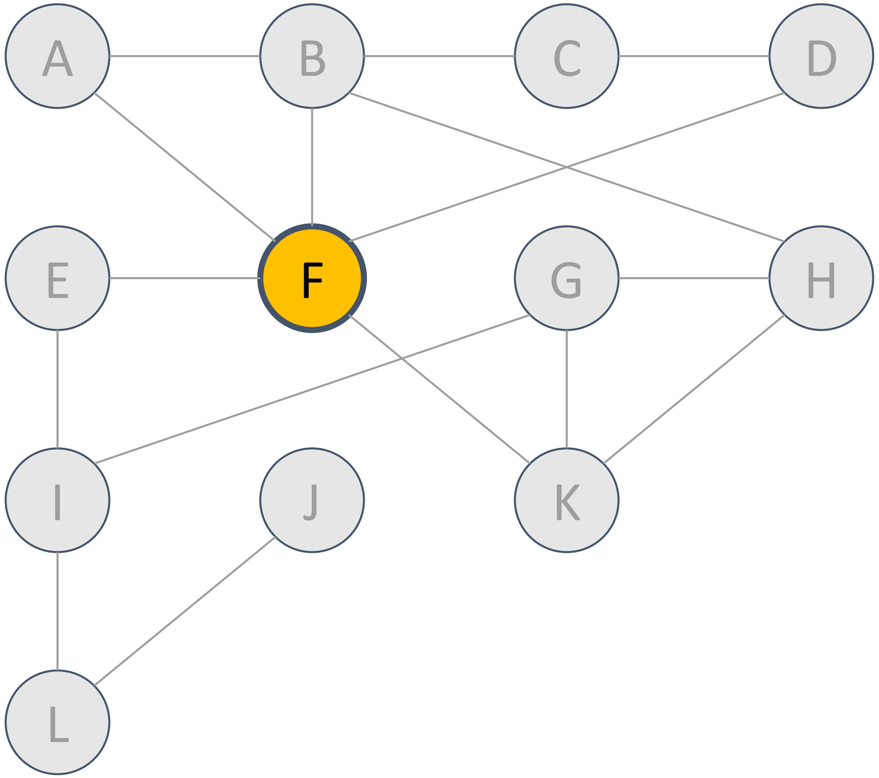


TO START:

1. Mark all nodes as unvisited
2. Make an empty stack
3. Push the desired start node and mark it as visited

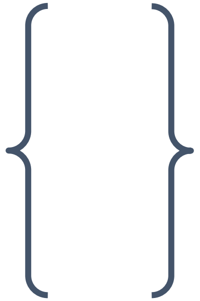


Depth-First Search

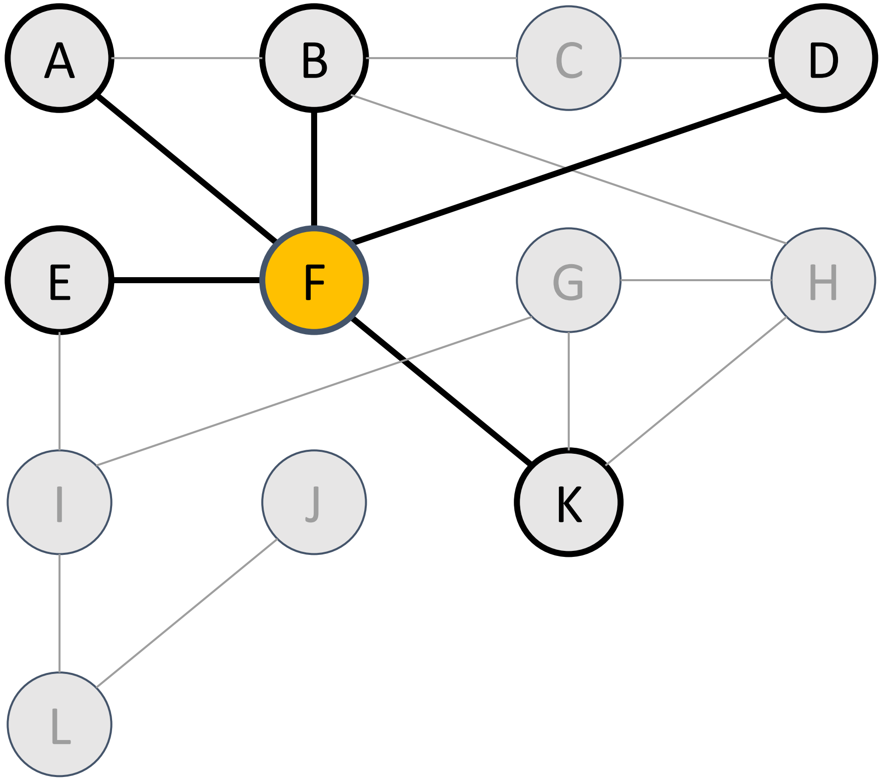


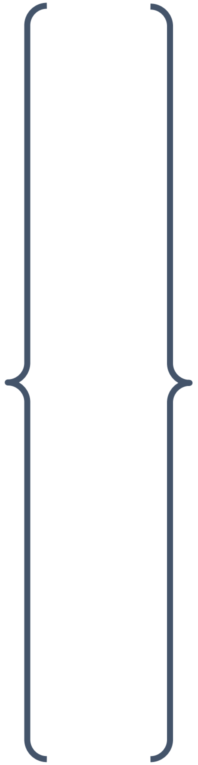
LOOP PROCEDURE:

1. Pop a node
2. For each adjacent node, if that node has never been pushed, then push

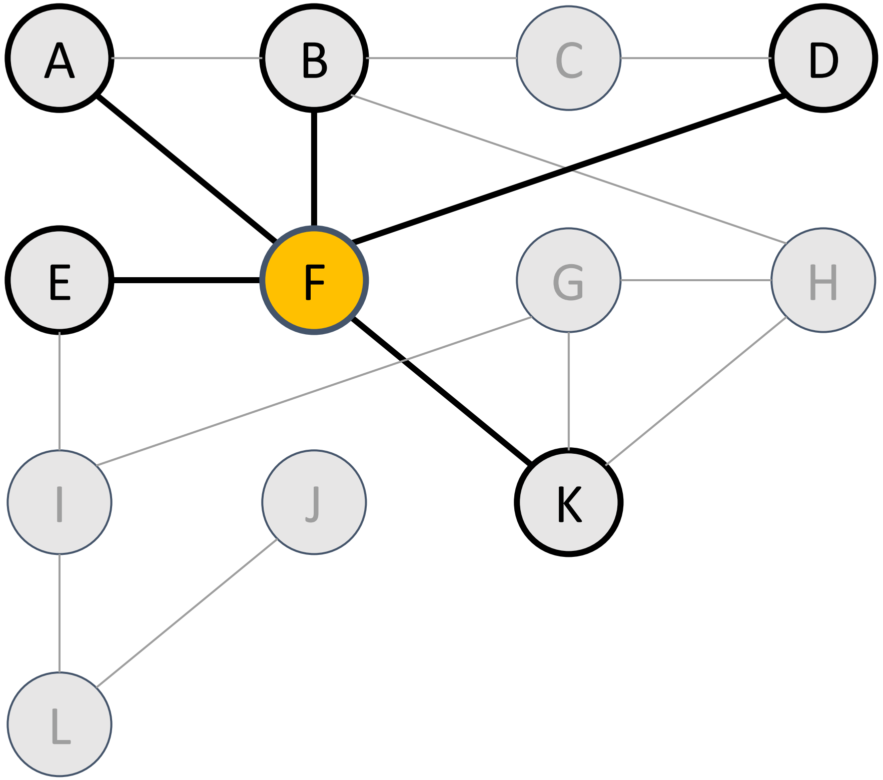


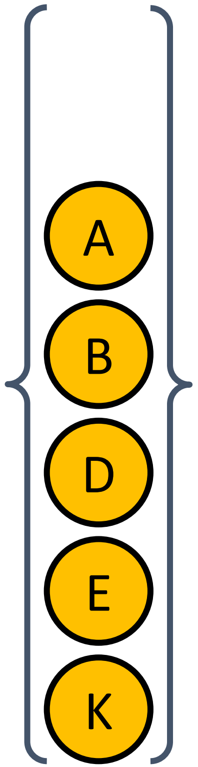
Depth-First Search



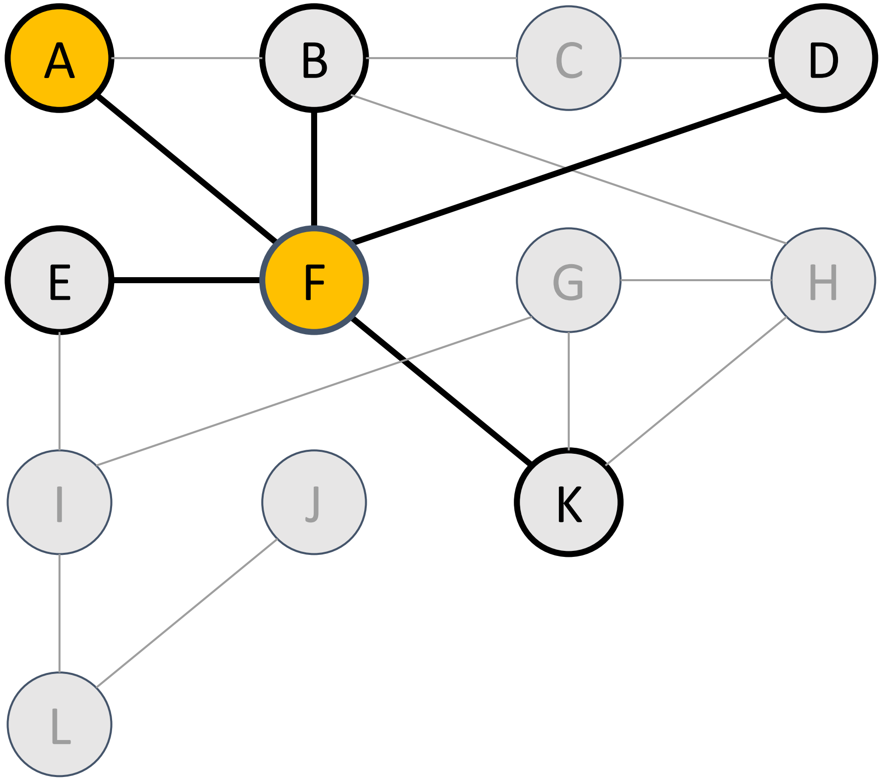


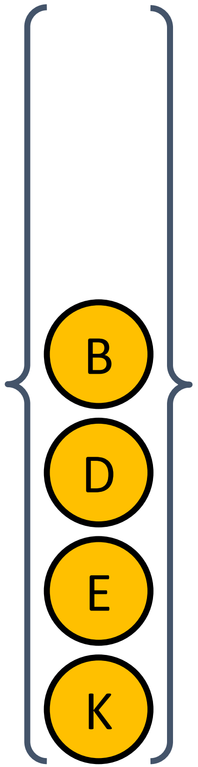
Depth-First Search



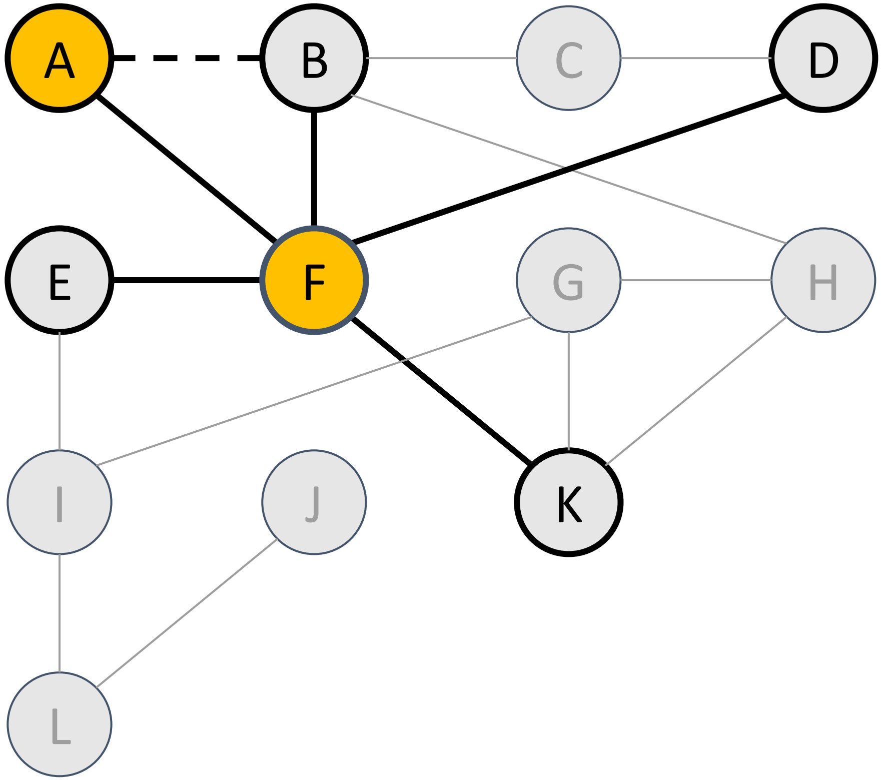


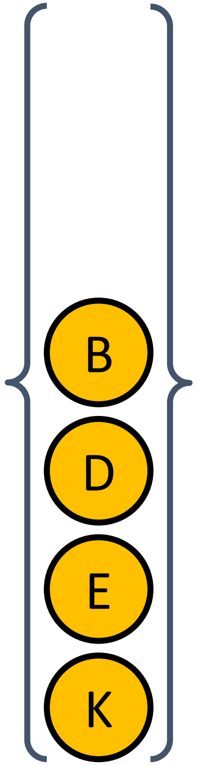
Depth-First Search



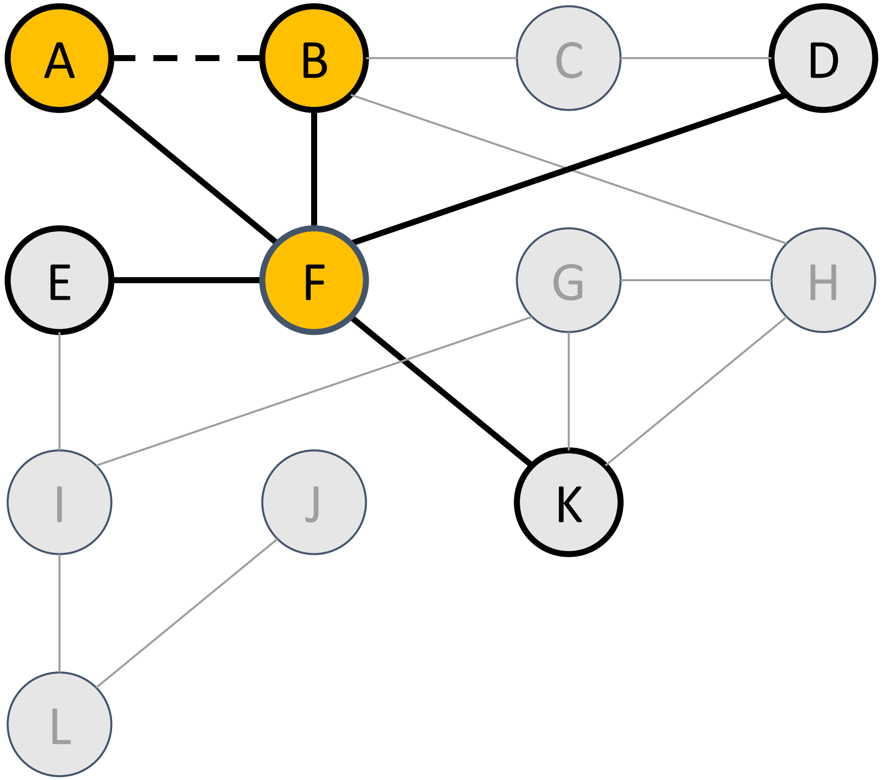


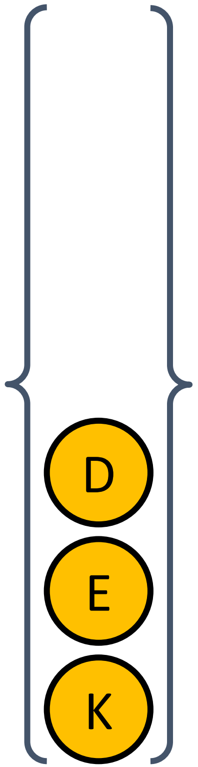
Depth-First Search



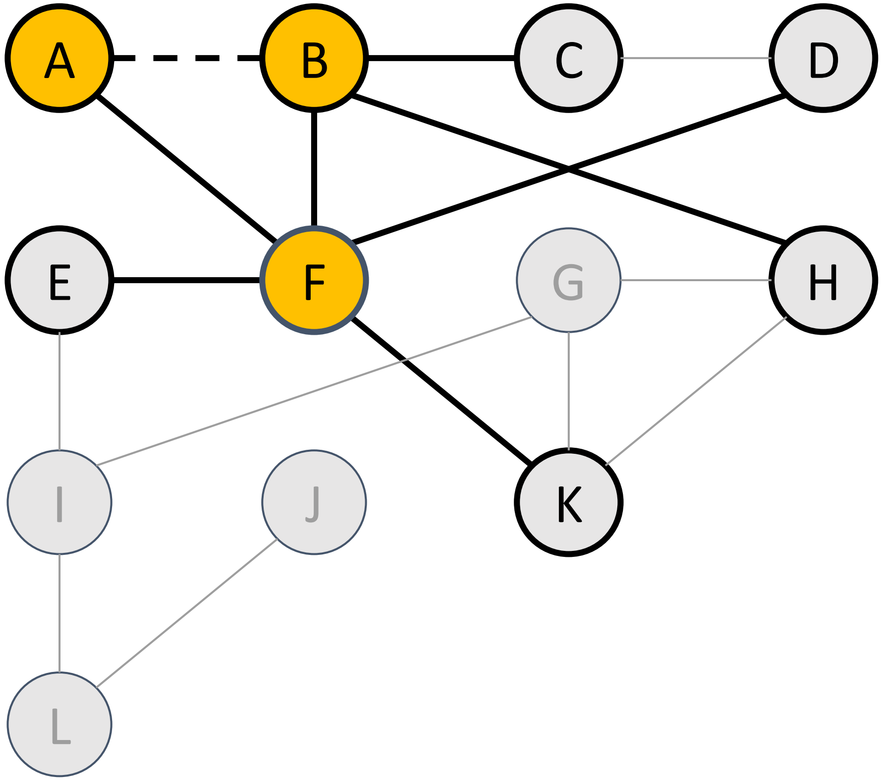


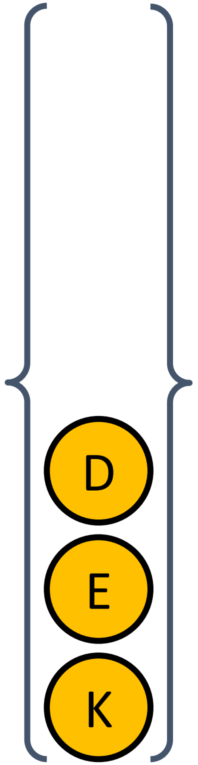
Depth-First Search



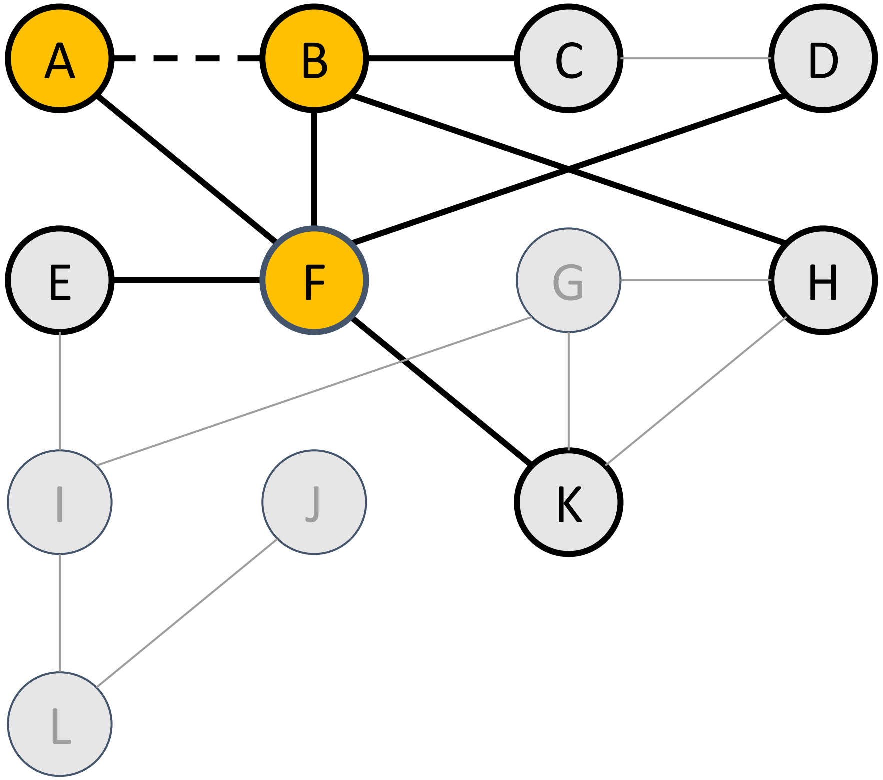


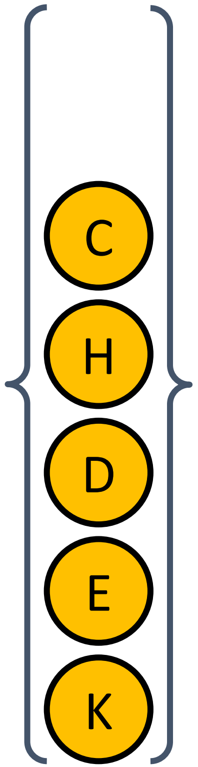
Depth-First Search



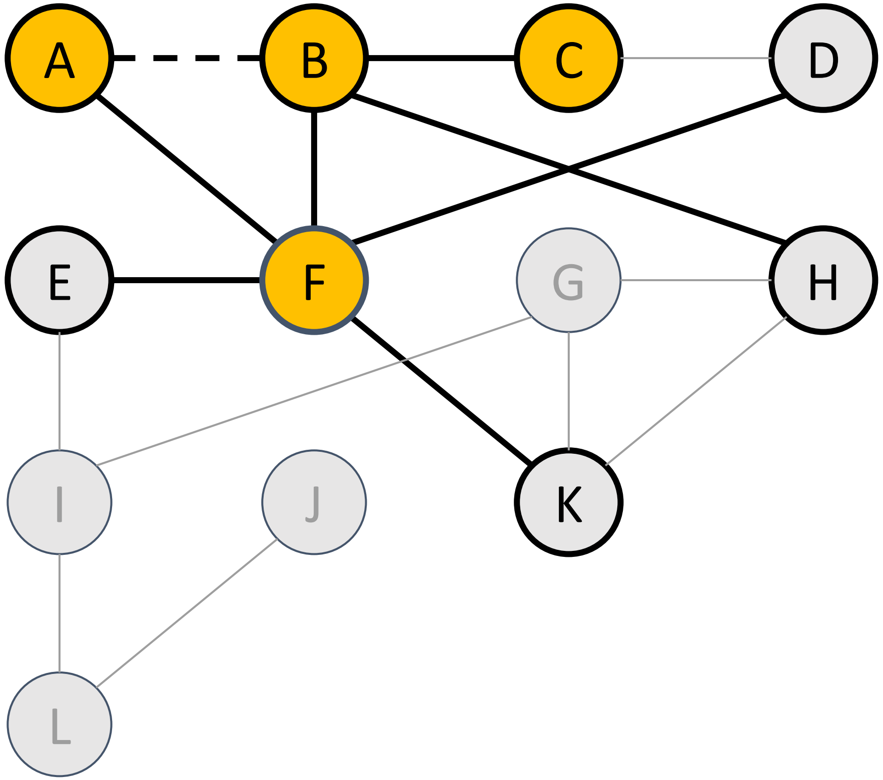


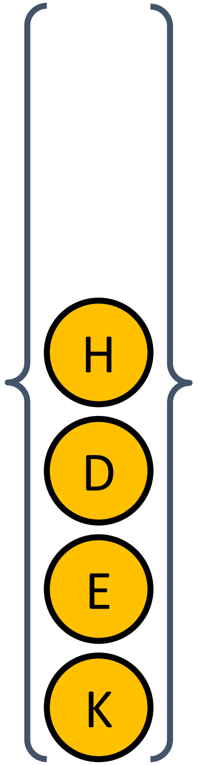
Depth-First Search



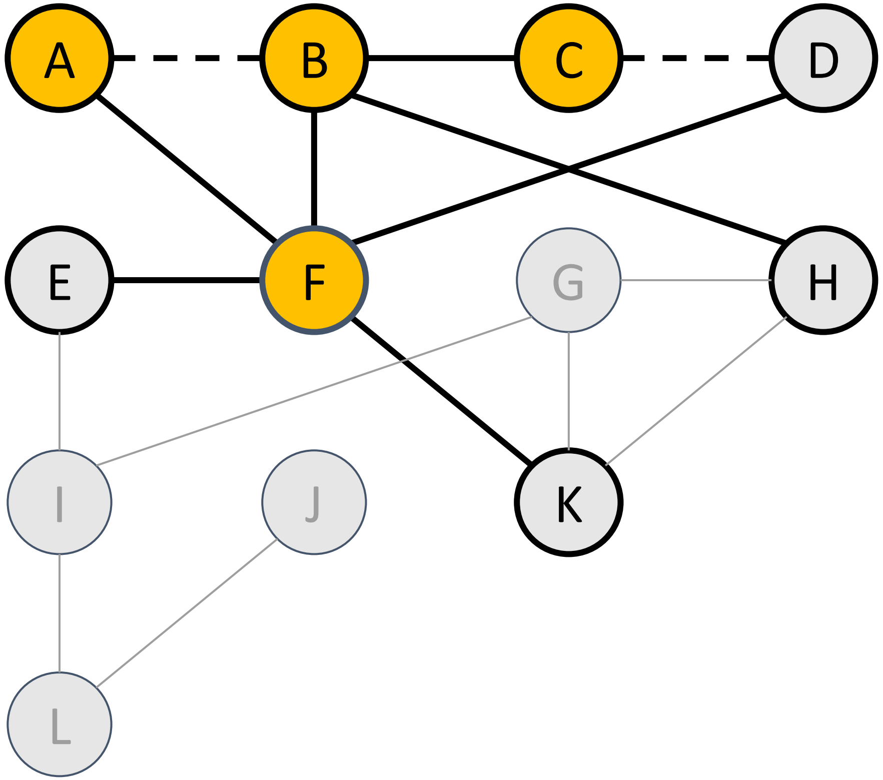


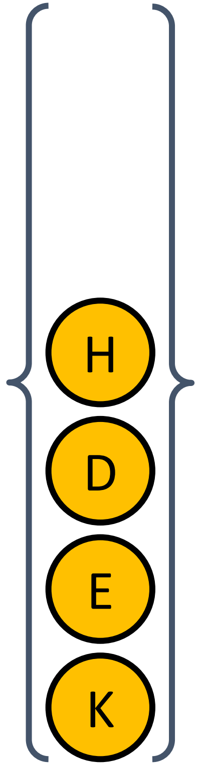
Depth-First Search



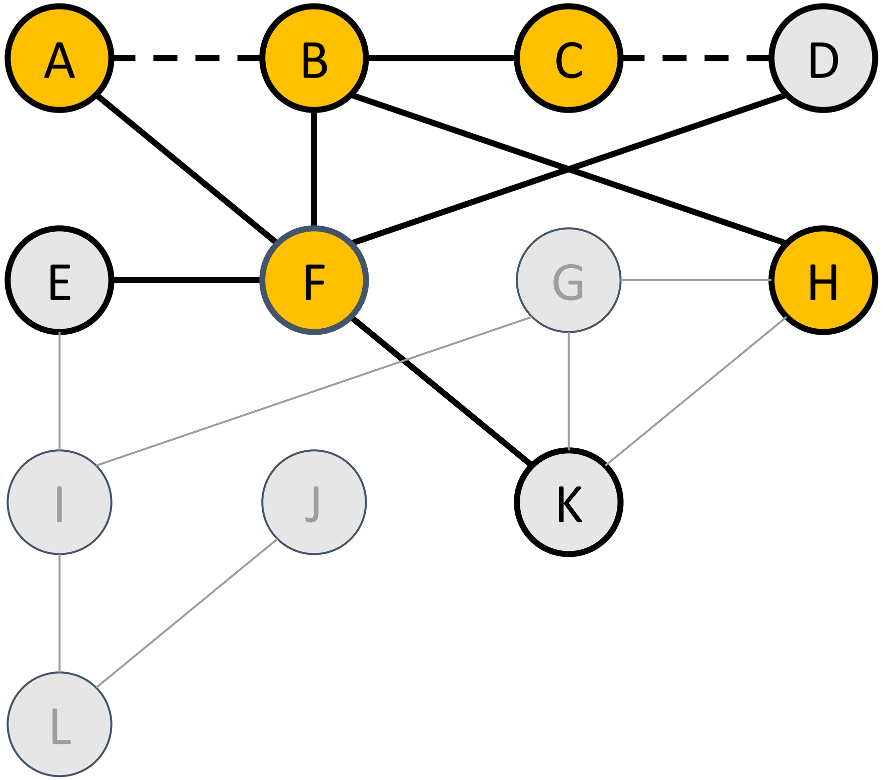


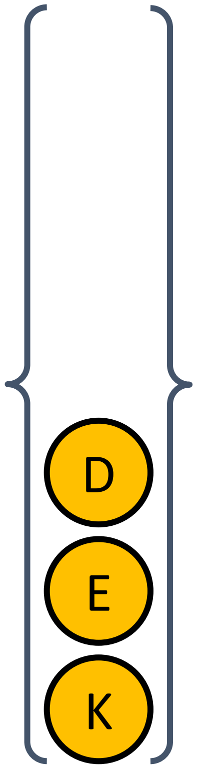
Depth-First Search



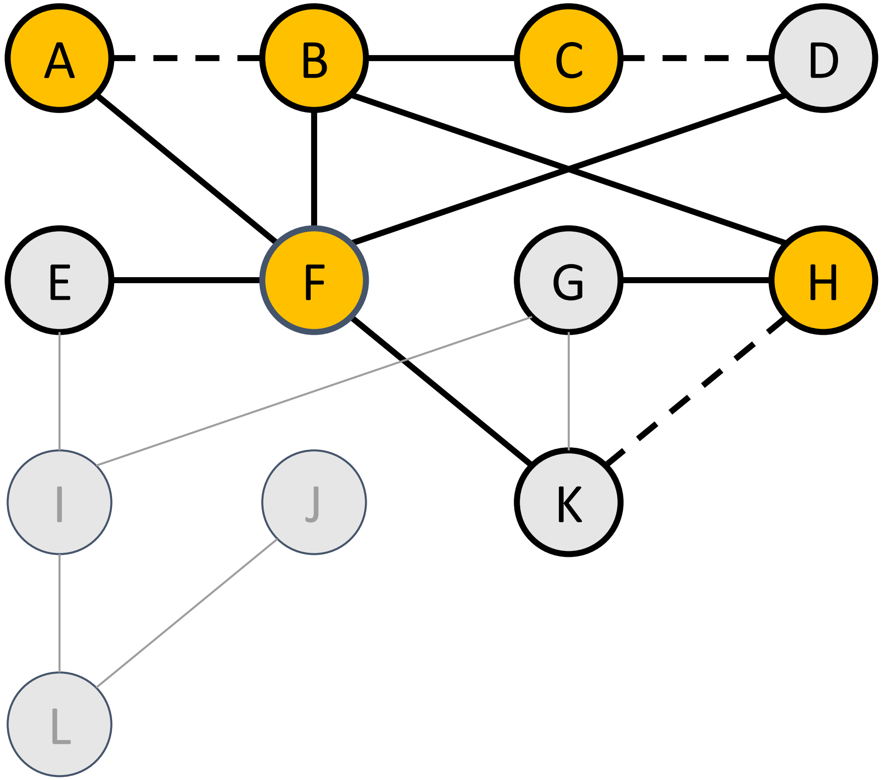


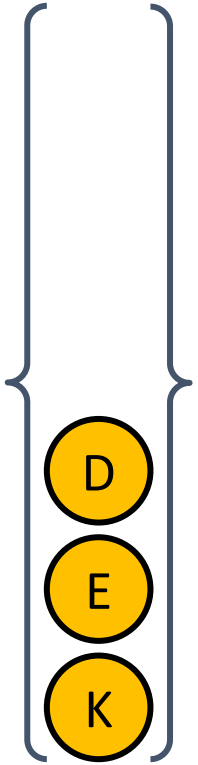
Depth-First Search



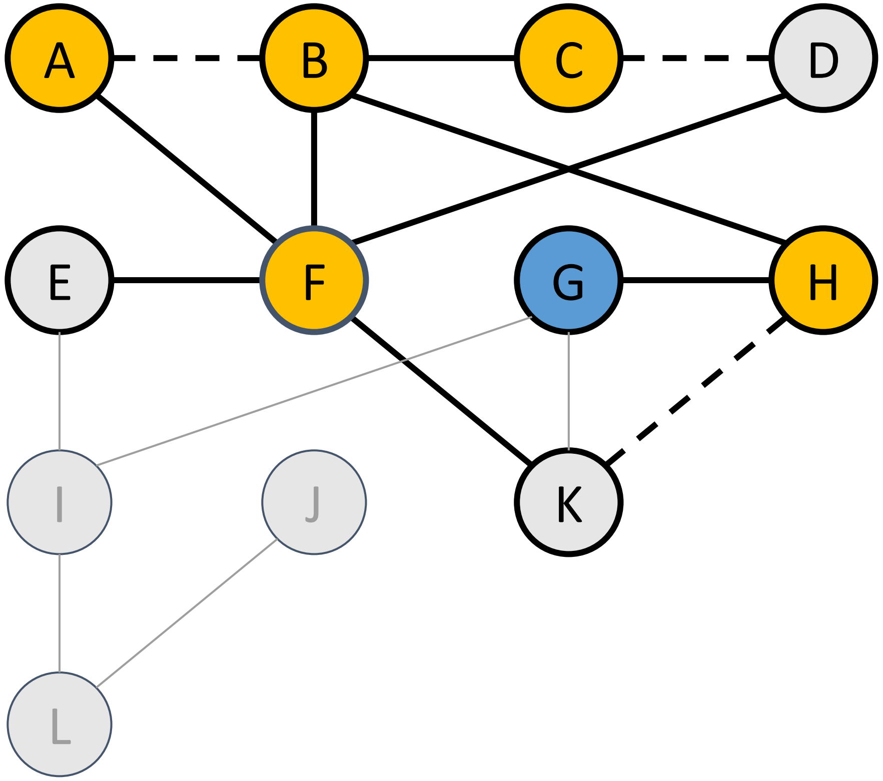


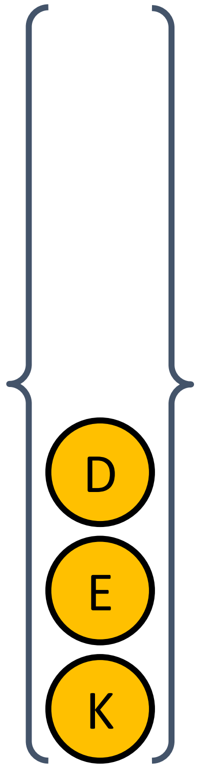
Depth-First Search



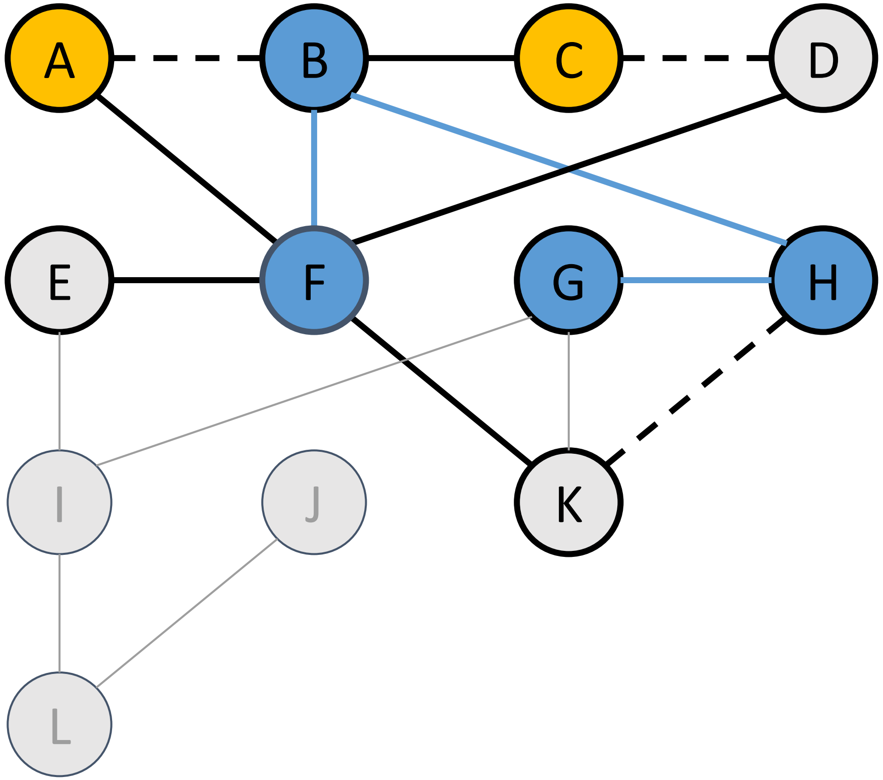


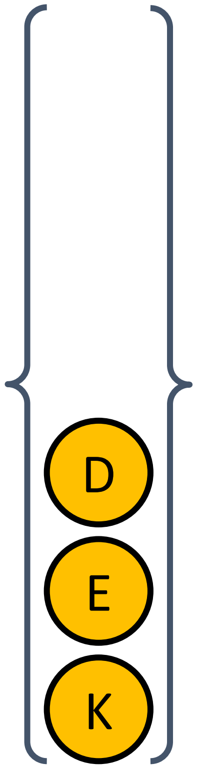
Depth-First Search





Depth-First Search



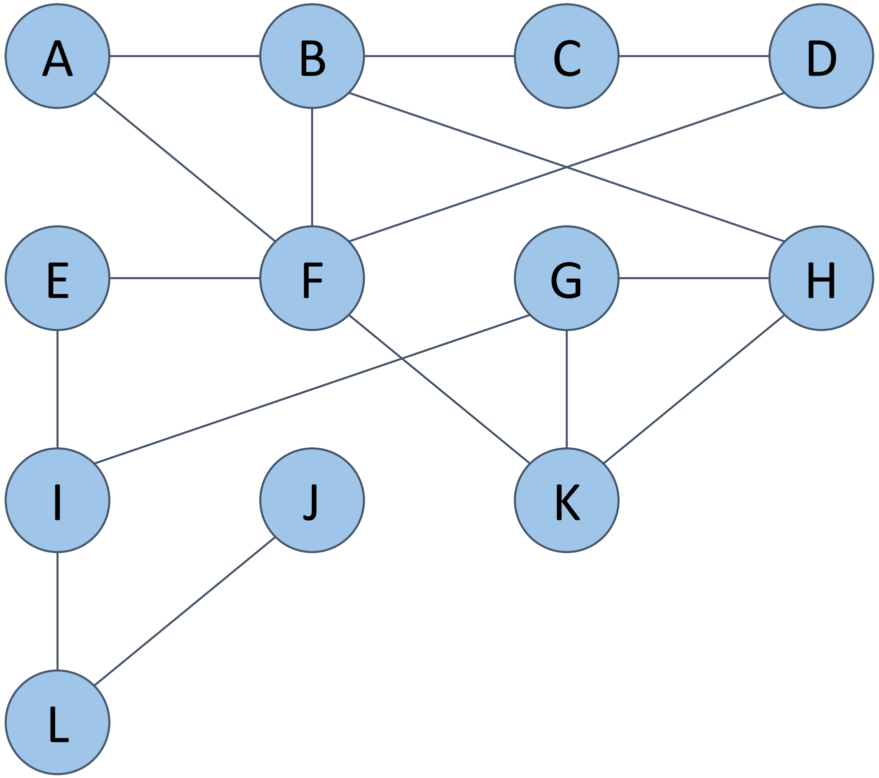


DFS Algorithm

```
dfs-from(node v) {
    make a stack of nodes, initially seeded with v.

    while the stack isn't empty:
        pop a node curr.
        process the node curr.

        for each node adjacent to curr:
            if that node has never been pushed:
                push that node.
}
```

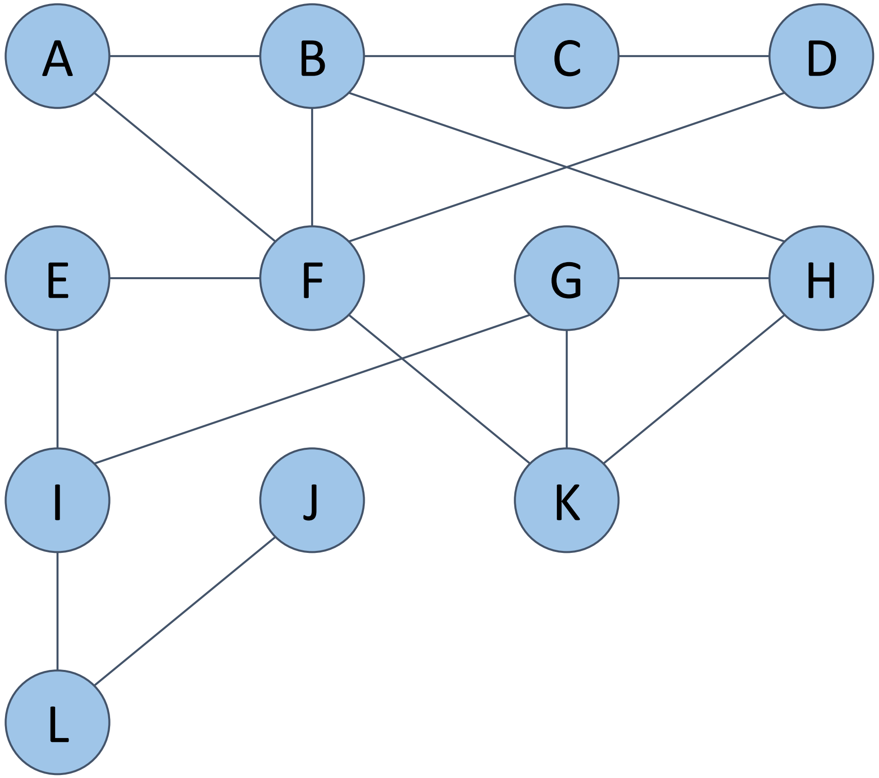
Breadth-First Search

Breadth-First Search



Use BFS to find the shortest path between **F** and **G**

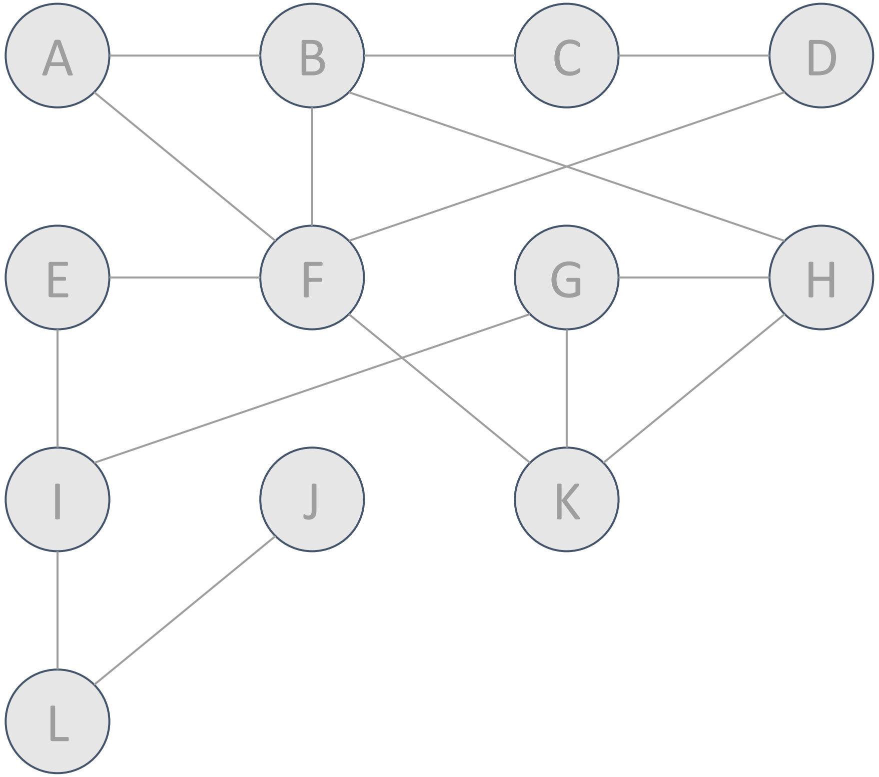
Breadth-First Search



TO START:

1. Mark all nodes as unvisited

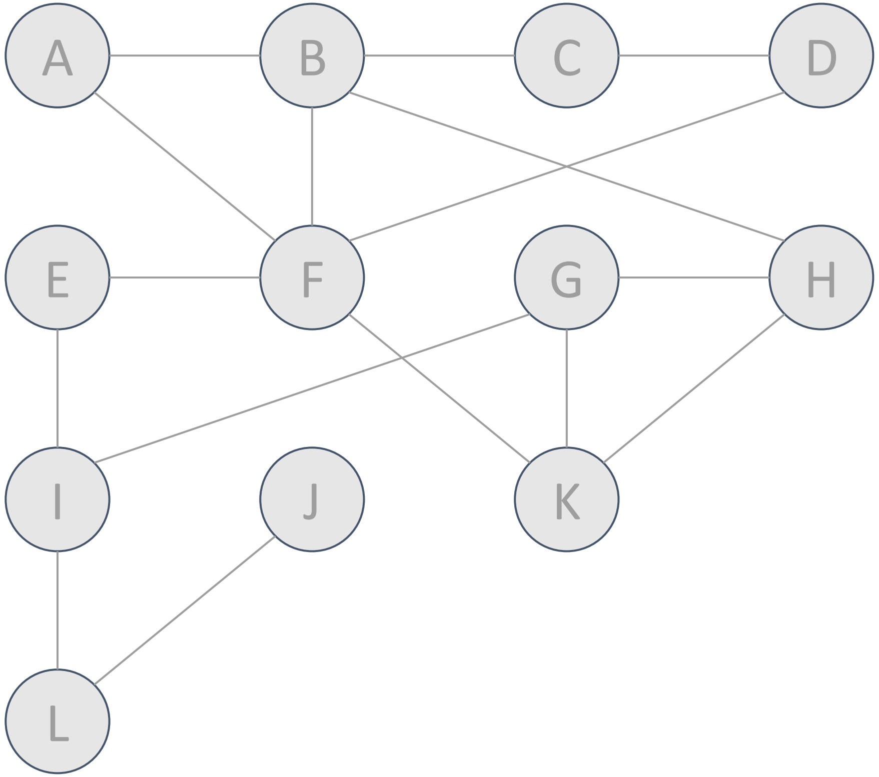
Breadth-First Search



TO START:

1. Mark all nodes as unvisited

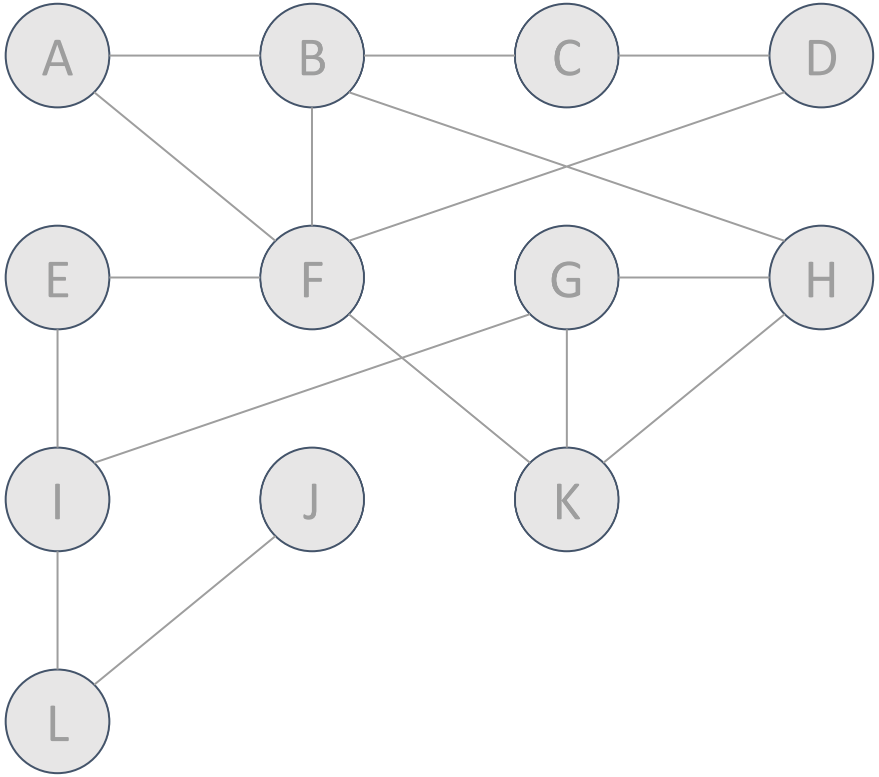
Breadth-First Search



TO START:

1. Mark all nodes as unvisited
2. Make an empty queue

Breadth-First Search

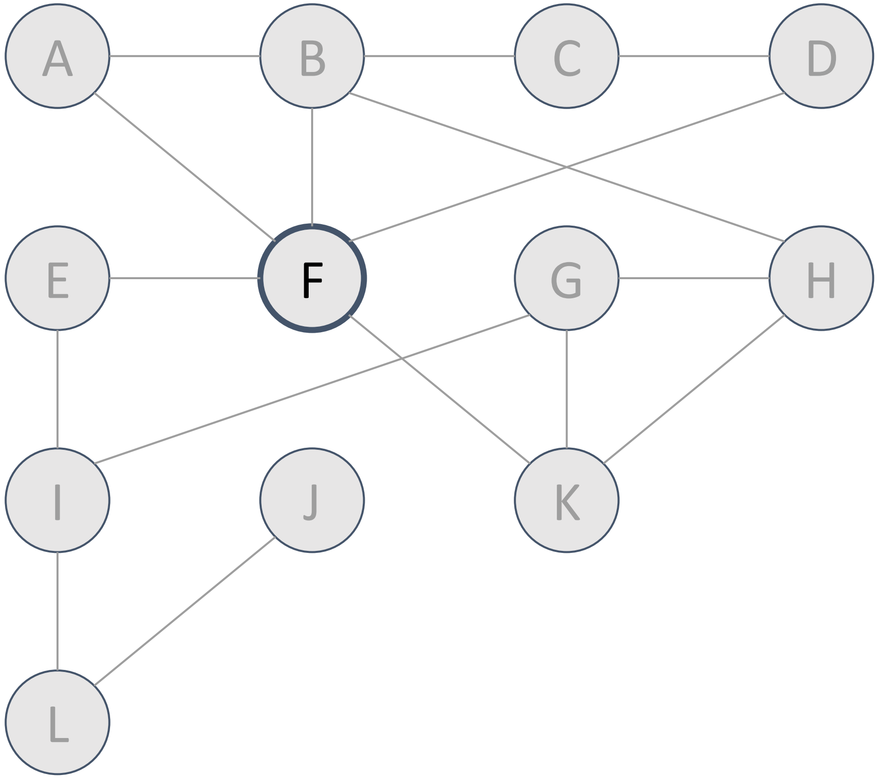


TO START:

1. Mark all nodes as unvisited
2. Make an empty queue

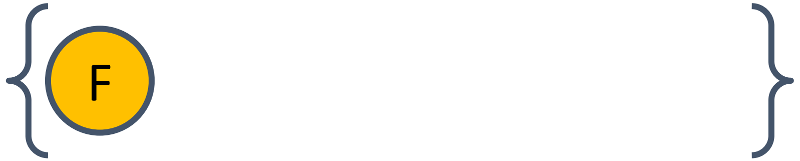
{ }

Breadth-First Search

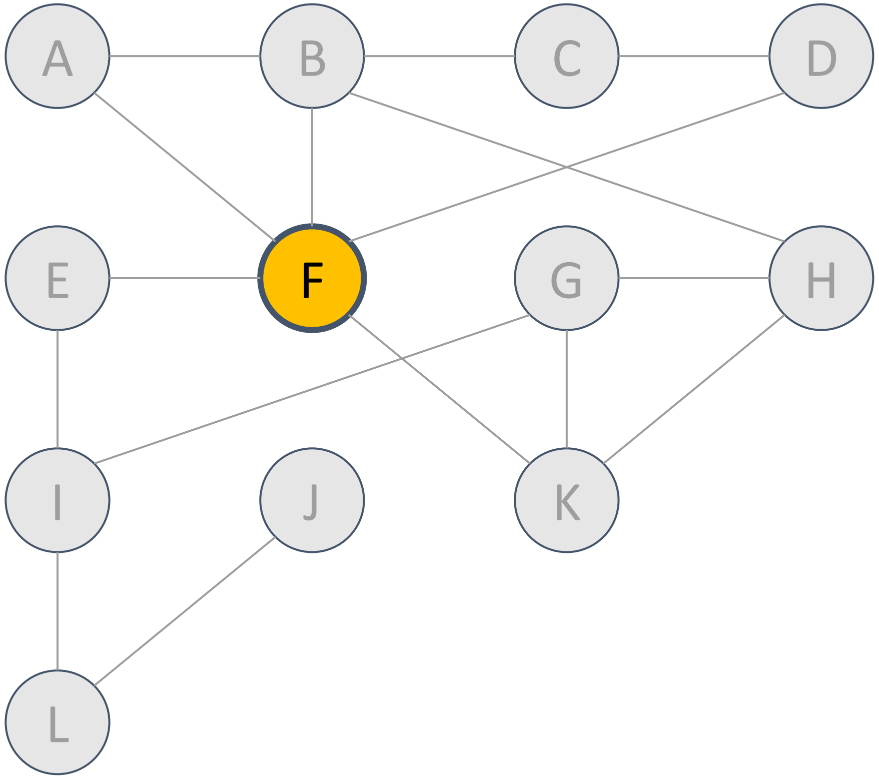


TO START:

1. Mark all nodes as unvisited
2. Make an empty queue
3. Enqueue the desired start node and mark it as visited



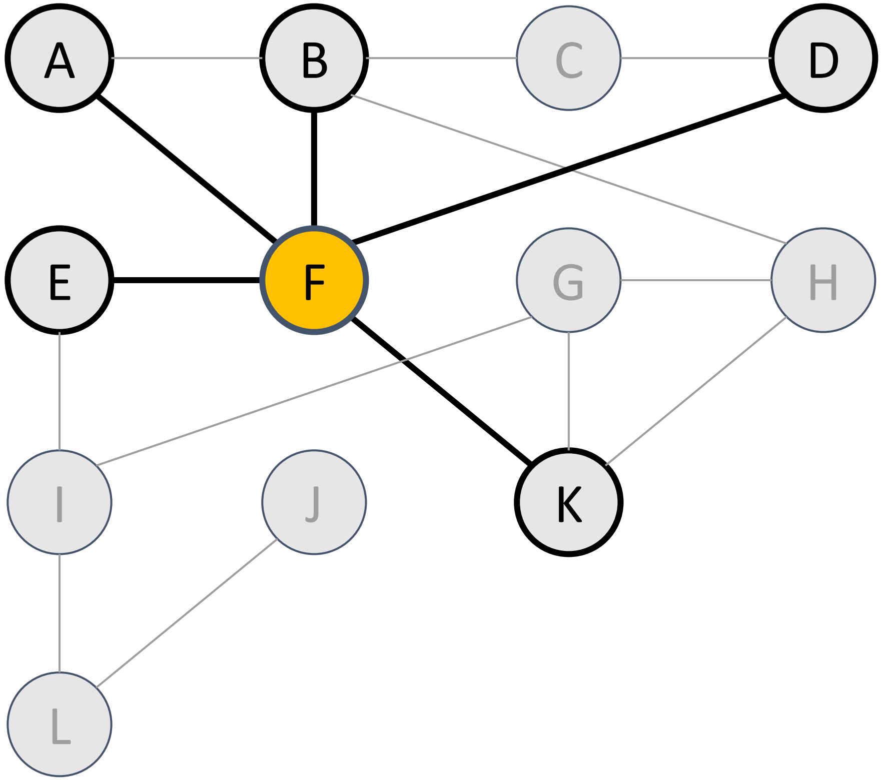
Breadth-First Search



LOOP PROCEDURE:

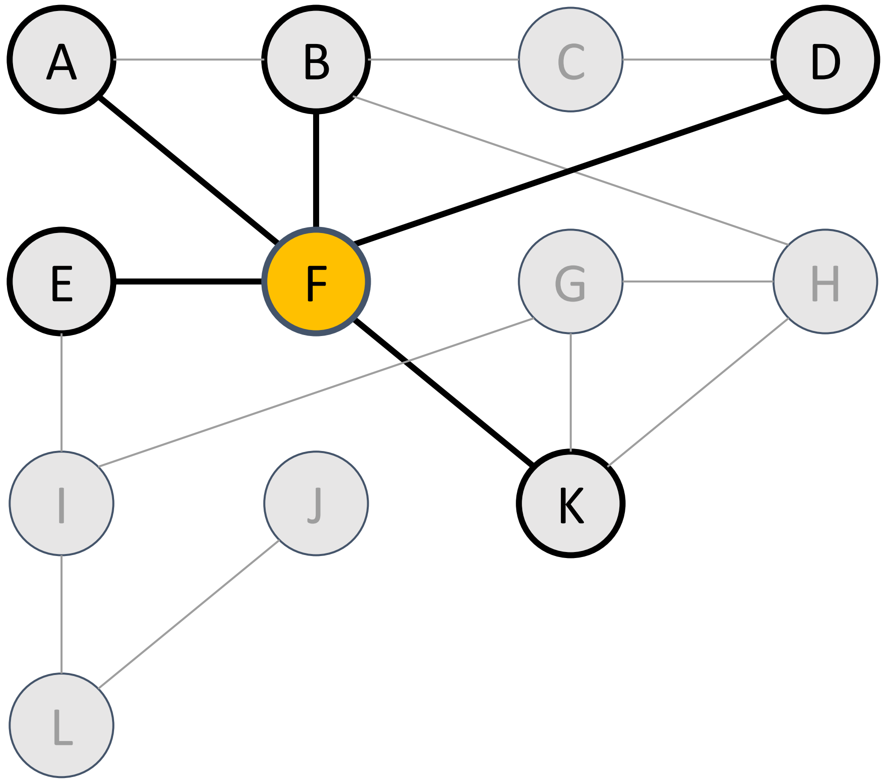
1. Dequeue a node
2. For each adjacent node, if that node has never been enqueued, then enqueue

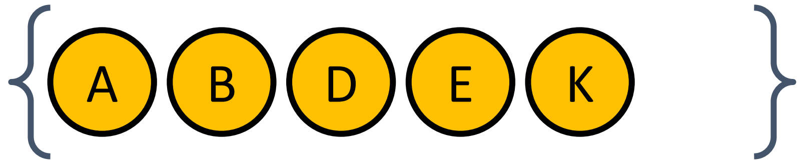
Breadth-First Search



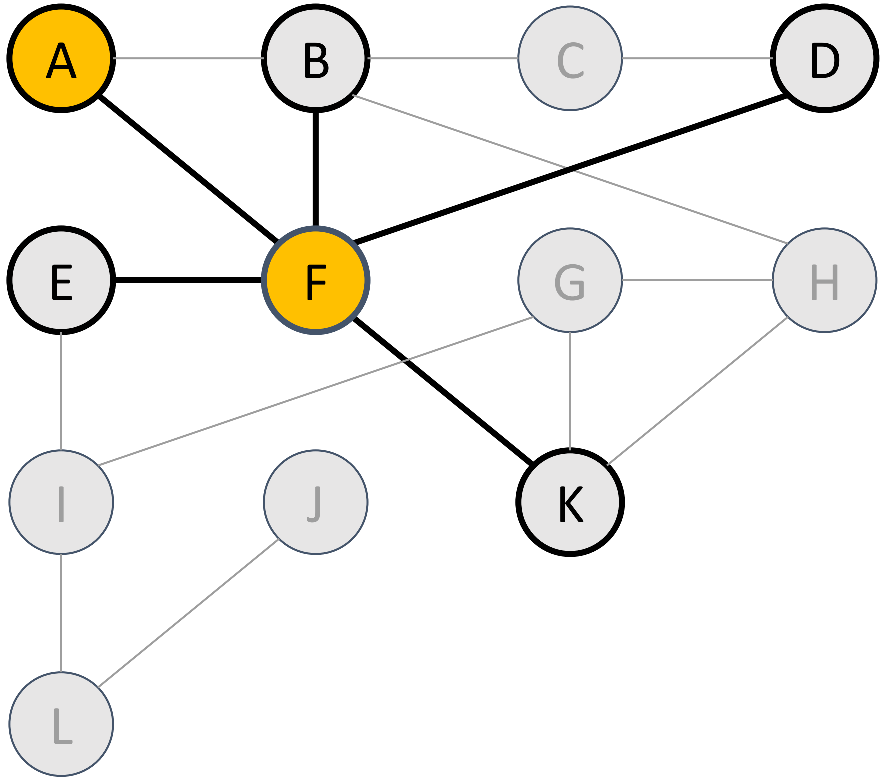
{ }

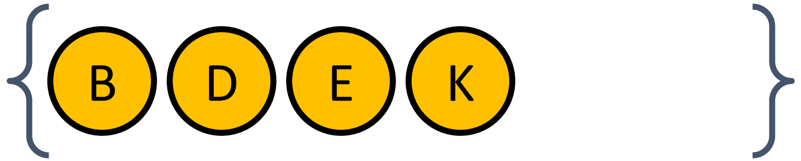
Breadth-First Search



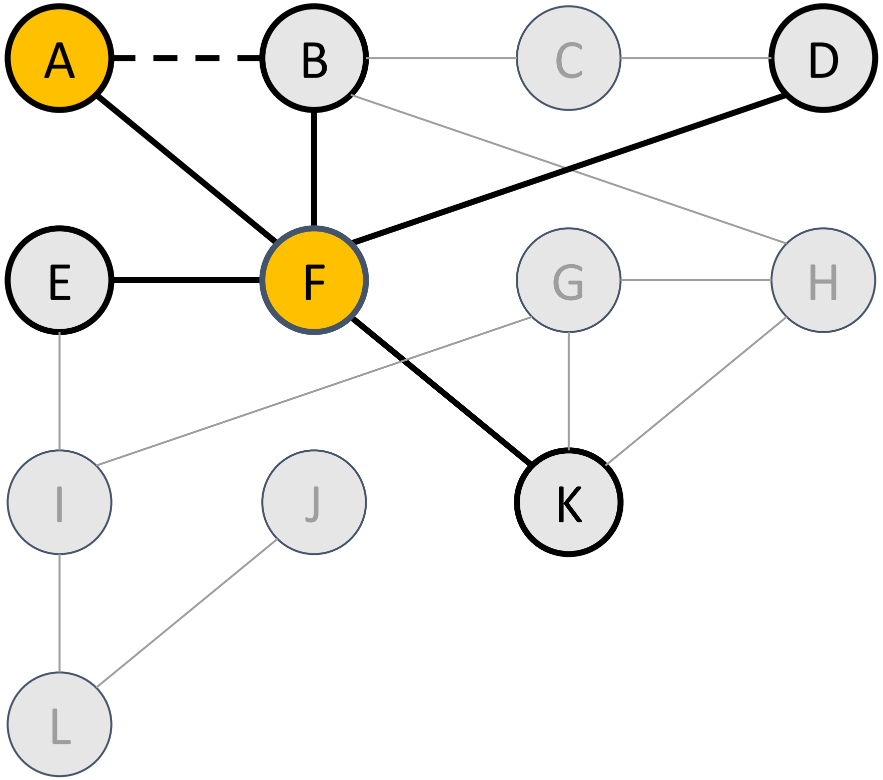


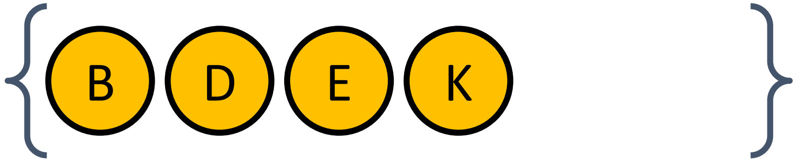
Breadth-First Search



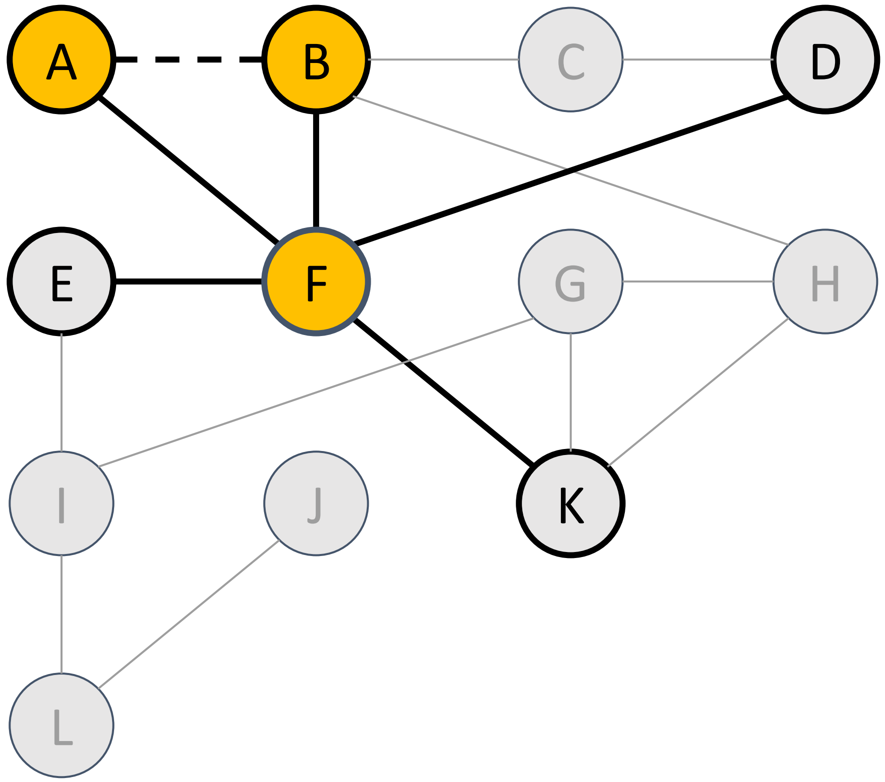


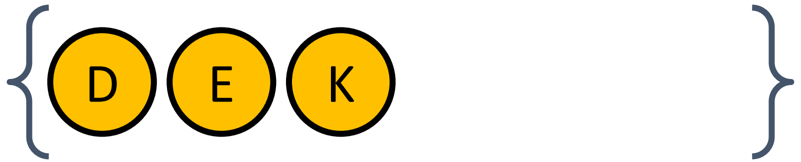
Breadth-First Search



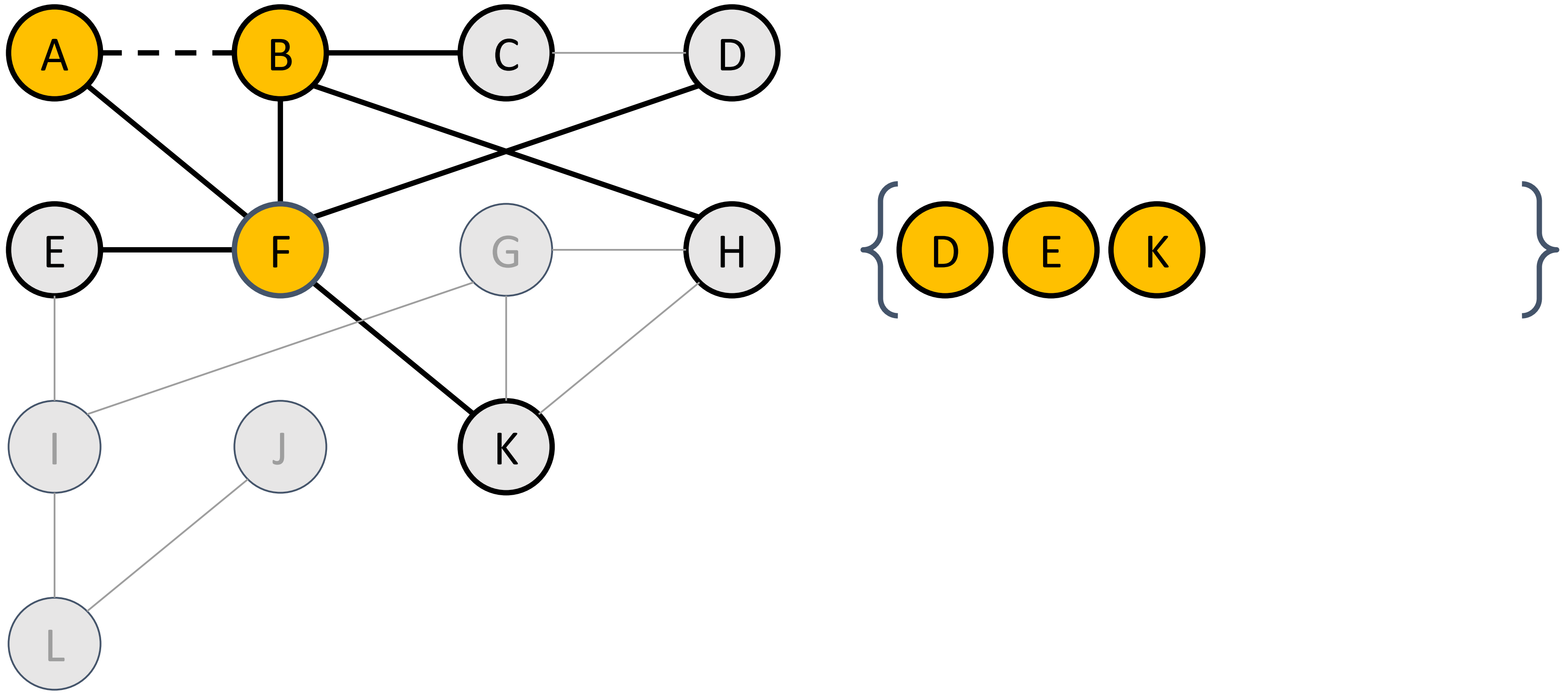


Breadth-First Search

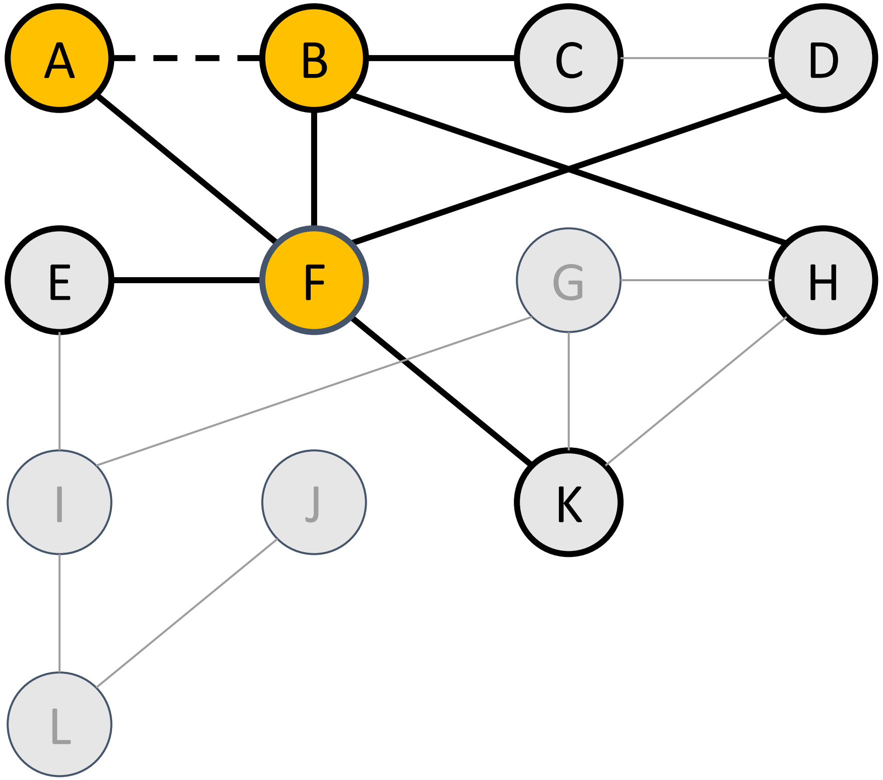


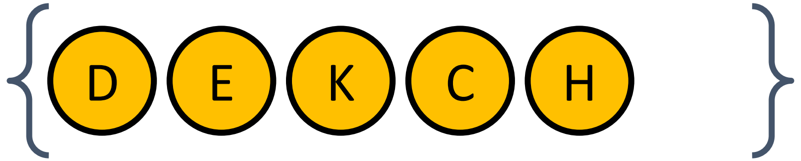


Breadth-First Search

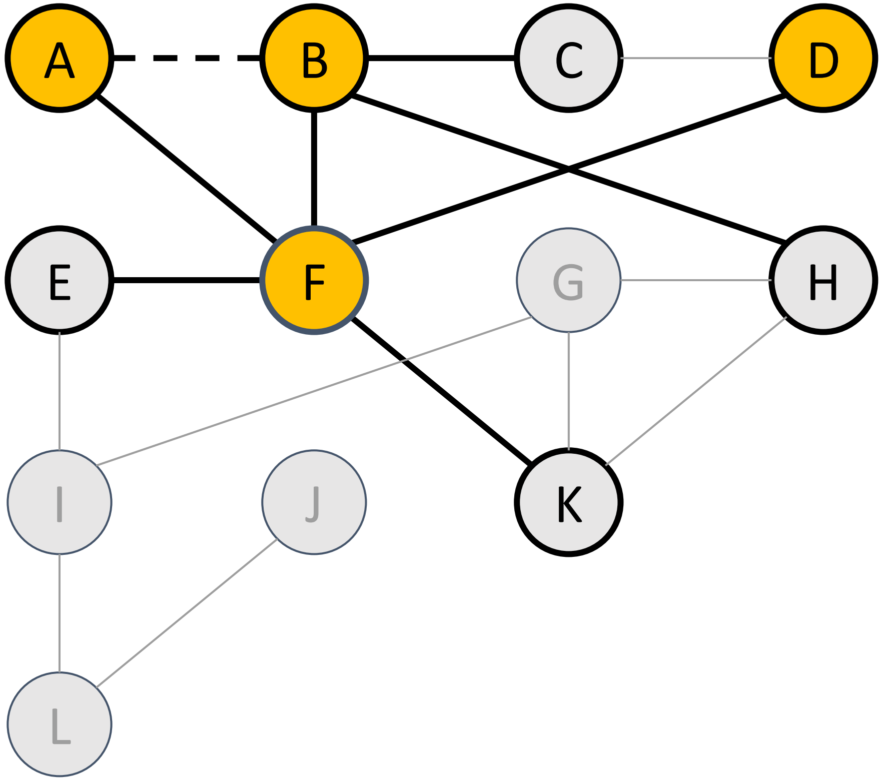


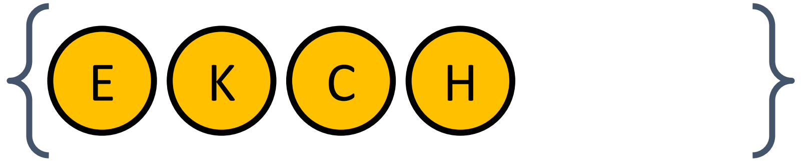
Breadth-First Search



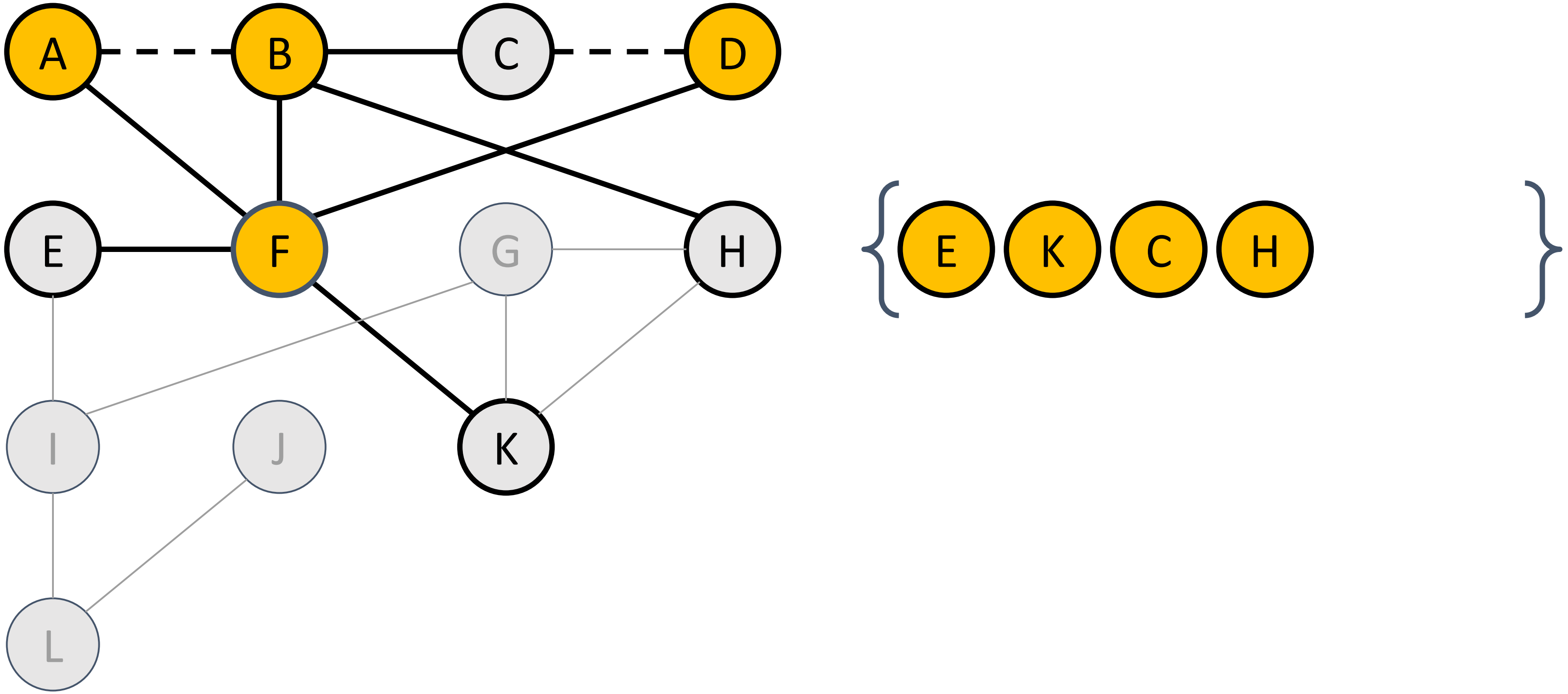


Breadth-First Search

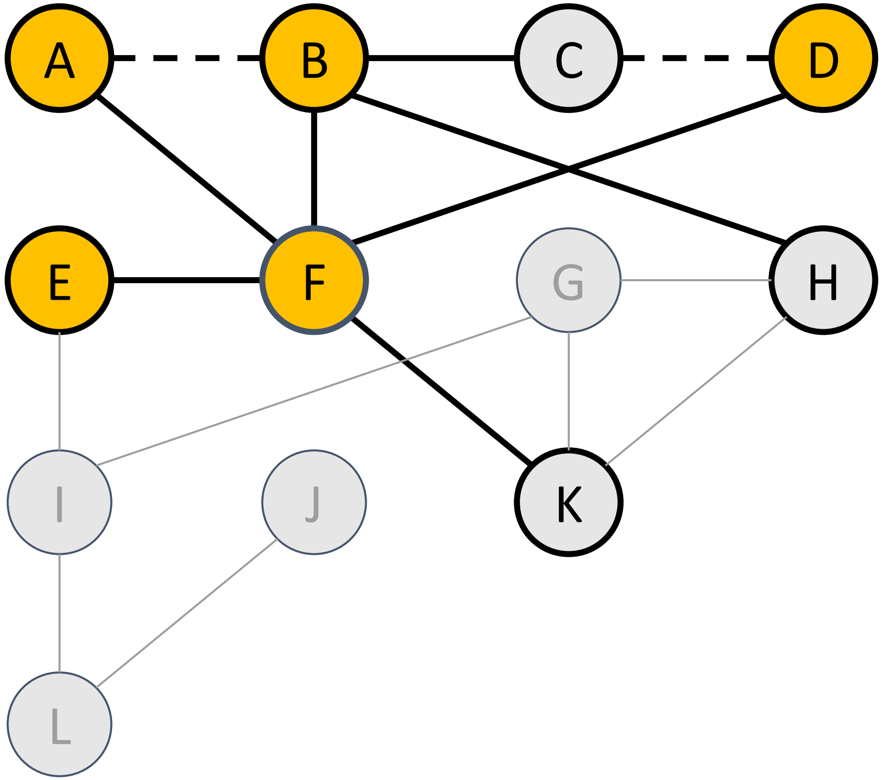


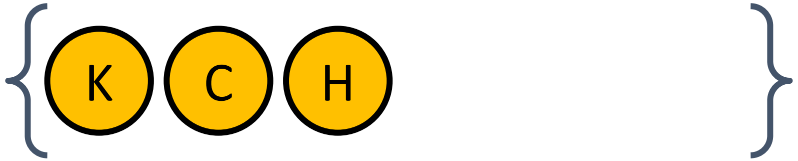


Breadth-First Search

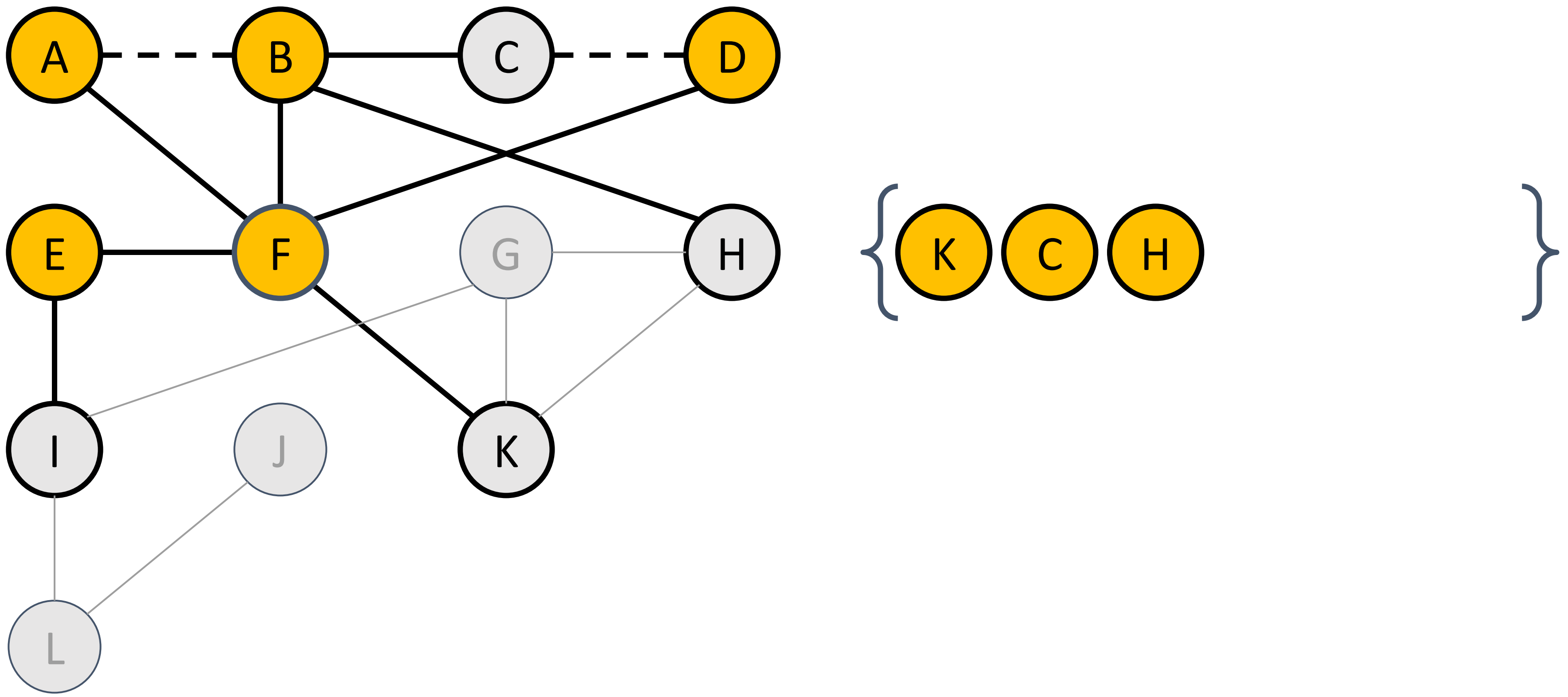


Breadth-First Search

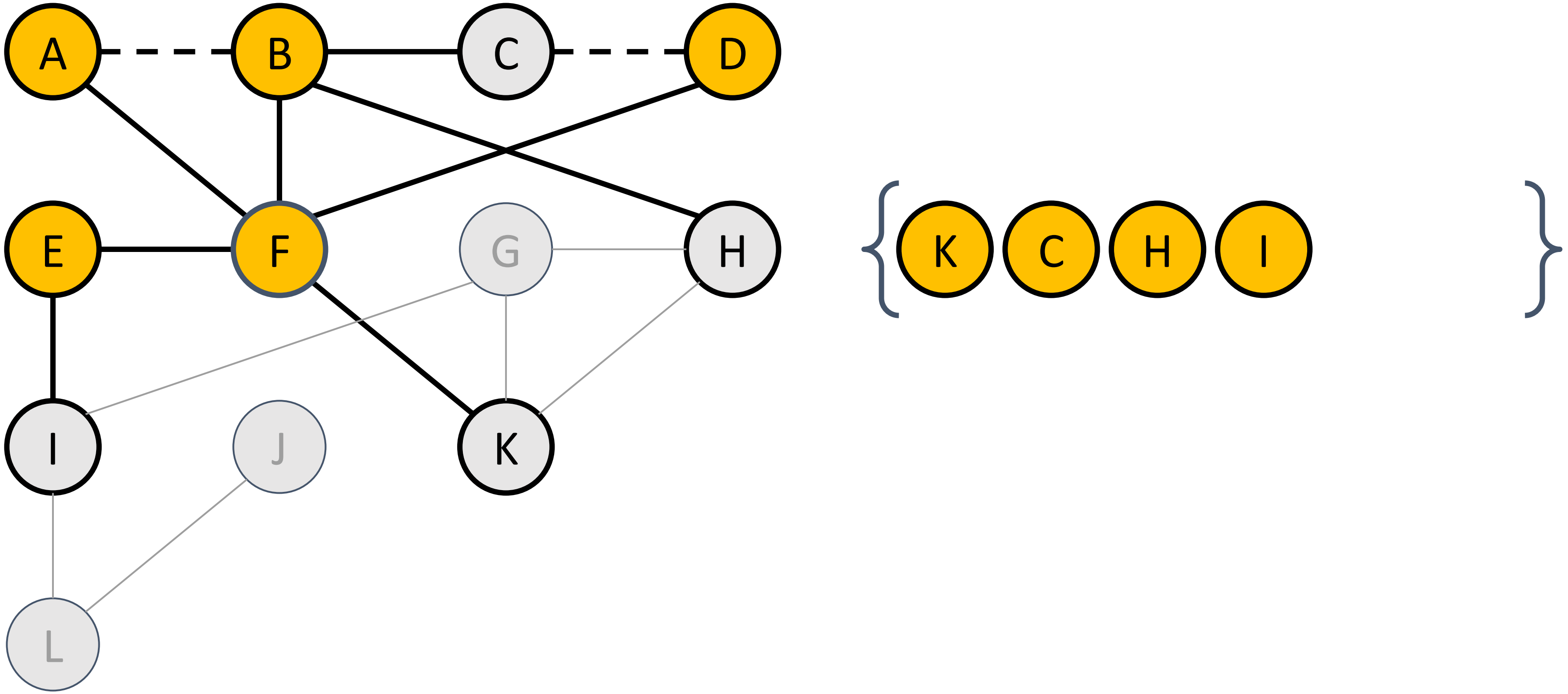




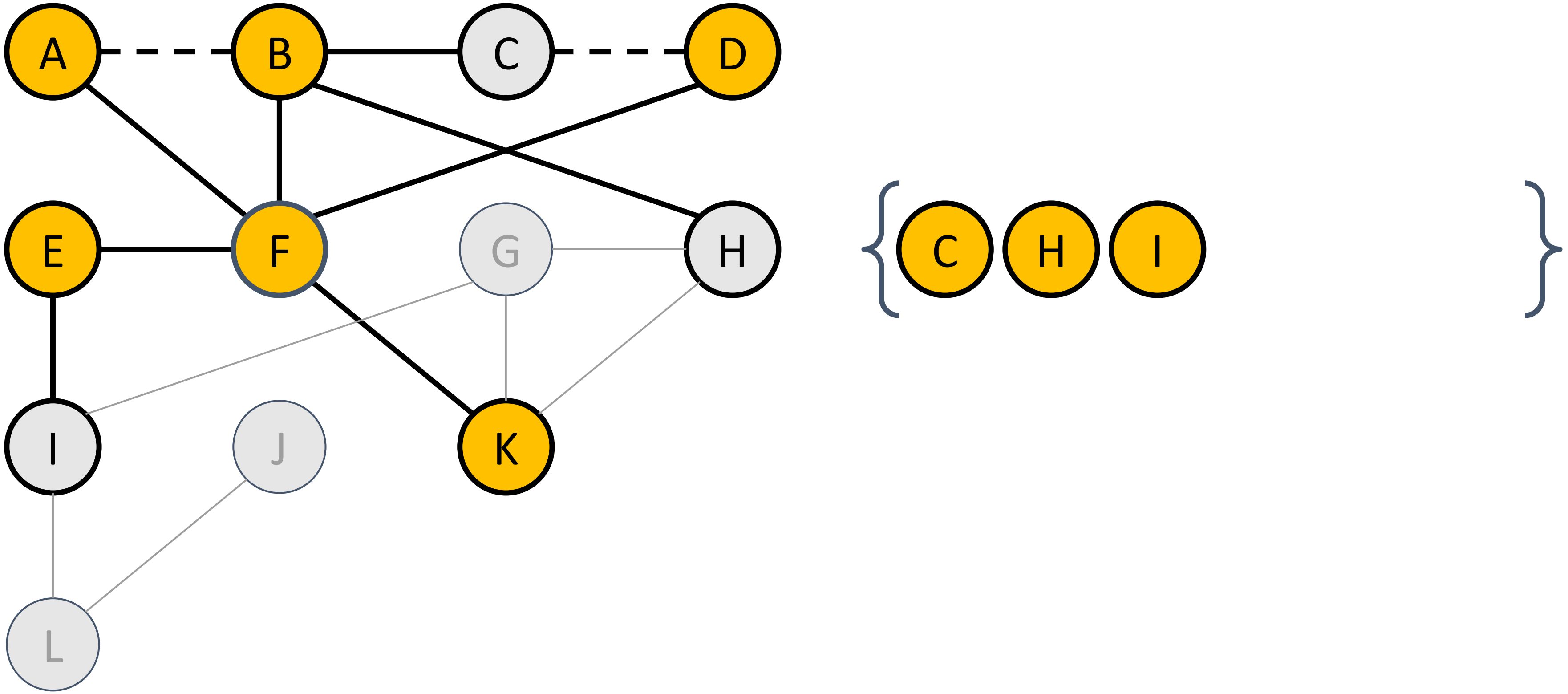
Breadth-First Search



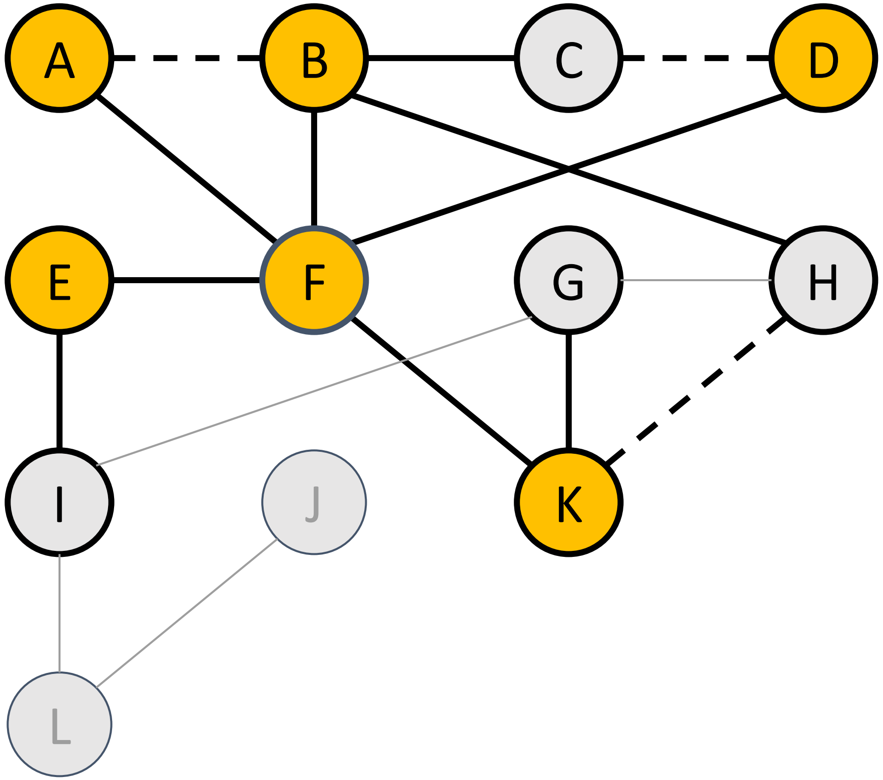
Breadth-First Search

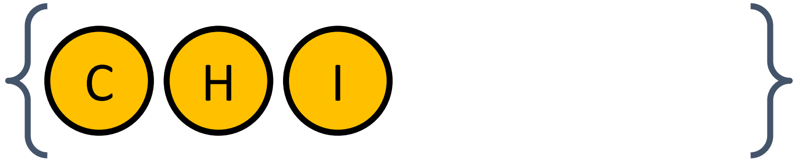


Breadth-First Search

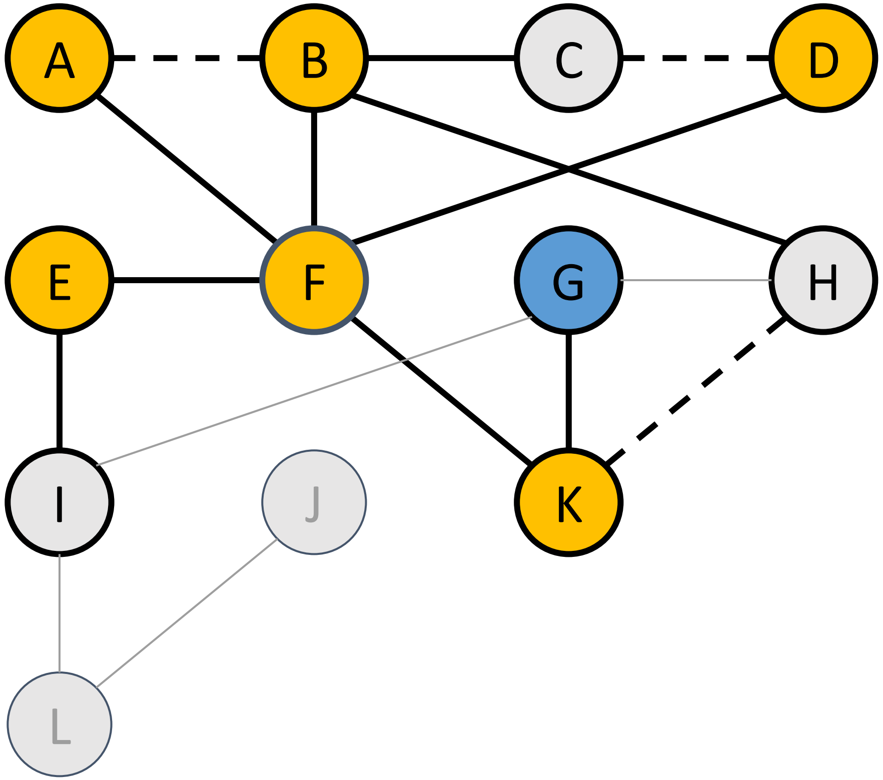


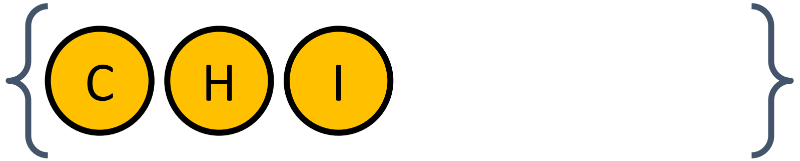
Breadth-First Search



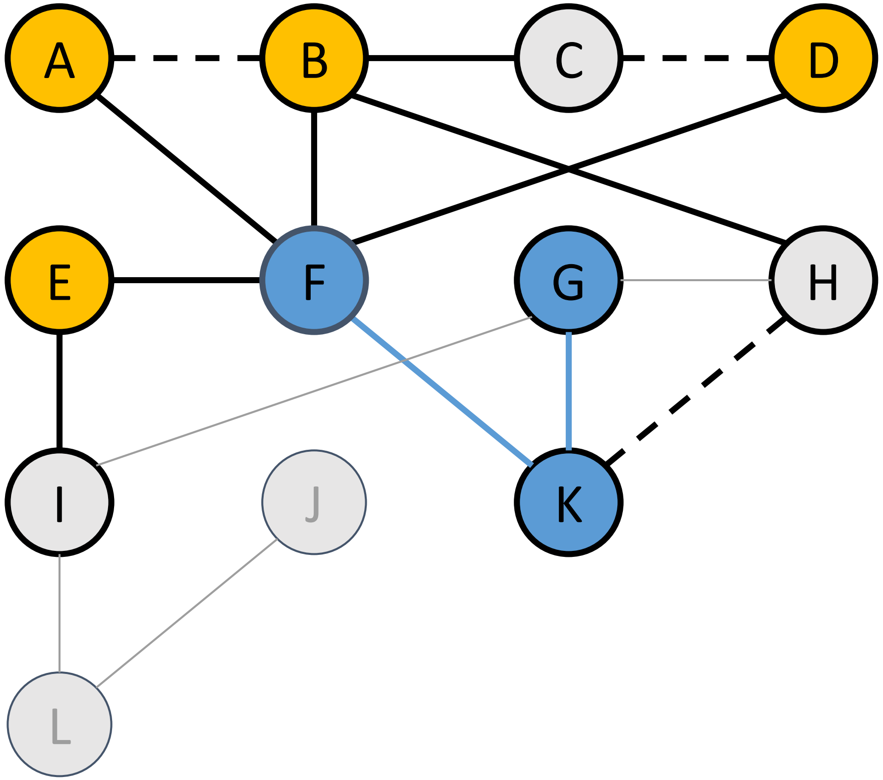


Breadth-First Search





Breadth-First Search



BFS Algorithm

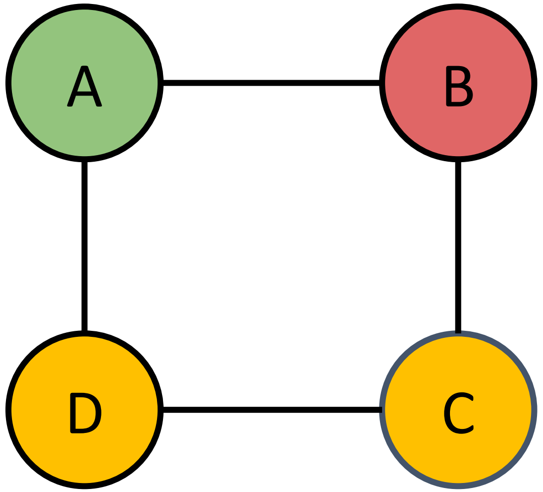
```
bfs-from(node v) {
    make a queue of nodes, initially seeded with v.

    while the queue isn't empty:
        dequeue a node curr.
        process the node curr.

        for each node adjacent to curr:
            if that node has never been enqueued:
                enqueue that node.
}
```

BFS vs DFS

- Running BFS or DFS from a node in a graph will visit the same set of nodes, but probably in a different order
- BFS will visit nodes in increasing order of distance
  - Will give you the shortest path
- DFS does visit nodes in some interesting order, but not order of distance
  - Take CS161 for more info

Shortest Path

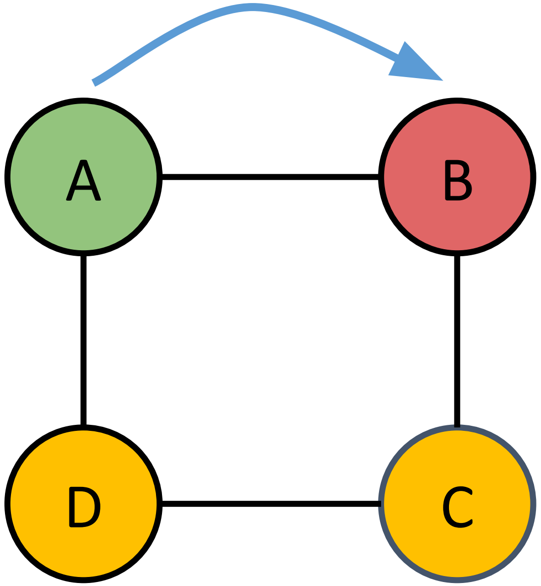
What is the shortest path from A to B?



Shortest Path

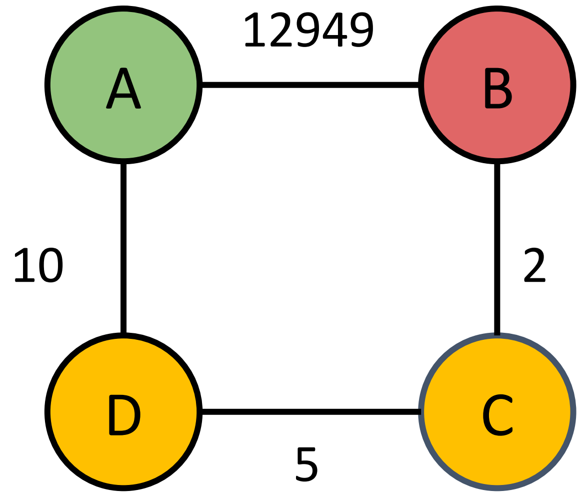
What is the shortest path from A to B?

- Use BFS!



Shortest Weighted Path

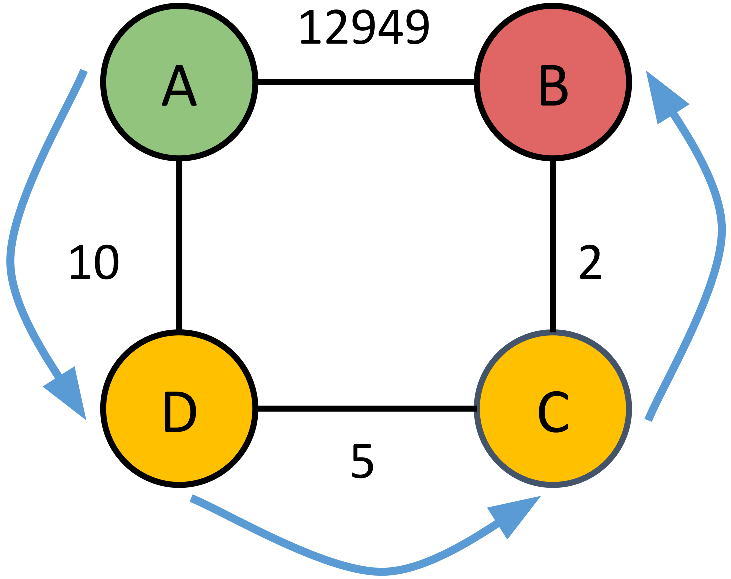
What is the shortest **weighted** path from A to B?



Shortest Weighted Path

What is the shortest **weighted** path from A to B?

- BFS doesn’t work here



Dijkstra's Algorithm

Dijkstra's Algorithm

- Finds the shortest weighted path from one node to another

Dijkstra's Algorithm

- Finds the shortest weighted path from one node to another
- Greedy algorithm
  - Prioritizes finding a solution by what is "best right now"
  - Looks at its options and always chooses whatever gets it closer to a solution in the best possible way given the current situation
  - Ex: Change We Can Believe In (Section 4, Problem 2)

Dijkstra's Algorithm

- Finds the shortest weighted path from one node to another
- Greedy algorithm
  - Prioritizes finding a solution by what is "best right now"
  - Looks at its options and always chooses whatever gets it closer to a solution in the best possible way given the current situation
  - Ex: Change We Can Believe In (Section 4, Problem 2)
- Many different ways to model this
  - Can use a priority queue, where weights become priorities
  - Can use a table of nodes

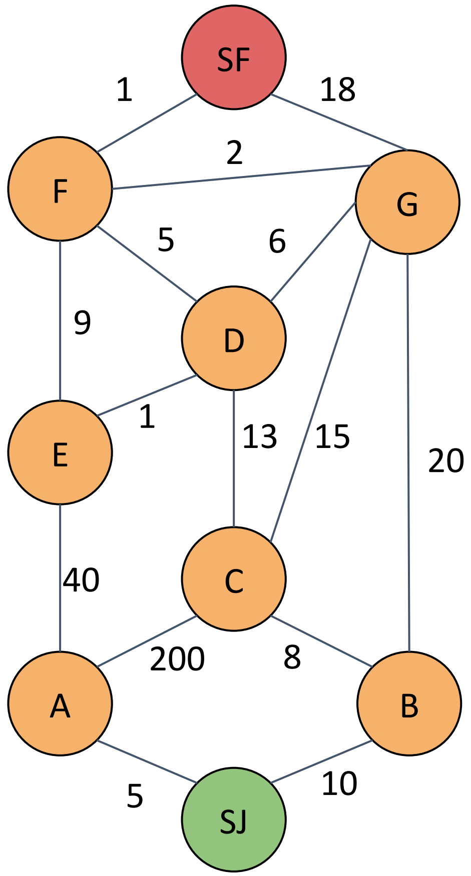
Dijkstra's Algorithm

- Finds the shortest weighted path from one node to another
- Greedy algorithm
  - Prioritizes finding a solution by what is "best right now"
  - Looks at its options and always chooses whatever gets it closer to a solution in the best possible way given the current situation
  - Ex: Change We Can Believe In (Section 4, Problem 2)
- Many different ways to model this
  - Can use a priority queue, where weights become priorities
  - Can use a table of nodes
- Real world uses: shortest paths on maps (Ethiopia), tracks of electricity lines and oil pipelines, network routing protocols

Dijkstra's Algorithm

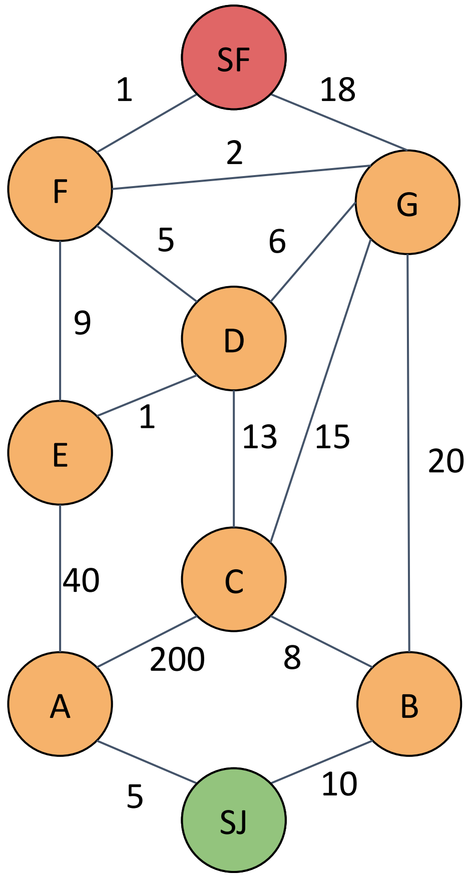
Algorithm:

1. Of the unseen nodes, find the node that currently has the shortest distance from the start
2. Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node.
3. If the node visited is the destination, stop
4. Repeat from step 1



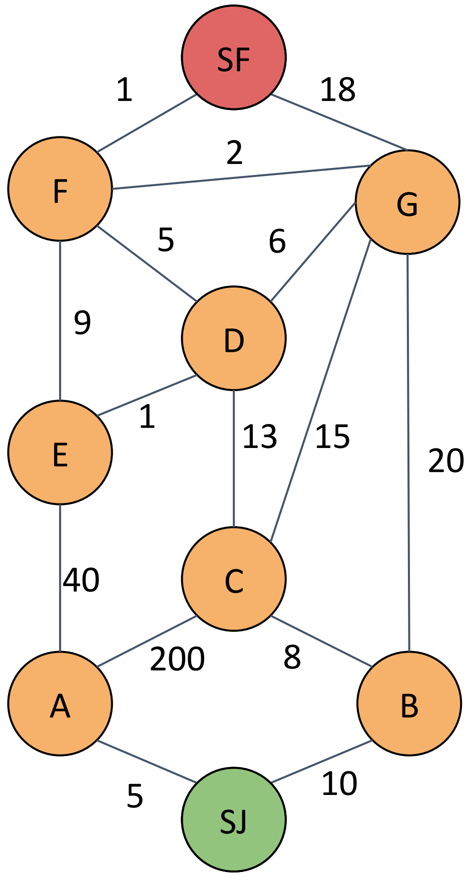
Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start									
Previous									
Seen?									



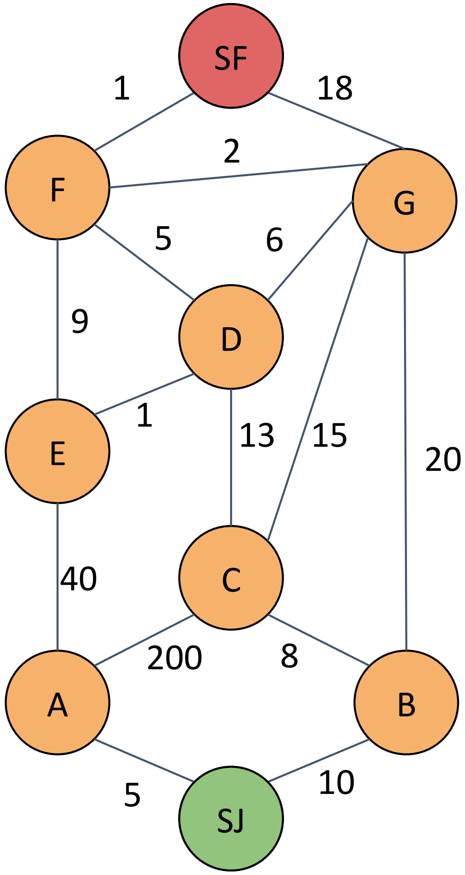
Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	∞	∞	∞	∞	∞	∞	∞	∞
Previous									
Seen?									



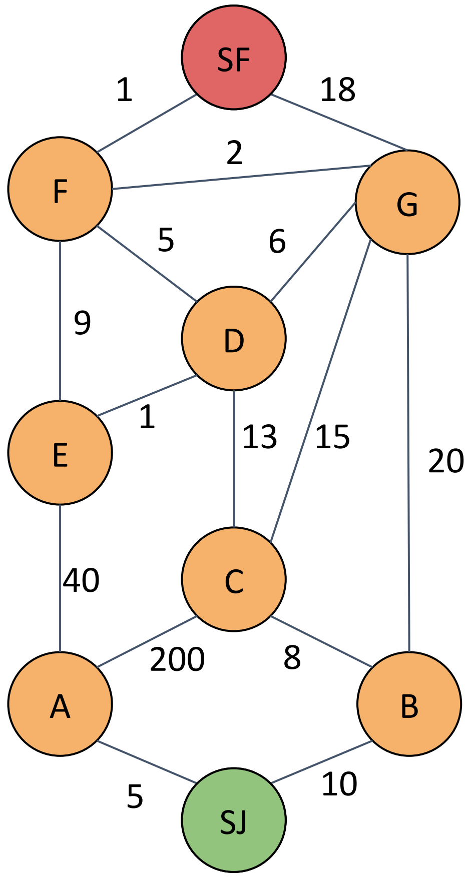
Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	∞	∞	∞	∞	∞	∞	∞	∞
Previous	-								
Seen?									



Dijkstra's Algorithm

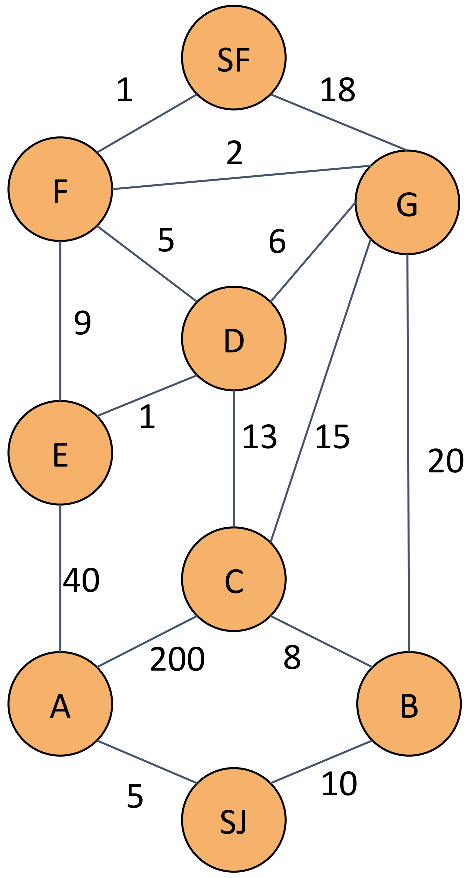
	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	∞	∞	∞	∞	∞	∞	∞	∞
Previous	-								
Seen?	N	N	N	N	N	N	N	N	N



Dijkstra's Algorithm

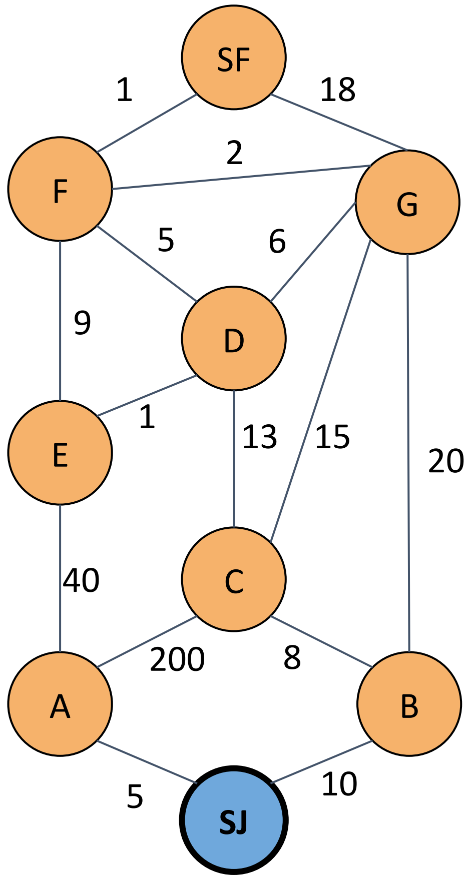
	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	∞	∞	∞	∞	∞	∞	∞	∞
Previous	-								
Seen?	N	N	N	N	N	N	N	N	N

Step 1: Of the unseen nodes, find the node that currently has the shortest distance from the start



Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	∞	∞	∞	∞	∞	∞	∞	∞
Previous	-								
Seen?	N	N	N	N	N	N	N	N	N

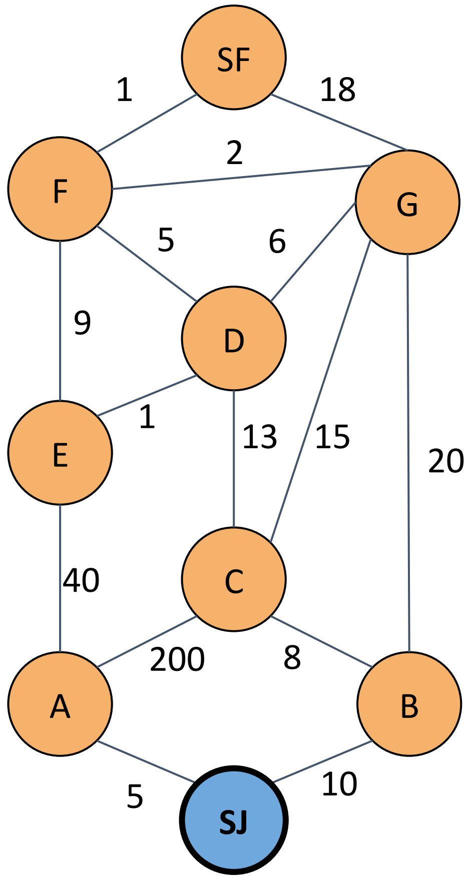


Step 1: Of the unseen nodes, find the node that currently has the shortest distance from the start

Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	∞	∞	∞	∞	∞	∞	∞	∞
Previous	-								
Seen?	N	N	N	N	N	N	N	N	N

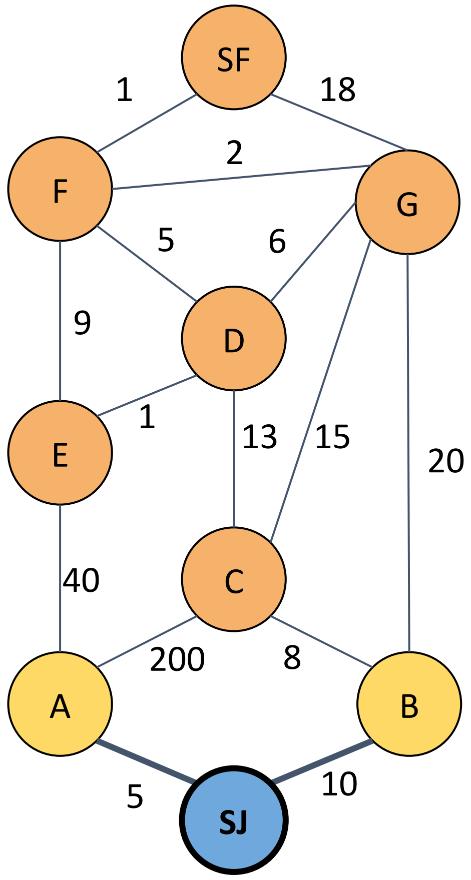
Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node



Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	∞	∞	∞	∞	∞	∞	∞	∞
Previous	-								
Seen?	N	N	N	N	N	N	N	N	N

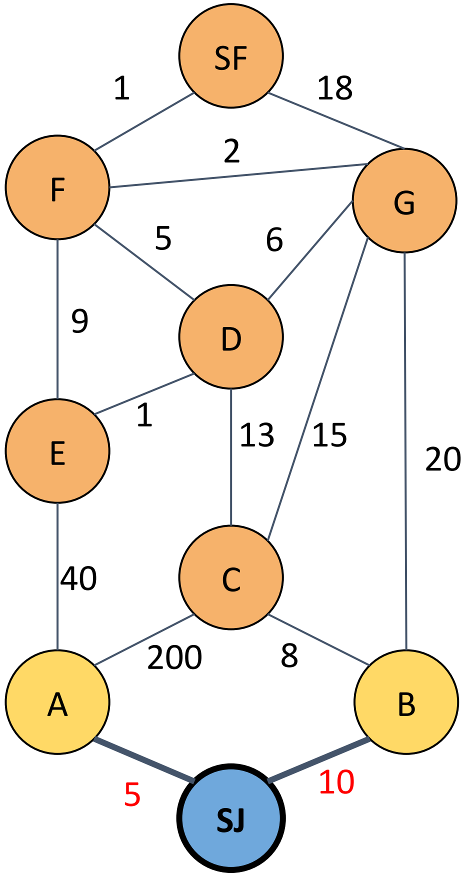
Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node



Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	∞	∞	∞	∞	∞	∞
Previous	-								
Seen?	N	N	N	N	N	N	N	N	N

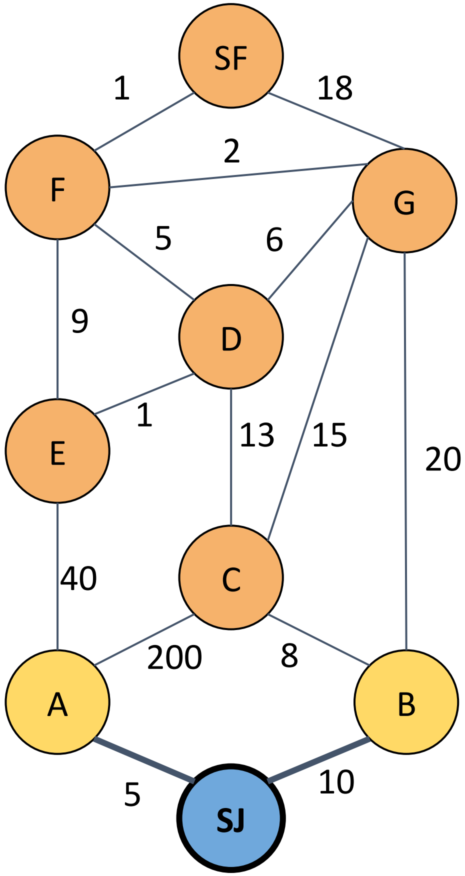
Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node



Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	∞	∞	∞	∞	∞	∞
Previous	-	SJ	SJ						
Seen?	N	N	N	N	N	N	N	N	N

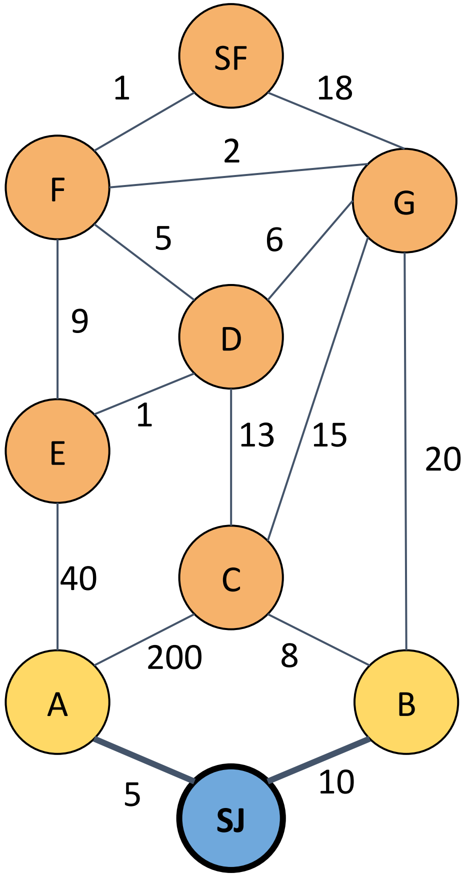
Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node



Dijkstra's Algorithm

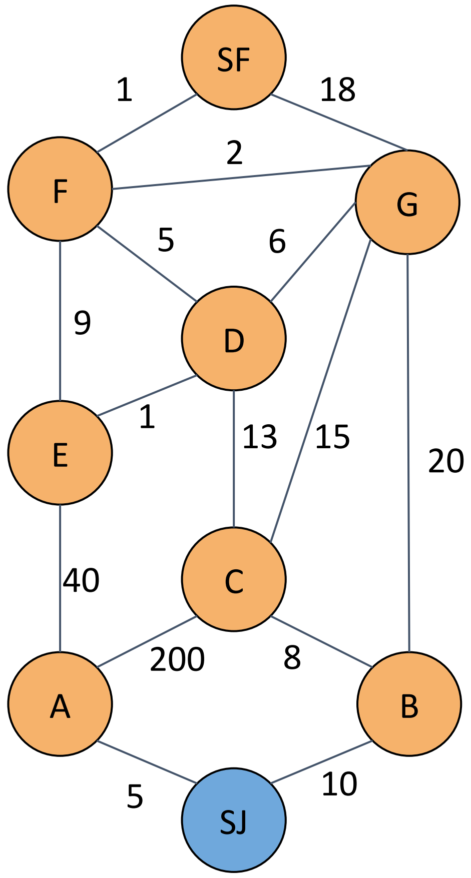
	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	∞	∞	∞	∞	∞	∞
Previous	-	SJ	SJ						
Seen?	Y	N	N	N	N	N	N	N	N

Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node



Dijkstra's Algorithm

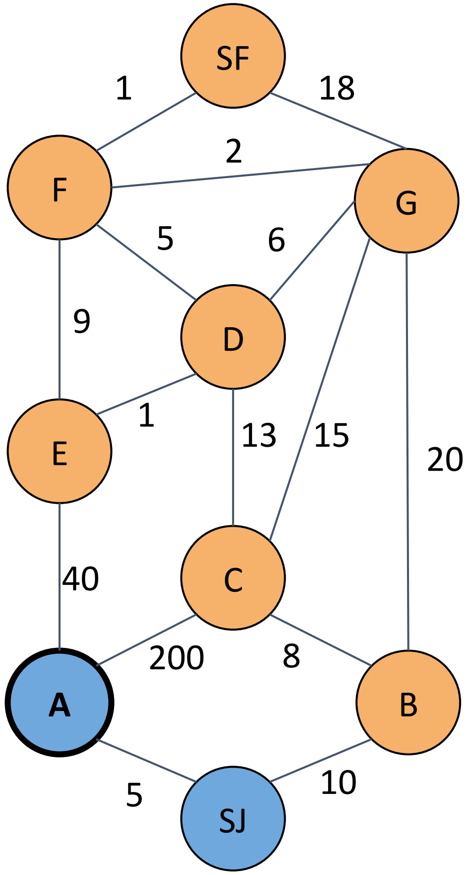
	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	∞	∞	∞	∞	∞	∞
Previous	-	SJ	SJ						
Seen?	Y	N	N	N	N	N	N	N	N



Step 1: Of the unseen nodes, find the node that currently has the shortest distance from the start

Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	∞	∞	∞	∞	∞	∞
Previous	-	SJ	SJ						
Seen?	Y	N	N	N	N	N	N	N	N

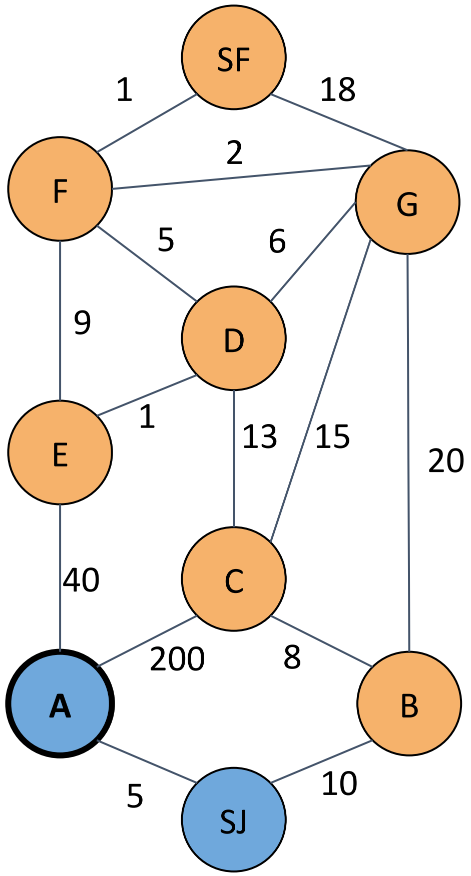


Step 1: Of the unseen nodes, find the node that currently has the shortest distance from the start

Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	∞	∞	∞	∞	∞	∞
Previous	-	SJ	SJ						
Seen?	Y	N	N	N	N	N	N	N	N

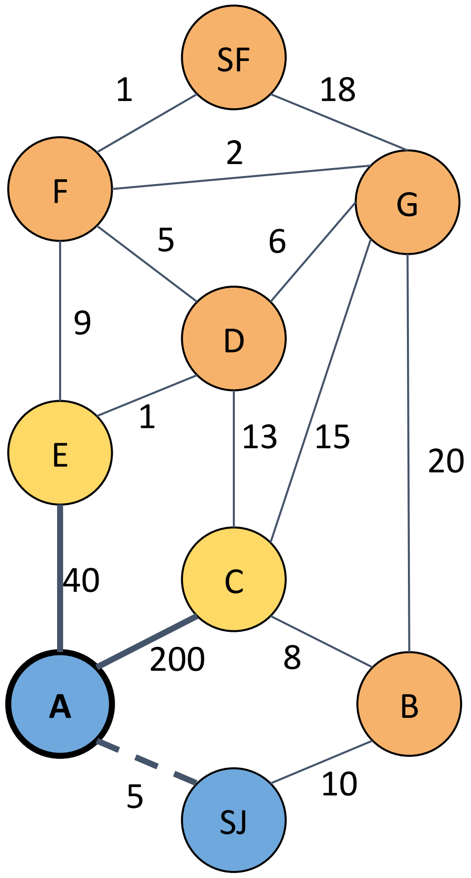
Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node



Dijkstra's Algorithm

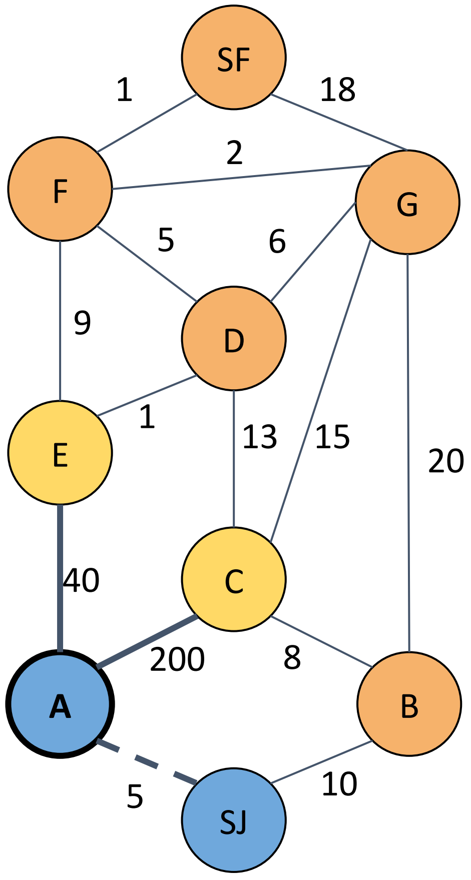
	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	∞	∞	∞	∞	∞	∞
Previous	-	SJ	SJ						
Seen?	Y	N	N	N	N	N	N	N	N

Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node



Dijkstra's Algorithm

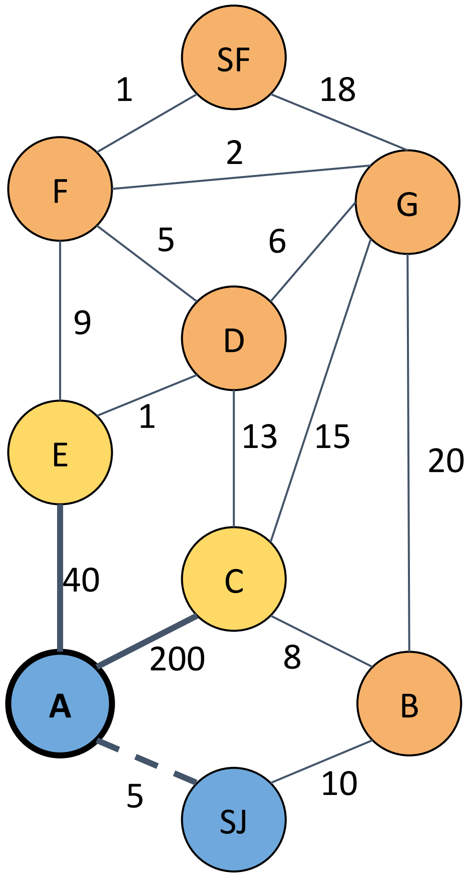
	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	205	∞	45	∞	∞	∞
Previous	-	SJ	SJ						
Seen?	Y	N	N	N	N	N	N	N	N



Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node

Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	205	∞	45	∞	∞	∞
Previous	-	SJ	SJ	A		A			
Seen?	Y	N	N	N	N	N	N	N	N

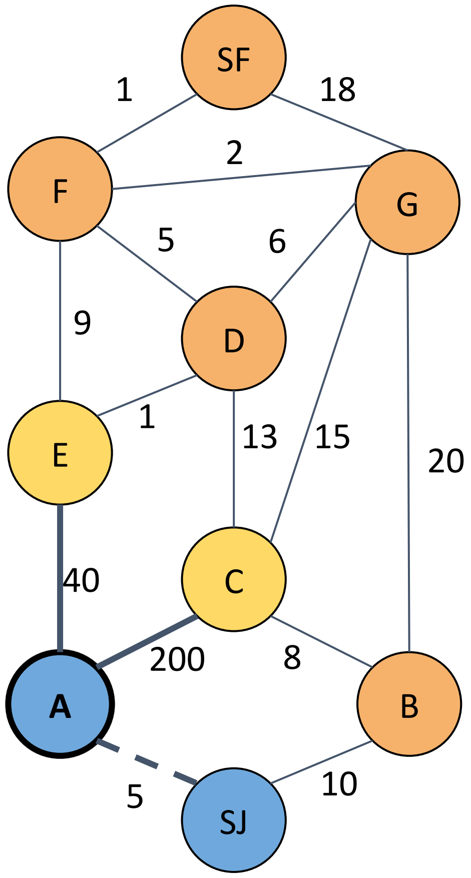


Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node

Dijkstra's Algorithm

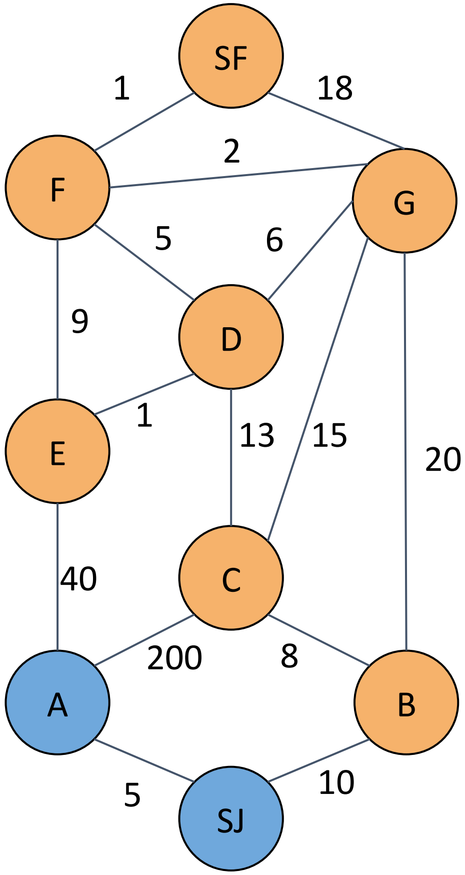
	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	205	∞	45	∞	∞	∞
Previous	-	SJ	SJ	A		A			
Seen?	Y	Y	N	N	N	N	N	N	N

Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node



Dijkstra's Algorithm

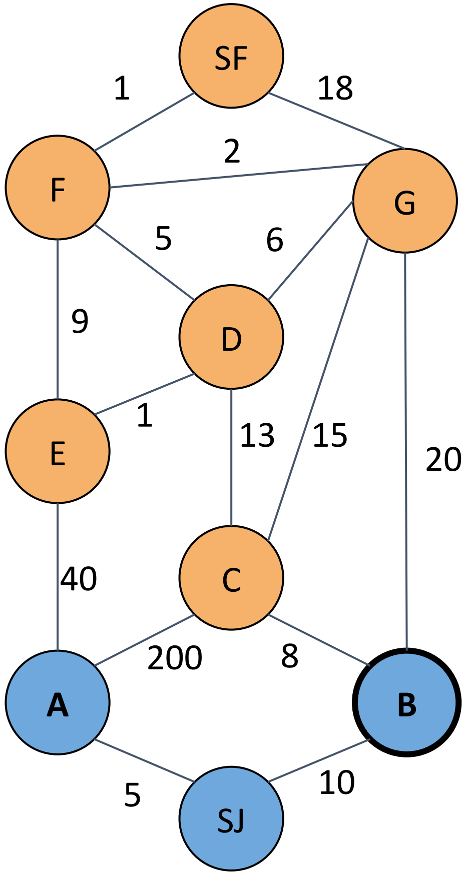
	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	205	∞	45	∞	∞	∞
Previous	-	SJ	SJ	A		A			
Seen?	Y	Y	N	N	N	N	N	N	N



Step 1: Of the unseen nodes, find the node that currently has the shortest distance from the start

Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	205	∞	45	∞	∞	∞
Previous	-	SJ	SJ	A		A			
Seen?	Y	Y	N	N	N	N	N	N	N

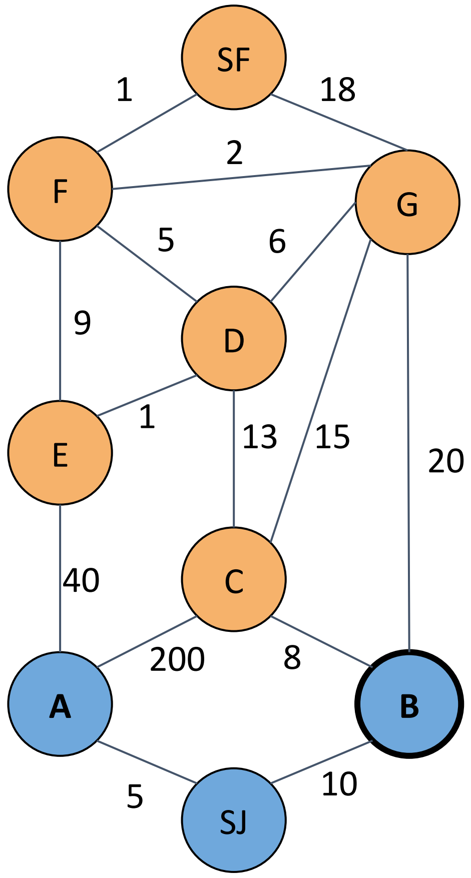


Step 1: Of the unseen nodes, find the node that currently has the shortest distance from the start

Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	205	∞	45	∞	∞	∞
Previous	-	SJ	SJ	A		A			
Seen?	Y	Y	N	N	N	N	N	N	N

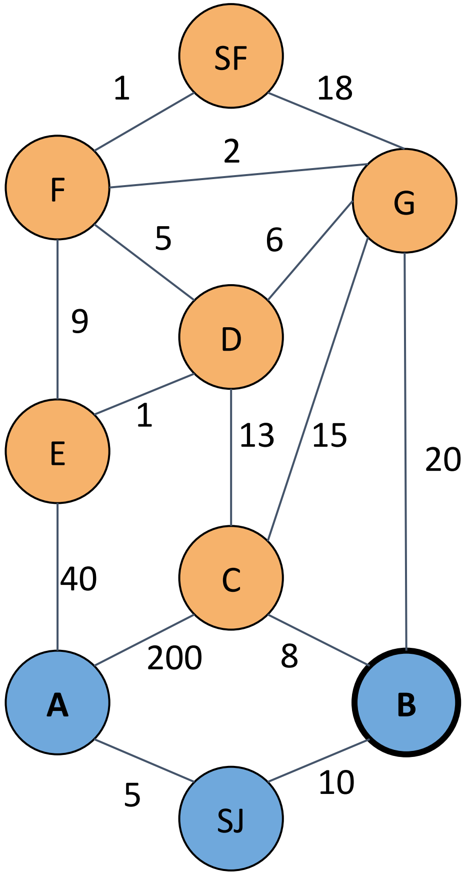
Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node



Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	205	∞	45	∞	∞	∞
Previous	-	SJ	SJ	A		A			
Seen?	Y	Y	N	N	N	N	N	N	N

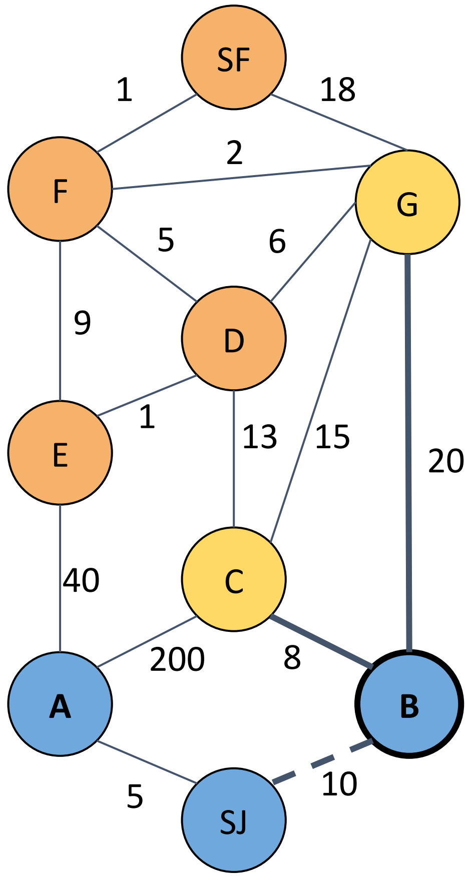
Which nodes' distances will be updated next in the table and what will those distances become?



Dijkstra's Algorithm

	SJ	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**SF**
Distance from start	0	5	10	205	∞	45	∞	∞	∞
Previous	-	SJ	SJ	A		A			
Seen?	Y	Y	N	N	N	N	N	N	N

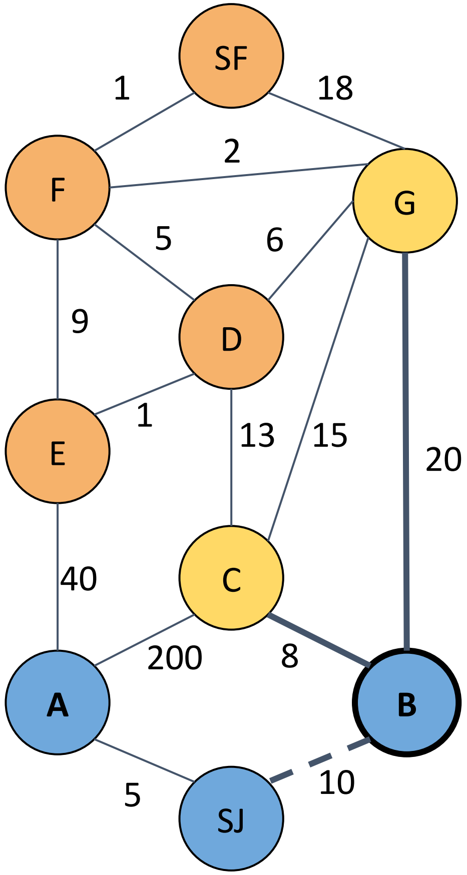
Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node



Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	18	∞	45	∞	30	∞
Previous	-	SJ	SJ	A		A			
Seen?	Y	Y	N	N	N	N	N	N	N

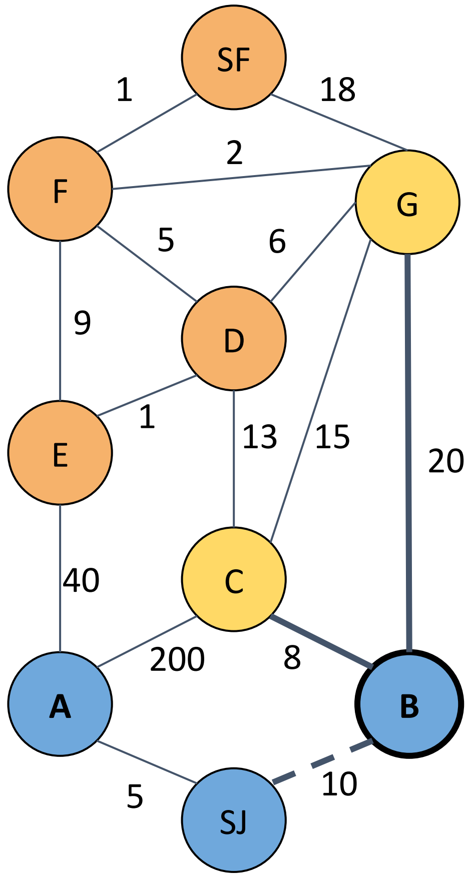
Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node



Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	18	∞	45	∞	30	∞
Previous	-	SJ	SJ	B		A		B	
Seen?	Y	Y	N	N	N	N	N	N	N

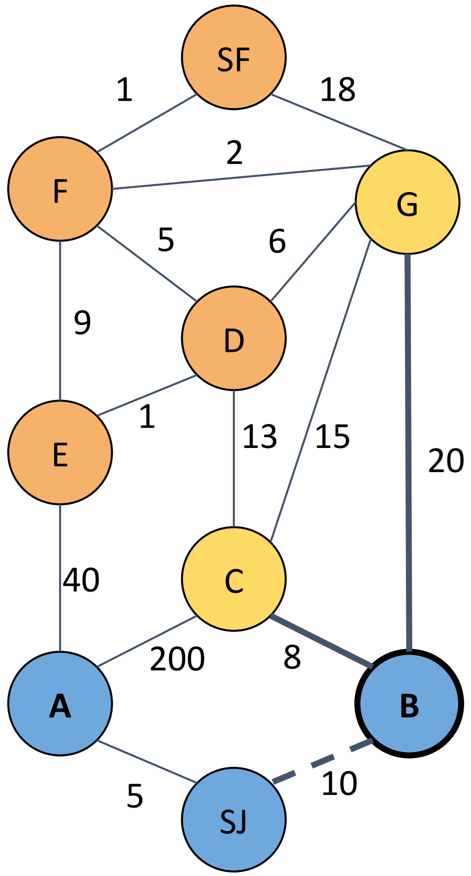
Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node



Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	18	∞	45	∞	30	∞
Previous	-	SJ	SJ	B		A		B	
Seen?	Y	Y	Y	N	N	N	N	N	N

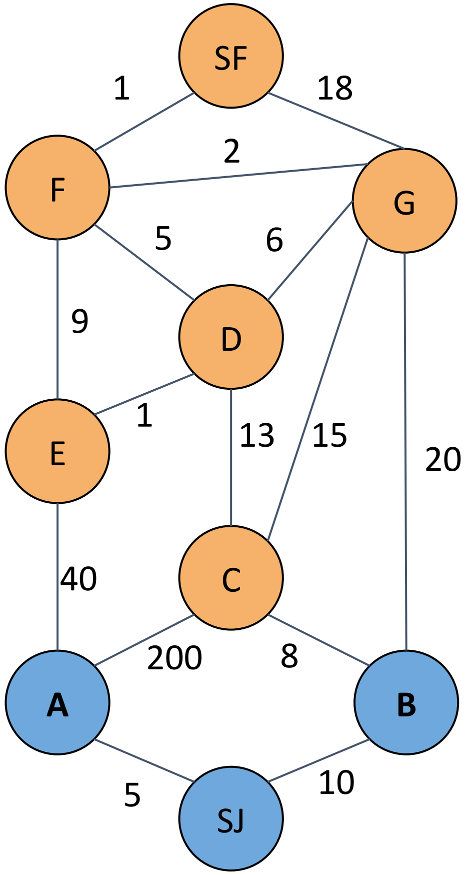
Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node



Dijkstra's Algorithm

	SJ	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**SF**
Distance from start	0	5	10	18	∞	45	∞	30	∞
Previous	-	SJ	SJ	B		A		B	
Seen?	Y	Y	Y	N	N	N	N	N	N

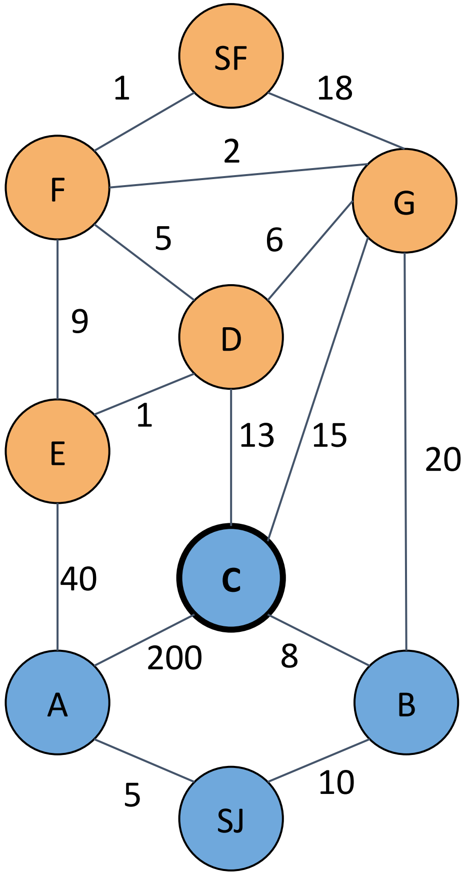
Step 1: Of the unseen nodes, find the node that currently has the shortest distance from the start



Dijkstra's Algorithm

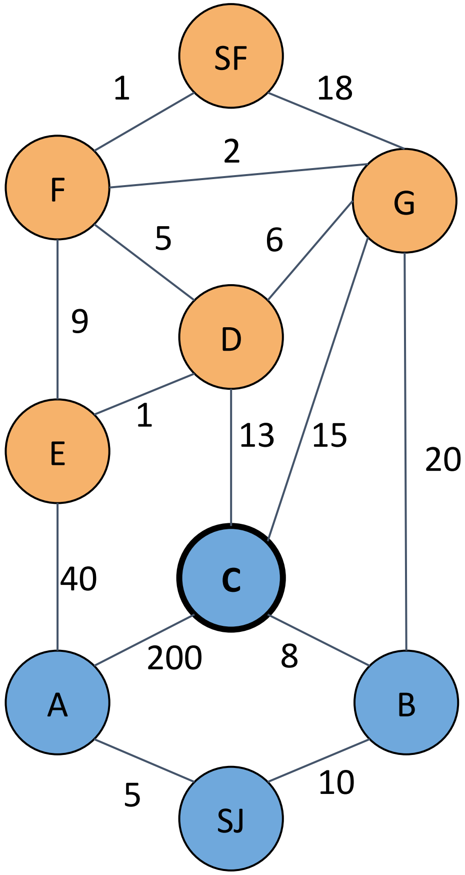
	SJ	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**SF**
Distance from start	0	5	10	18	∞	45	∞	30	∞
Previous	-	SJ	SJ	B		A		B	
Seen?	Y	Y	Y	N	N	N	N	N	N

Step 1: Of the unseen nodes, find the node that currently has the shortest distance from the start



Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	18	∞	45	∞	30	∞
Previous	-	SJ	SJ	B		A		B	
Seen?	Y	Y	Y	N	N	N	N	N	N

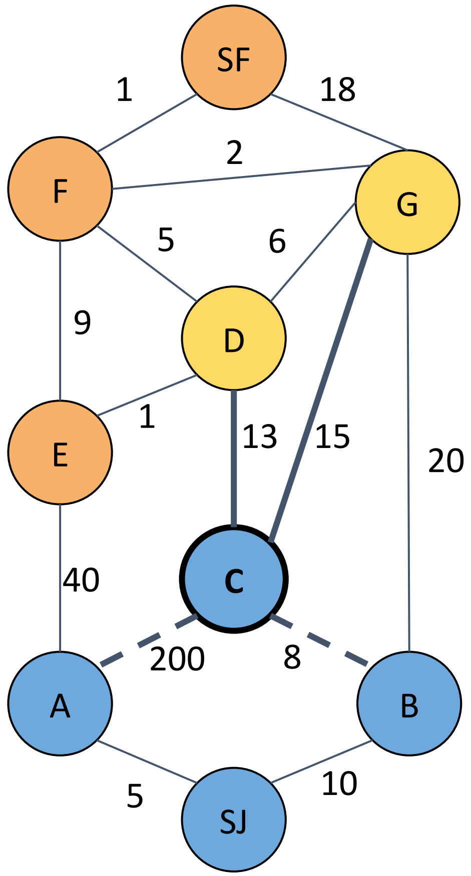


Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node

Dijkstra's Algorithm

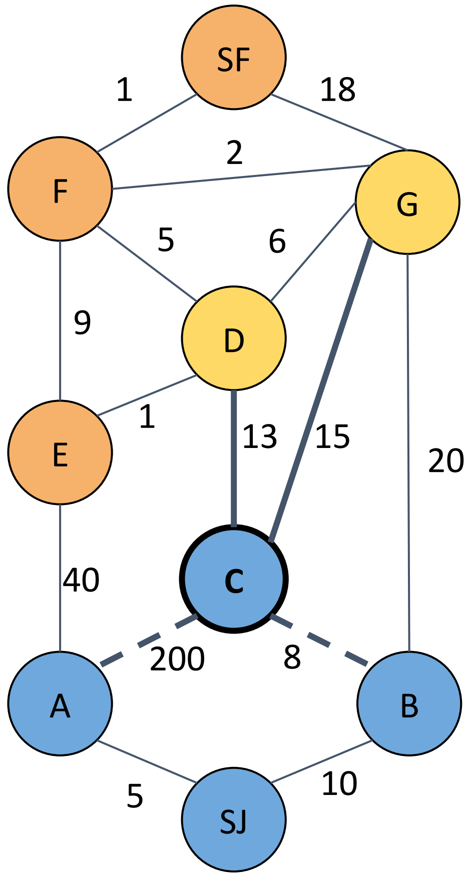
	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	18	∞	45	∞	30	∞
Previous	-	SJ	SJ	B		A		B	
Seen?	Y	Y	Y	N	N	N	N	N	N

Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node



Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	18	31	45	∞	30	∞
Previous	-	SJ	SJ	B		A		B	
Seen?	Y	Y	Y	N	N	N	N	N	N

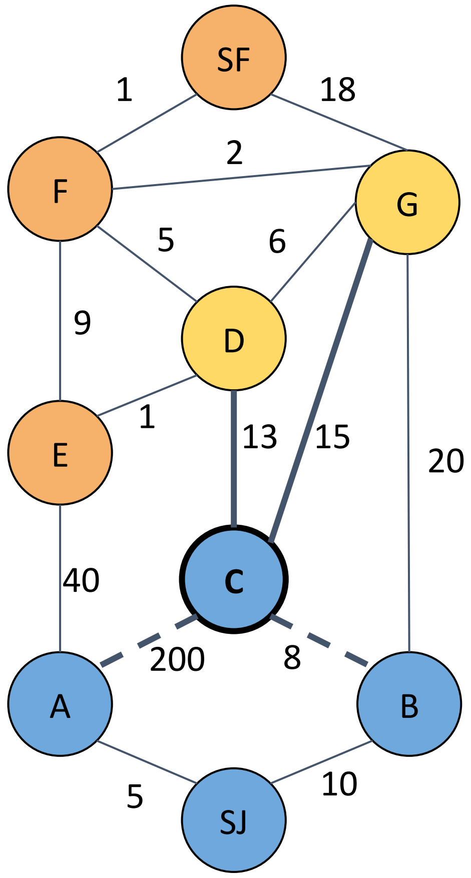


Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node

Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	18	31	45	∞	30	∞
Previous	-	SJ	SJ	B	C	A		B	
Seen?	Y	Y	Y	N	N	N	N	N	N

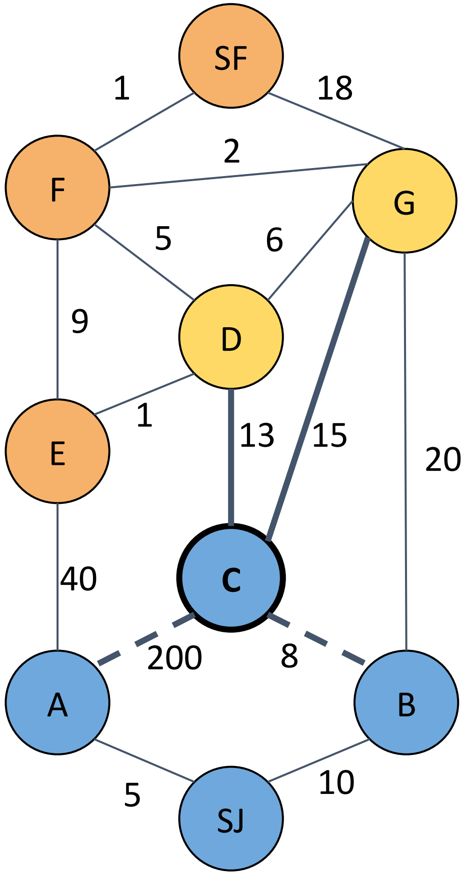
Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node



Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	18	31	45	∞	30	∞
Previous	-	SJ	SJ	B	C	A		B	
Seen?	Y	Y	Y	Y	N	N	N	N	N

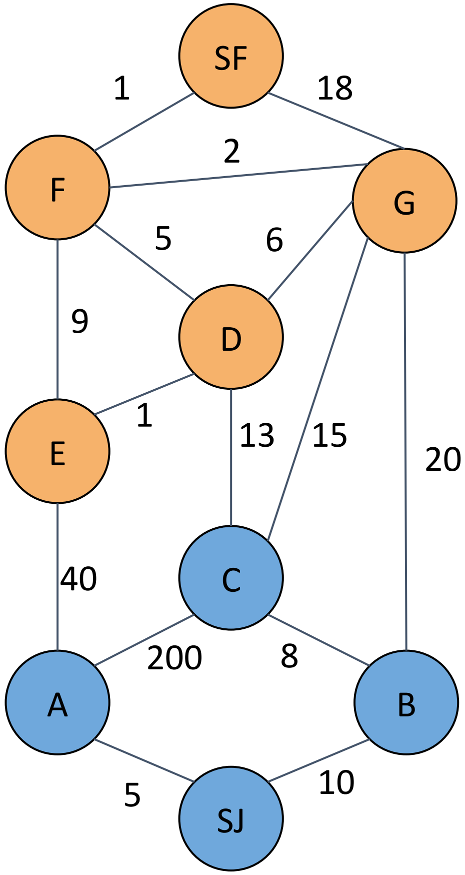
Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node



Dijkstra's Algorithm

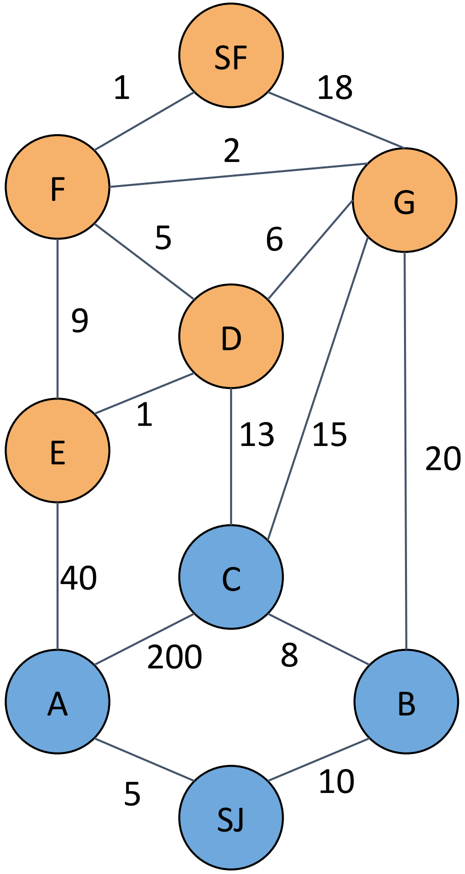
	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	18	31	45	∞	30	∞
Previous	-	SJ	SJ	B	C	A		B	
Seen?	Y	Y	Y	Y	N	N	N	N	N

Try to find the shortest weighted path from SJ



Dijkstra's Algorithm

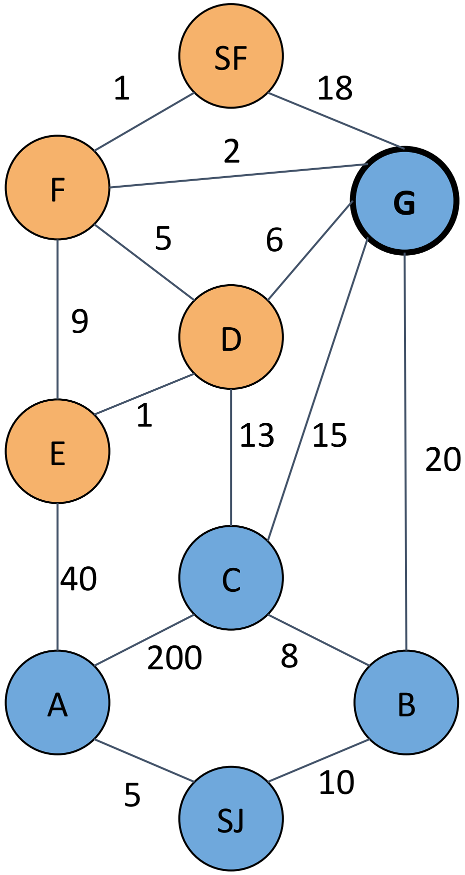
	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	18	31	45	∞	30	∞
Previous	-	SJ	SJ	B	C	A		B	
Seen?	Y	Y	Y	Y	N	N	N	N	N



Step 1: Of the unseen nodes, find the node that currently has the shortest distance from the start

Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	18	31	45	∞	30	∞
Previous	-	SJ	SJ	B	C	A		B	
Seen?	Y	Y	Y	Y	N	N	N	N	N

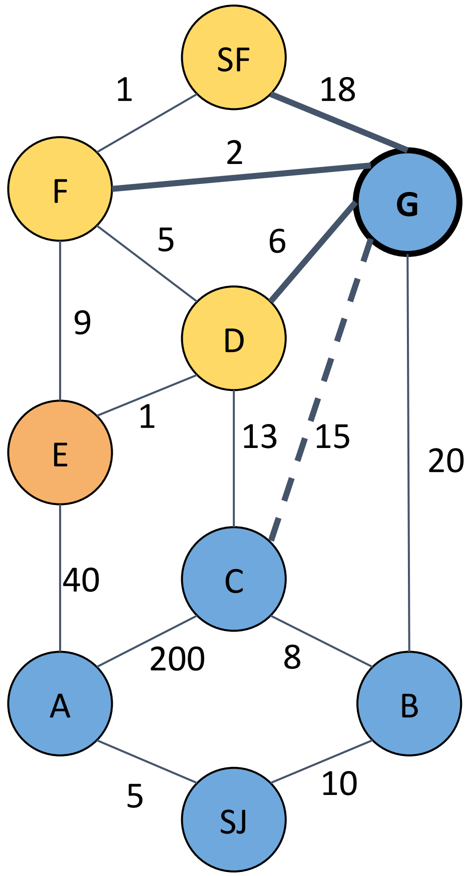


Step 1: Of the unseen nodes, find the node that currently has the shortest distance from the start

Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	18	31	45	∞	30	∞
Previous	-	SJ	SJ	B	C	A		B	
Seen?	Y	Y	Y	Y	N	N	N	N	N

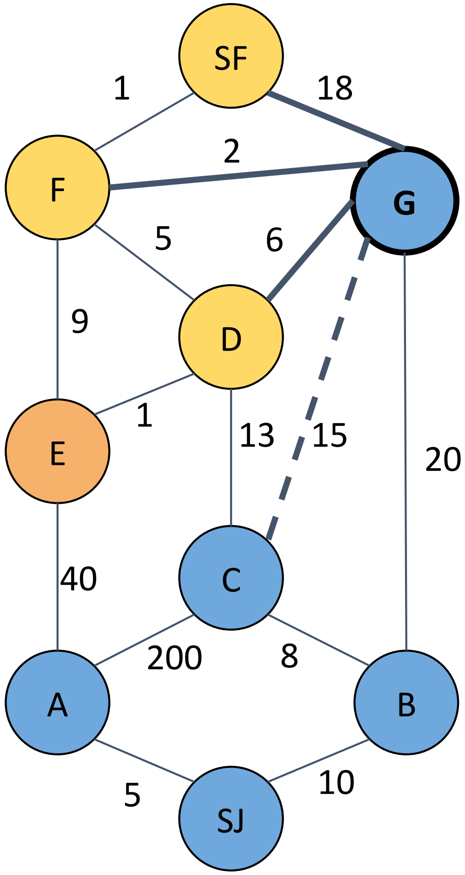
Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node



Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	18	31	45	32	30	48
Previous	-	SJ	SJ	B	C	A		B	
Seen?	Y	Y	Y	Y	N	N	N	N	N

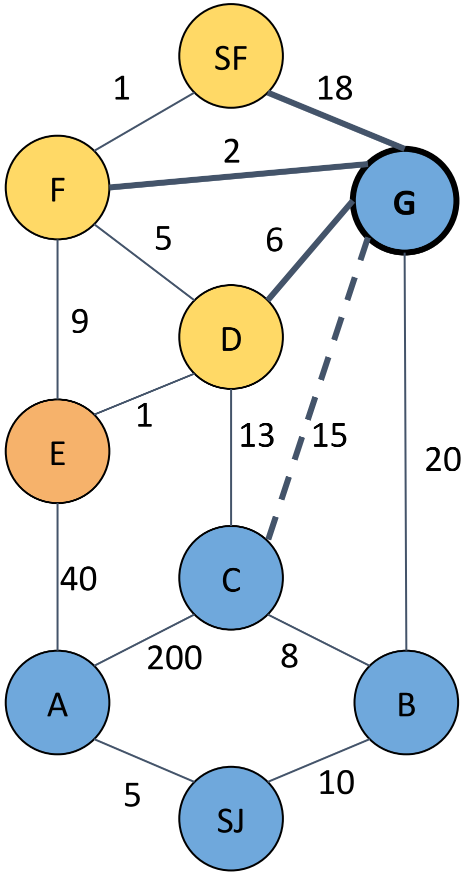
Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node



Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	18	31	45	32	30	48
Previous	-	SJ	SJ	B	C	A	G	B	G
Seen?	Y	Y	Y	Y	N	N	N	N	N

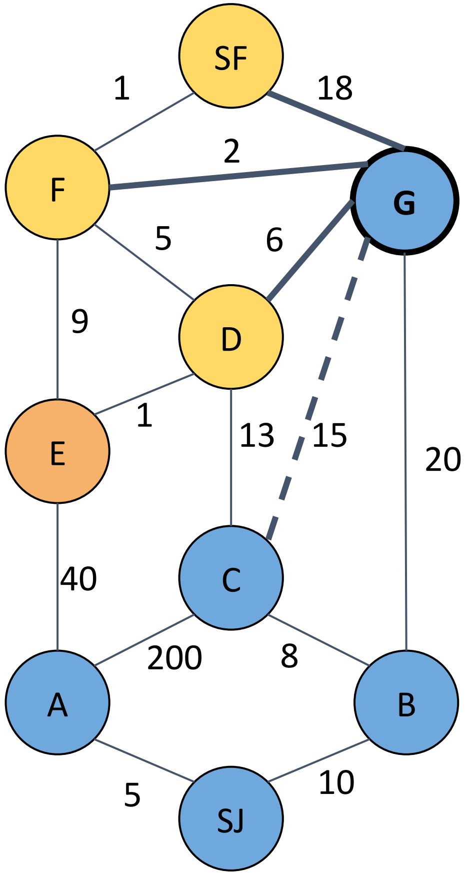
Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node



Dijkstra's Algorithm

	SJ	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**SF**
Distance from start	0	5	10	18	31	45	32	30	48
Previous	-	SJ	SJ	B	C	A	G	B	G
Seen?	Y	Y	Y	Y	N	N	N	Y	N

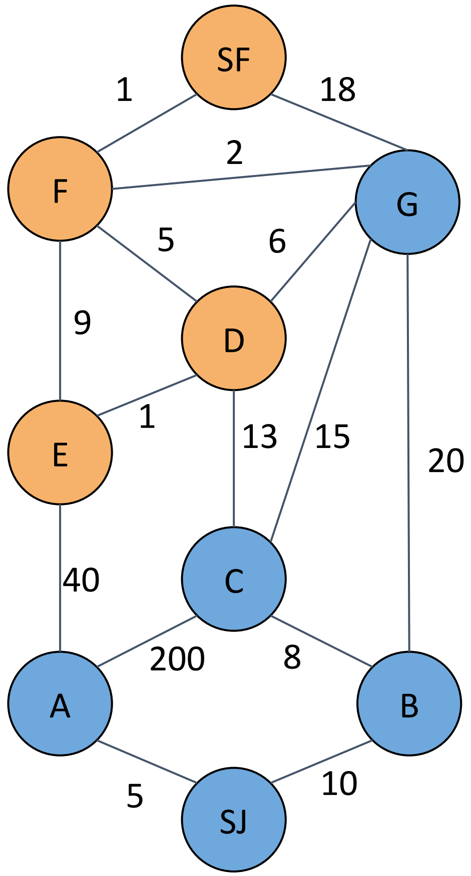
Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node



Dijkstra's Algorithm

	SJ	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**SF**
Distance from start	0	5	10	18	31	45	32	30	48
Previous	-	SJ	SJ	B	C	A	G	B	G
Seen?	Y	Y	Y	Y	N	N	N	Y	N

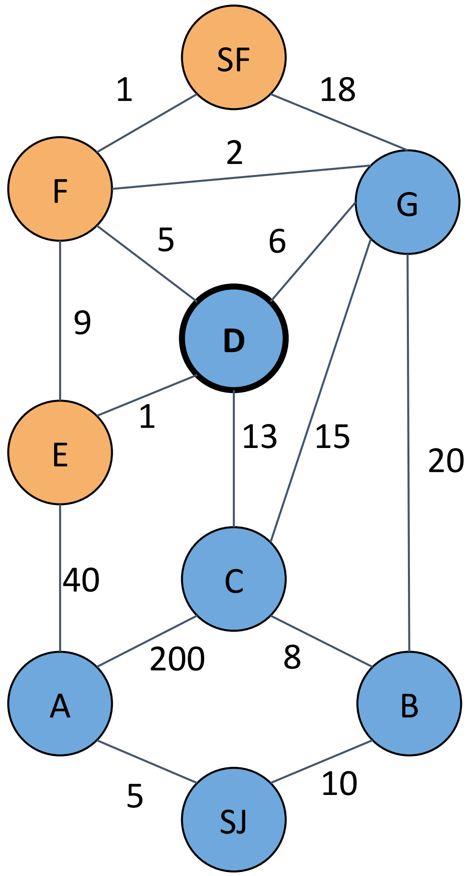
Step 1: Of the unseen nodes, find the node that currently has the shortest distance from the start



Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	18	31	45	32	30	48
Previous	-	SJ	SJ	B	C	A	G	B	G
Seen?	Y	Y	Y	Y	N	N	N	Y	N

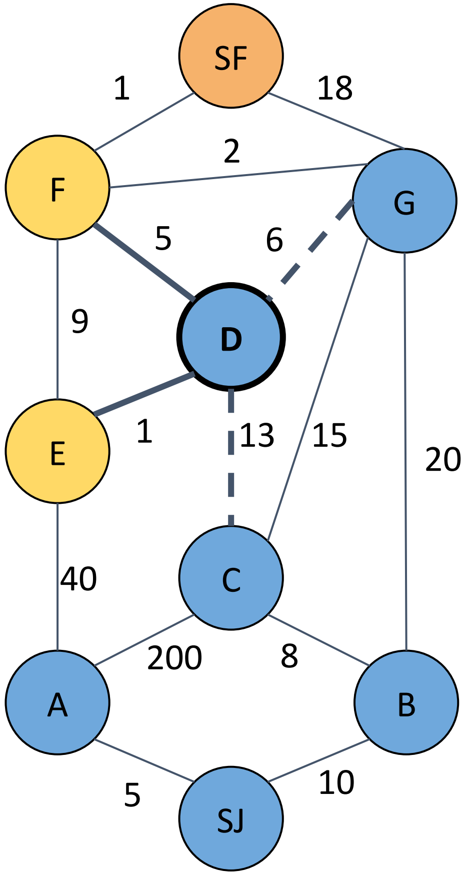
Step 1: Of the unseen nodes, find the node that currently has the shortest distance from the start



Dijkstra's Algorithm

	SJ	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**SF**
Distance from start	0	5	10	18	31	45	32	30	48
Previous	-	SJ	SJ	B	C	A	G	B	G
Seen?	Y	Y	Y	Y	N	N	N	Y	N

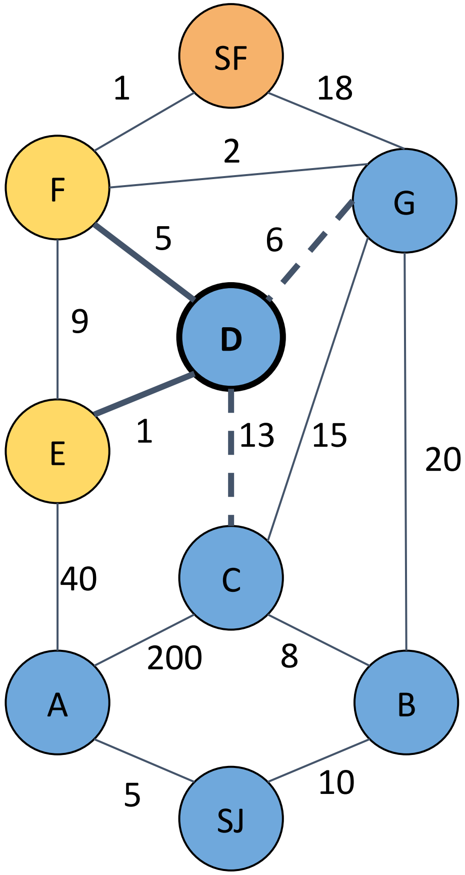
Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node



Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	18	31	32	32	30	48
Previous	-	SJ	SJ	B	C	A	G	B	G
Seen?	Y	Y	Y	Y	N	N	N	Y	N

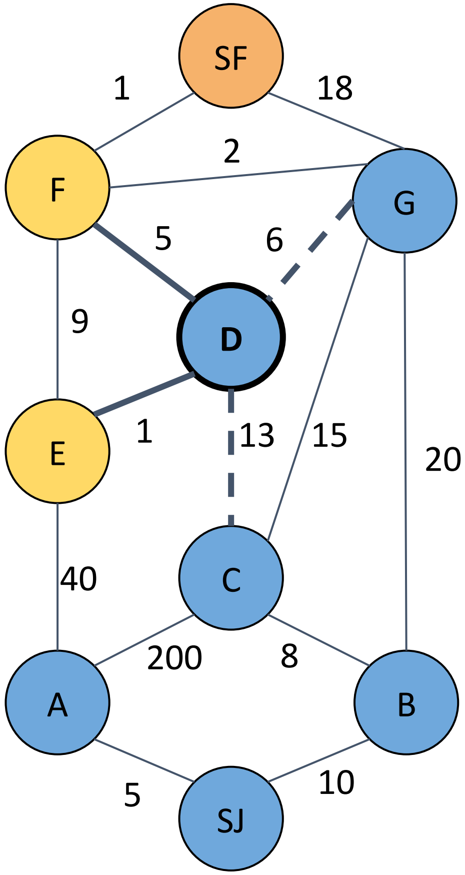
Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node



Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	18	31	32	32	30	48
Previous	-	SJ	SJ	B	C	D	G	B	G
Seen?	Y	Y	Y	Y	N	N	N	Y	N

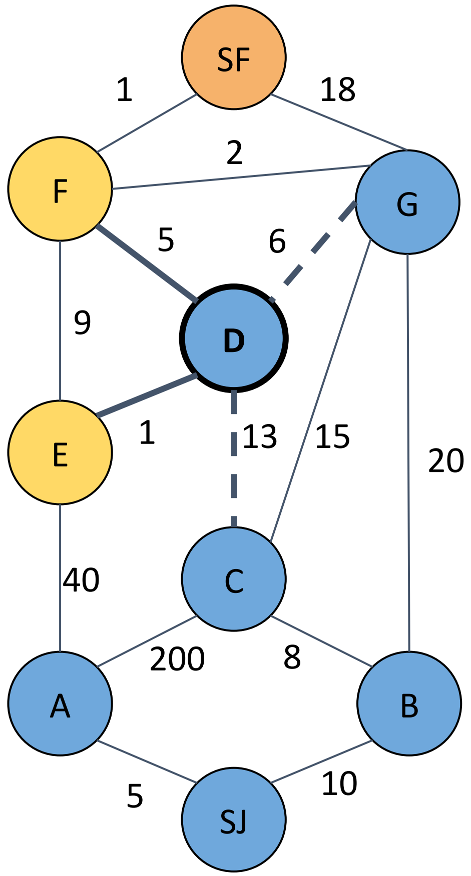
Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node



Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	18	31	32	32	30	48
Previous	-	SJ	SJ	B	C	D	G	B	G
Seen?	Y	Y	Y	Y	Y	N	N	Y	N

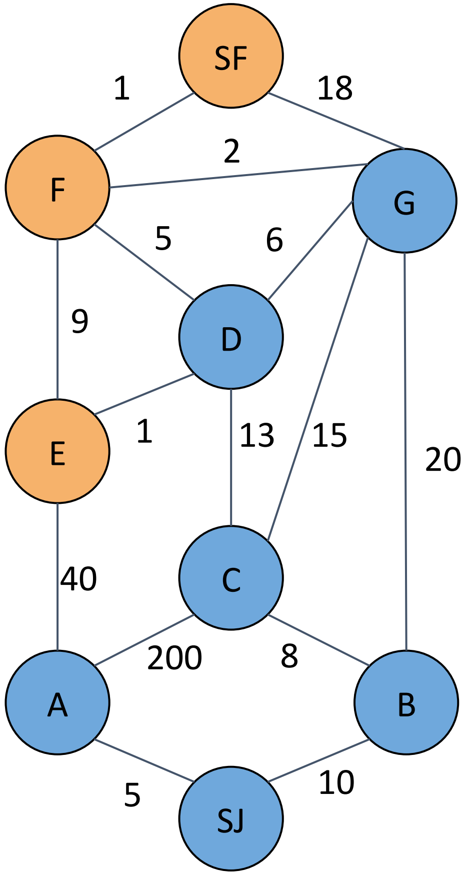
Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node



Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	18	31	32	32	30	48
Previous	-	SJ	SJ	B	C	D	G	B	G
Seen?	Y	Y	Y	Y	Y	N	N	Y	N

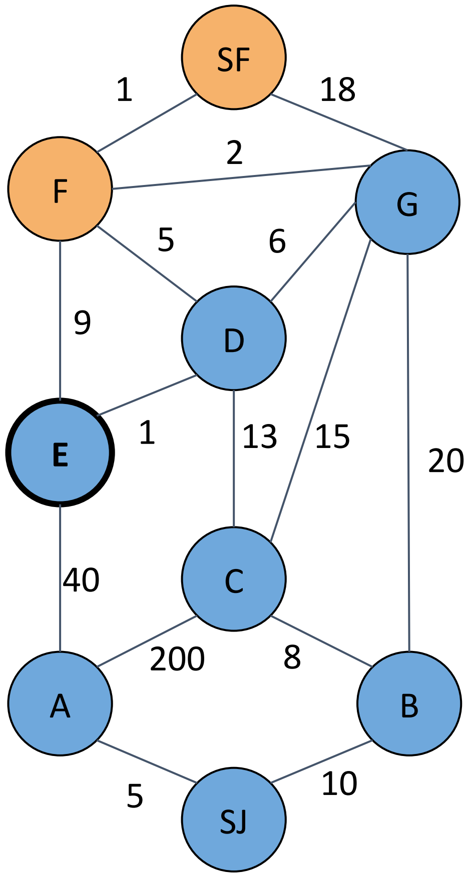
Step 1: Of the unseen nodes, find the node that currently has the shortest distance from the start



Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	18	31	32	32	30	48
Previous	-	SJ	SJ	B	C	D	G	B	G
Seen?	Y	Y	Y	Y	Y	N	N	Y	N

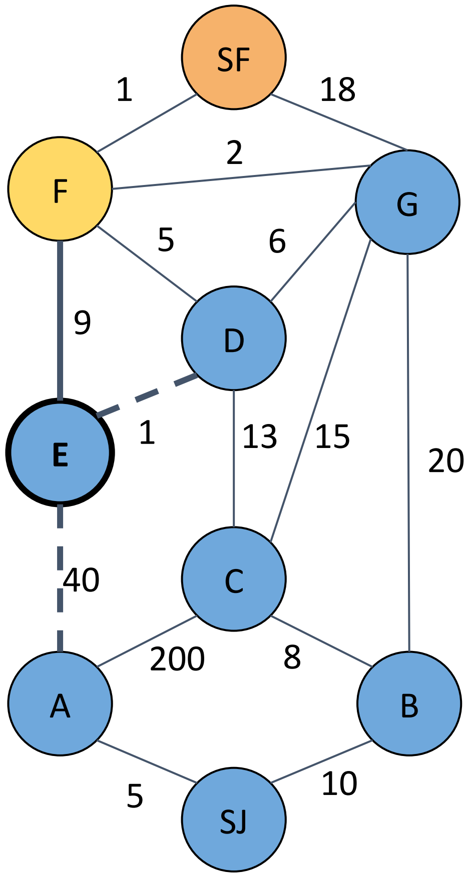
Step 1: Of the unseen nodes, find the node that currently has the shortest distance from the start



Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	18	31	32	32	30	48
Previous	-	SJ	SJ	B	C	D	G	B	G
Seen?	Y	Y	Y	Y	Y	N	N	Y	N

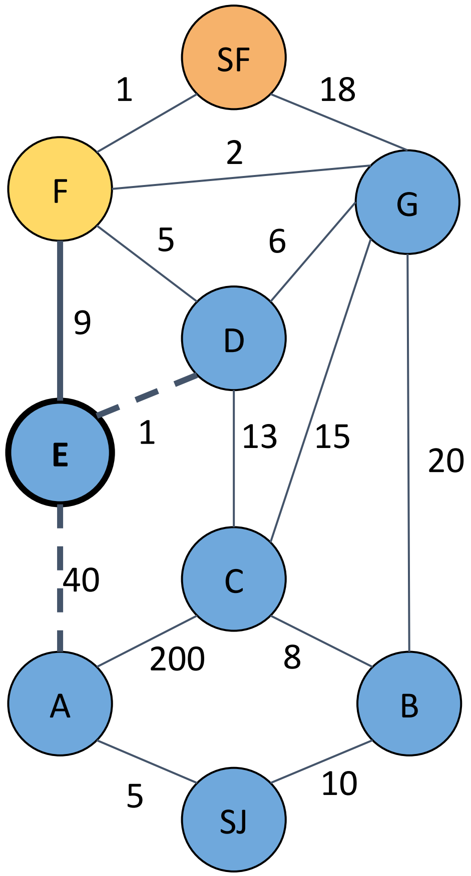
Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node



Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	18	31	32	32	30	48
Previous	-	SJ	SJ	B	C	D	G	B	G
Seen?	Y	Y	Y	Y	Y	Y	N	Y	N

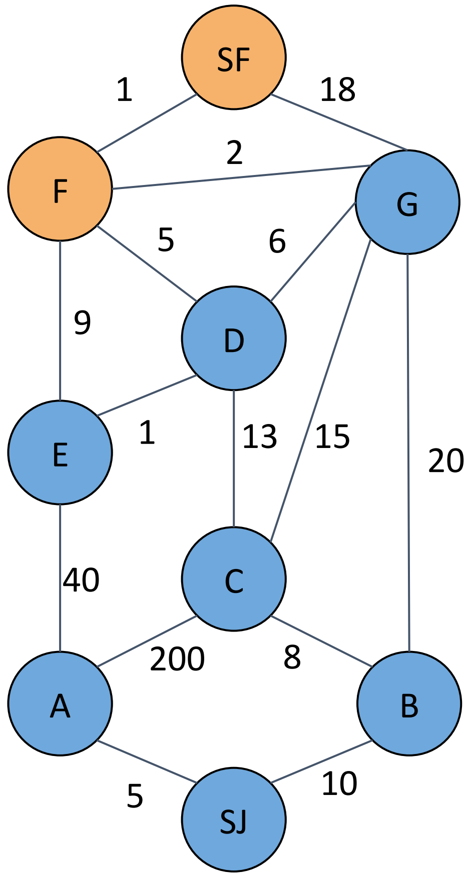
Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node



Dijkstra's Algorithm

	SJ	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**SF**
Distance from start	0	5	10	18	31	32	32	30	48
Previous	-	SJ	SJ	B	C	D	G	B	G
Seen?	Y	Y	Y	Y	Y	Y	N	Y	N

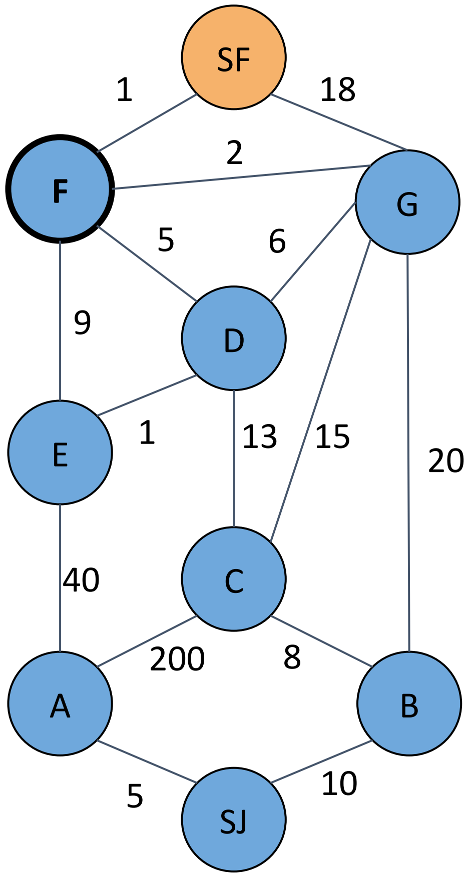
Step 1: Of the unseen nodes, find the node that currently has the shortest distance from the start



Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	18	31	32	32	30	48
Previous	-	SJ	SJ	B	C	D	G	B	G
Seen?	Y	Y	Y	Y	Y	Y	N	Y	N

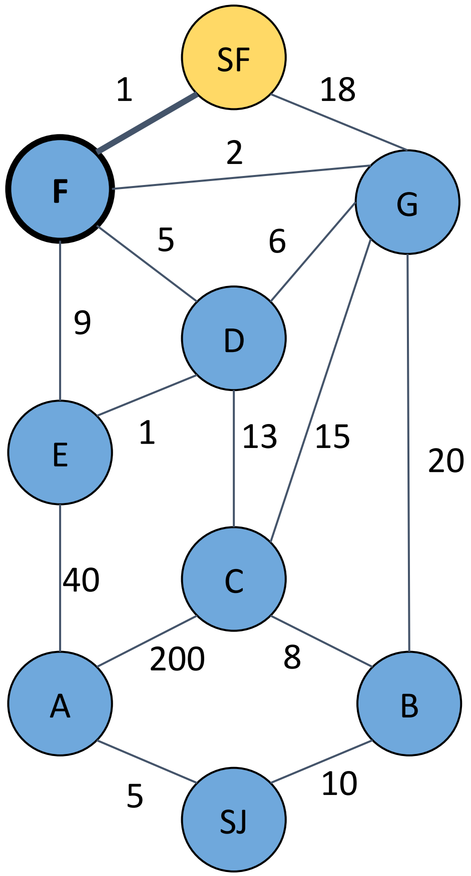
Step 1: Of the unseen nodes, find the node that currently has the shortest distance from the start



Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	18	31	32	32	30	48
Previous	-	SJ	SJ	B	C	D	G	B	G
Seen?	Y	Y	Y	Y	Y	Y	N	Y	N

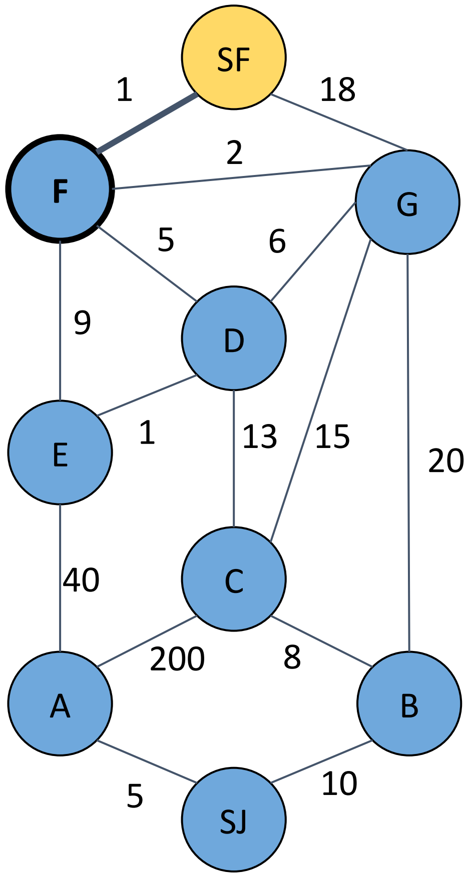
Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node



Dijkstra's Algorithm

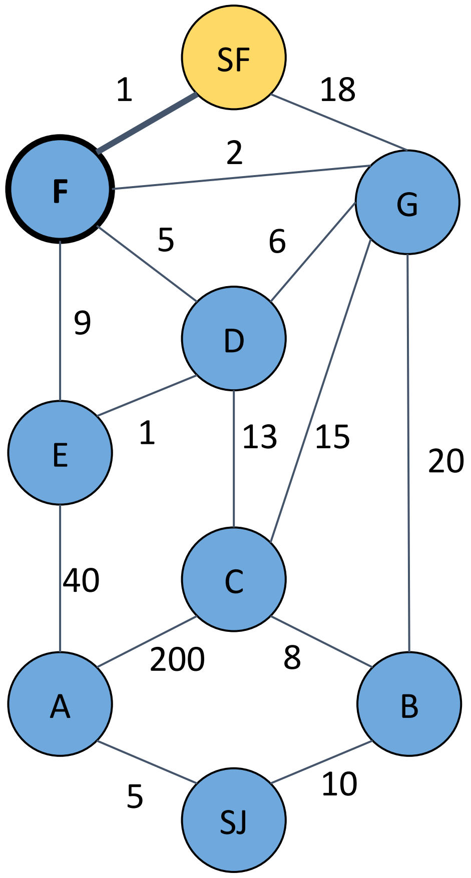
	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	18	31	32	32	30	33
Previous	-	SJ	SJ	B	C	D	G	B	G
Seen?	Y	Y	Y	Y	Y	Y	N	Y	N

Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node



Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	18	31	32	32	30	33
Previous	-	SJ	SJ	B	C	D	G	B	F
Seen?	Y	Y	Y	Y	Y	Y	N	Y	N

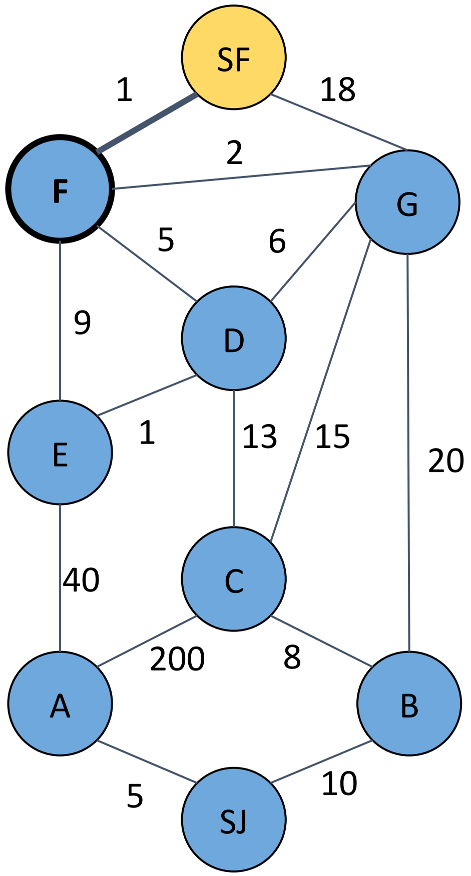


Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node

Dijkstra's Algorithm

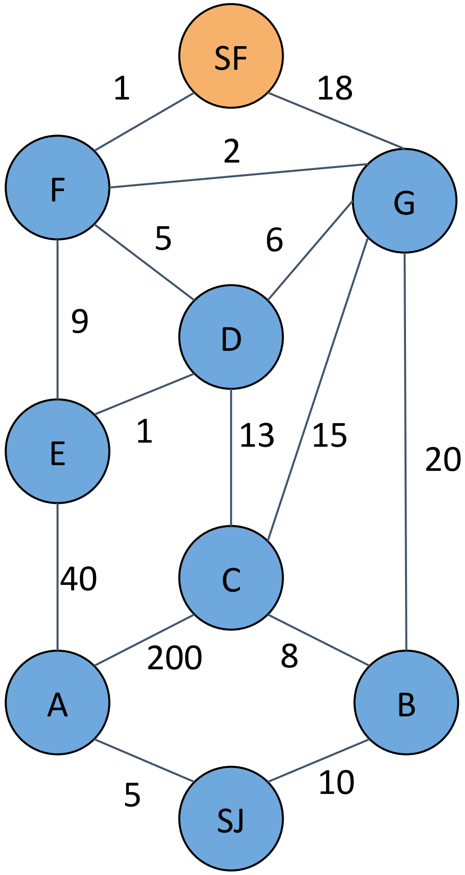
	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	18	31	32	32	30	33
Previous	-	SJ	SJ	B	C	D	G	B	F
Seen?	Y	Y	Y	Y	Y	Y	Y	Y	N

Step 2: Look at this node's neighbors, and update the total distance to the neighbors based on their distance and the distance already to this node



Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	18	31	32	32	30	33
Previous	-	SJ	SJ	B	C	D	G	B	F
Seen?	Y	Y	Y	Y	Y	Y	Y	Y	N

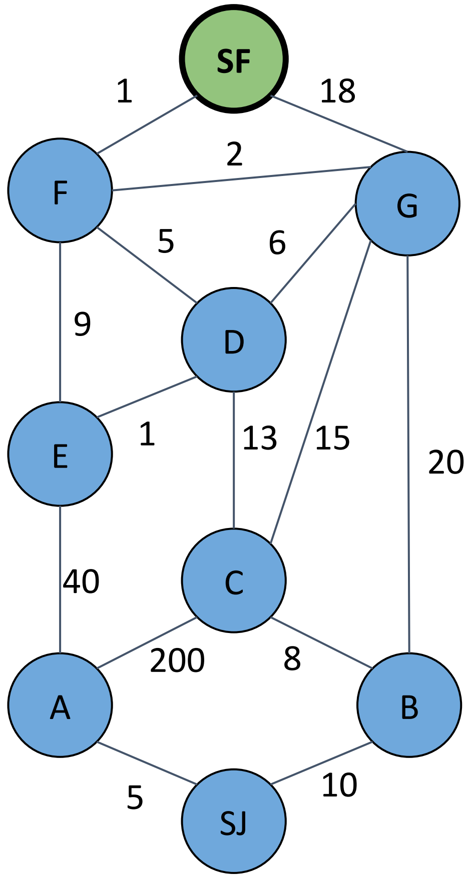


Step 1: Of the unseen nodes, find the node that currently has the shortest distance from the start

Dijkstra's Algorithm

	SJ	A	B	C	D	E	F	G	SF
Distance from start	0	5	10	18	31	32	32	30	33
Previous	-	SJ	SJ	B	C	D	G	B	F
Seen?	Y	Y	Y	Y	Y	Y	Y	Y	N

Step 1: Of the unseen nodes, find the node that currently has the shortest distance from the start

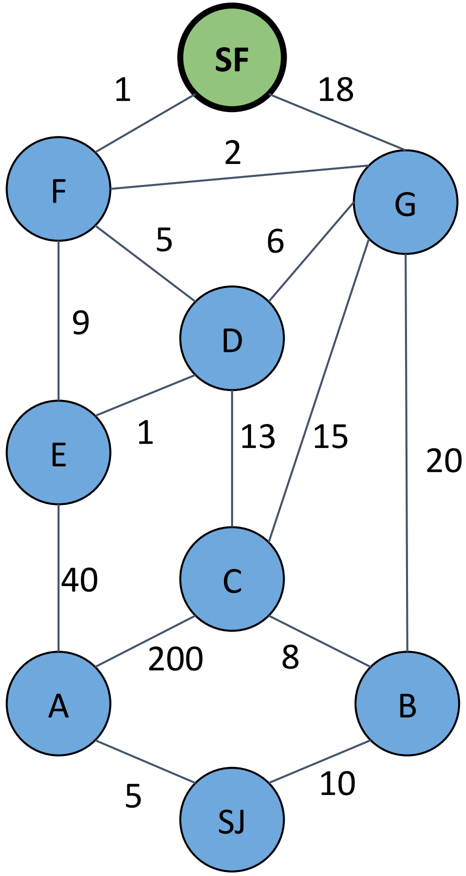


Dijkstra's Algorithm

	SJ	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**SF**
Distance from start	0	5	10	18	31	32	32	30	33
Previous	-	SJ	SJ	B	C	D	G	B	F
Seen?	Y	Y	Y	Y	Y	Y	Y	Y	N

We're done! Shortest weighted path is of length **33**

with a path of ??

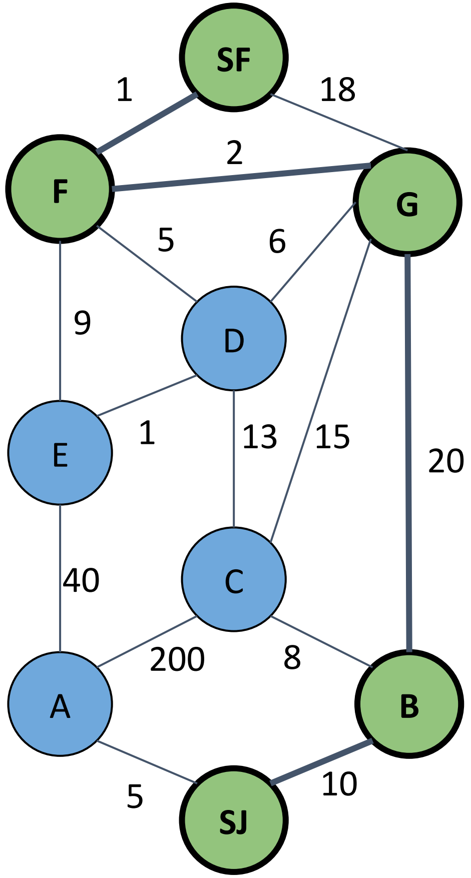


Dijkstra's Algorithm

	SJ	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**SF**
Distance from start	0	5	10	18	31	32	32	30	33
Previous	-	SJ	SJ	B	C	D	G	B	F
Seen?	Y	Y	Y	Y	Y	Y	Y	Y	N

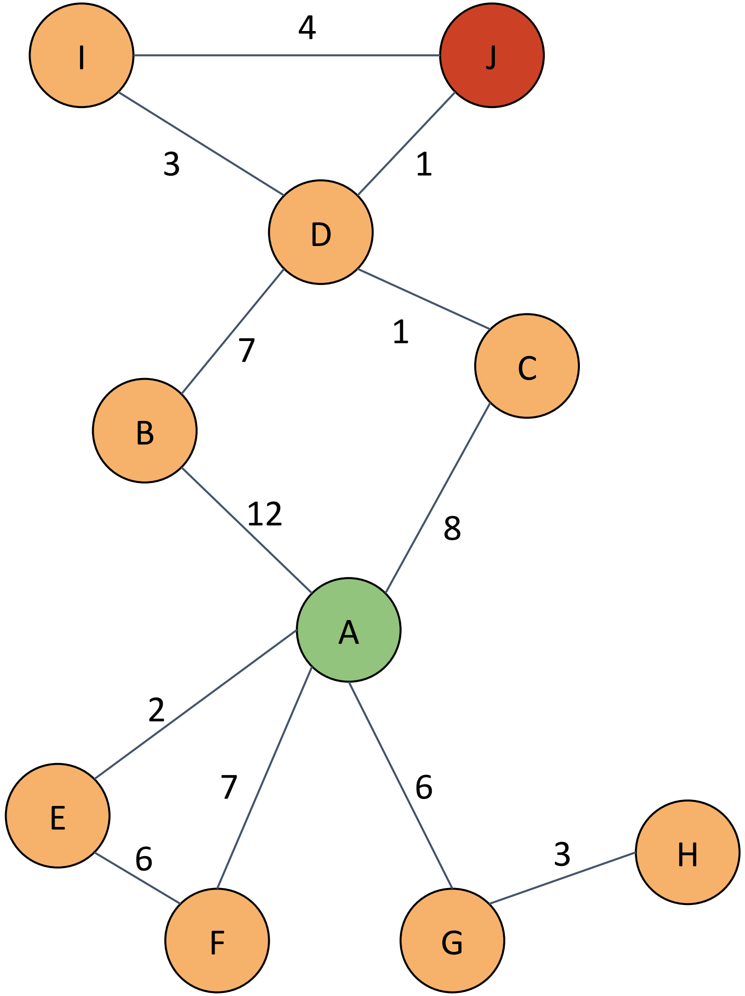
We're done! Shortest weighted path is of length **33**

with a path of SJ⟶B⟶G⟶F⟶SF



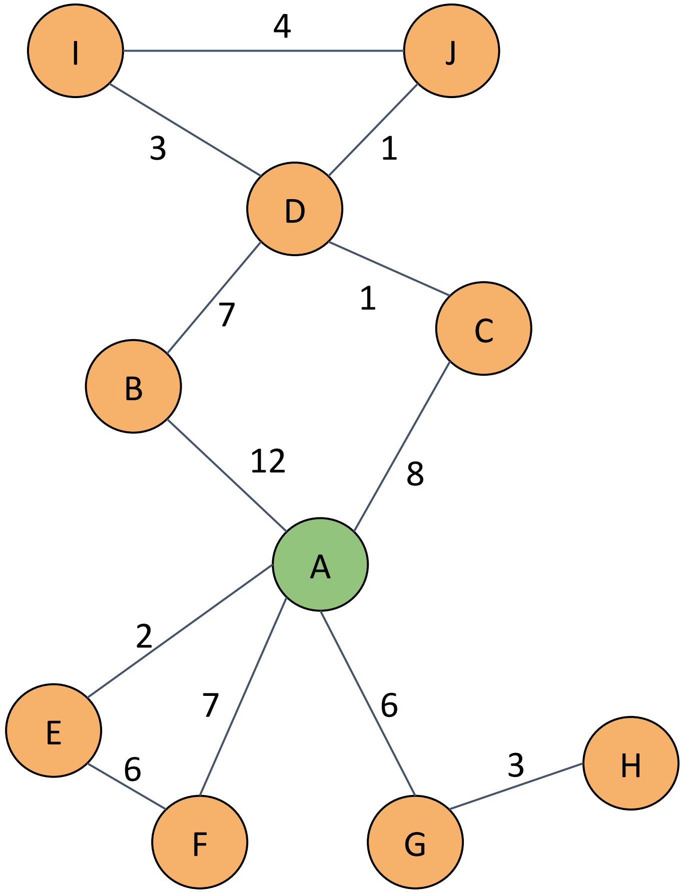
Demo

https://bit.ly/graph_demo


I
J
4
3
1
D
7
1
C
B
12
8
A
2
7
6
E
6
3
H
F
G

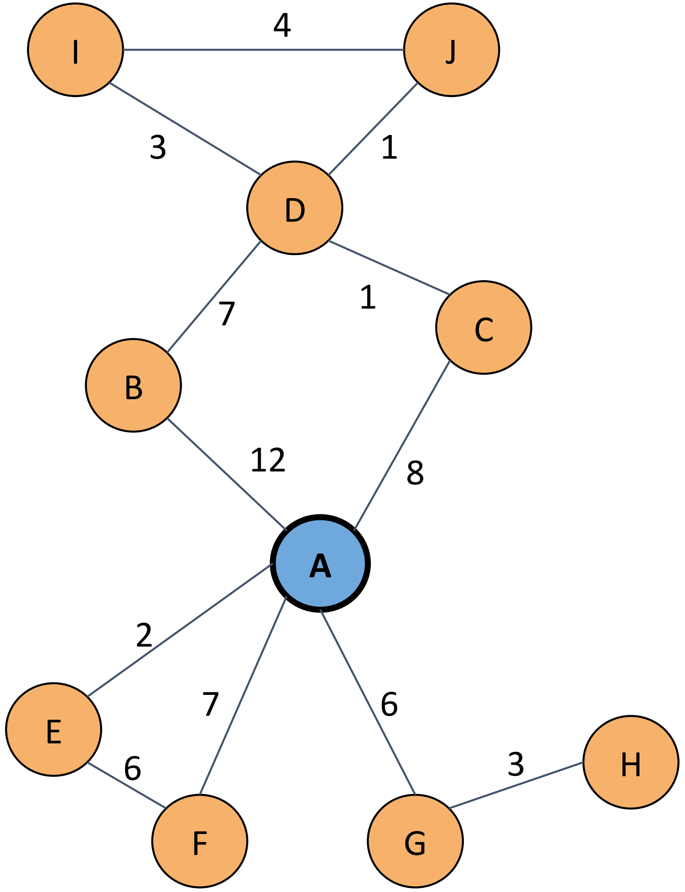
Dijkstra's Algorithm

	A	B	C	D	E	F	G	H	I	J
Distance from start	0	∞	∞	∞	∞	∞	∞	∞	∞	∞
Previous	-									
Seen?	N	N	N	N	N	N	N	N	N	N



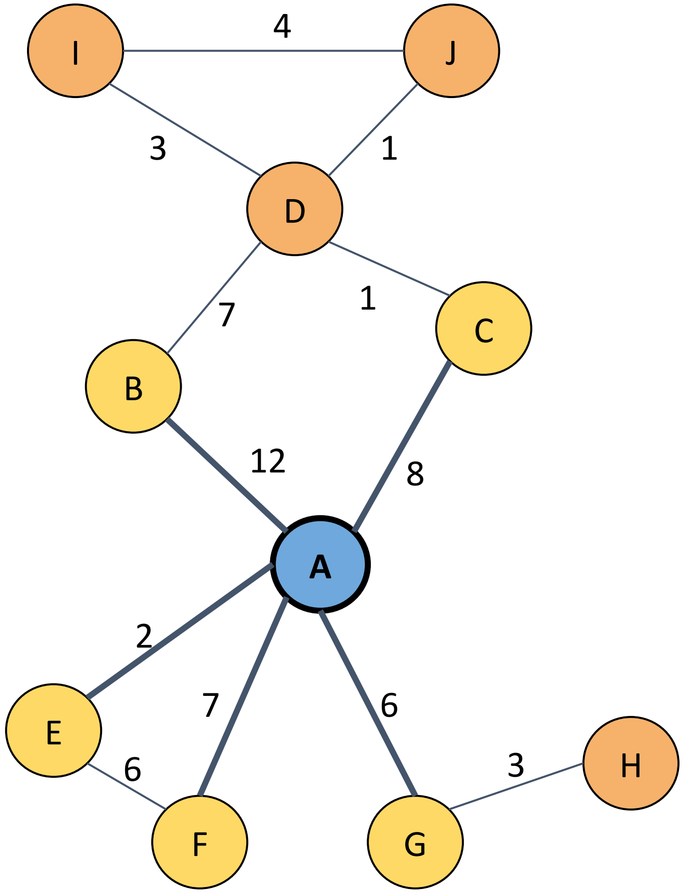
Dijkstra's Algorithm

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
Distance from start	0	∞	∞	∞	∞	∞	∞	∞	∞	∞
Previous	-									
Seen?	N	N	N	N	N	N	N	N	N	N



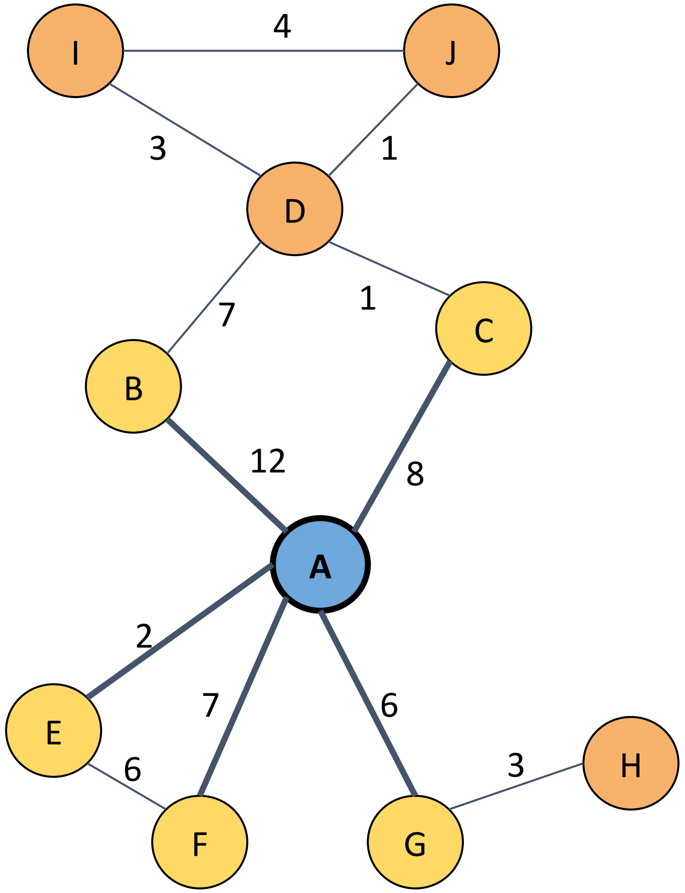
Dijkstra's Algorithm

	A	B	C	D	E	F	G	H	I	J
Distance from start	0	∞	∞	∞	∞	∞	∞	∞	∞	∞
Previous	-									
Seen?	N	N	N	N	N	N	N	N	N	N



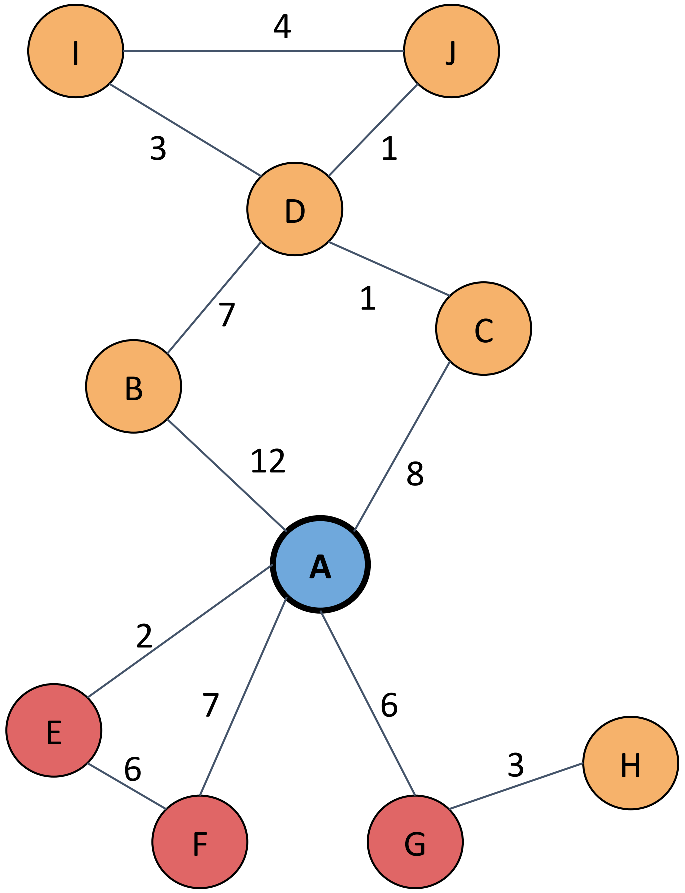
Dijkstra's Algorithm

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
Distance from start	0	12	8	∞	2	7	6	∞	∞	∞
Previous	-	A	A		A	A	A			
Seen?	Y	N	N	N	N	N	N	N	N	N



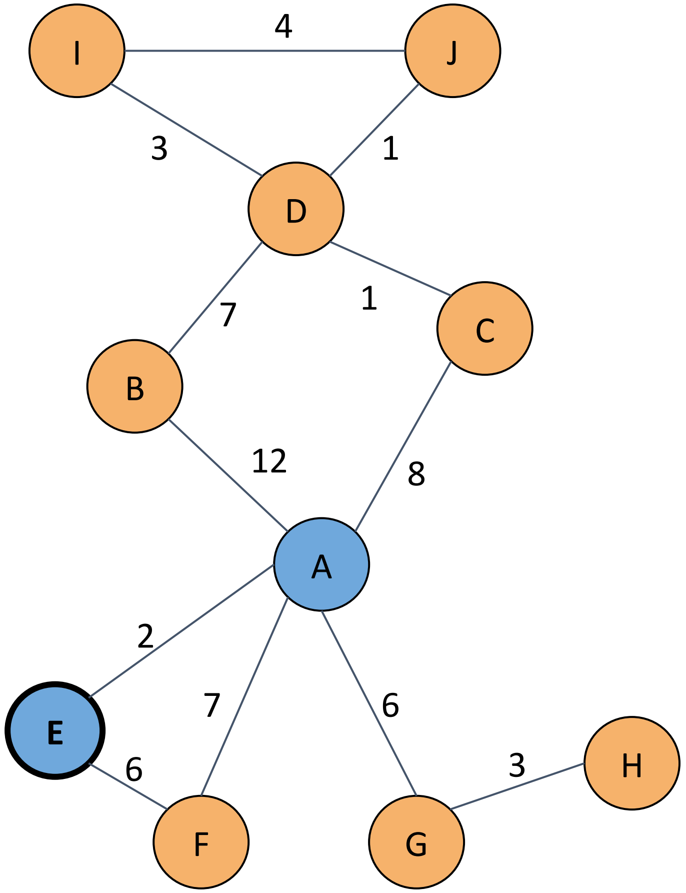
Dijkstra's Algorithm

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
Distance from start	0	12	8	∞	2	7	6	∞	∞	∞
Previous	-	A	A		A	A	A			
Seen?	Y	N	N	N	N	N	N	N	N	N



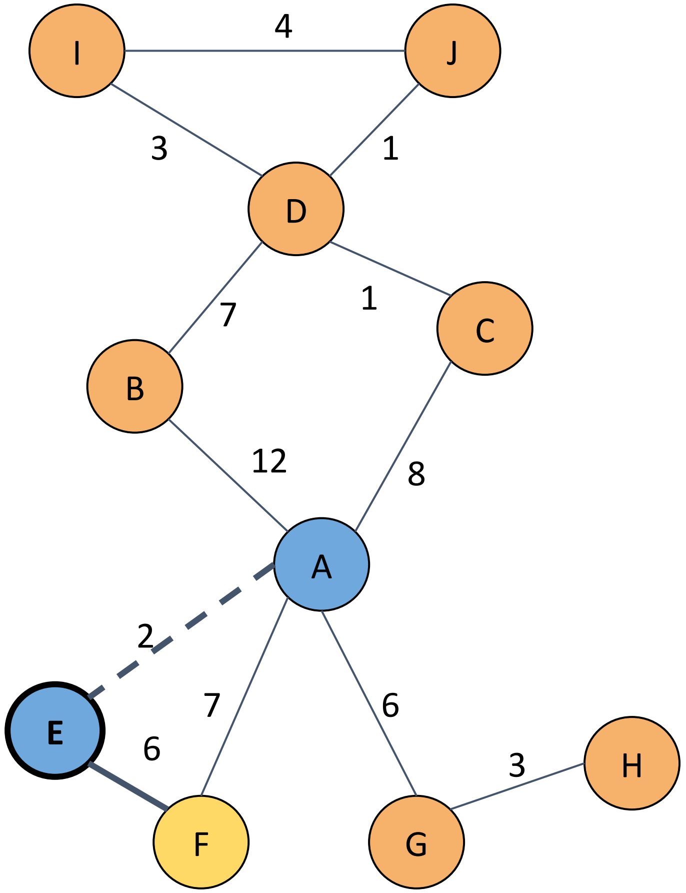
Dijkstra's Algorithm

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
Distance from start	0	12	8	∞	2	7	6	∞	∞	∞
Previous	-	A	A		A	A	A			
Seen?	Y	N	N	N	N	N	N	N	N	N



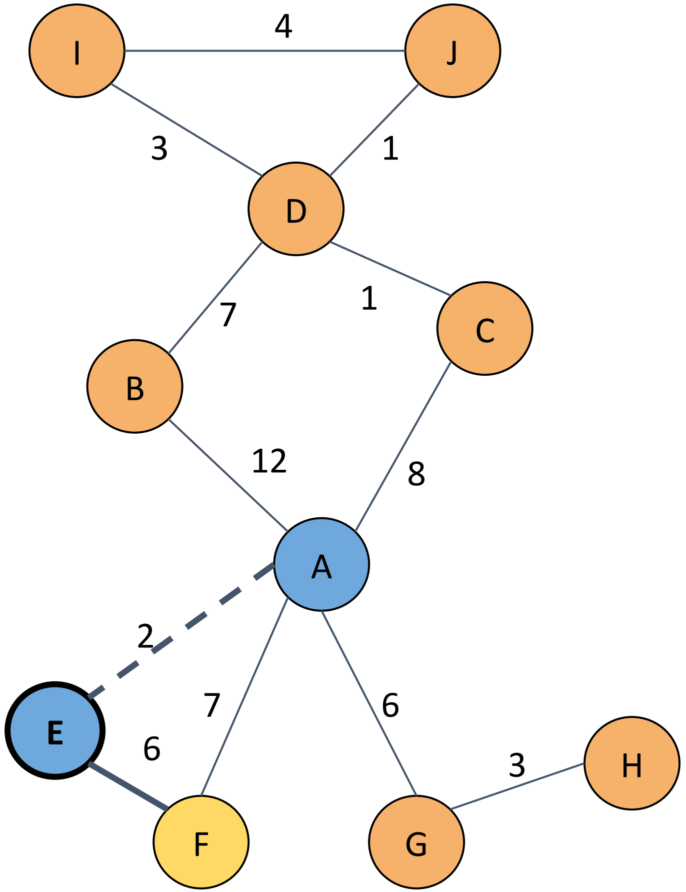
Dijkstra's Algorithm

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
Distance from start	0	12	8	∞	2	7	6	∞	∞	∞
Previous	-	A	A		A	A	A			
Seen?	Y	N	N	N	N	N	N	N	N	N



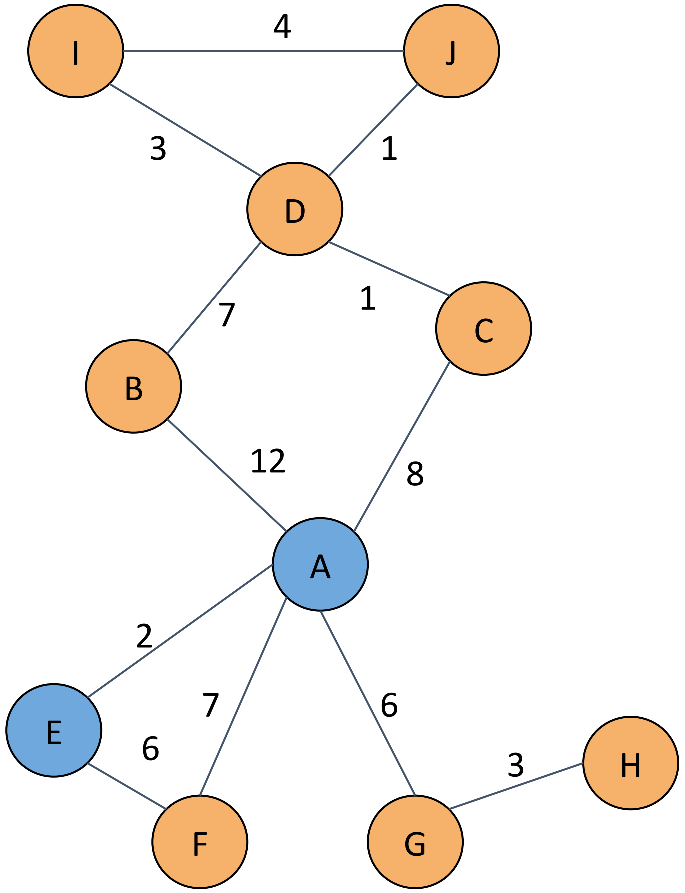
Dijkstra's Algorithm

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
Distance from start	0	12	8	∞	2	7	6	∞	∞	∞
Previous	-	A	A		A	A	A			
Seen?	Y	N	N	N	Y	N	N	N	N	N



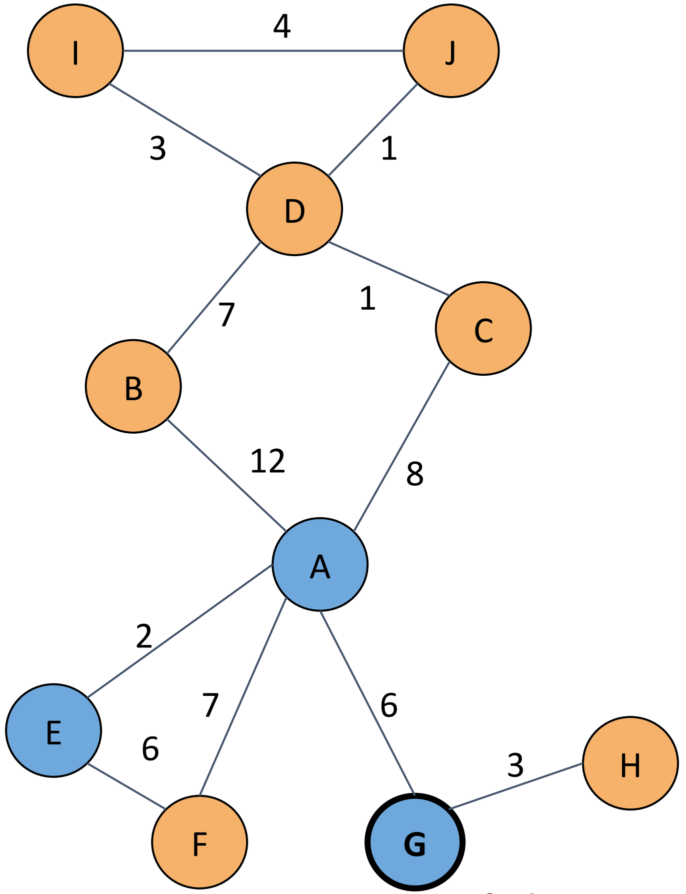
Dijkstra's Algorithm

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
Distance from start	0	12	8	∞	2	7	6	∞	∞	∞
Previous	-	A	A		A	A	A			
Seen?	Y	N	N	N	Y	N	N	N	N	N



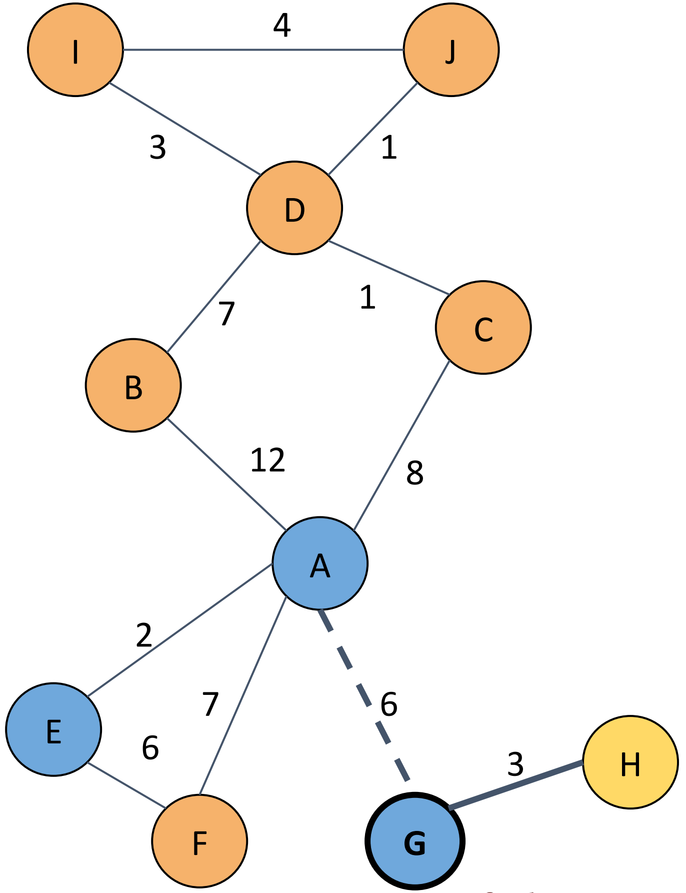
Dijkstra's Algorithm

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
Distance from start	0	12	8	∞	2	7	6	∞	∞	∞
Previous	-	A	A		A	A	A			
Seen?	Y	N	N	N	Y	N	N	N	N	N



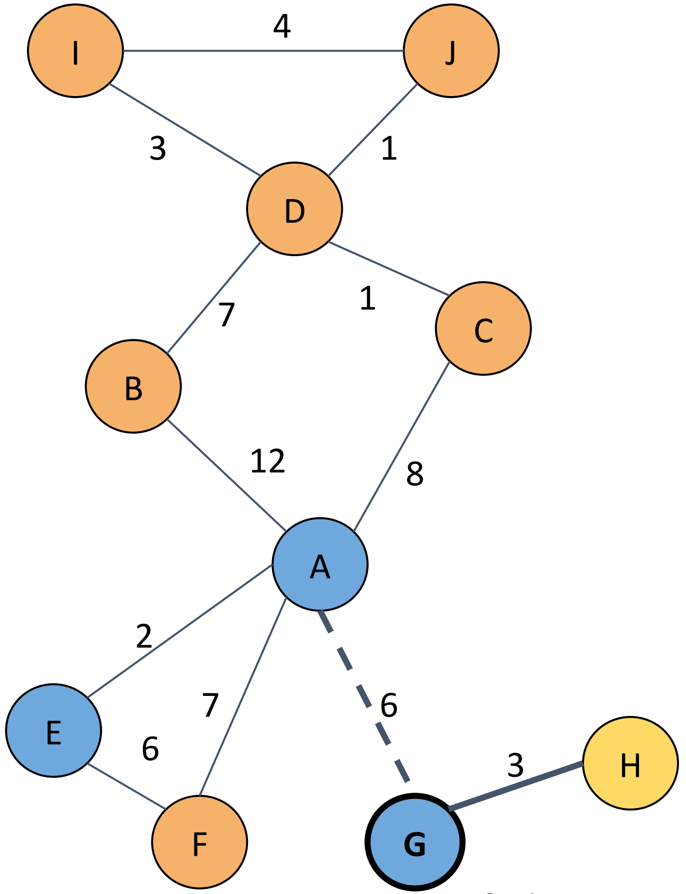
Dijkstra's Algorithm

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
Distance from start	0	12	8	∞	2	7	6	∞	∞	∞
Previous	-	A	A		A	A	A			
Seen?	Y	N	N	N	Y	N	N	N	N	N



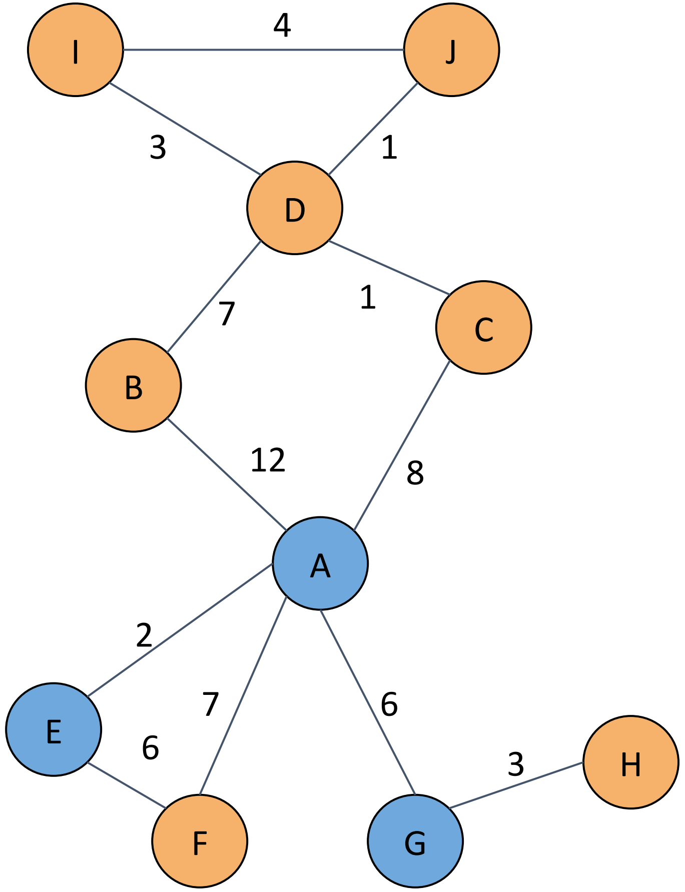
Dijkstra's Algorithm

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
Distance from start	0	12	8	∞	2	7	6	9	∞	∞
Previous	-	A	A		A	A	A	G		
Seen?	Y	N	N	N	Y	N	Y	N	N	N



Dijkstra's Algorithm

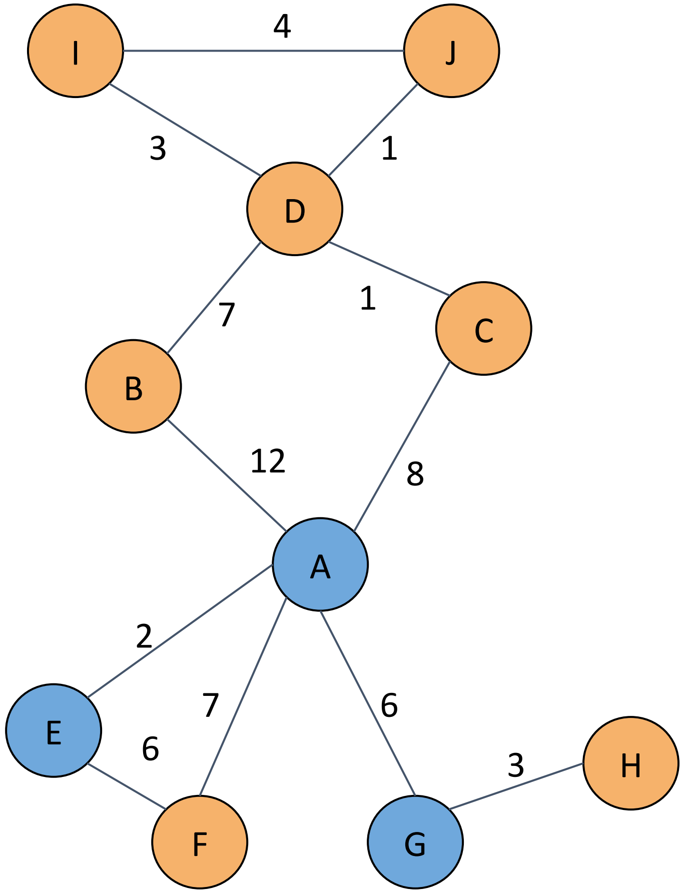
	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
Distance from start	0	12	8	∞	2	7	6	9	∞	∞
Previous	-	A	A		A	A	A	G		
Seen?	Y	N	N	N	Y	N	Y	N	N	N



Dijkstra's Algorithm

	A	B	C	D	E	F	G	H	I	J
Distance from start	0	12	8	∞	2	7	6	9	∞	∞
Previous	-	A	A		A	A	A	G		
Seen?	Y	N	N	N	Y	N	Y	N	N	N

Doesn’t seem very efficient



A* Algorithm

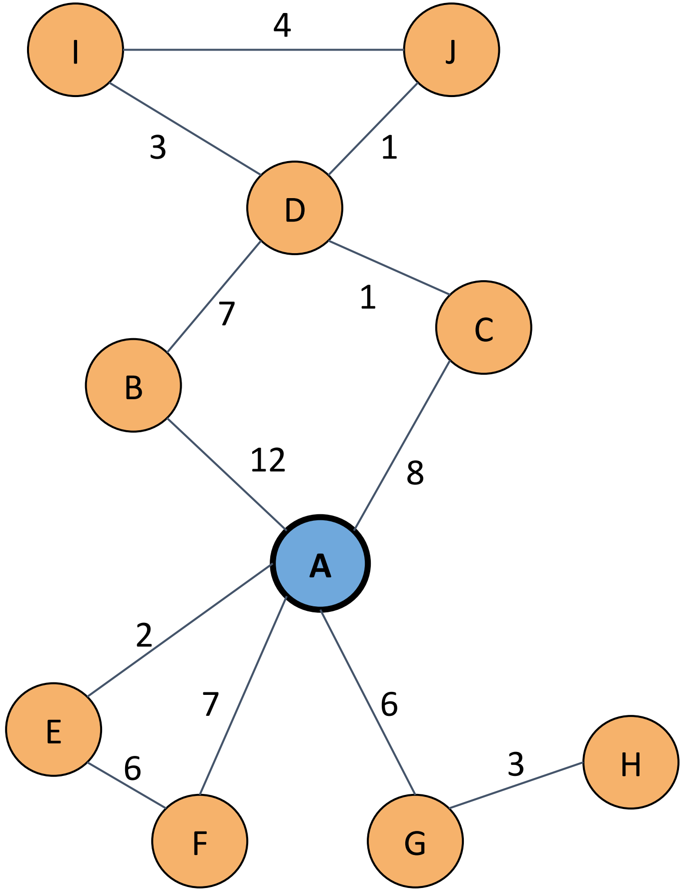
A* Algorithm

- Finds the shortest weighted path from one node to another
- Uses external information about the graph
- Heuristic: estimates the cost of the cheapest path to the goal
  - Should always underestimate the distance to the goal, because if it overestimates, it could find a non-optimal solution
- If the distance to the destination is closer, weight the nodes in that direction to be preferable
  - `priority(u) = weight(s, u) + heuristic(u, d)`

A* Algorithm

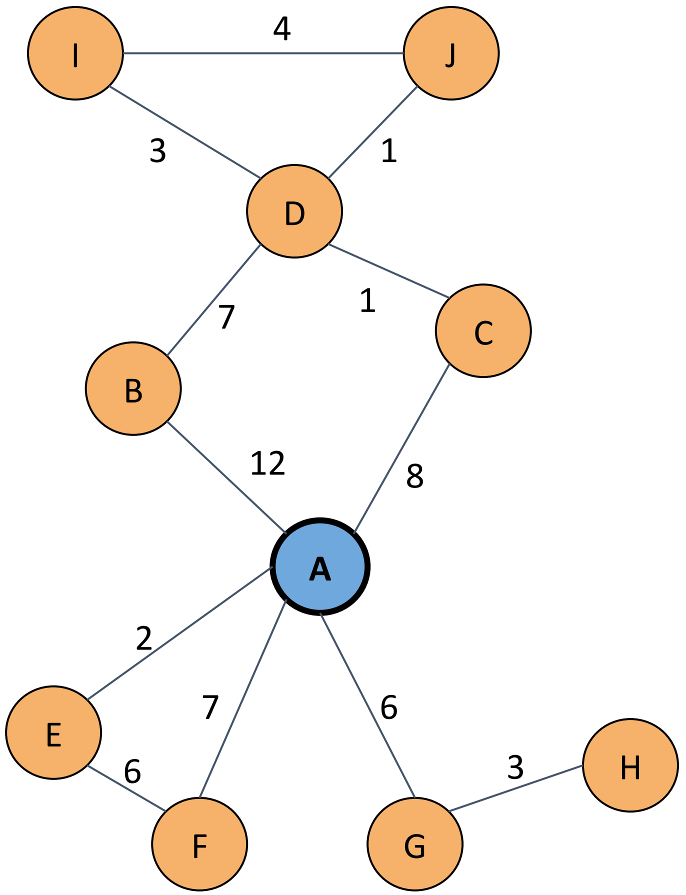
What is the shortest weighted path from A⟶J?

	A	B	C	D	E	F	G	H	I	J
Distance from start	0	∞	∞	∞	∞	∞	∞	∞	∞	∞
Distance + future										
Previous	-									
Seen?	N	N	N	N	N	N	N	N	N	N



What is the shortest weighted path from A⟶J?

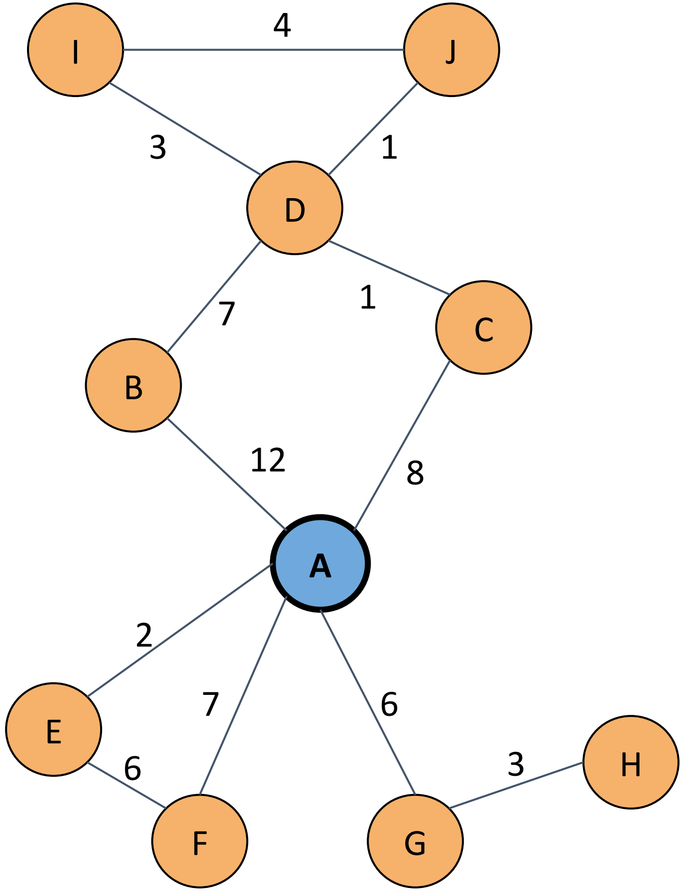
Node	Distance to J	Dist/Smallest Dist
A	301	2.6
B	232	2
C	180	1.6
D	116	1
E	400	3.4
F	441	3.8
G	425	3.7
H	386	3.3
I	154	1.3
J	0	0



A* Algorithm

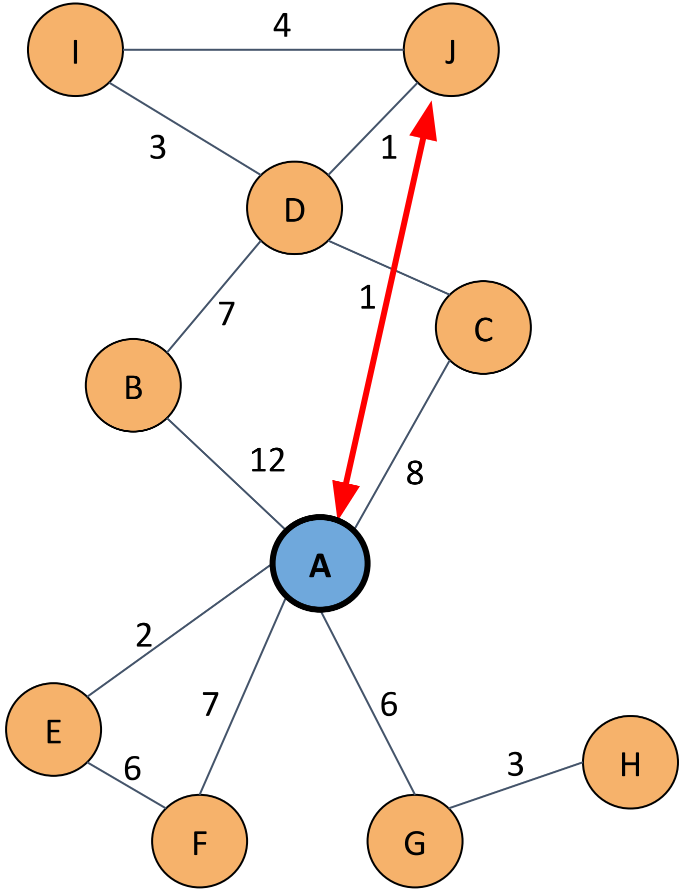
What is the shortest weighted path from A⟶J?

	A	B	C	D	E	F	G	H	I	J
Distance from start	0	∞	∞	∞	∞	∞	∞	∞	∞	∞
Distance + future										
Previous	-									
Seen?	N	N	N	N	N	N	N	N	N	N



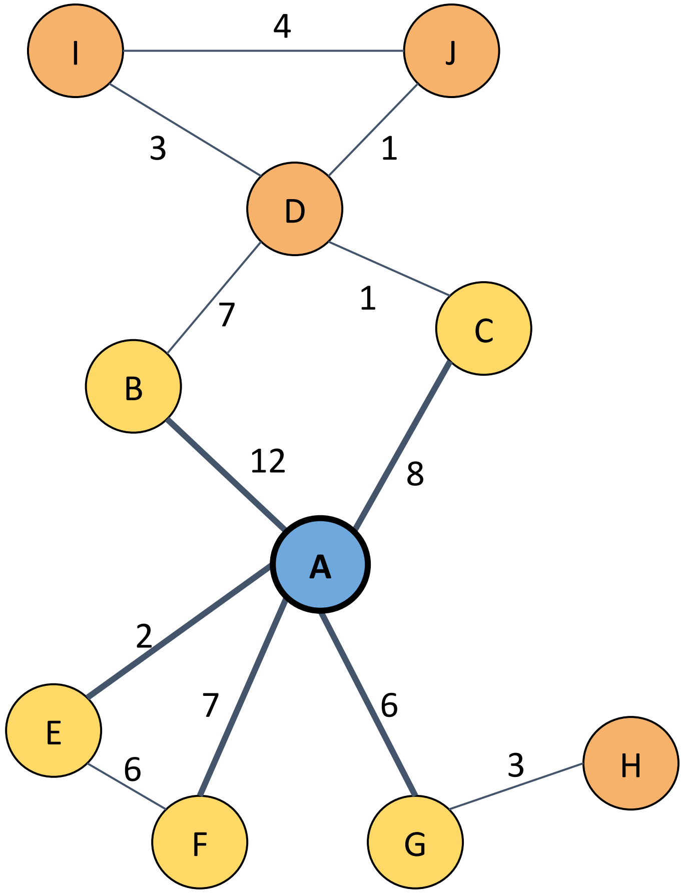
A* Algorithm

	A	B	C	D	E	F	G	H	I	J
Distance from start	0	∞	∞	∞	∞	∞	∞	∞	∞	∞
Distance + future	2.6									
Previous	-									
Seen?	N	N	N	N	N	N	N	N	N	N



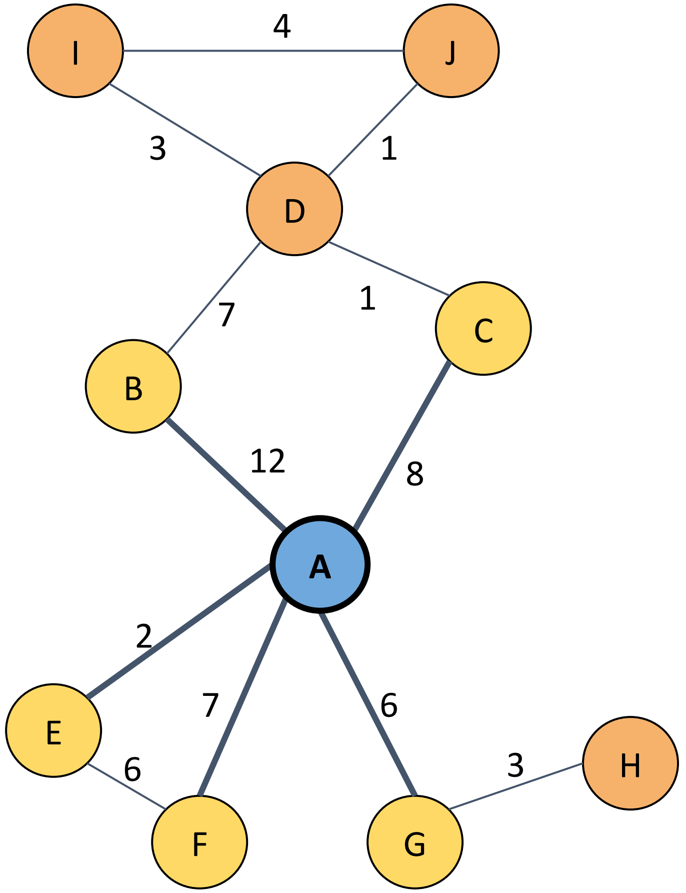
A* Algorithm

	A	B	C	D	E	F	G	H	I	J
Distance from start	0	∞	∞	∞	∞	∞	∞	∞	∞	∞
Distance + future	2.6									
Previous	-									
Seen?	N	N	N	N	N	N	N	N	N	N



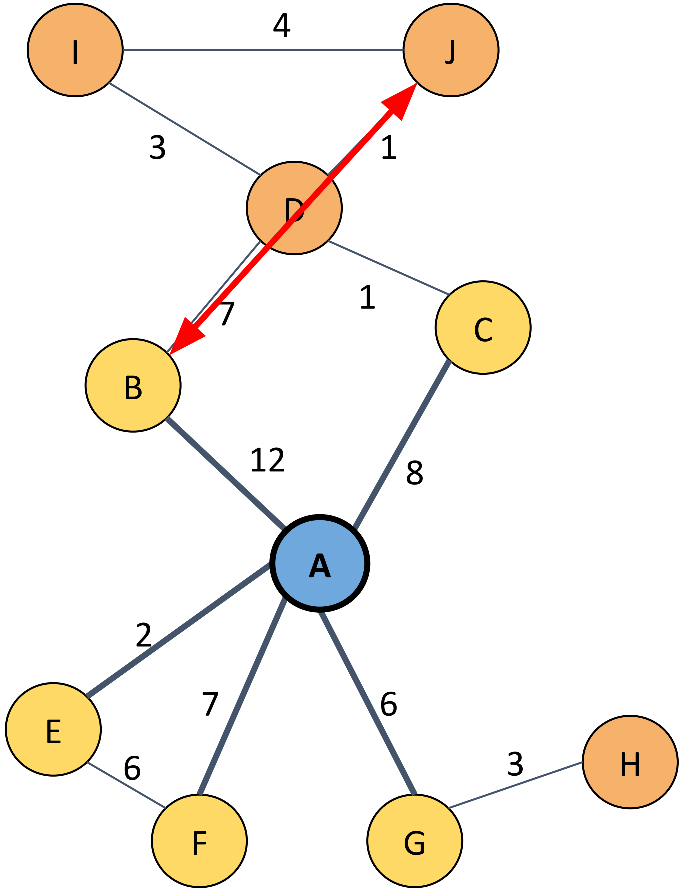
A* Algorithm

	A	B	C	D	E	F	G	H	I	J
Distance from start	0	12	8	∞	2	7	6	∞	∞	∞
Distance + future	2.6									
Previous	-	A	A		A	A	A			
Seen?	N	N	N	N	N	N	N	N	N	N



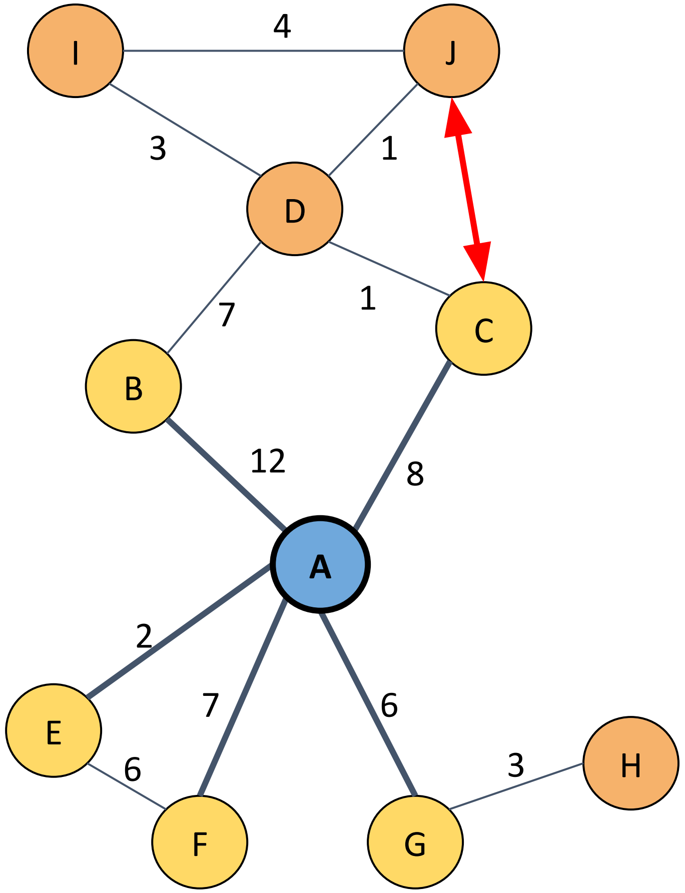
A* Algorithm

	A	B	C	D	E	F	G	H	I	J
Distance from start	0	12	8	∞	2	7	6	∞	∞	∞
Distance + future	2.6	14								
Previous	-	A	A		A	A	A			
Seen?	N	N	N	N	N	N	N	N	N	N



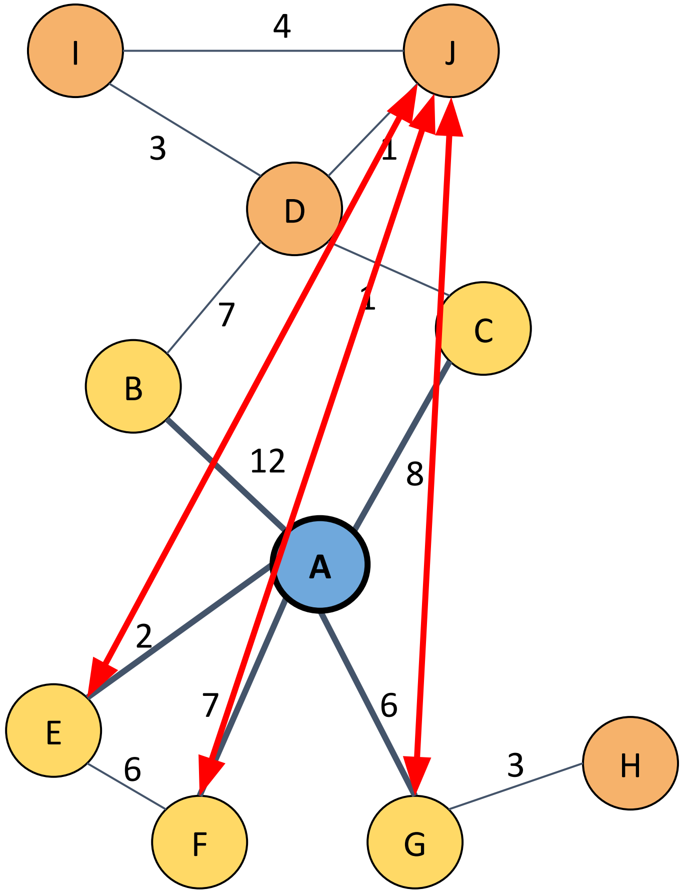
A* Algorithm

	A	B	C	D	E	F	G	H	I	J
Distance from start	0	12	8	∞	2	7	6	∞	∞	∞
Distance + future	2.6	14	9.6							
Previous	-	A	A		A	A	A			
Seen?	N	N	N	N	N	N	N	N	N	N



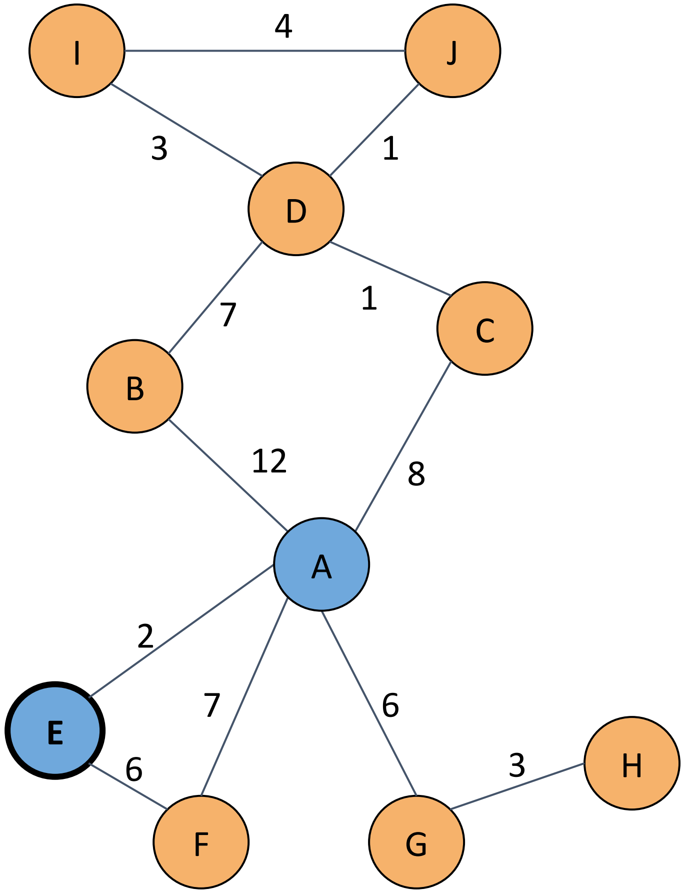
A* Algorithm

	A	B	C	D	E	F	G	H	I	J
Distance from start	0	12	8	∞	2	7	6	∞	∞	∞
Distance + future	2.6	14	9.6		5.4	11.8	9.7			
Previous	-	A	A		A	A	A			
Seen?	N	N	N	N	N	N	N	N	N	N



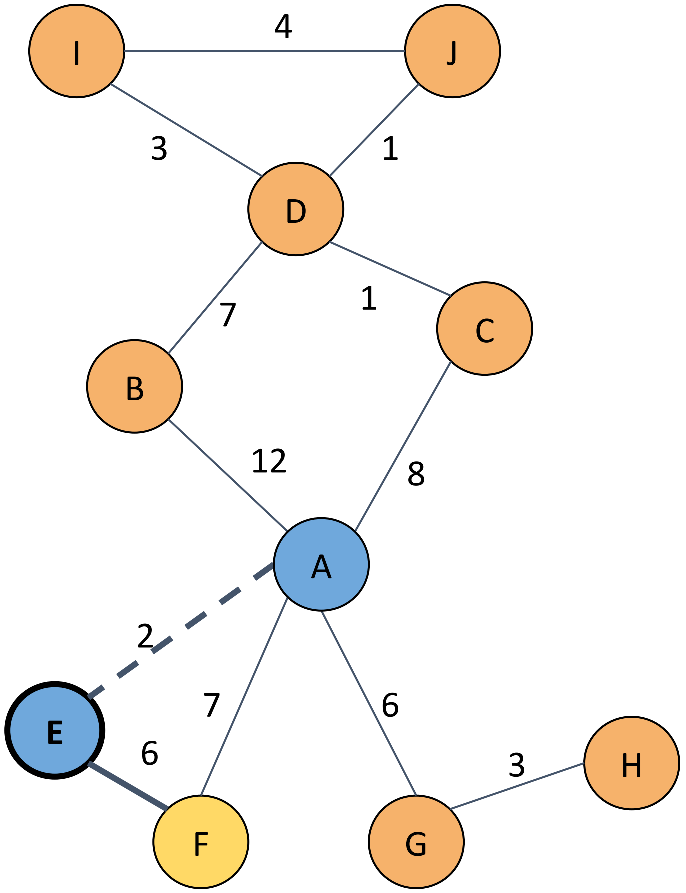
A* Algorithm

	A	B	C	D	E	F	G	H	I	J
Distance from start	0	12	8	∞	2	7	6	∞	∞	∞
Distance + future	2.6	14	9.6		5.4	11.8	9.7			
Previous	-	A	A		A	A	A			
Seen?	Y	N	N	N	N	N	N	N	N	N



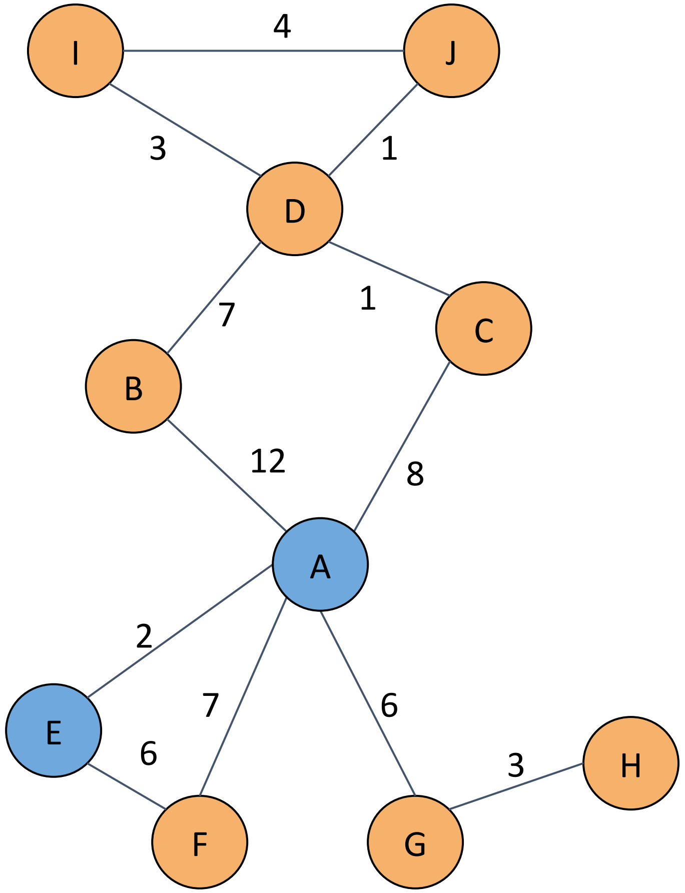
A* Algorithm

	A	B	C	D	E	F	G	H	I	J
Distance from start	0	12	8	∞	2	7	6	∞	∞	∞
Distance + future	2.6	14	9.6		5.4	11.8	9.7			
Previous	-	A	A		A	A	A			
Seen?	Y	N	N	N	N	N	N	N	N	N



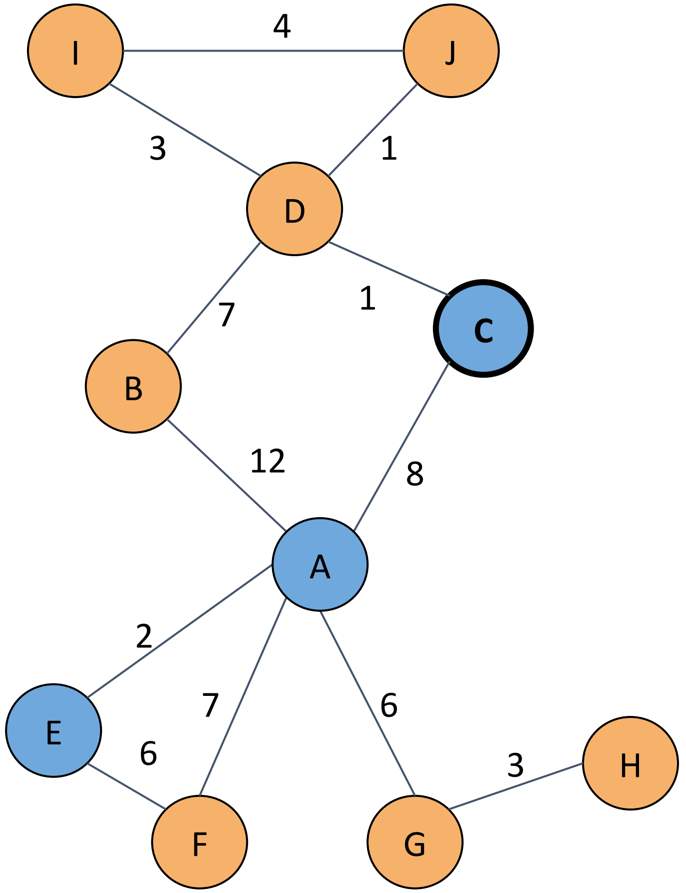
A* Algorithm

	A	B	C	D	E	F	G	H	I	J
Distance from start	0	12	8	∞	2	7	6	∞	∞	∞
Distance + future	2.6	14	9.6		5.4	11.8	9.7			
Previous	-	A	A		A	A	A			
Seen?	Y	N	N	N	Y	N	N	N	N	N



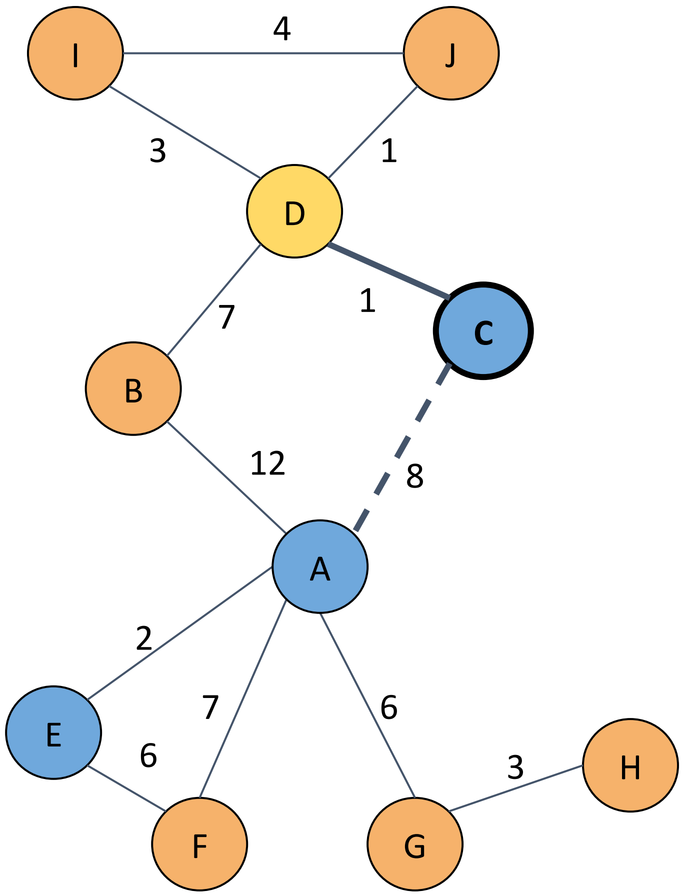
A* Algorithm

	A	B	C	D	E	F	G	H	I	J
Distance from start	0	12	8	∞	2	7	6	∞	∞	∞
Distance + future	2.6	14	9.6		5.4	11.8	9.7			
Previous	-	A	A		A	A	A			
Seen?	Y	N	N	N	Y	N	N	N	N	N



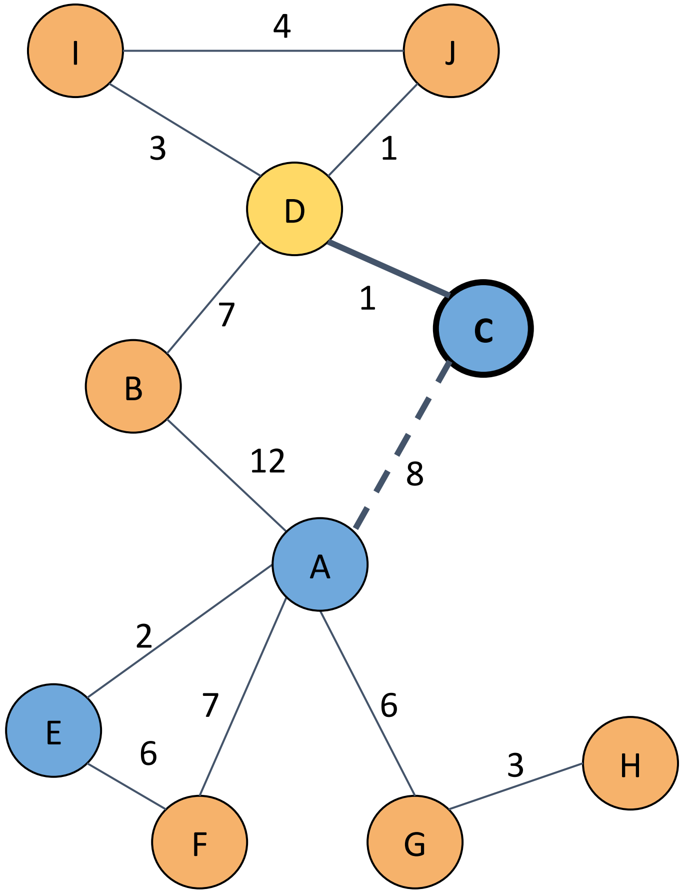
A* Algorithm

	A	B	C	D	E	F	G	H	I	J
Distance from start	0	12	8	∞	2	7	6	∞	∞	∞
Distance + future	2.6	14	9.6		5.4	11.8	9.7			
Previous	-	A	A		A	A	A			
Seen?	Y	N	N	N	Y	N	N	N	N	N



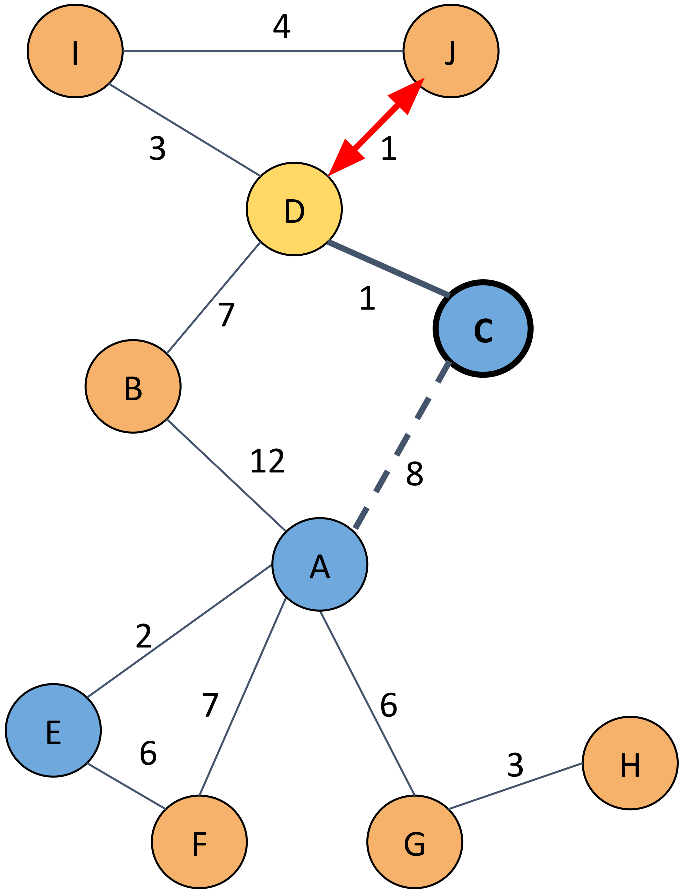
A* Algorithm

	A	B	C	D	E	F	G	H	I	J
Distance from start	0	12	8	9	2	7	6	∞	∞	∞
Distance + future	2.6	14	9.6		5.4	11.8	9.7			
Previous	-	A	A		A	A	A			
Seen?	Y	N	N	N	Y	N	N	N	N	N



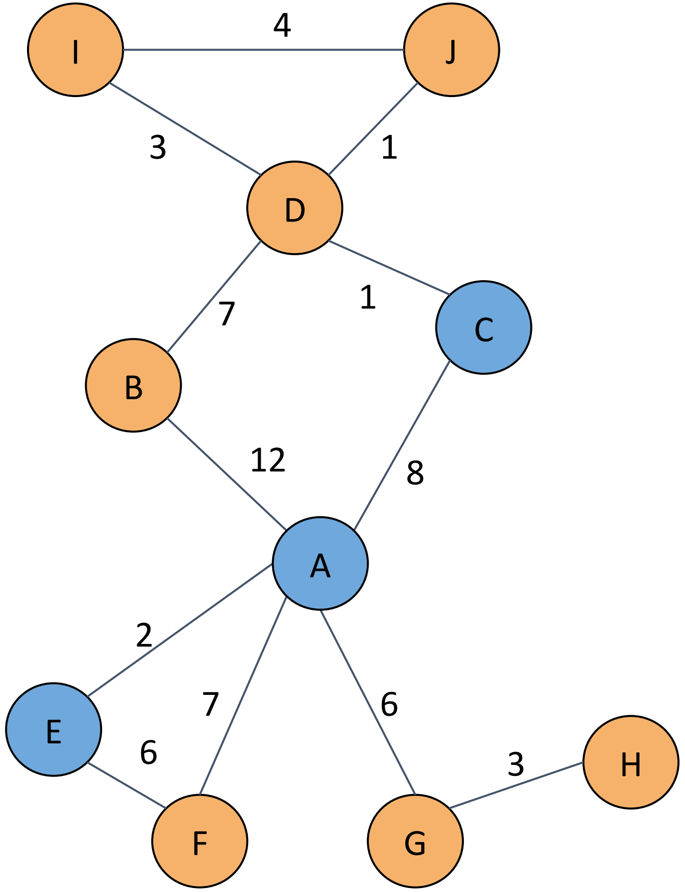
A* Algorithm

	A	B	C	D	E	F	G	H	I	J
Distance from start	0	12	8	9	2	7	6	∞	∞	∞
Distance + future	2.6	14	9.6	10	5.4	11.8	9.7			
Previous	-	A	A	C	A	A	A			
Seen?	Y	N	Y	N	Y	N	N	N	N	N



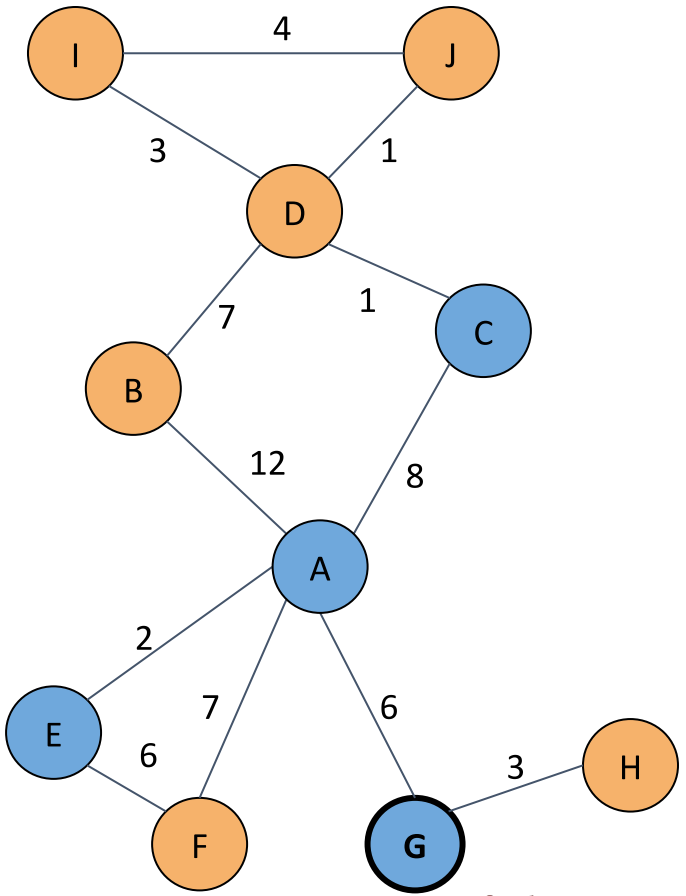
A* Algorithm

	A	B	C	D	E	F	G	H	I	J
Distance from start	0	12	8	9	2	7	6	∞	∞	∞
Distance + future	2.6	14	9.6	10	5.4	11.8	9.7			
Previous	-	A	A	C	A	A	A			
Seen?	Y	N	Y	N	Y	N	N	N	N	N



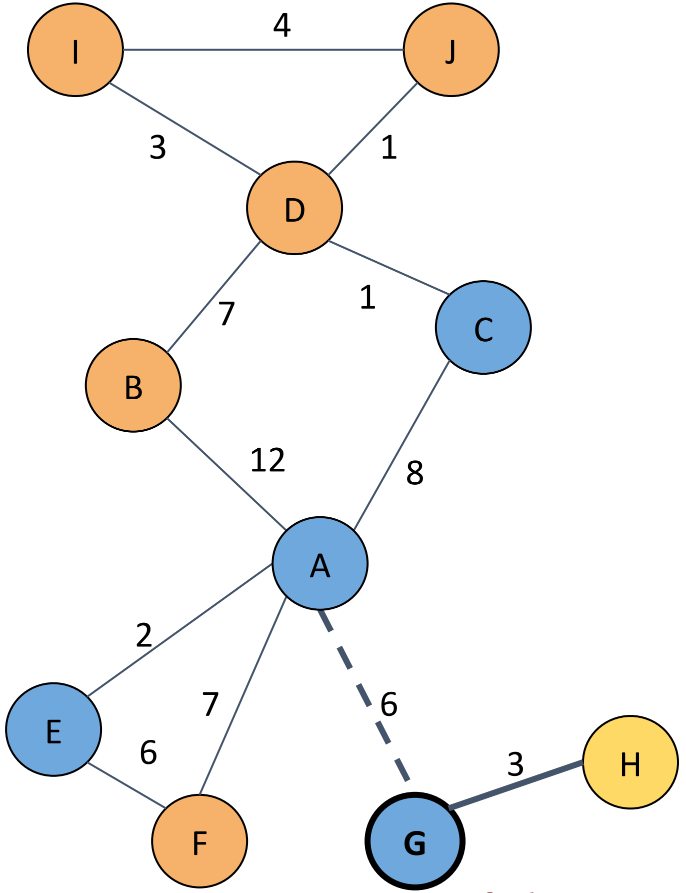
A* Algorithm

	A	B	C	D	E	F	G	H	I	J
Distance from start	0	12	8	9	2	7	6	∞	∞	∞
Distance + future	2.6	14	9.6	10	5.4	11.8	9.7			
Previous	-	A	A	C	A	A	A			
Seen?	Y	N	Y	N	Y	N	N	N	N	N



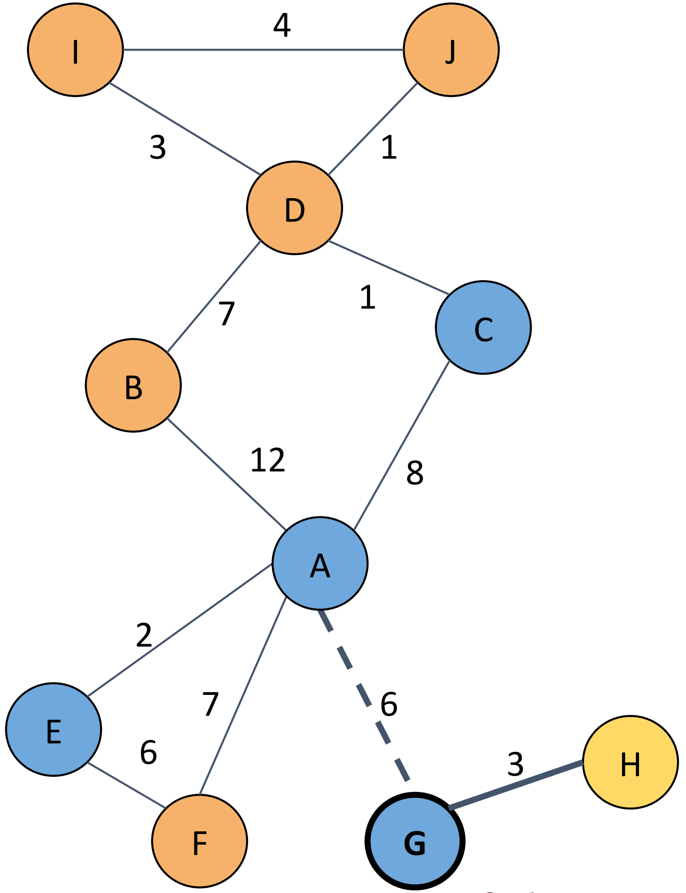
A* Algorithm

	A	B	C	D	E	F	G	H	I	J
Distance from start	0	12	8	9	2	7	6	∞	∞	∞
Distance + future	2.6	14	9.6	10	5.4	11.8	9.7			
Previous	-	A	A	C	A	A	A			
Seen?	Y	N	Y	N	Y	N	N	N	N	N



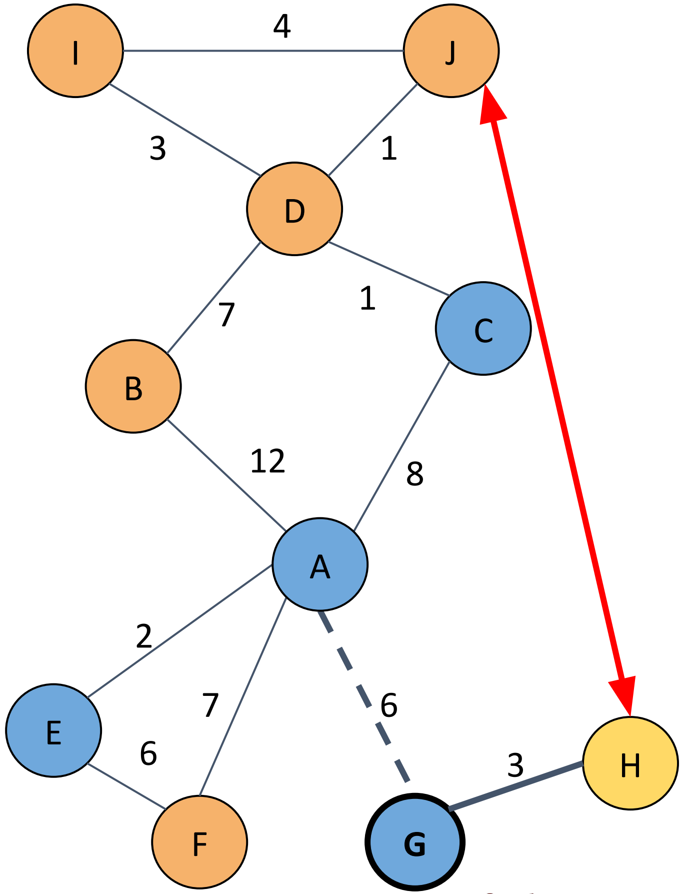
A* Algorithm

	A	B	C	D	E	F	G	H	I	J
Distance from start	0	12	8	9	2	7	6	9	∞	∞
Distance + future	2.6	14	9.6	10	5.4	11.8	9.7			
Previous	-	A	A	C	A	A	A			
Seen?	Y	N	Y	N	Y	N	N	N	N	N



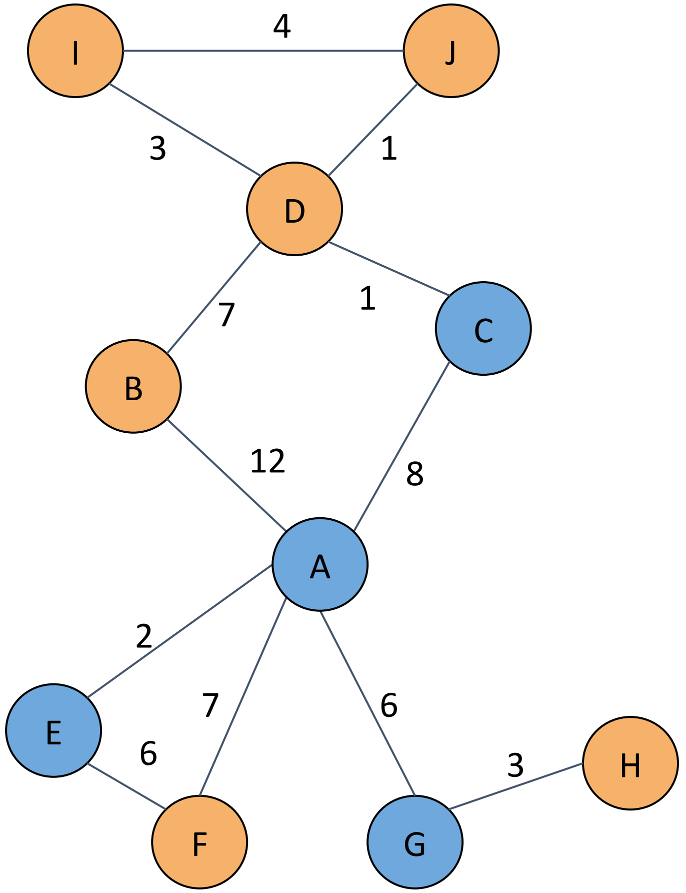
A* Algorithm

	A	B	C	D	E	F	G	H	I	J
Distance from start	0	12	8	9	2	7	6	9	∞	∞
Distance + future	2.6	14	9.6	10	5.4	11.8	9.7	12.3		
Previous	-	A	A	C	A	A	A	G		
Seen?	Y	N	Y	N	Y	N	Y	N	N	N



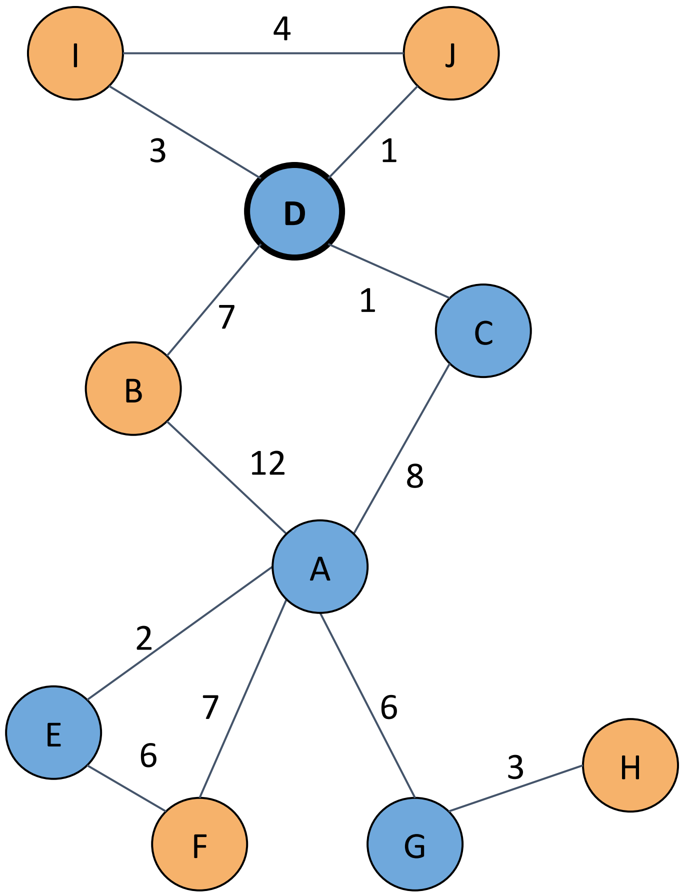
A* Algorithm

	A	B	C	D	E	F	G	H	I	J
Distance from start	0	12	8	9	2	7	6	9	∞	∞
Distance + future	2.6	14	9.6	10	5.4	11.8	9.7	12.3		
Previous	-	A	A	C	A	A	A	G		
Seen?	Y	N	Y	N	Y	N	Y	N	N	N



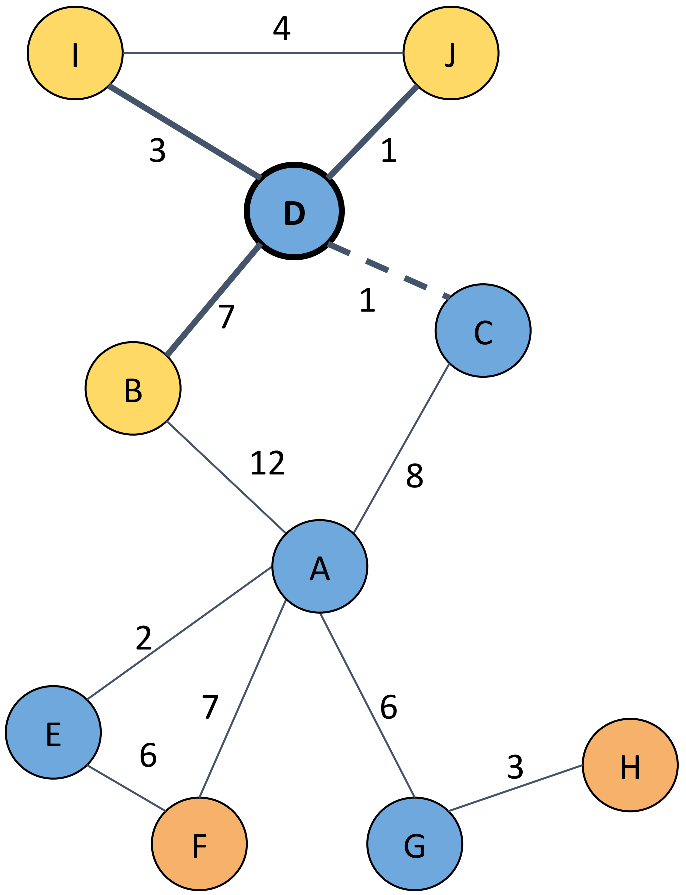
A* Algorithm

	A	B	C	D	E	F	G	H	I	J
Distance from start	0	12	8	9	2	7	6	9	∞	∞
Distance + future	2.6	14	9.6	10	5.4	11.8	9.7	12.3		
Previous	-	A	A	C	A	A	A	G		
Seen?	Y	N	Y	N	Y	N	Y	N	N	N



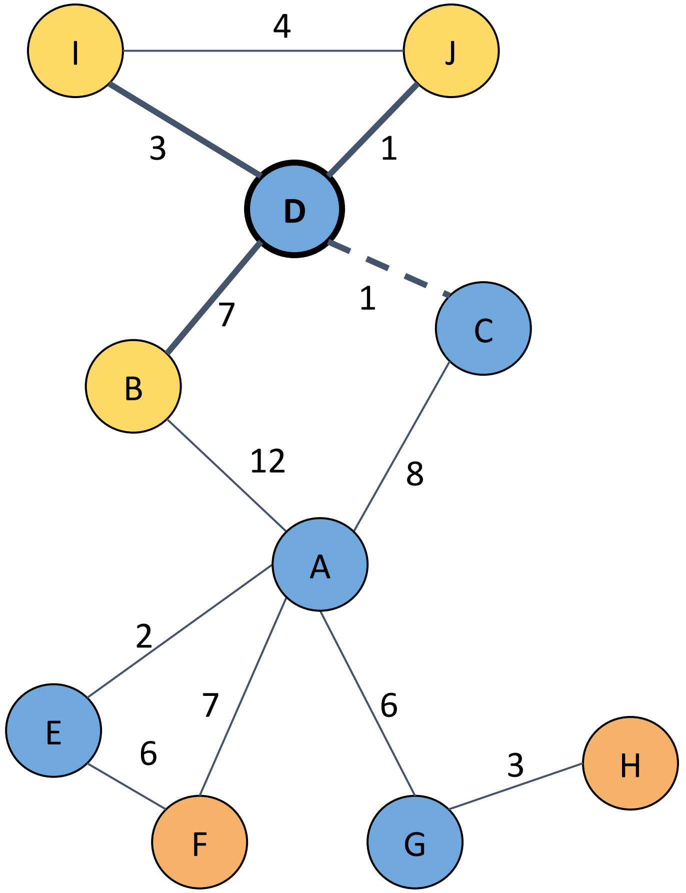
A* Algorithm

	A	B	C	D	E	F	G	H	I	J
Distance from start	0	12	8	9	2	7	6	9	∞	∞
Distance + future	2.6	14	9.6	10	5.4	11.8	9.7	12.3		
Previous	-	A	A	C	A	A	A	G		
Seen?	Y	N	Y	N	Y	N	Y	N	N	N



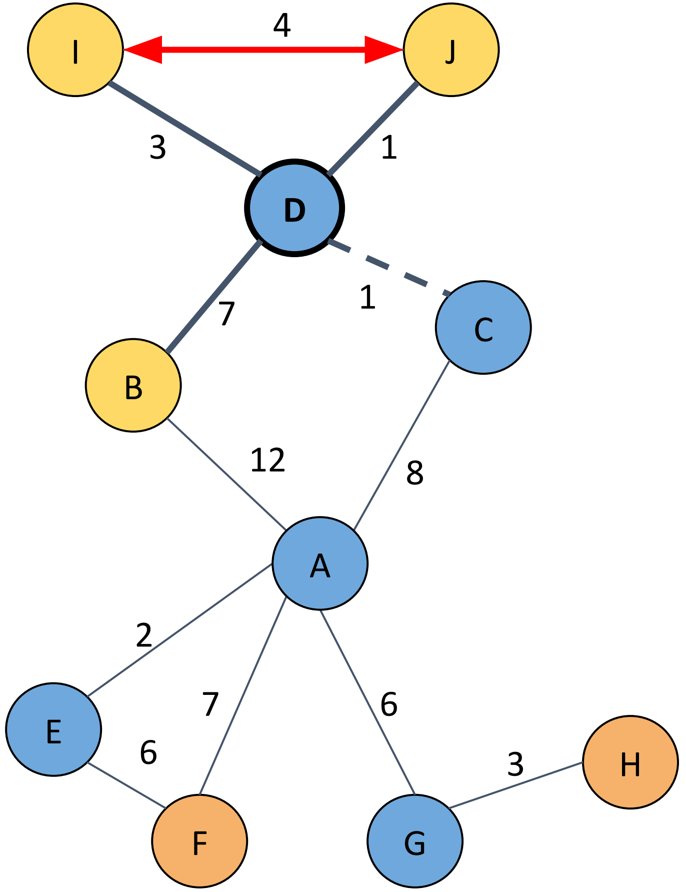
A* Algorithm

	A	B	C	D	E	F	G	H	I	J
Distance from start	0	12	8	9	2	7	6	9	12	
Distance + future	2.6	14	9.6	10	5.4	11.8	9.7	12.3		
Previous	-	A	A	C	A	A	A	G		
Seen?	Y	N	Y	N	Y	N	Y	N	N	N



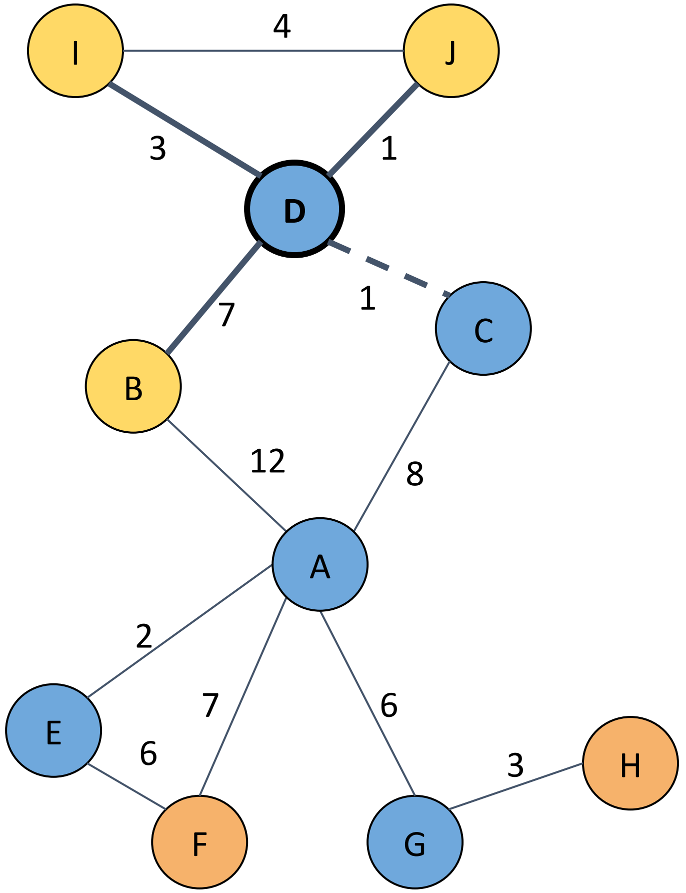
A* Algorithm

	A	B	C	D	E	F	G	H	I	J
Distance from start	0	12	8	9	2	7	6	9	12	
Distance + future	2.6	14	9.6	10	5.4	11.8	9.7	12.3	4	
Previous	-	A	A	C	A	A	A	G	D	
Seen?	Y	N	Y	N	Y	N	Y	N	N	N



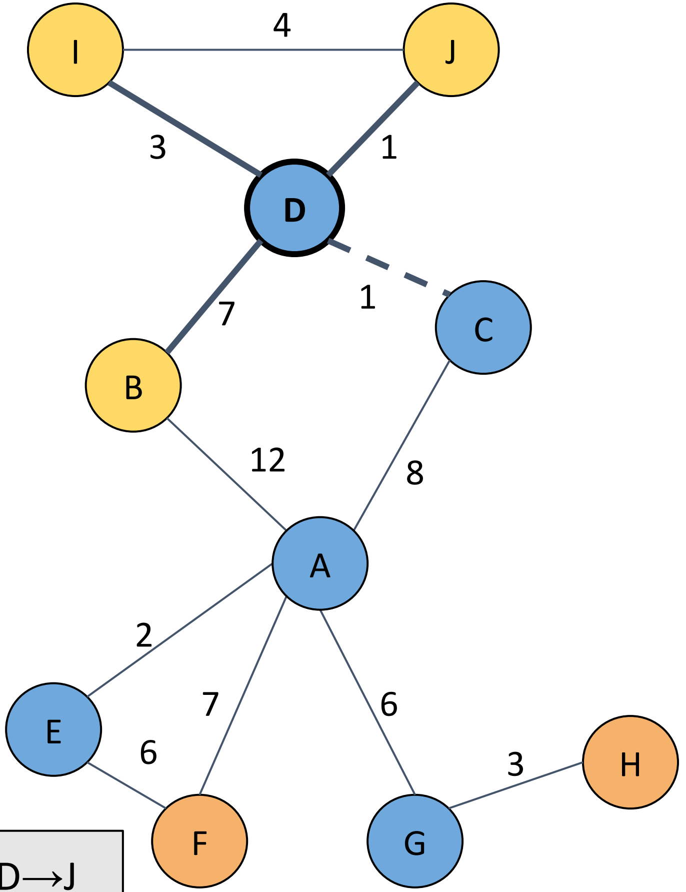
A* Algorithm

	A	B	C	D	E	F	G	H	I	J
Distance from start	0	12	8	9	2	7	6	9	12	11
Distance + future	2.6	14	9.6	10	5.4	11.8	9.7	12.3	4	11
Previous	-	A	A	C	A	A	A	G	D	D
Seen?	Y	N	Y	Y	Y	N	Y	N	N	N



A* Algorithm

	A	B	C	D	E	F	G	H	I	J
Distance from start	0	12	8	9	2	7	6	9	12	11
Distance + future	2.6	14	9.6	10	5.4	11.8	9.7	12.3	4	11
Previous	-	A	A	C	A	A	A	G	D	D
Seen?	Y	N	Y	Y	Y	N	Y	N	N	N



We're done! Shortest weighted path is of length **11** from A⟶C⟶D⟶J

Extensions

- There are many, many different graph algorithms out there
- Other famous graph algorithms:
  - Kruskal's Algorithm: Find a minimum spanning tree from a given graph.
  - Topological Sort: "Sort" the nodes in a dependency graph in such a way that traversing the nodes in order results in all dependencies being fulfilled at each point in time.
  - Traveling salesman: Given a map of cities and the distances between them, find the shortest path that traverses all cities in the map.

Recap

- Graphs are a linked data structure with almost no rules
  - Represent in code with either an adjacency list or matrix
- Depth-First Search: does not always return the shortest path, though it may be faster in some cases
- Breadth-First Search: returns the shortest path, but it only works on unweighted graphs
- Dijkstra’s Algorithm: returns the shortest weighted path, but not necessarily the most efficient
- A* Algorithm: returns the shortest weighted path using heuristics, and is often thought of as gold standard

Have a great weekend! 🌻