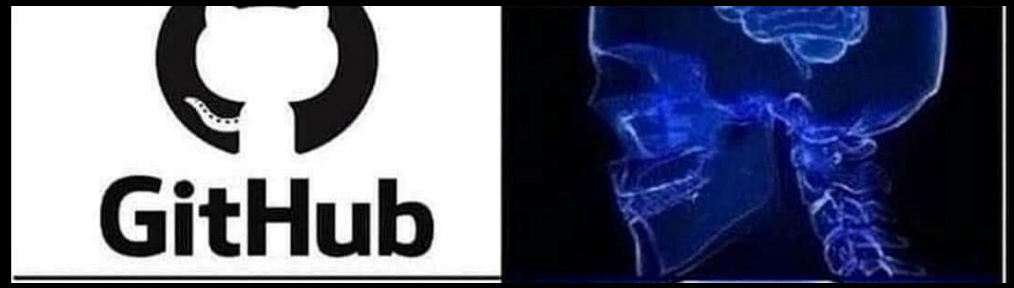
Housekeeping

- Syllabus
- Project
- Feedback so far
  - What’s going well
  - Not so well?
  - What can WE do?

Meme sourced from my sister’s repository



Meme sourced from my sister’s repository

Meme sourced from my sister’s repository

Meme sourced from my sister’s repository

gitolite.engsci.utoronto.ca

Graphs: Searching

Week 3

Content

Today

- Graphs
  - Terminology, examples
  - Notation

Friday

- Graph algorithms
  - Conceptual

Next week

  - Advanced graph algorithm(s)
  - Implementation and code

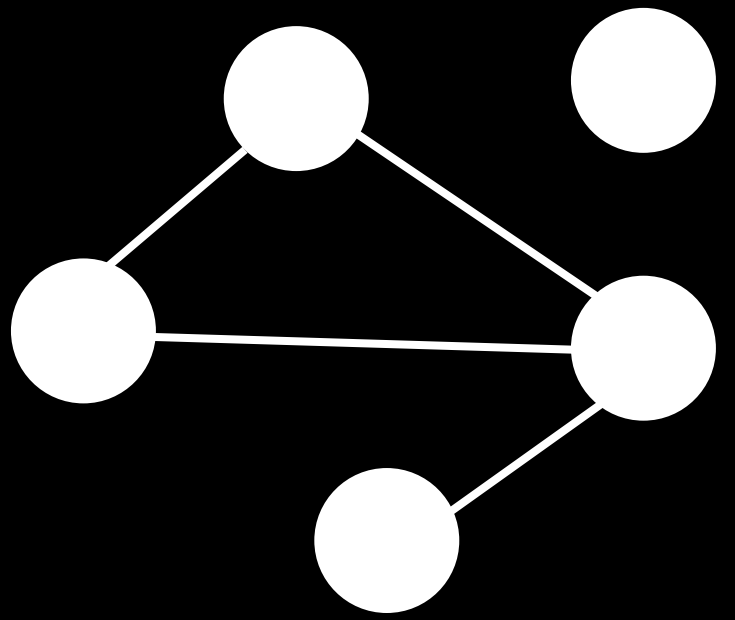
Graphs

What I used to think graphs are

Graphs

What I used to think graphs are

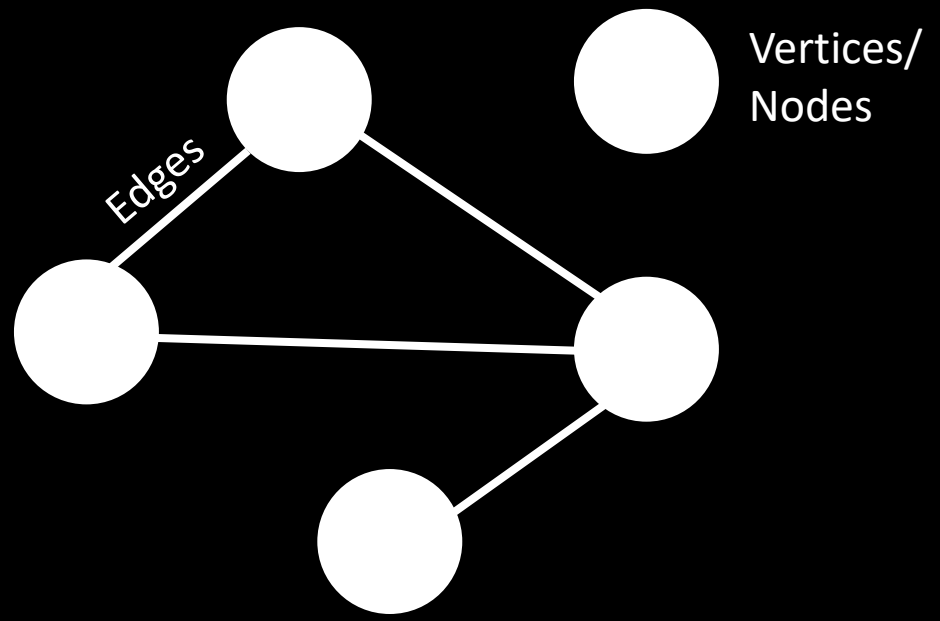
What graphs really are



Graphs

What I used to think graphs are

What graphs really are



Graphs

- A way to show relationships
- Vertices/nodes store some sort of data
- Edges indicate the relationship

Graphs

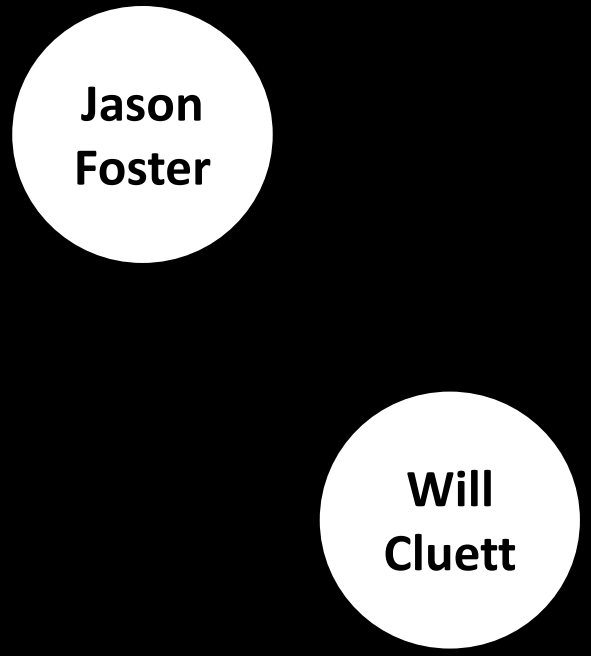
- A way to show relationships
- Vertices/nodes store some sort of data
- Edges indicate the relationship
- Trees are a type of graph!
  - Heaps (a type of tree) are also a type of graph

Graphs

An EngSci Social Network (Acebook?)

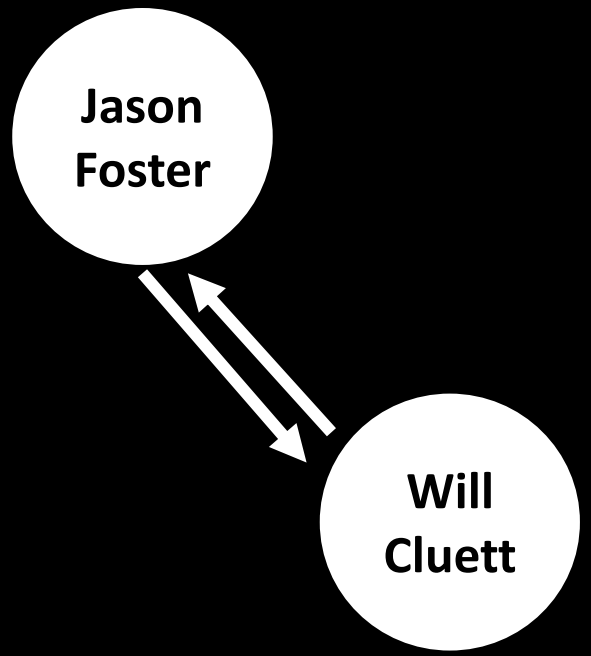
Graphs

An EngSci Social Network (Acebook?)



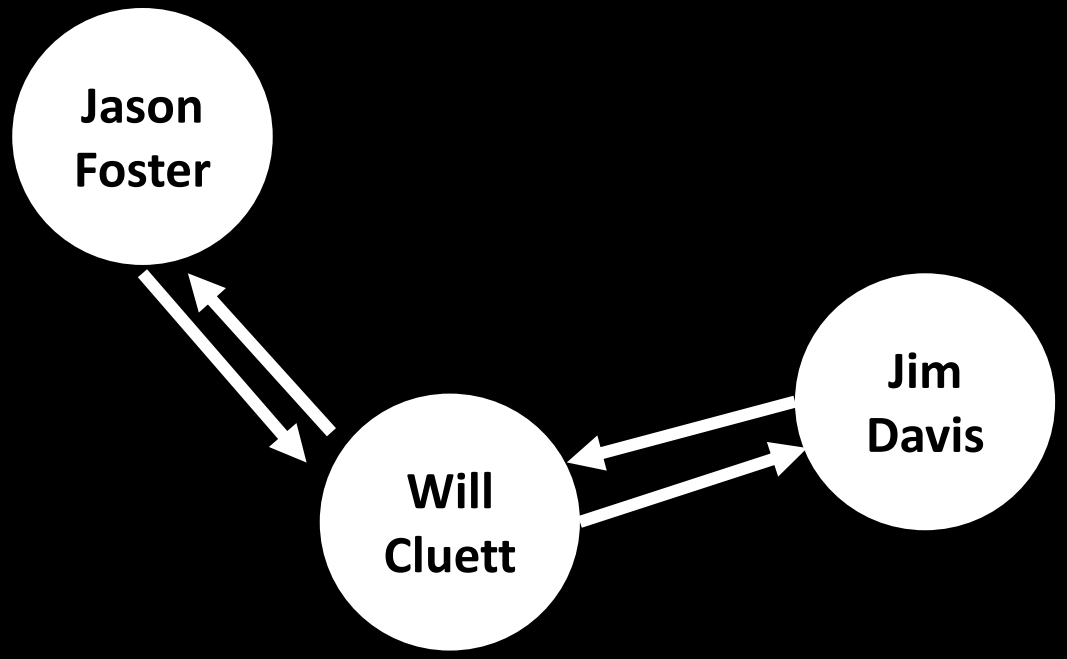
Graphs

An EngSci Social Network (Acebook?)



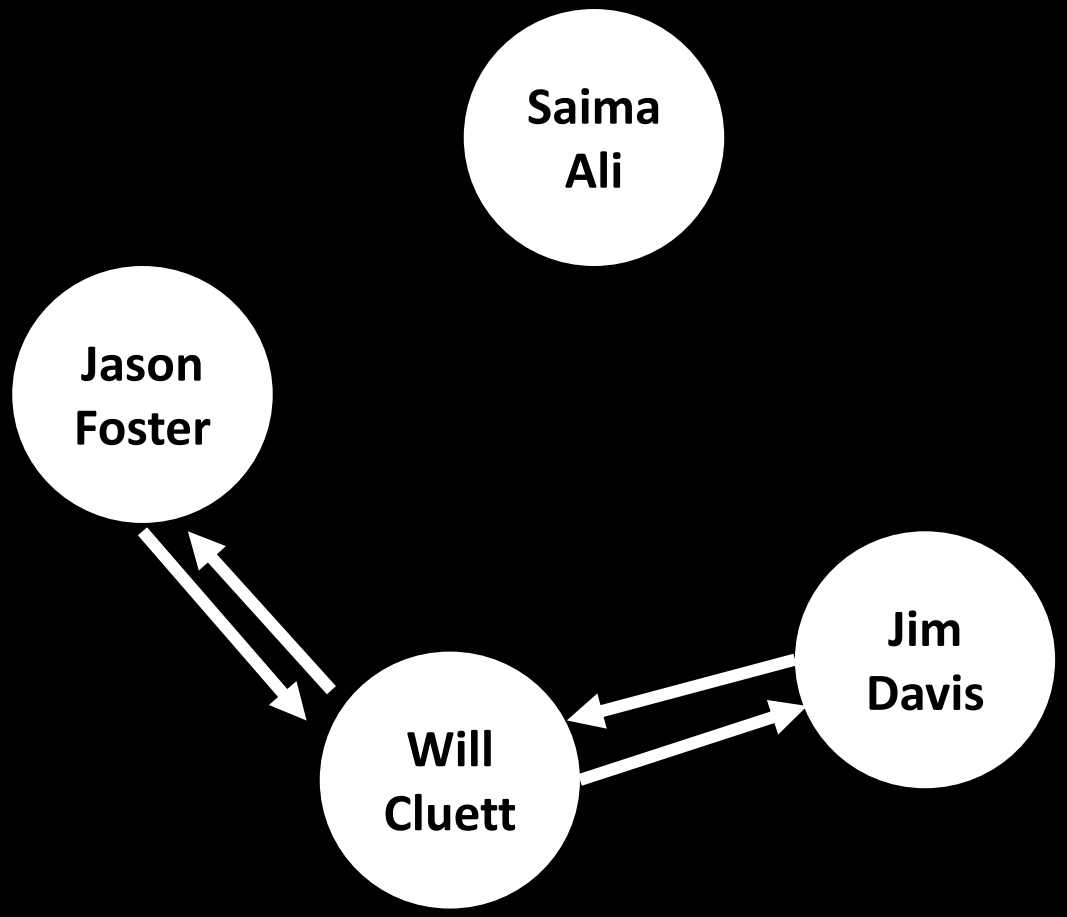
Graphs

An EngSci Social Network (Acebook?)



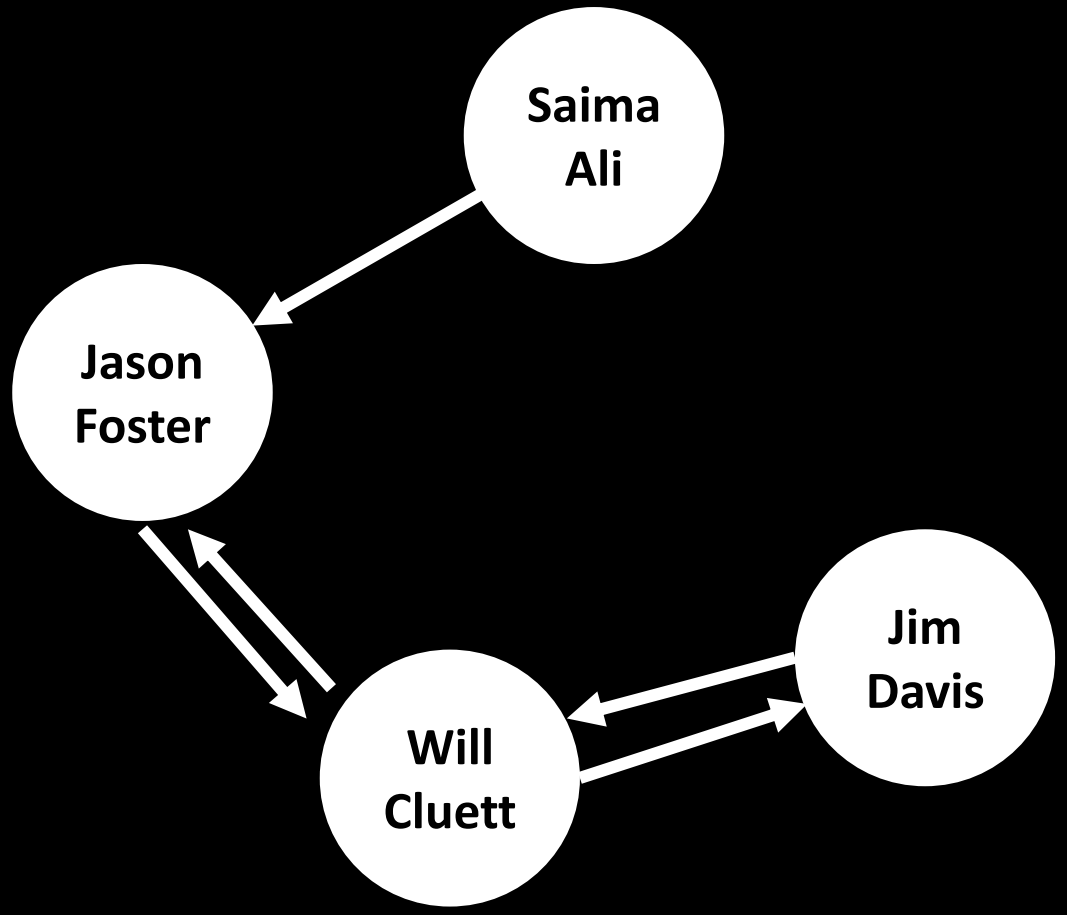
Graphs

An EngSci Social Network (Acebook?)



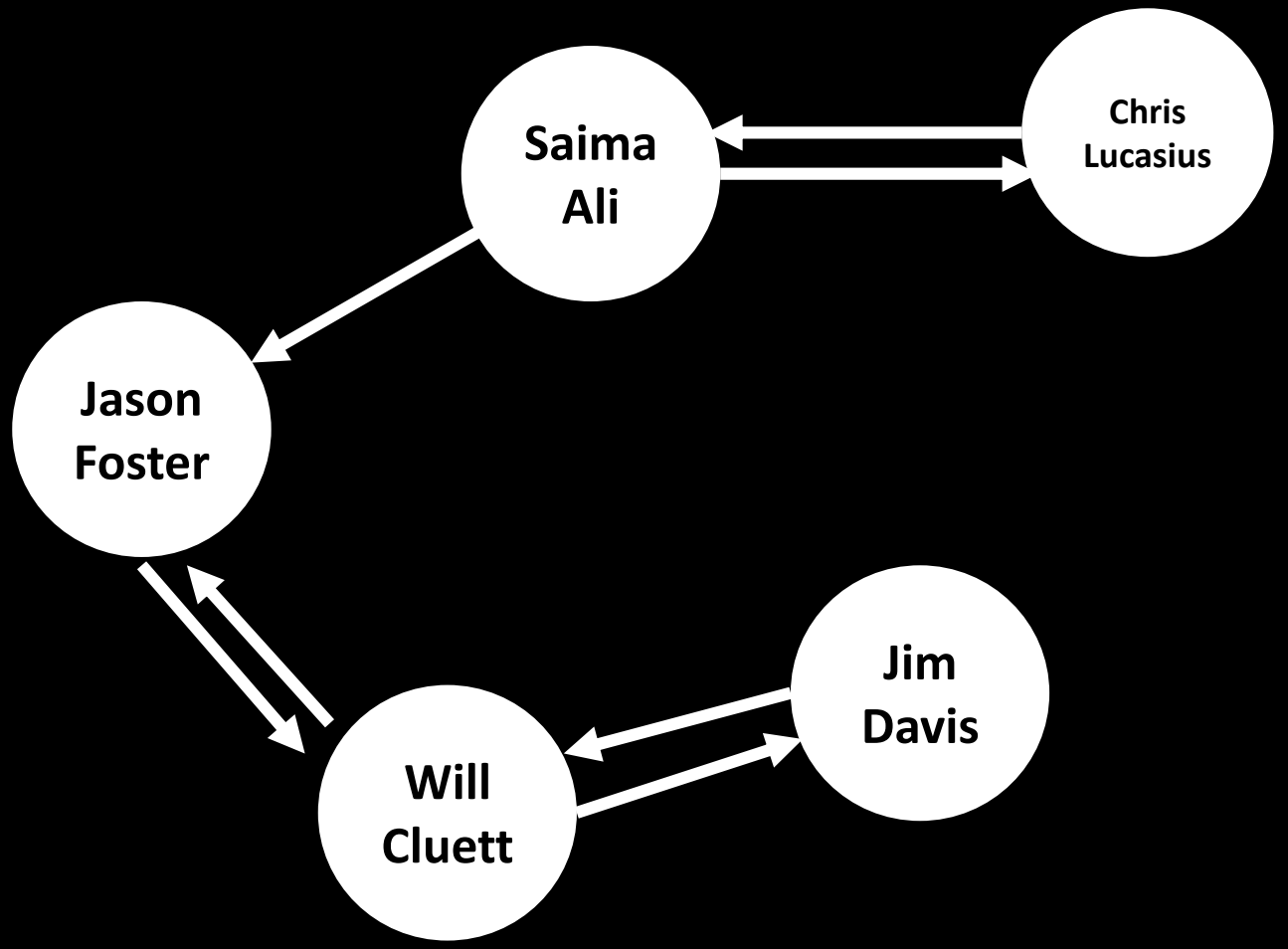
Graphs

An EngSci Social Network (Acebook?)



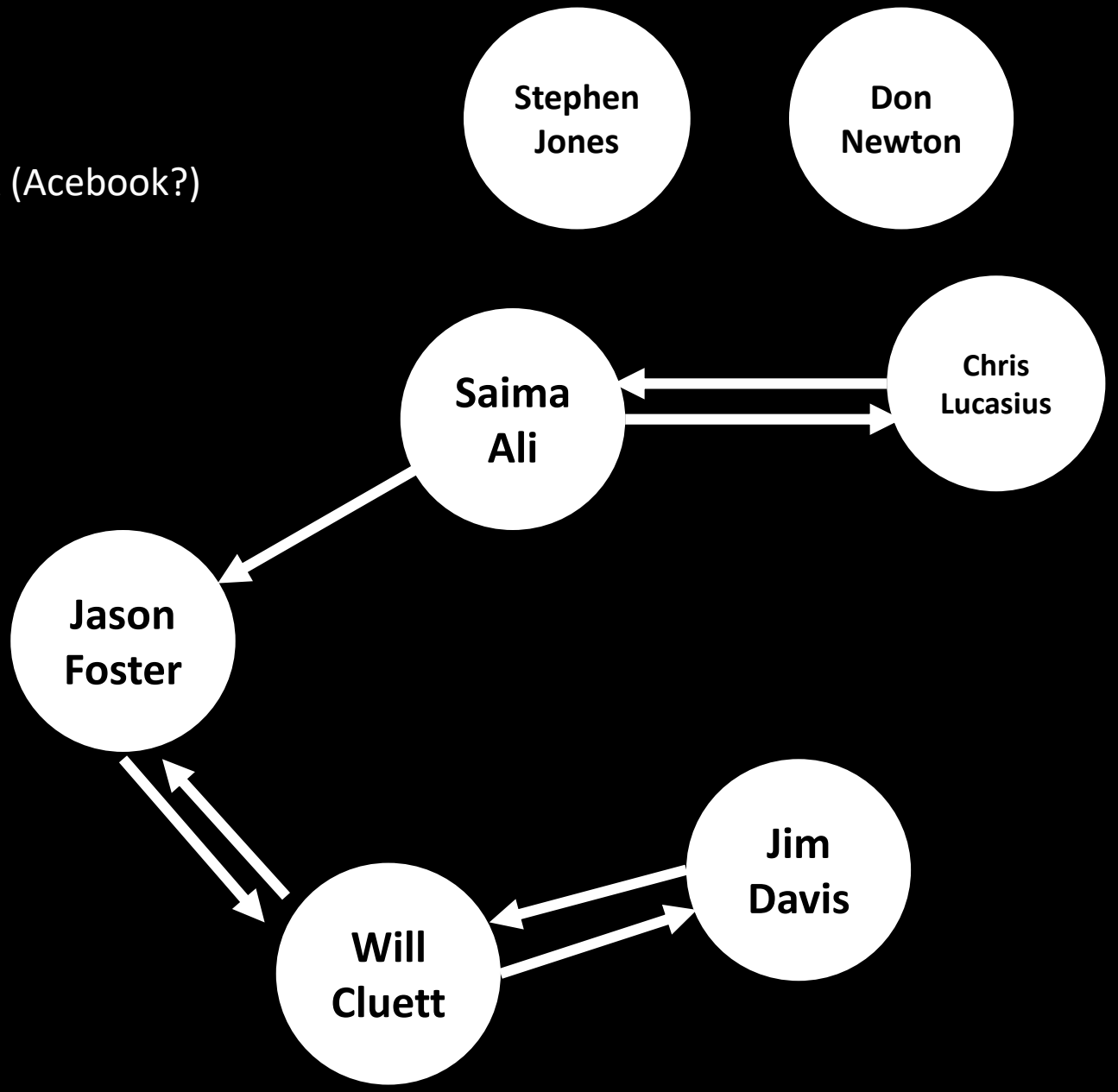
Graphs

An EngSci Social Network (Acebook?)



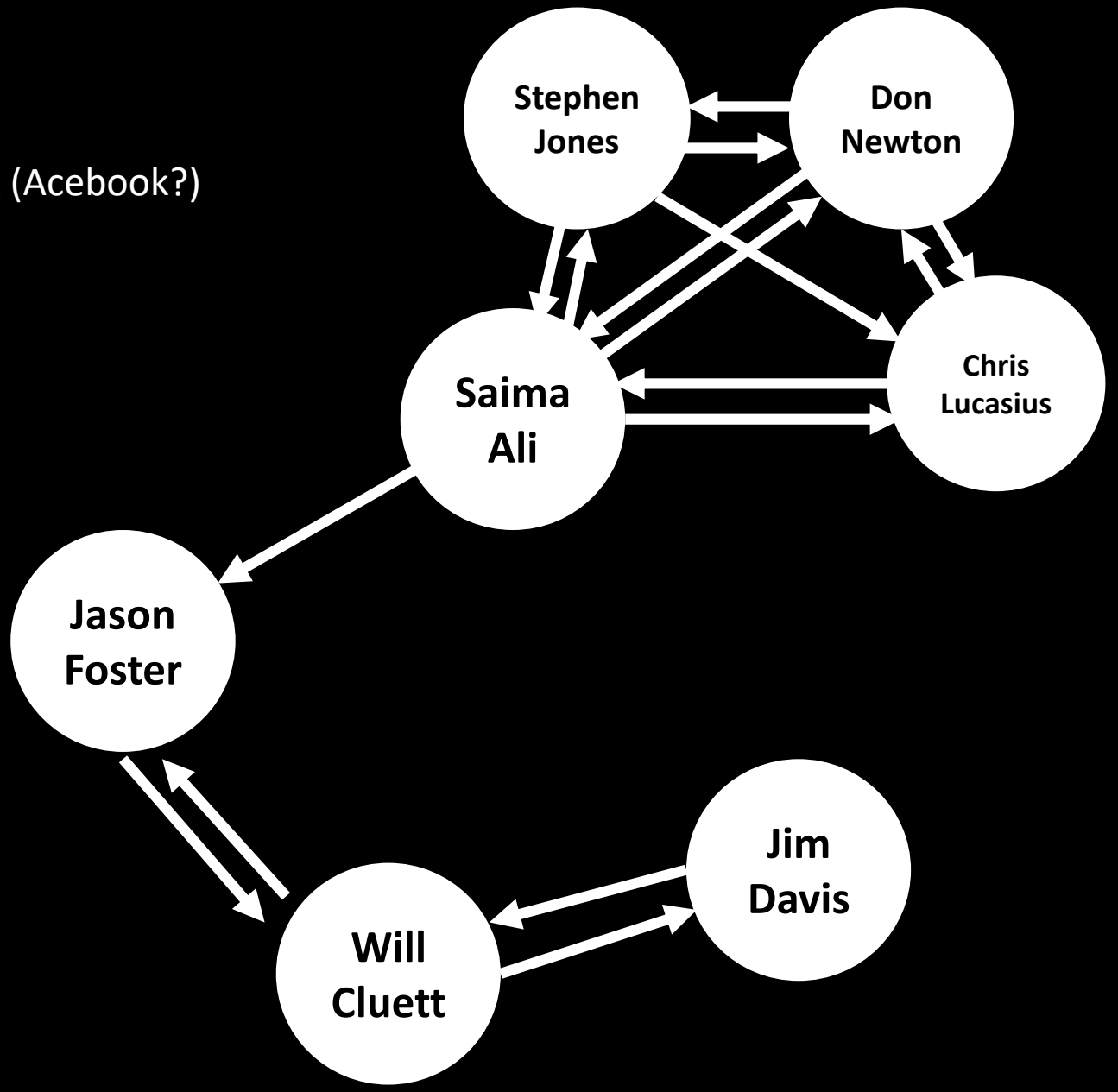
Graphs

An EngSci Social Network (Acebook?)



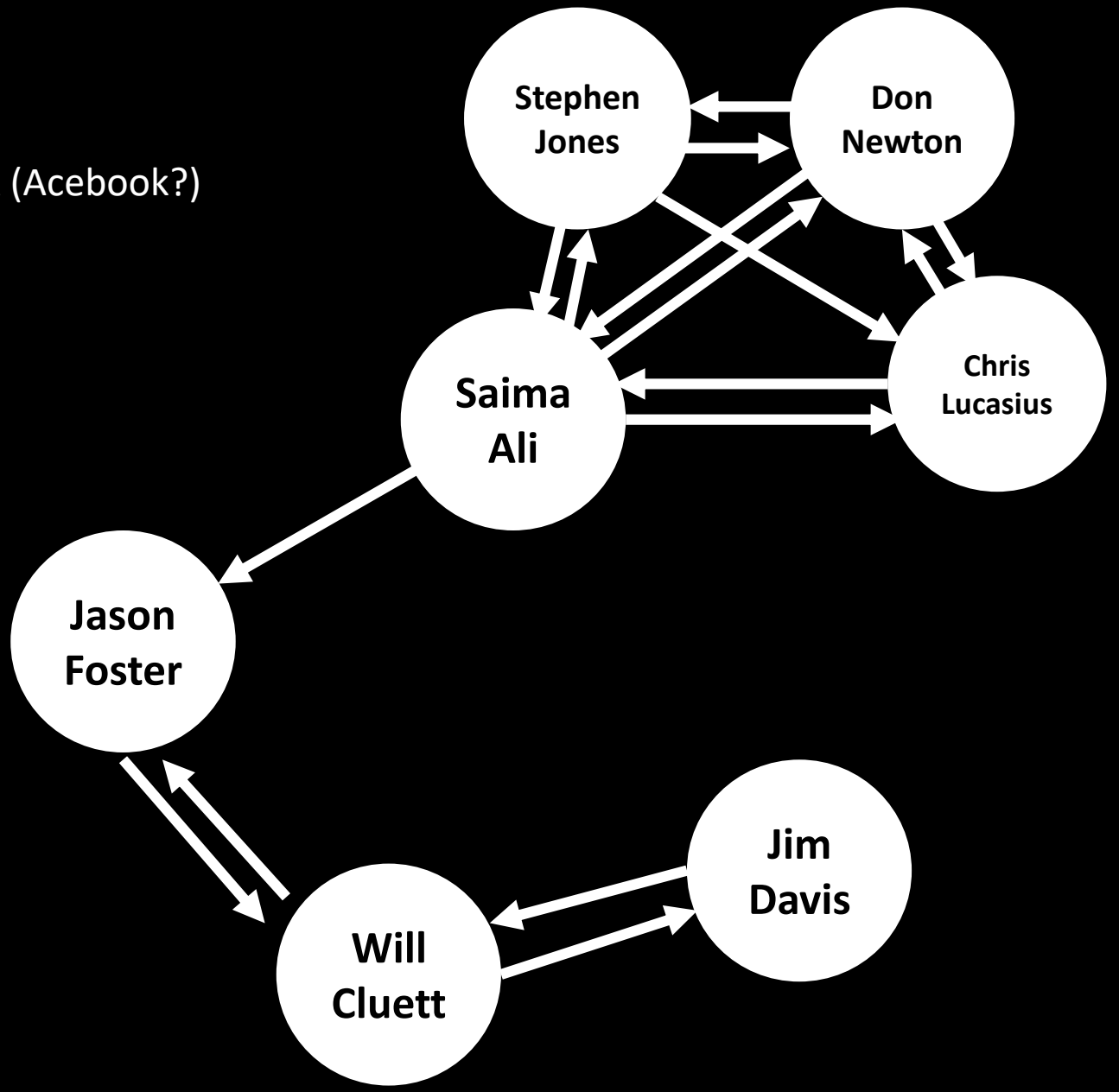
Graphs

An EngSci Social Network (Acebook?)



Graphs

An EngSci Social Network (Acebook?)

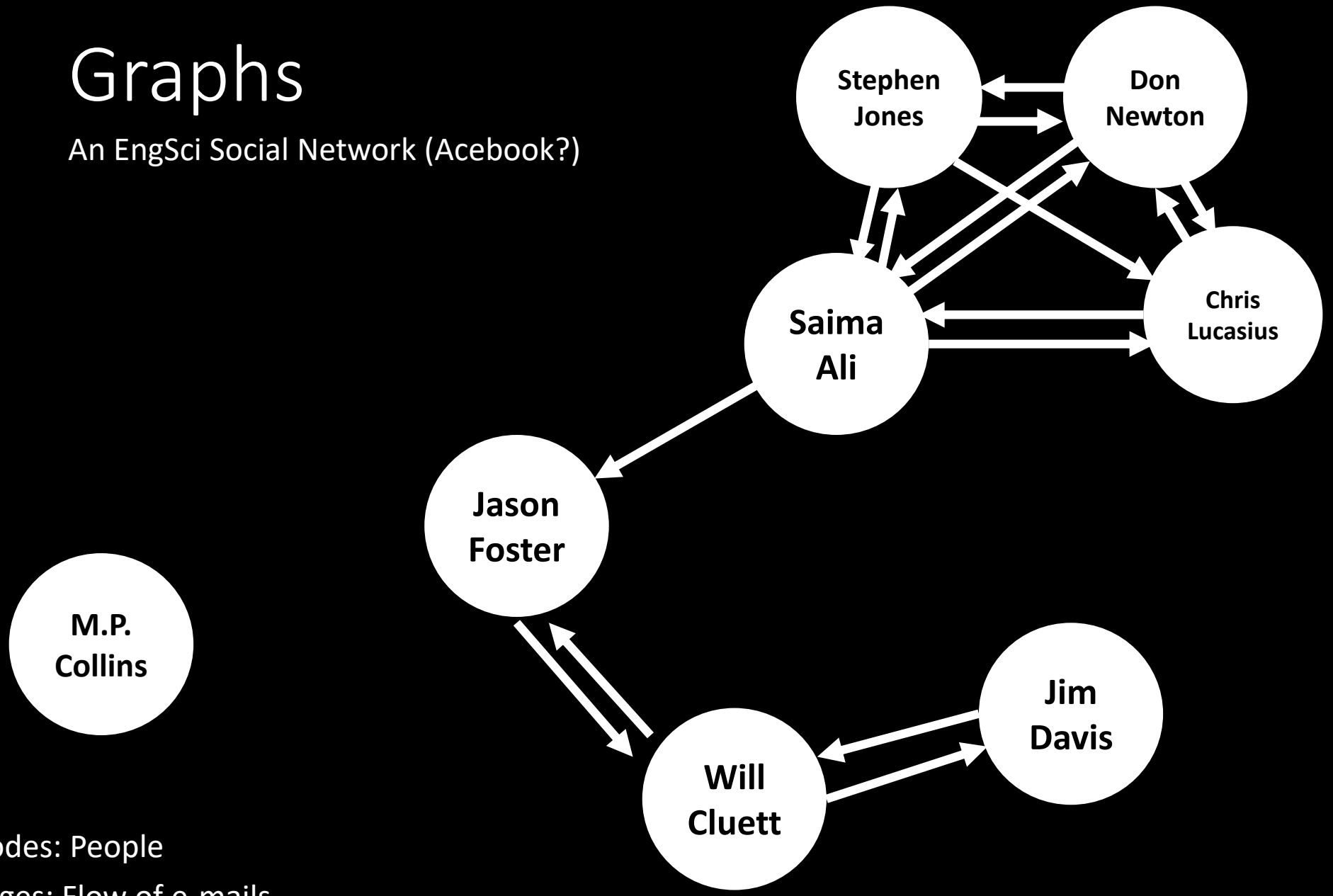


Nodes: People

Edges: Flow of e-mails

Graphs

An EngSci Social Network (Acebook?)

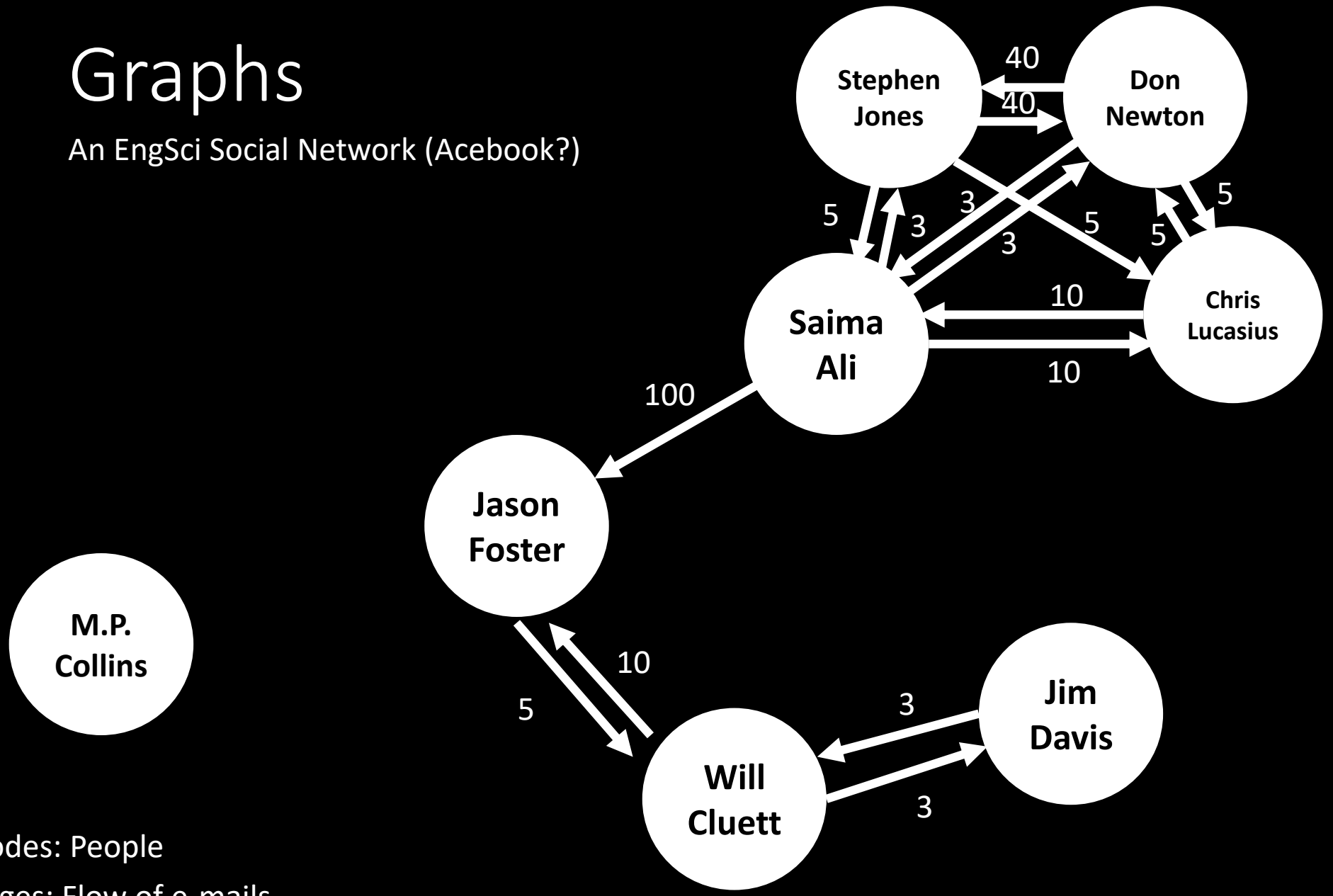


Nodes: People

Edges: Flow of e-mails

Graphs

An EngSci Social Network (Acebook?)



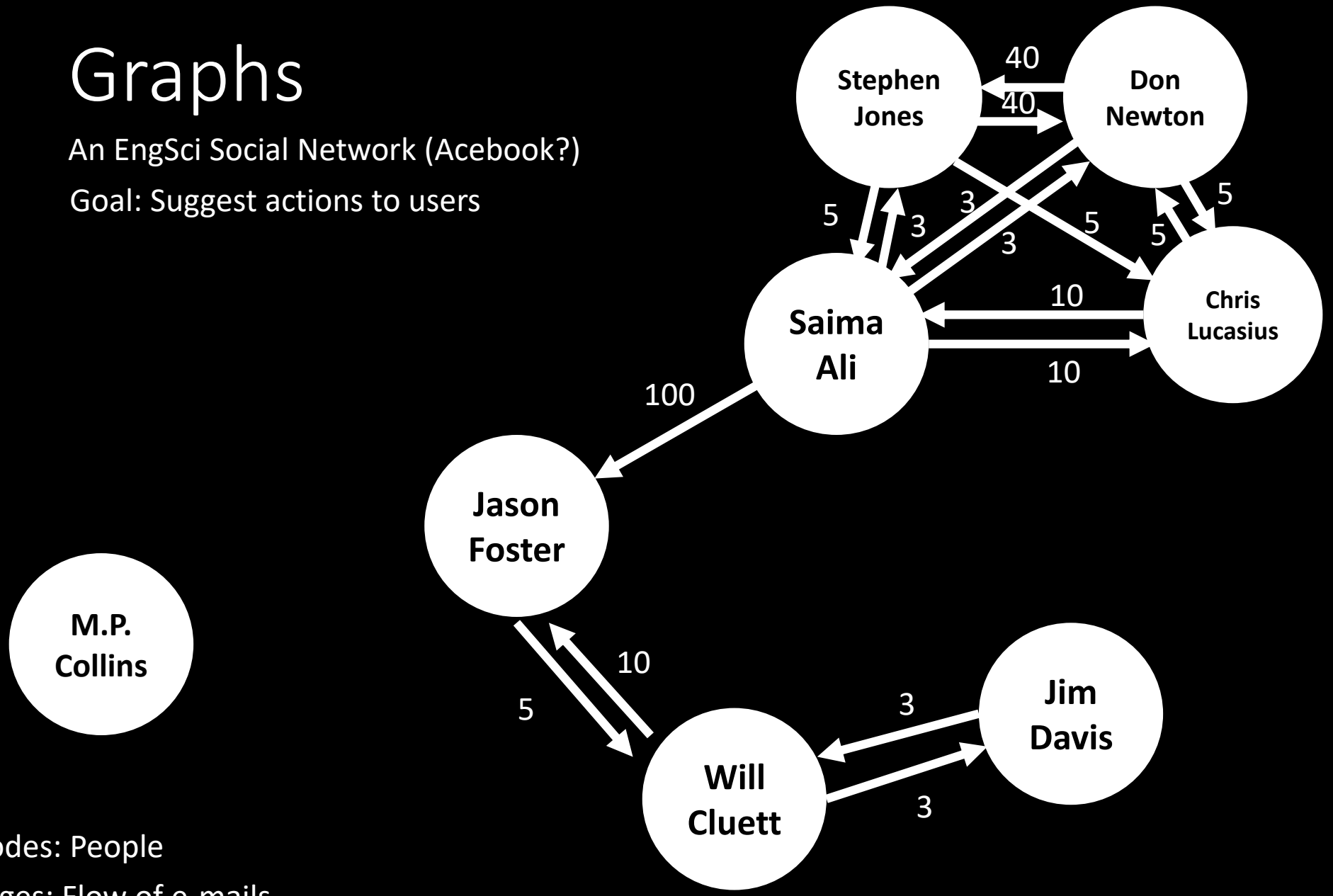
Nodes: People

Edges: Flow of e-mails

Graphs

An EngSci Social Network (Acebook?)

Goal: Suggest actions to users



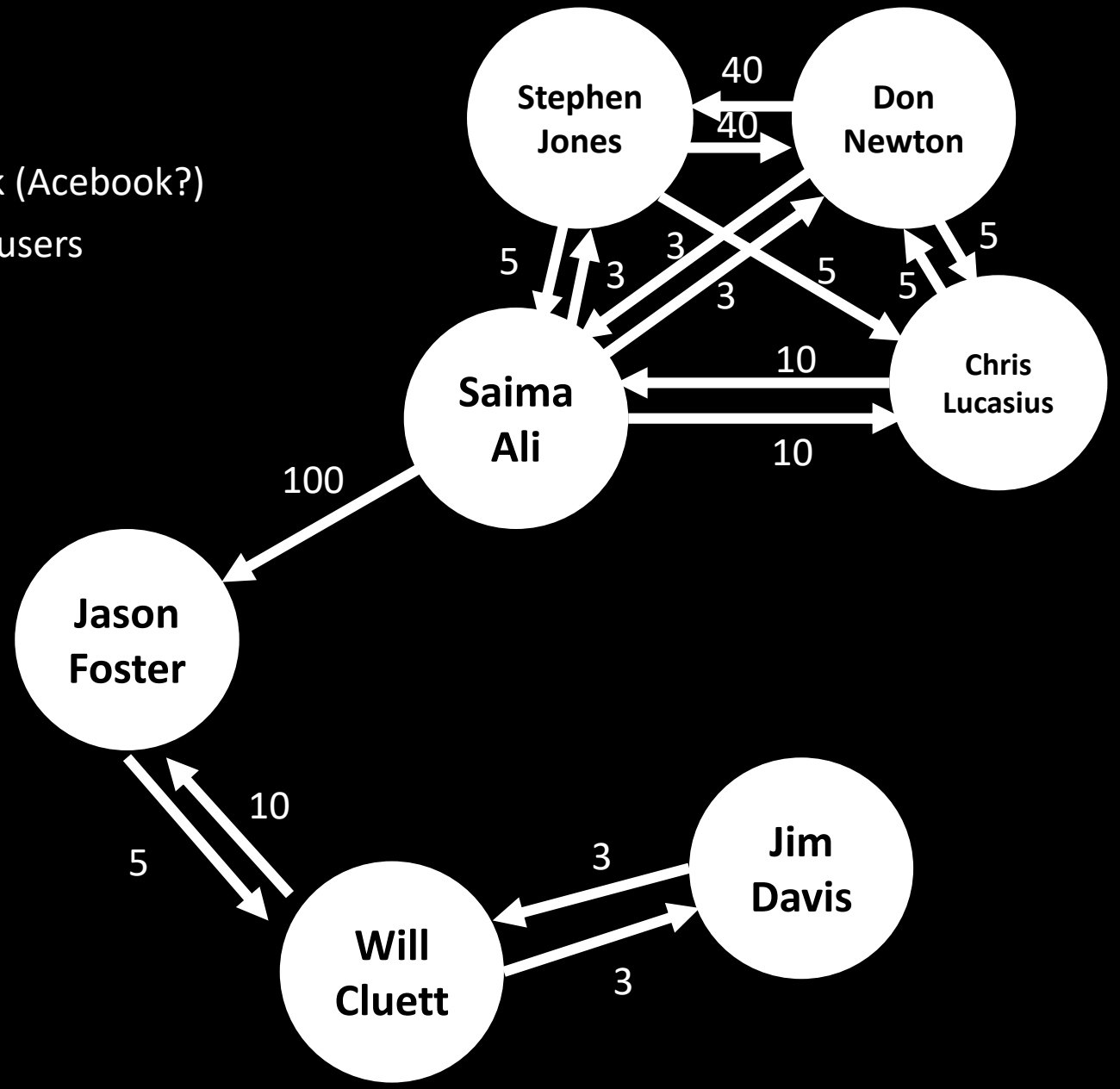
Nodes: People

Edges: Flow of e-mails

Graphs

An EngSci Social Network (Acebook?)

Goal: Suggest actions to users



You're missing out.

M.P. Collins

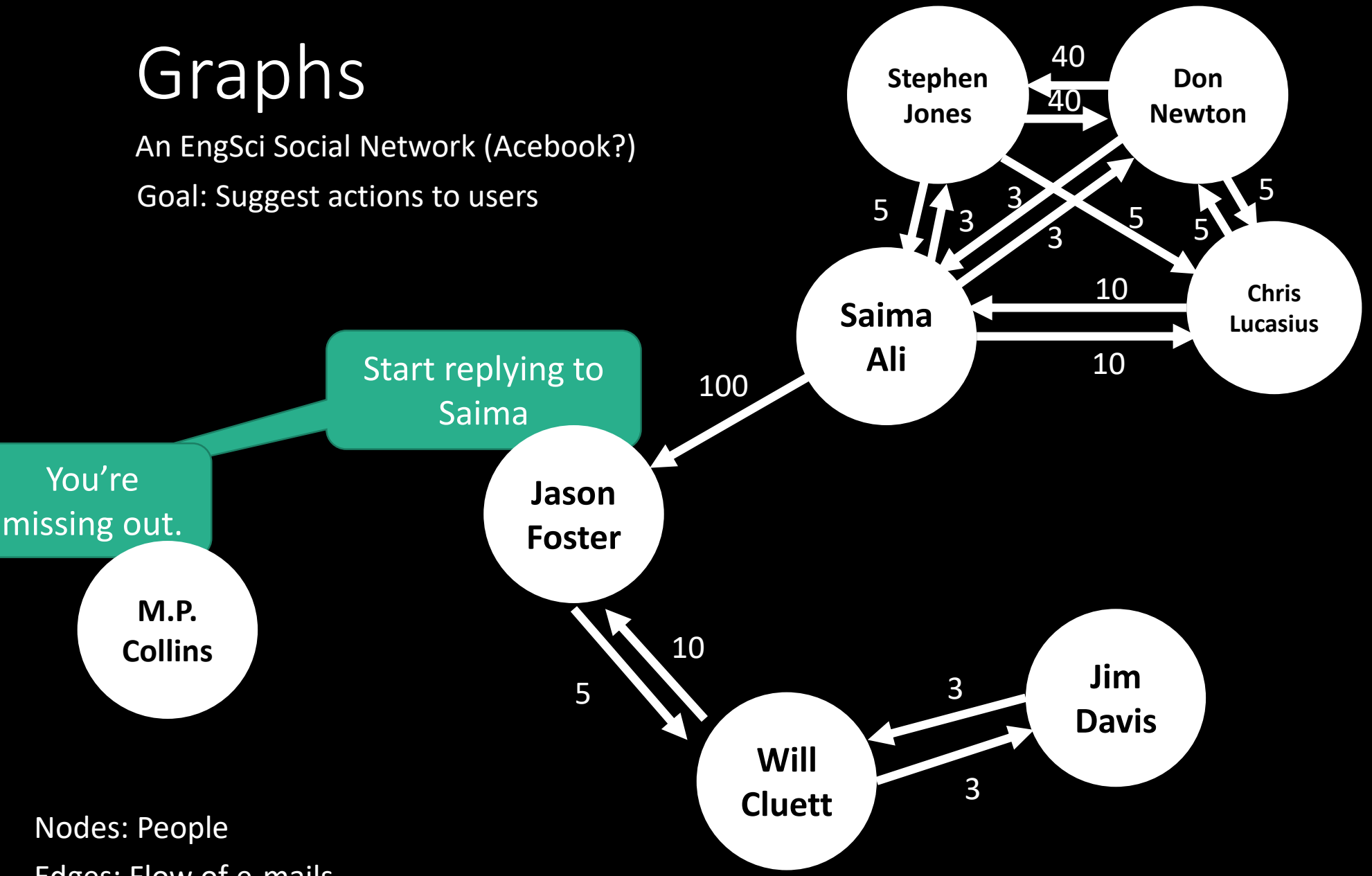
Nodes: People

Edges: Flow of e-mails

Graphs

An EngSci Social Network (Acebook?)

Goal: Suggest actions to users



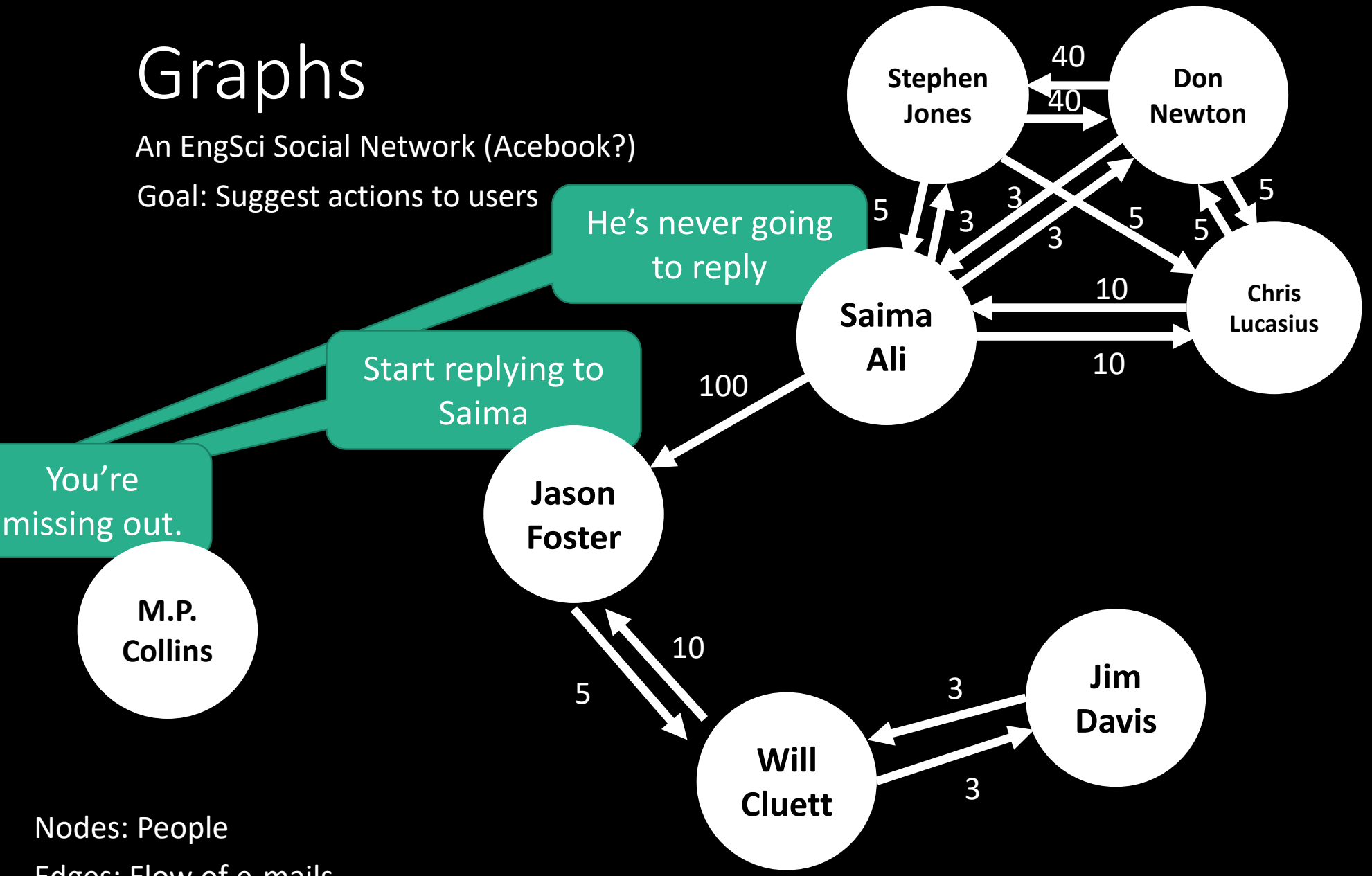
Nodes: People

Edges: Flow of e-mails

Graphs

An EngSci Social Network (Acebook?)

Goal: Suggest actions to users



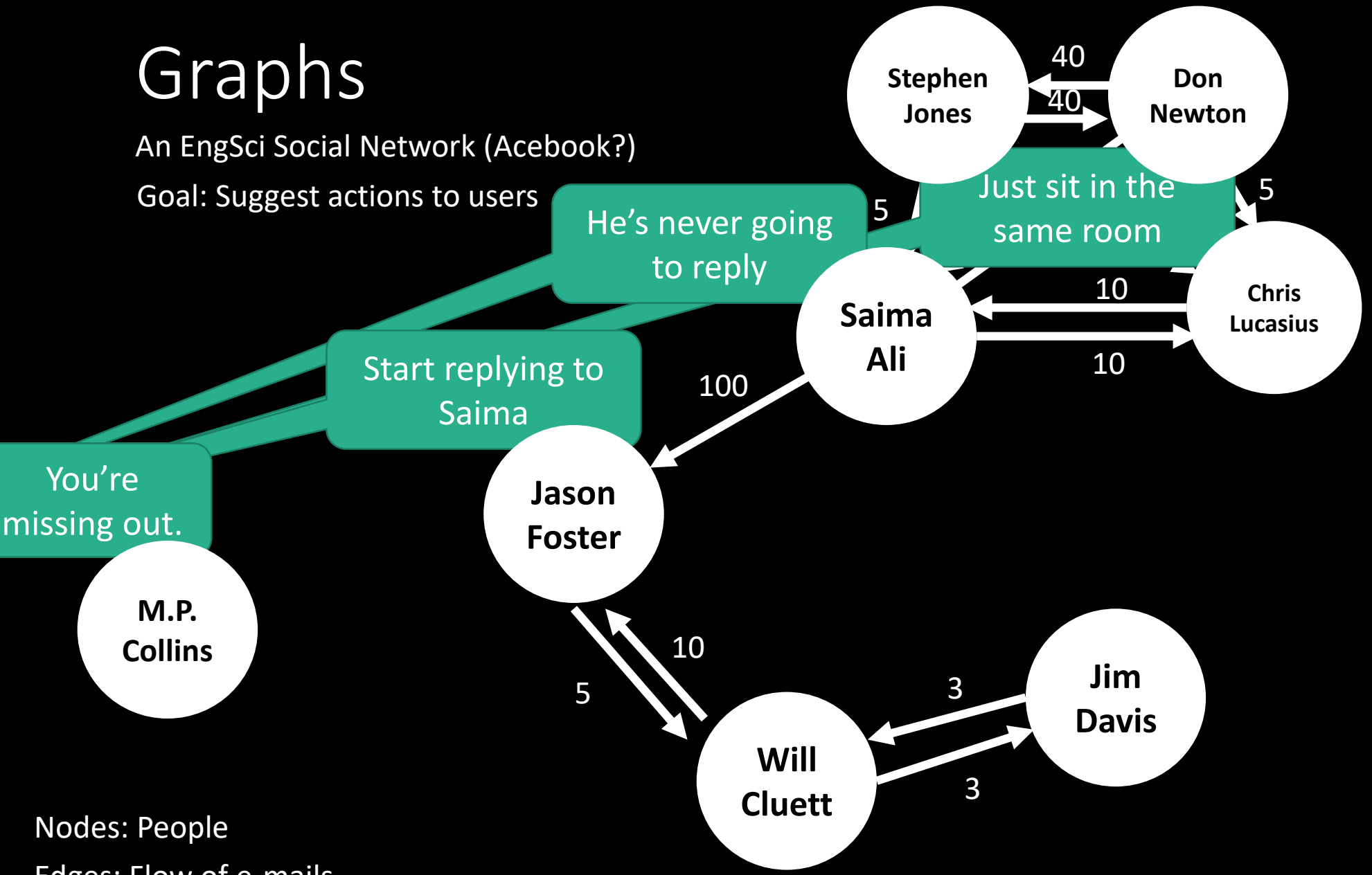
Nodes: People

Edges: Flow of e-mails

Graphs

An EngSci Social Network (Acebook?)

Goal: Suggest actions to users



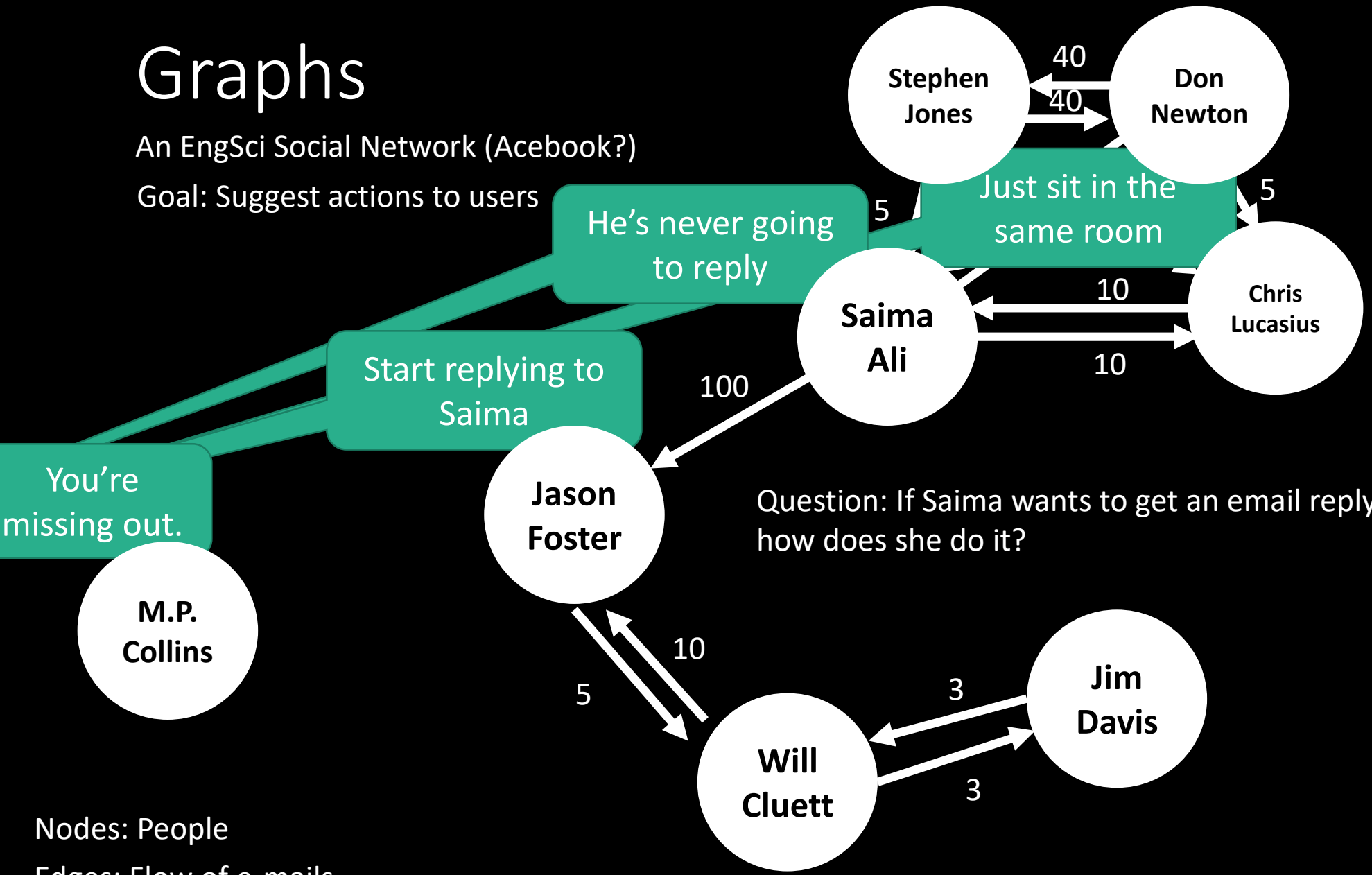
Nodes: People

Edges: Flow of e-mails

Graphs

An EngSci Social Network (Acebook?)

Goal: Suggest actions to users



Question: If Saima wants to get an email reply from Prof Foster, how does she do it?

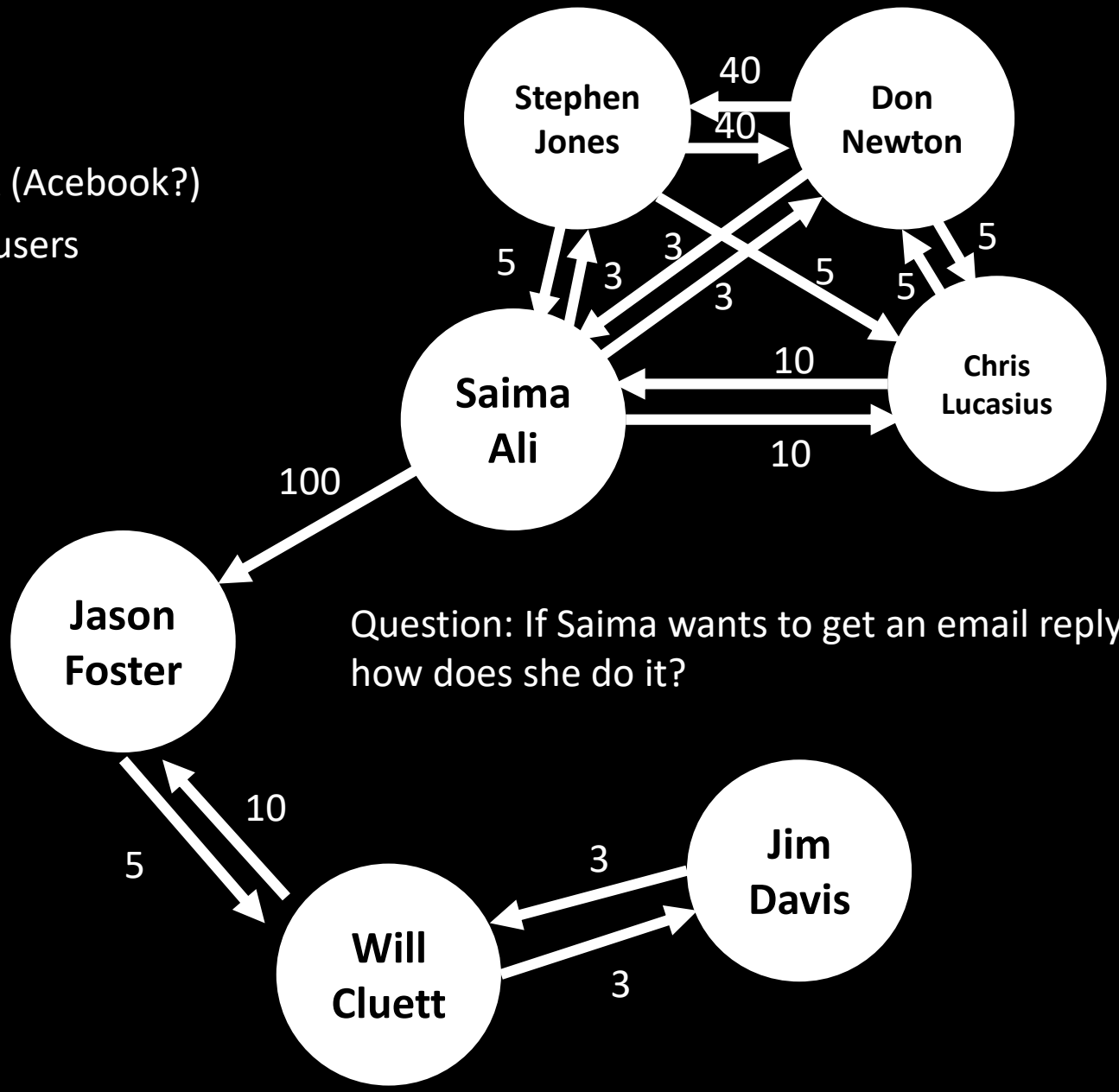
Nodes: People

Edges: Flow of e-mails

Graphs

An EngSci Social Network (Acebook?)

Goal: Suggest actions to users



Question: If Saima wants to get an email reply from Prof Foster, how does she do it?

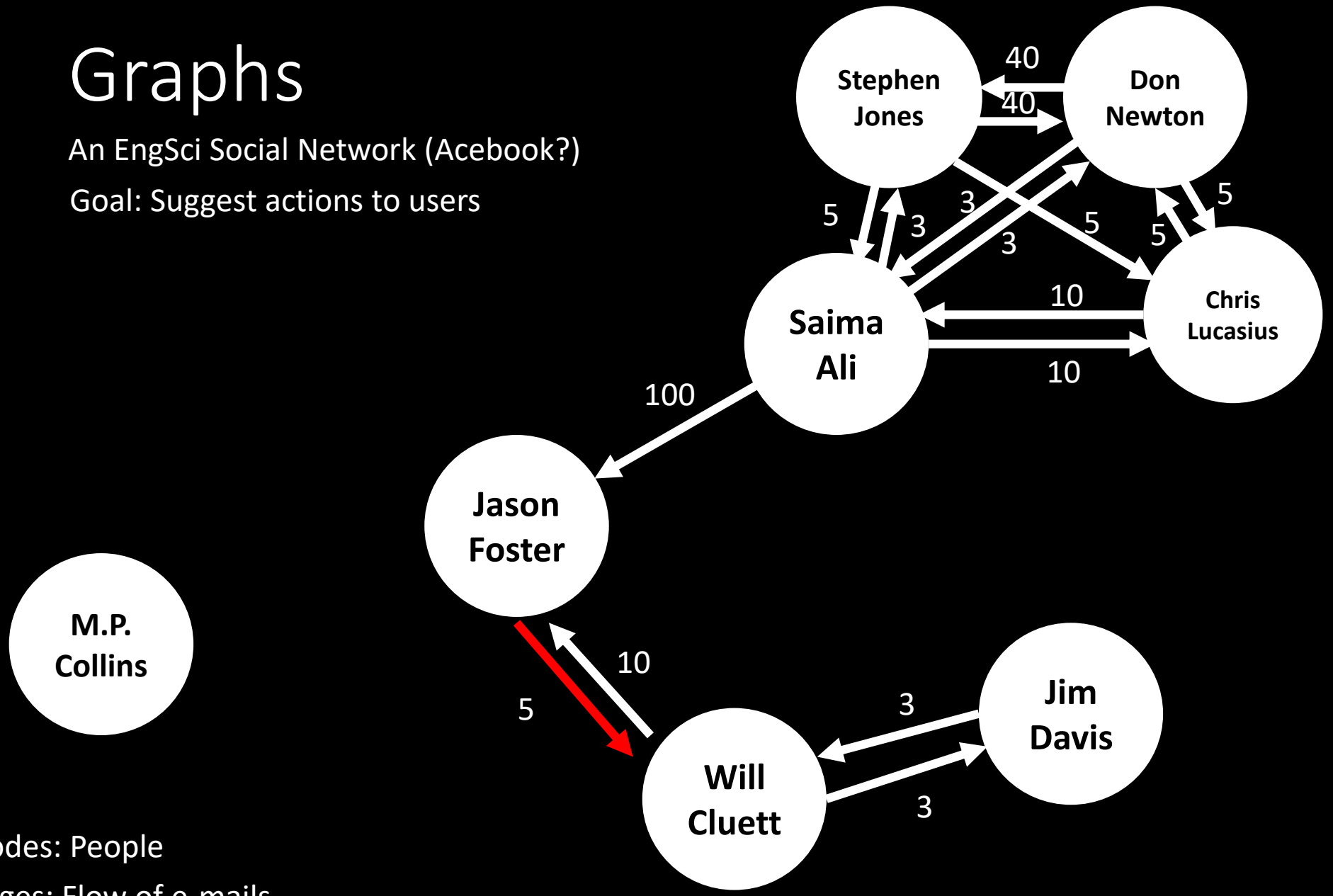
Nodes: People

Edges: Flow of e-mails

Graphs

An EngSci Social Network (Acebook?)

Goal: Suggest actions to users



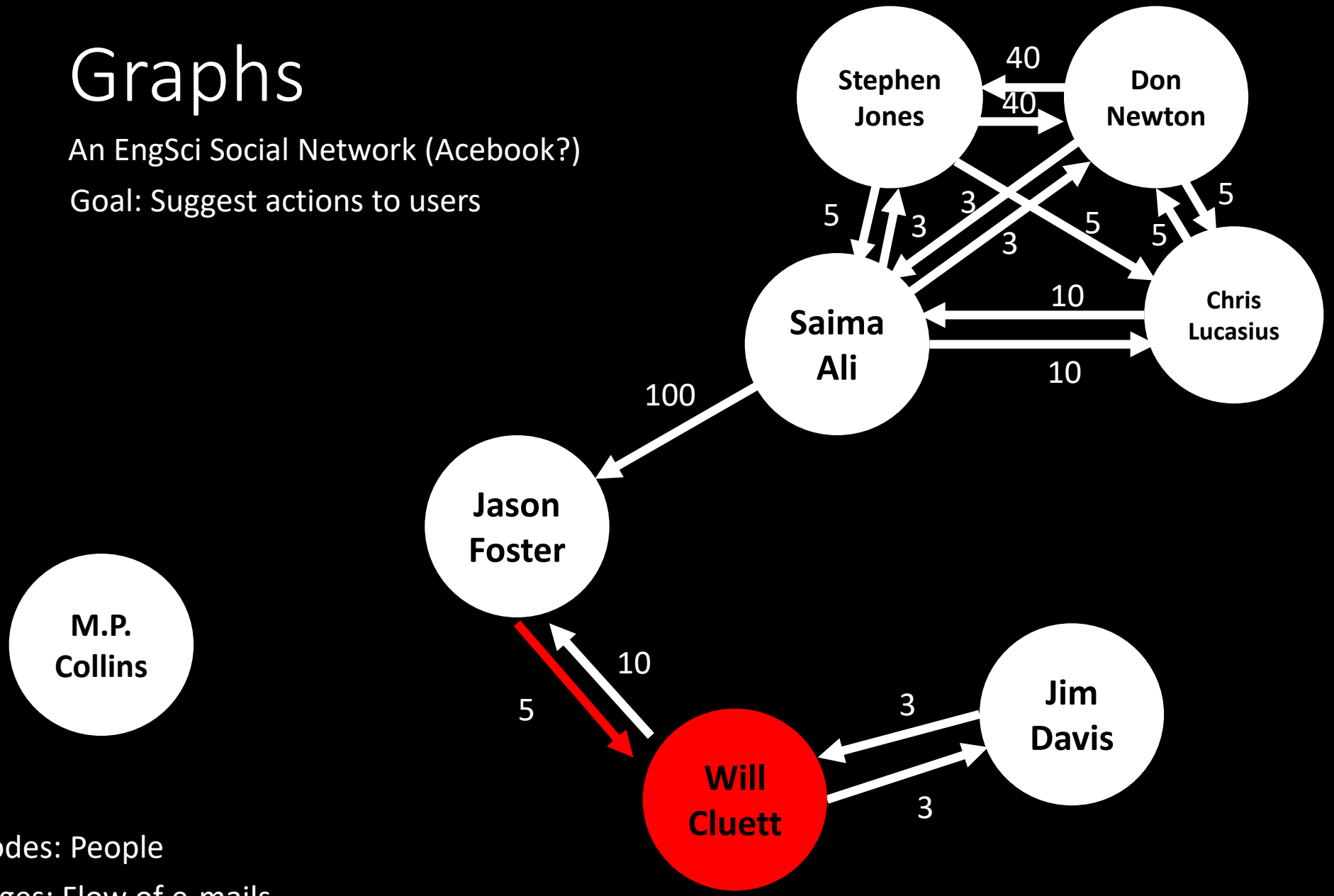
Nodes: People

Edges: Flow of e-mails

Graphs

An EngSci Social Network (Acebook?)

Goal: Suggest actions to users



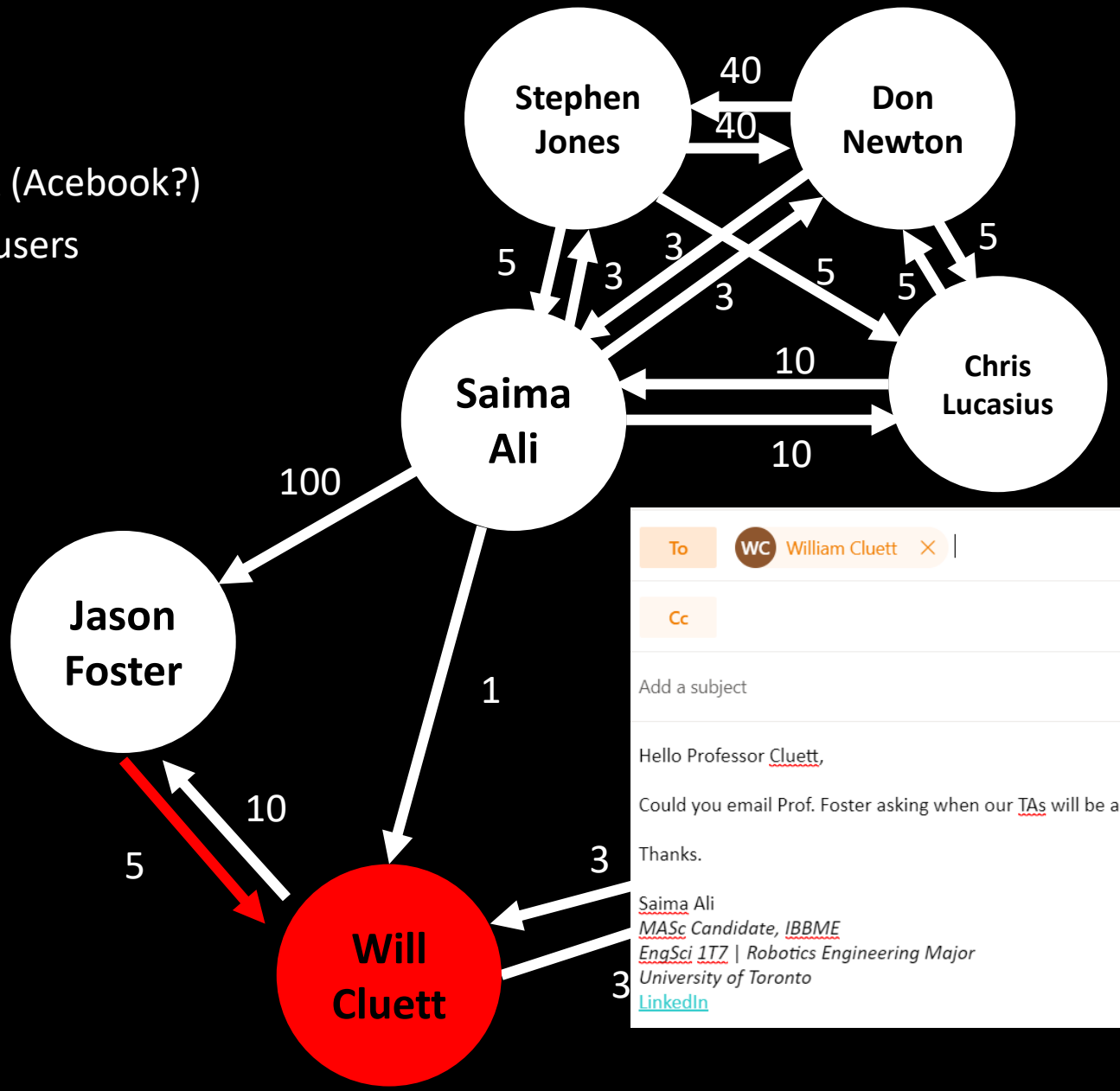
Nodes: People

Edges: Flow of e-mails

Graphs

An EngSci Social Network (Acebook?)

Goal: Suggest actions to users



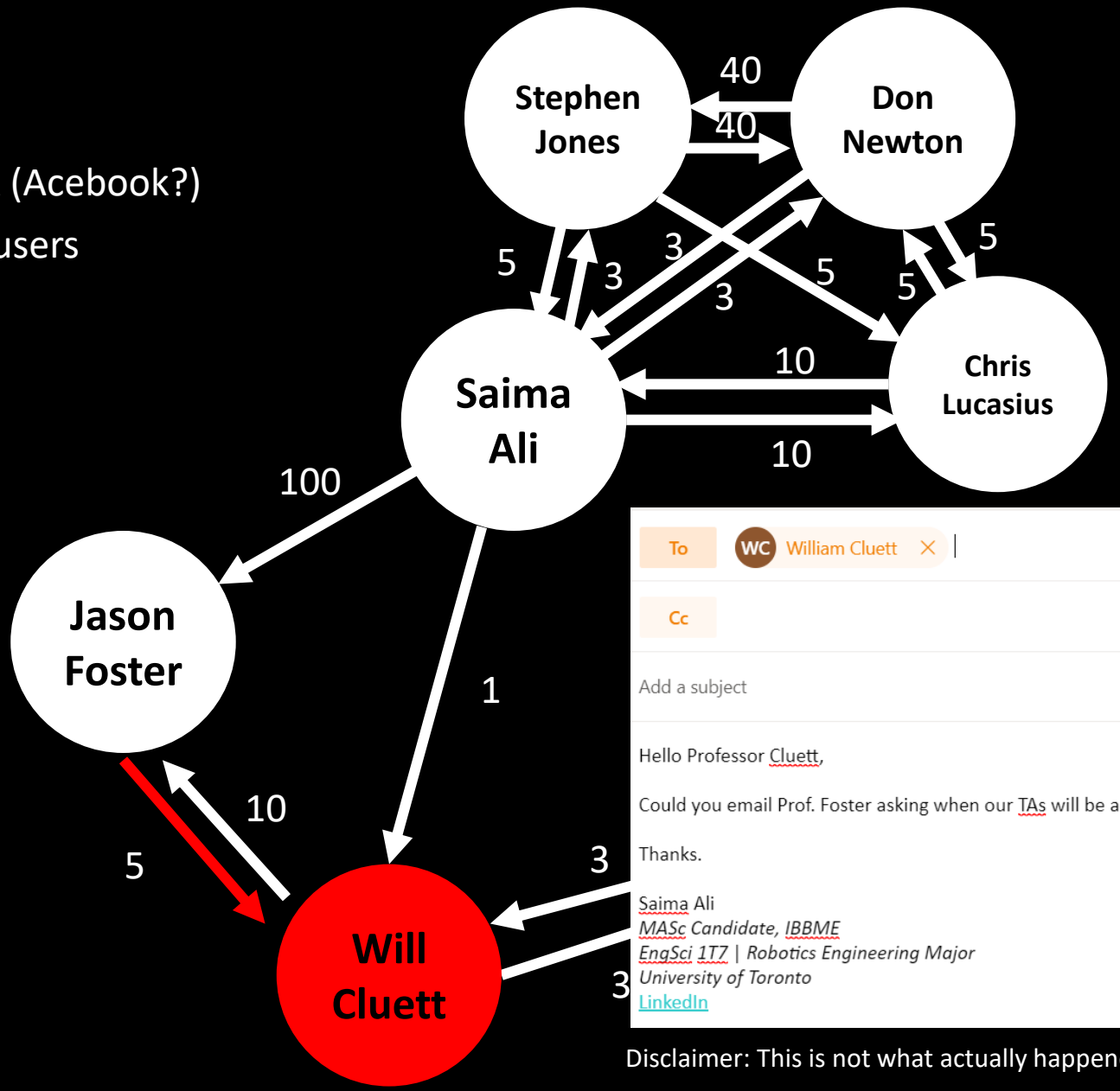
Nodes: People

Edges: Flow of e-mails

Graphs

An EngSci Social Network (Acebook?)

Goal: Suggest actions to users



To WC William Cluett

Cc

Add a subject

Hello Professor Cluett,

Could you email Prof. Foster asking when our TAs will be added to Discourse?

Thanks.

Saima Ali
MASc Candidate, IBBME
EngSci 1T7 | Robotics Engineering Major
University of Toronto
LinkedIn

Disclaimer: This is not what actually happened

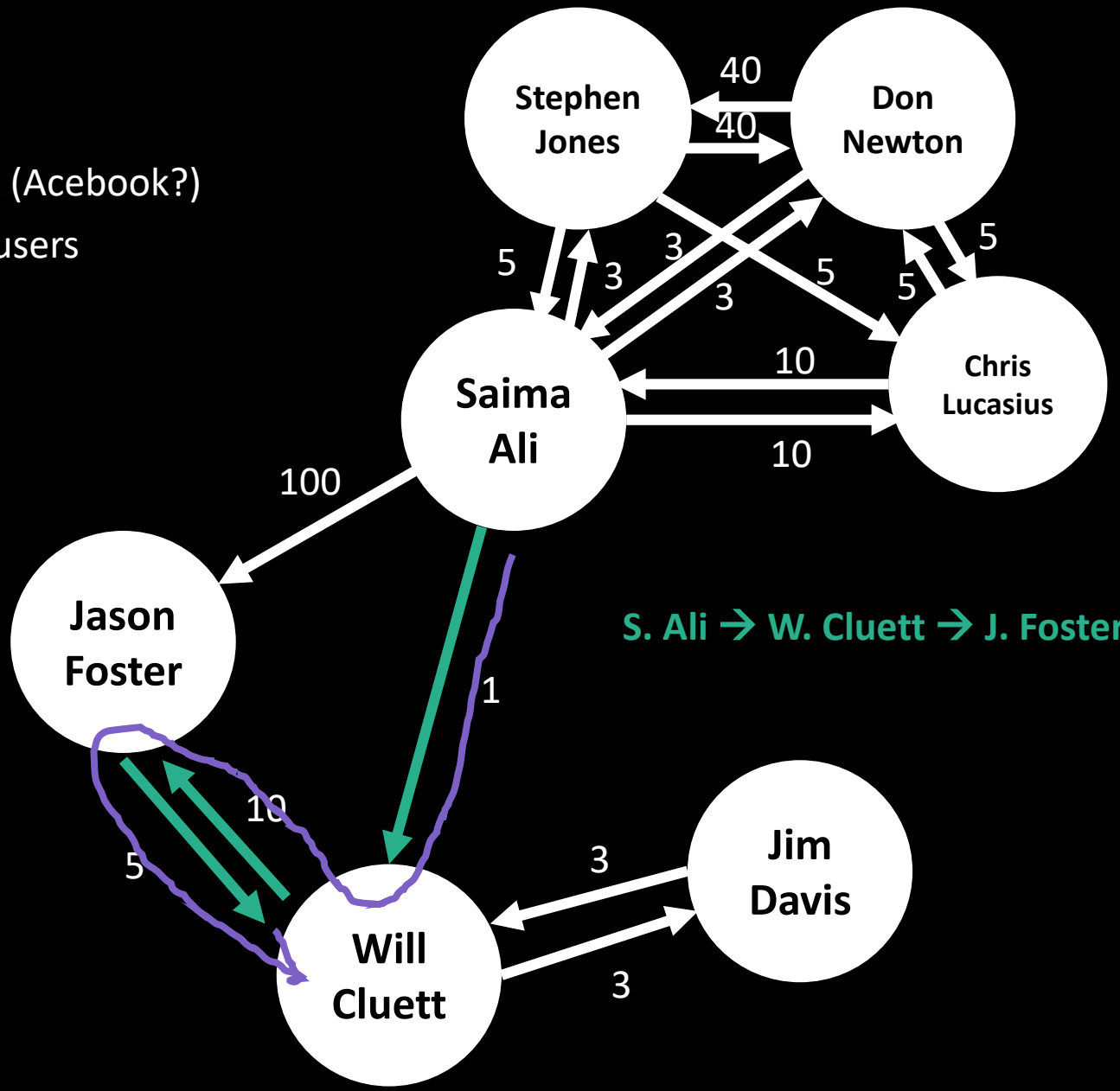
Nodes: People

Edges: Flow of e-mails

Graphs

An EngSci Social Network (Acebook?)

Goal: Suggest actions to users



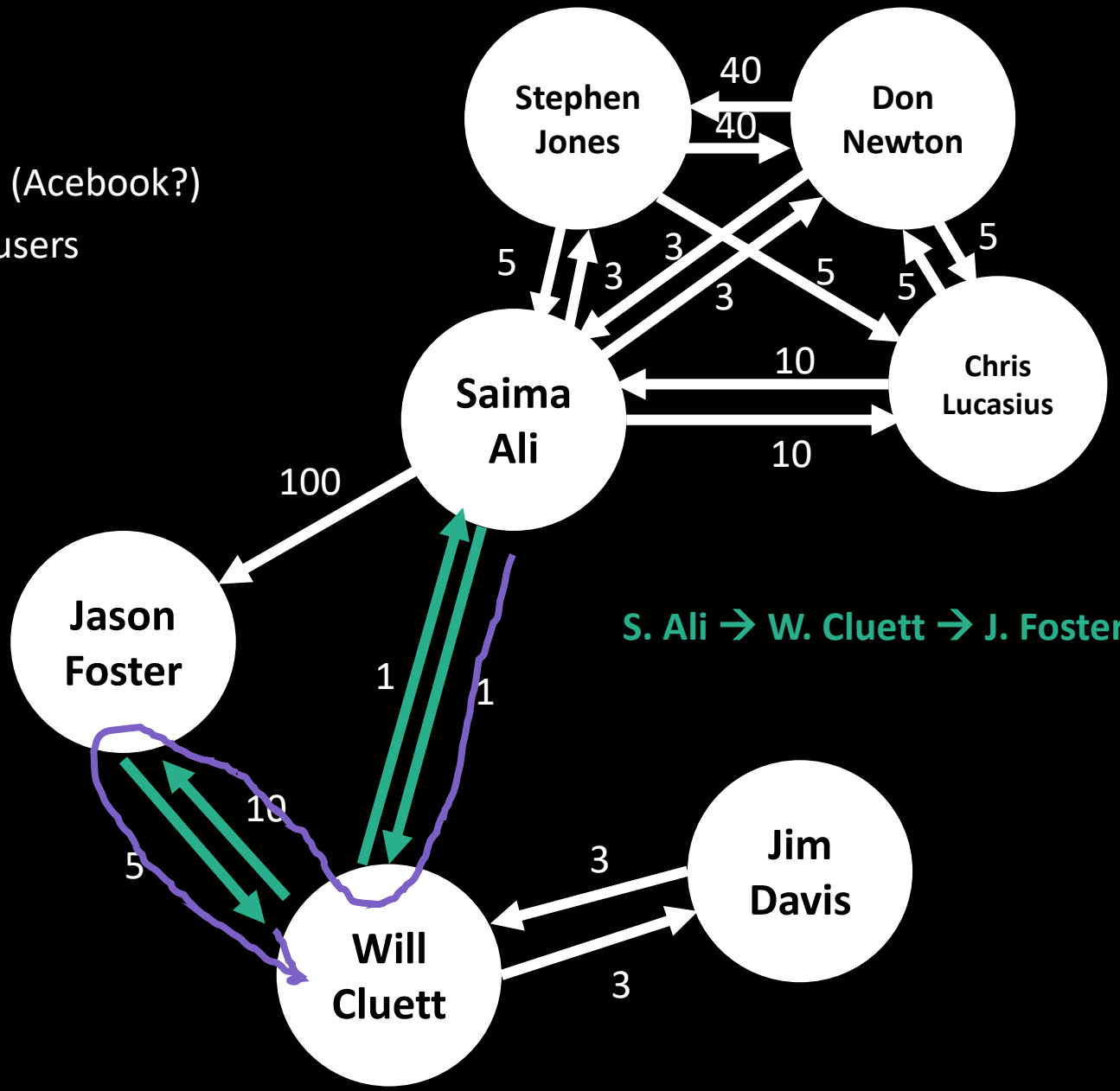
Nodes: People

Edges: Flow of e-mails

Graphs

An EngSci Social Network (Acebook?)

Goal: Suggest actions to users



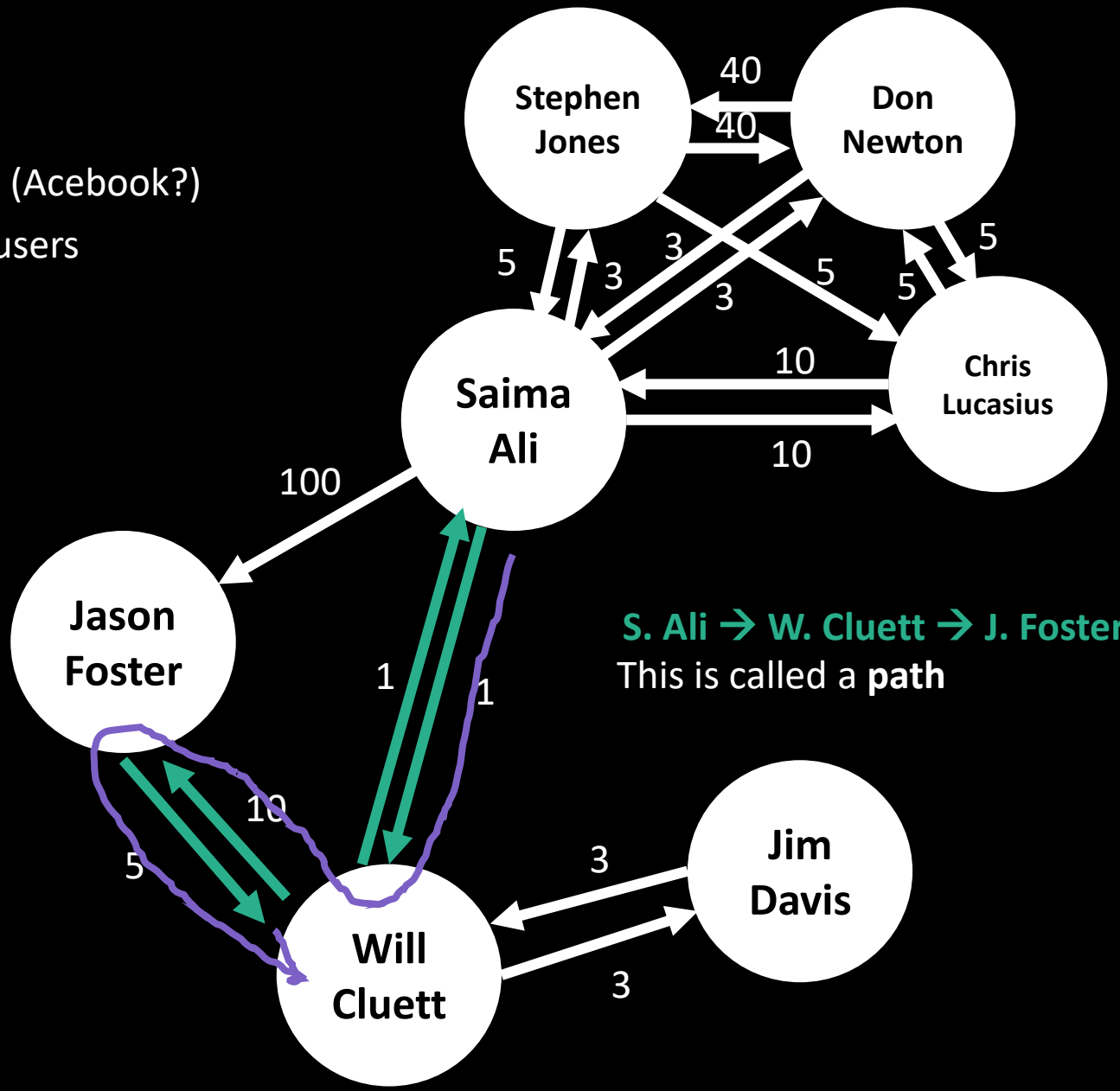
Nodes: People

Edges: Flow of e-mails

Graphs

An EngSci Social Network (Acebook?)

Goal: Suggest actions to users



S. Ali → W. Cluett → J. Foster→ S. Ali

This is called a **path**

Nodes: People

Edges: Flow of e-mails

Recall: Sets

- Sets are collections of **unique entities**
- Our treatment of graphs will heavily rely on set notation

$\forall$ $\subset$

$\in$ $\cap$

$\exists$

Recall: Sets

- Sets are collections of **unique entities**
- Our treatment of graphs will heavily rely on set notation

$\forall x \in \mathbb{R}$

$\in$

$\exists$

$\subset$

$\cap$

Recall: Sets

- Sets are collections of **unique entities**
- Our treatment of graphs will heavily rely on set notation

For all items x in the set of real numbers (for every real number)

$\forall x \in \mathbb{R}$

$\in$

$\exists$

$\subset$

$\cap$

Recall: Sets

- Sets are collections of **unique entities**
- Our treatment of graphs will heavily rely on set notation

For all items x in the set of real numbers (for every real number)

$$\forall x \in \mathbb{R}, \exists y \in \mathbb{R}$$

$$\subset$$

$$\cap$$

$$\exists$$

Recall: Sets

- Sets are collections of **unique entities**
- Our treatment of graphs will heavily rely on set notation

For all items x in the set of real numbers (for every real number)

$$\forall x \in \mathbb{R}, \exists y \in \mathbb{R}$$

There exists an item y in the set of real numbers

$\subset$

$\cap$

$\exists$

Recall: Sets

- Sets are collections of **unique entities**
- Our treatment of graphs will heavily rely on set notation

For all items x in the set of real numbers (for every real number)

$$\forall x \in \mathbb{R}, \exists y \in \mathbb{R} \text{ s.t.}$$

There exists an item y in the set of real numbers

$$\subset$$

$$\cap$$

Recall: Sets

- Sets are collections of **unique entities**
- Our treatment of graphs will heavily rely on set notation

For all items x in the set of real numbers (for every real number)

$$\forall x \in \mathbb{R}, \exists y \in \mathbb{R} \text{ s.t.}$$

There exists an item y in the set of real numbers such that

$$\subset$$

$$\cap$$

Recall: Sets

- Sets are collections of **unique entities**
- Our treatment of graphs will heavily rely on set notation

For all items x in the set of real numbers (for every real number)

$$\forall x \in \mathbb{R}, \exists y \in \mathbb{R} \text{ s.t. } x \cdot y = x$$

There exists an item y in the set of real numbers such that

$\subset$

$\cap$

Recall: Sets

- Sets are collections of **unique entities**
- Our treatment of graphs will heavily rely on set notation

$$\forall x \in \mathbb{R}, \exists y \in \mathbb{R} \text{ s.t. } x \cdot y = x$$

For all items x in the set of real numbers (for every real number)
There exists an item y in the set of real numbers such that…

$\subset$

$\cap$

Recall: Sets

- Sets are collections of **unique entities**
- Our treatment of graphs will heavily rely on set notation

$$\forall x \in \mathbb{R}, \exists y \in \mathbb{R} \text{ s.t. } x \cdot y = x$$

For all items x in the set of real numbers (for every real number)
There exists an item y in the set of real numbers such that…

$$\mathbb{Z} \subset \mathbb{R}$$

The set of integers is a **subset** of the set of real numbers.

$$\{1, 2, 3\} \subseteq \{1, 2, 3\}$$

$$\cap$$

Recall: Sets

- Sets are collections of **unique entities**
- Our treatment of graphs will heavily rely on set notation

$$\forall x \in \mathbb{R}, \exists y \in \mathbb{R} \text{ s.t. } x \cdot y = x$$

For all items x in the set of real numbers (for every real number)
There exists an item y in the set of real numbers such that…

$$\mathbb{Z} \subset \mathbb{R}$$

The set of integers is a **subset** of the set of real numbers.

$$\{1, 2, 3\} \subseteq \{1, 2, 3\}$$

$$\mathbb{Z} \subseteq \mathbb{Z} \cap \mathbb{R}$$

The set of integers is a **subset or equal to** the **intersection** between the set of integers and the set of real numbers.

Graphs

- Collection of **nodes** and **edges**

$$G(V, E)$$

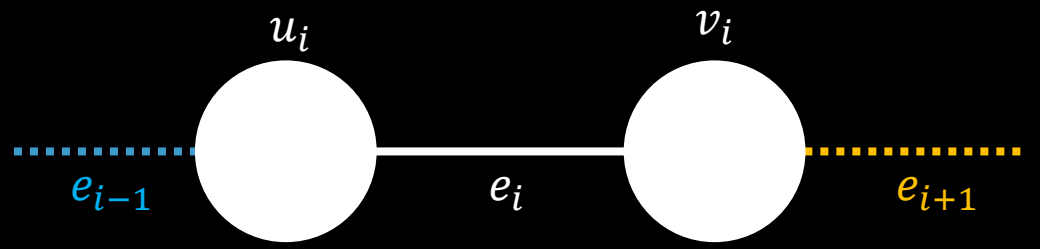
Graphs

- Nodes: Denoted by a set V
- Edges: Denoted by a set E
  - Impose a rule that $\forall e \in E$, e connects exactly 2 nodes $u, v \in V$

Graphs

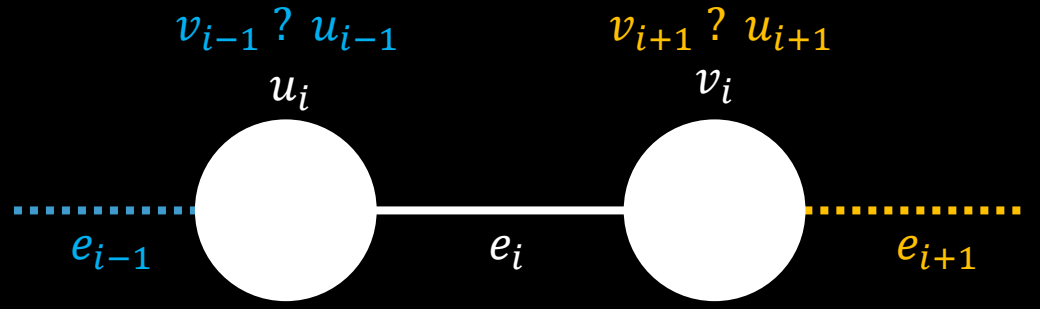
- Nodes: Denoted by a set V
- Edges: Denoted by a set E
  - Impose a rule that $\forall e \in E$, e connects exactly 2 nodes $u, v \in V$
- Path: A set of unique edges $\{e_1, e_2, \dots, e_n\} \subseteq E$ for which the following is satisfied:
  - $e_i, i = 1 \dots n$ connects two nodes denoted $v_i, u_i \in V$
  - Then, $\forall i = \{1 \dots n-1\}$, $\exists w \in \{u_{i+1}, v_{i+1}\}$ such that $w \in \{v_i, u_i\}$
    - I.e. $\exists w \in \{u_{i+1}, v_{i+1}\}$ such that $w \in \{u_{i+1}, v_{i+1}\} \cap \{v_i, u_i\}$
    - The intersection must exist

Graphs



- Nodes: Denoted by a set V
- Edges: Denoted by a set E
  - Impose a rule that $\forall e \in E$, e connects exactly 2 nodes $u, v \in V$
- Path: A set of unique edges $\{e_1, e_2, \dots, e_n\} \subseteq E$ for which the following is satisfied:
  - $e_i, i = 1 \dots n$ connects two nodes denoted $v_i, u_i \in V$
  - Then, $\forall i = \{1 \dots n-1\}$, $\exists w \in \{u_{i+1}, v_{i+1}\}$ such that $w \in \{v_i, u_i\}$
    - I.e. $\exists w \in \{u_{i+1}, v_{i+1}\}$ such that $w \in \{u_{i+1}, v_{i+1}\} \cap \{v_i, u_i\}$
    - The intersection must exist

Graphs



- Nodes: Denoted by a set V
- Edges: Denoted by a set E
  - Impose a rule that $\forall e \in E$, e connects exactly 2 nodes $u, v \in V$
- Path: A set of unique edges $\{e_1, e_2, \dots, e_n\} \subseteq E$ for which the following is satisfied:
  - $e_i, i = 1 \dots n$ connects two nodes denoted $v_i, u_i \in V$
  - Then, $\forall i = \{1 \dots n-1\}$, $\exists w \in \{u_{i+1}, v_{i+1}\}$ such that $w \in \{v_i, u_i\}$
    - I.e. $\exists w \in \{u_{i+1}, v_{i+1}\}$ such that $w \in \{u_{i+1}, v_{i+1}\} \cap \{v_i, u_i\}$
    - The intersection must exist

Graphs

- Nodes: Denoted by a set V
- Edges: Denoted by a set E
  - Impose a rule that $\forall e \in E$, e connects exactly 2 nodes $u, v \in V$
- Path: A set of unique edges $\{e_1, e_2, \dots, e_n\} \subseteq E$ f ich the following is satisfied:
  - $e_i, i = 1 \dots n-1$ co
  - Ther
    $\dots, v_{i+1}\} \cap \{v_i, u_i\}$

Just make sure your path consists of lines (edges) which are always connected by circles (nodes/vertices)

Graphs

- Neighbors:

  Let $u, v \in V$ be 2 nodes on a graph, $G(V, E)$.

  u and v are neighbors $\Leftrightarrow \exists e \in E$ connecting u, v

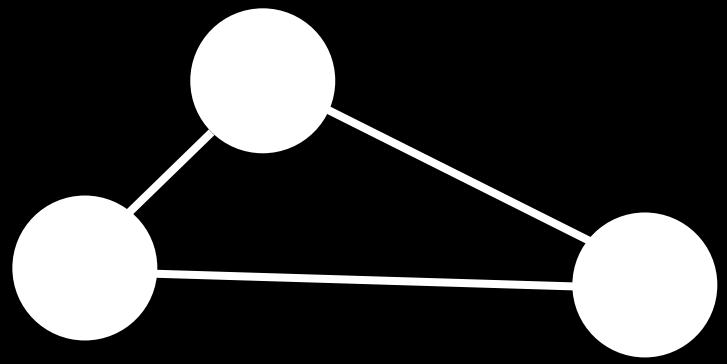
Graphs

- Neighbors:

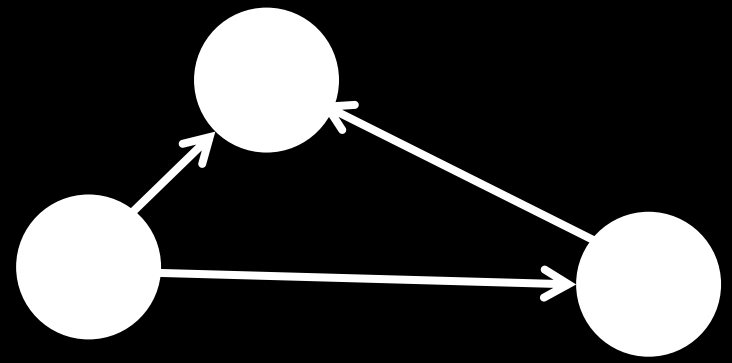
  Let $u, v \in V$ be 2 nodes on a graph, $G(V, E)$.

  u and v are neighbors $\Leftrightarrow \exists e \in E$ connecting u, v
- Just adjacent nodes (connected by an edge)

Graphs

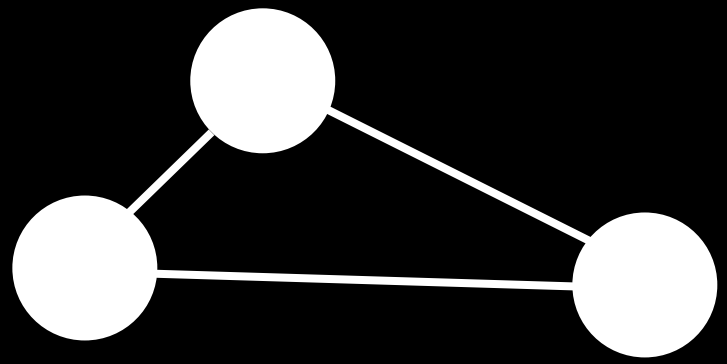


Undirected

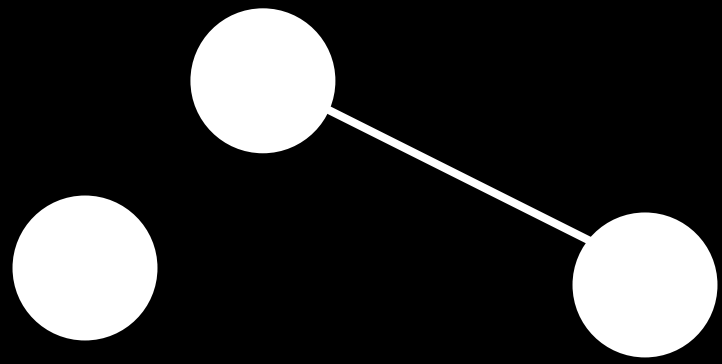


Directed

Graphs

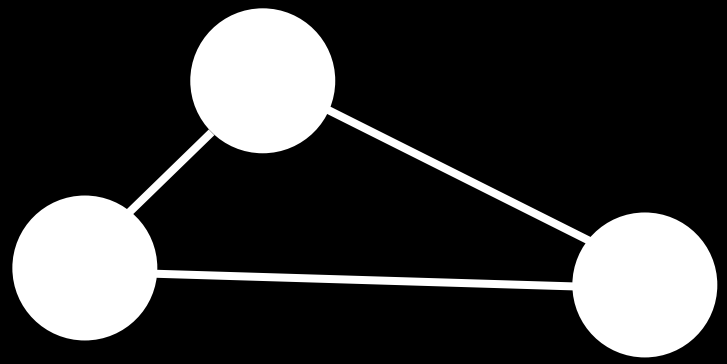


Connected



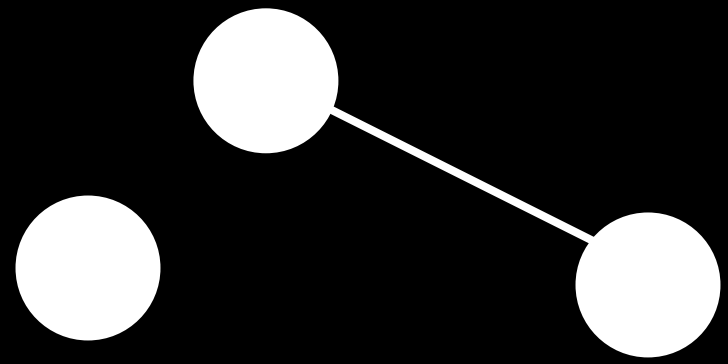
Disconnected

Graphs



Connected

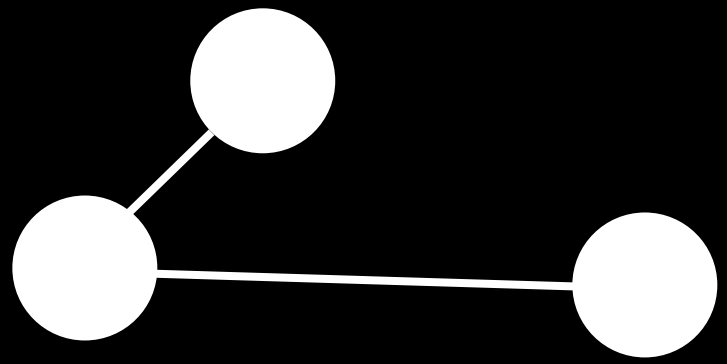
From any starting node, you can reach all other nodes



Disconnected

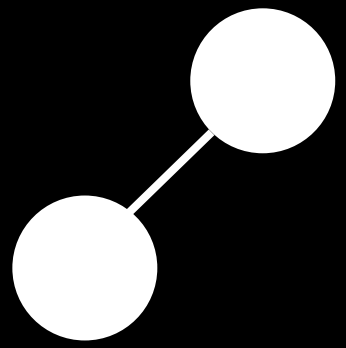
Graphs

Connected?



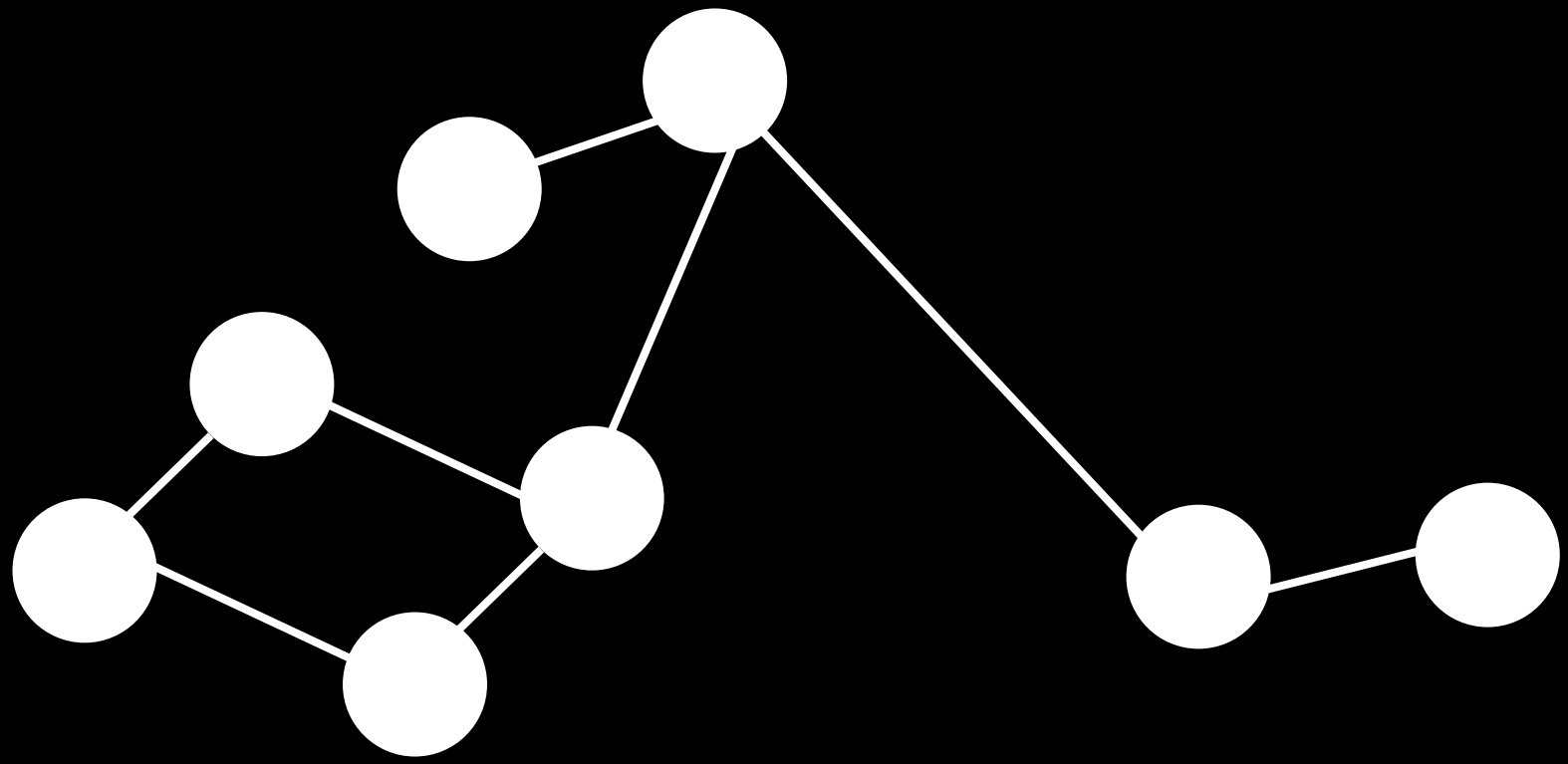
Graphs

Connected?



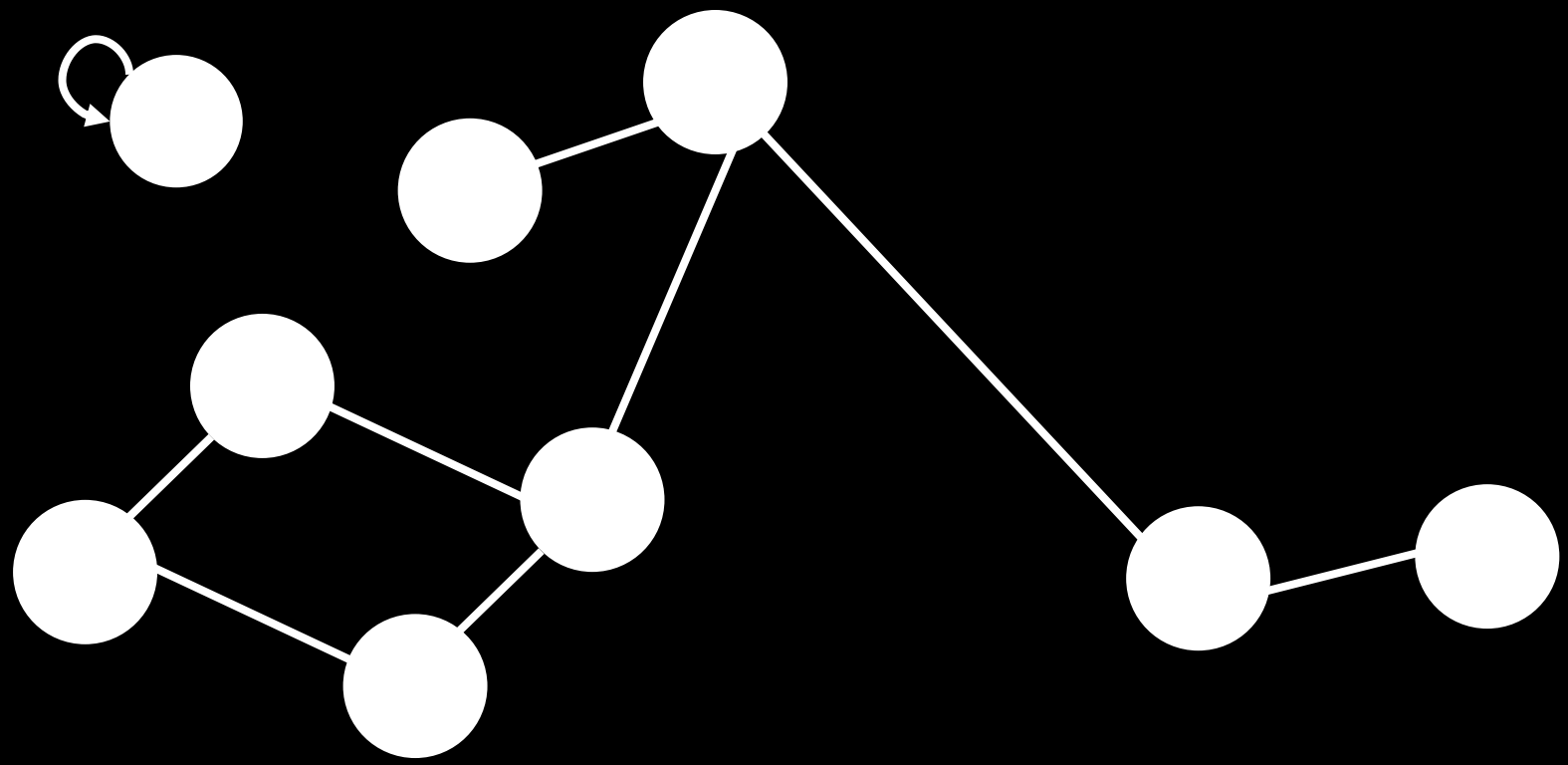
Graphs

Connected?

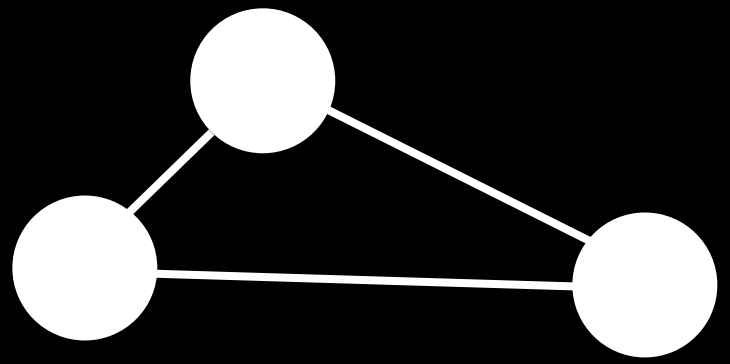


Graphs

Connected?

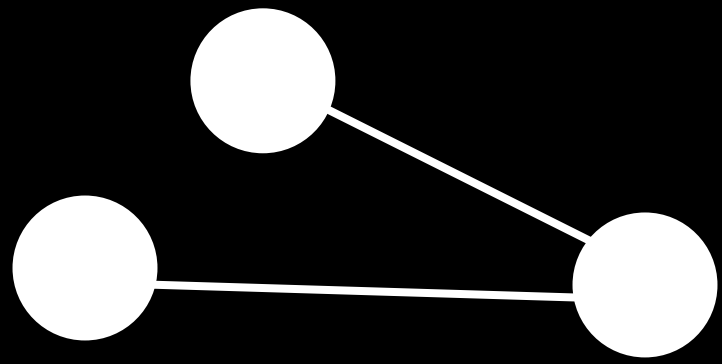


Graphs



Cyclic

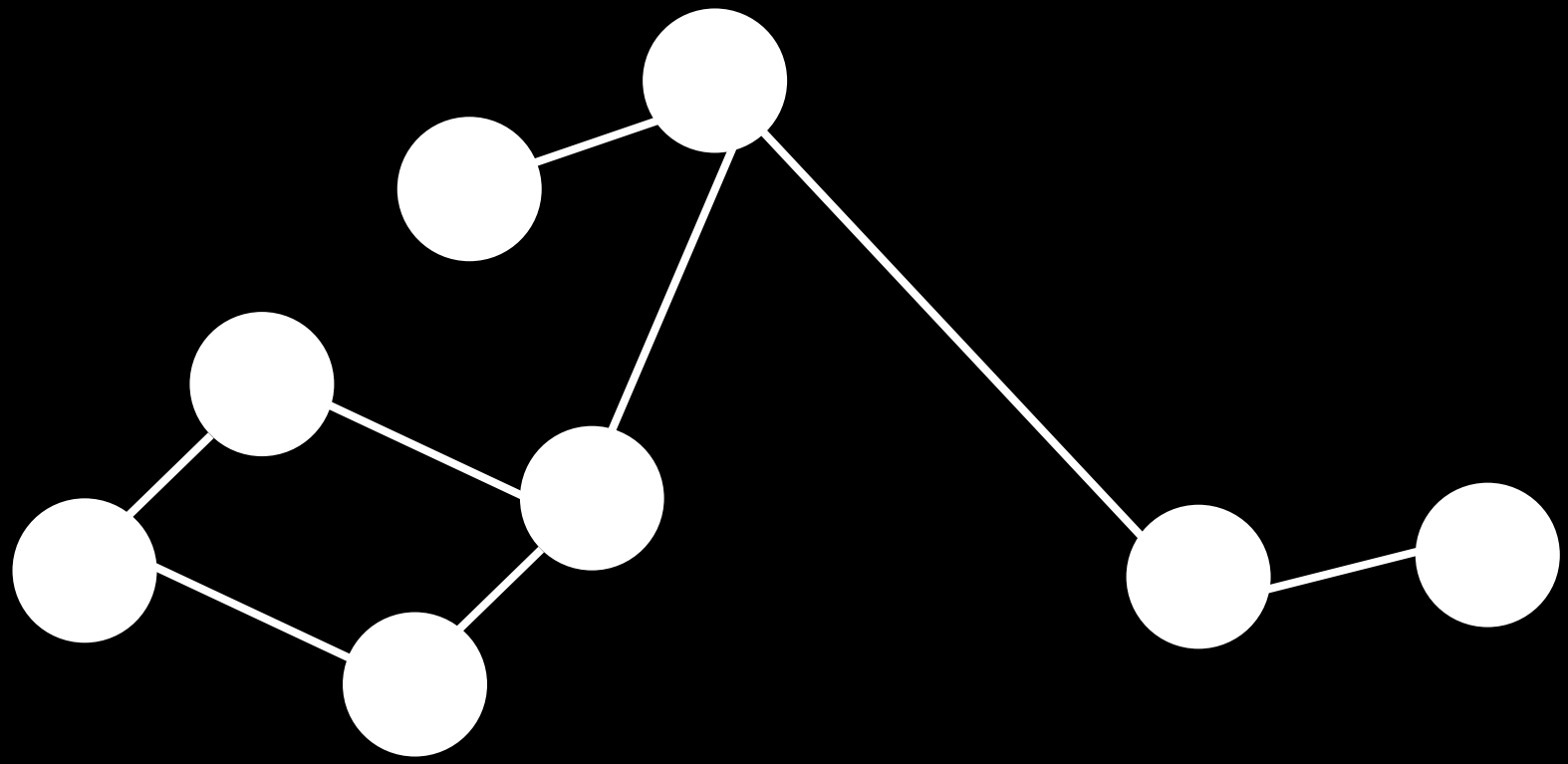
There exists a starting node for which there is a path to itself



Acyclic

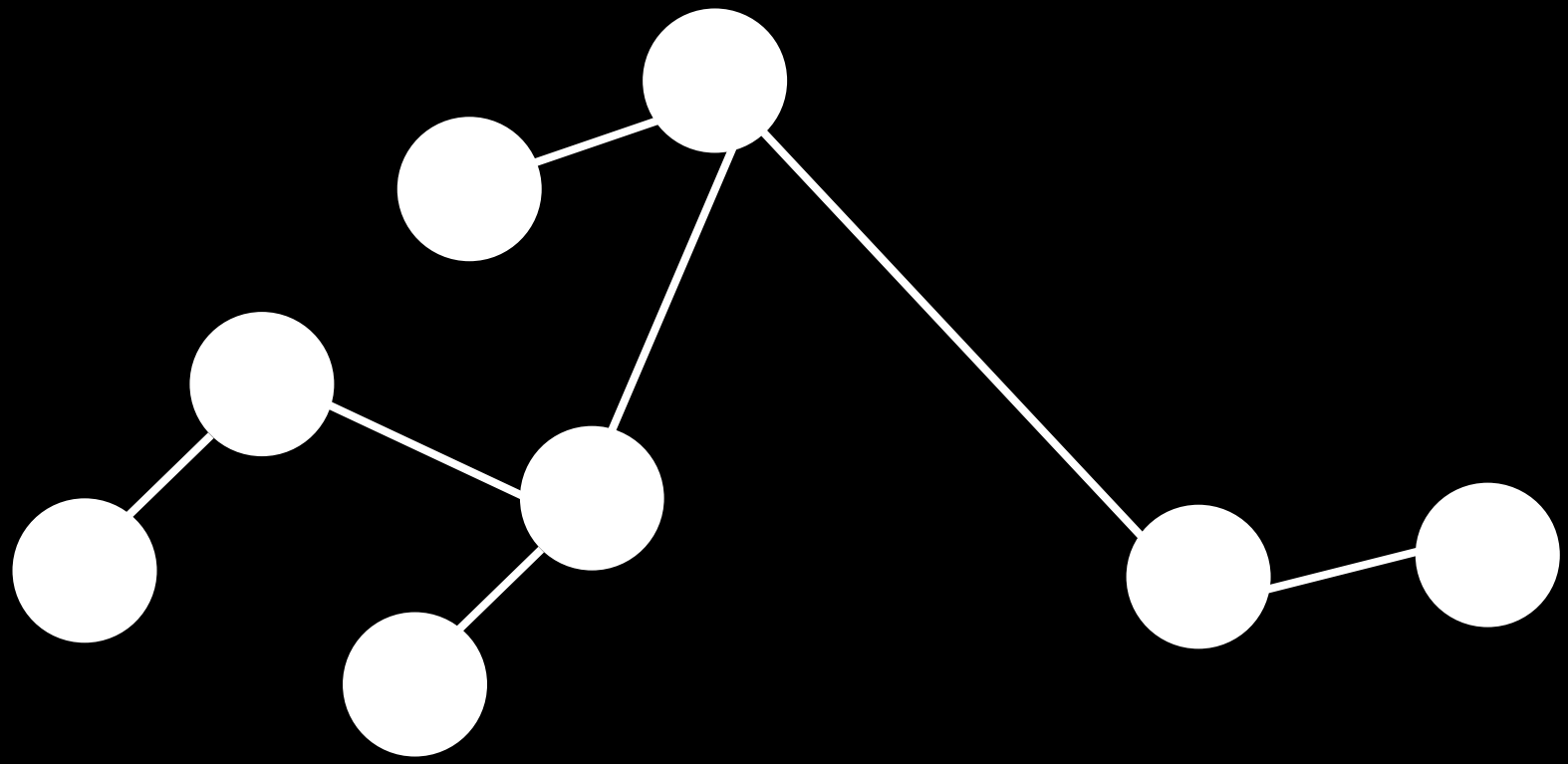
Graphs

Cyclic?



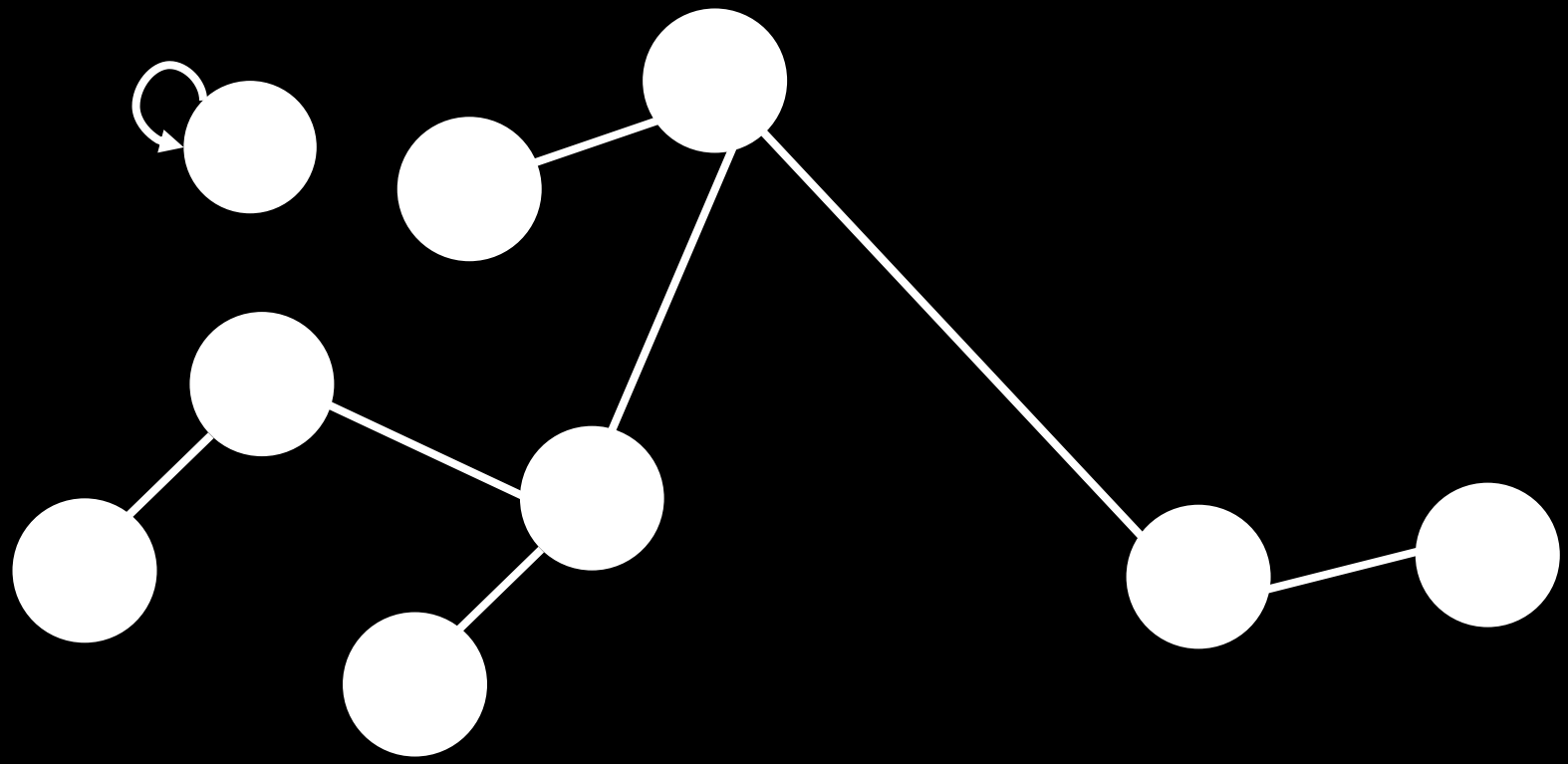
Graphs

Cyclic?

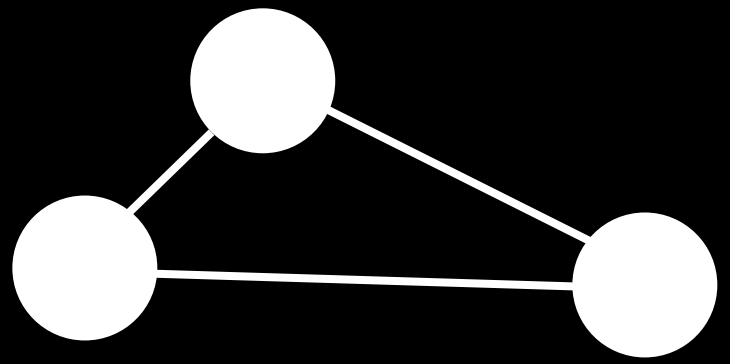


Graphs

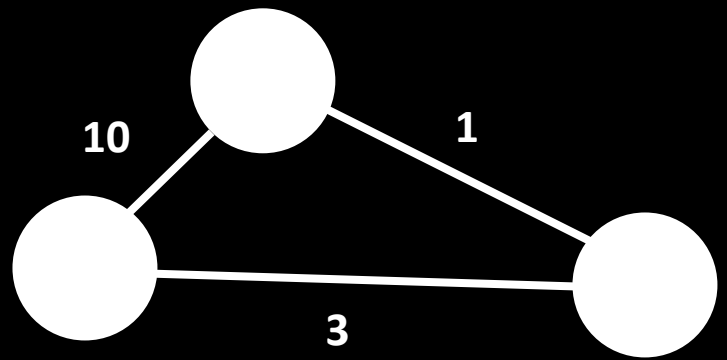
Cyclic?



Graphs

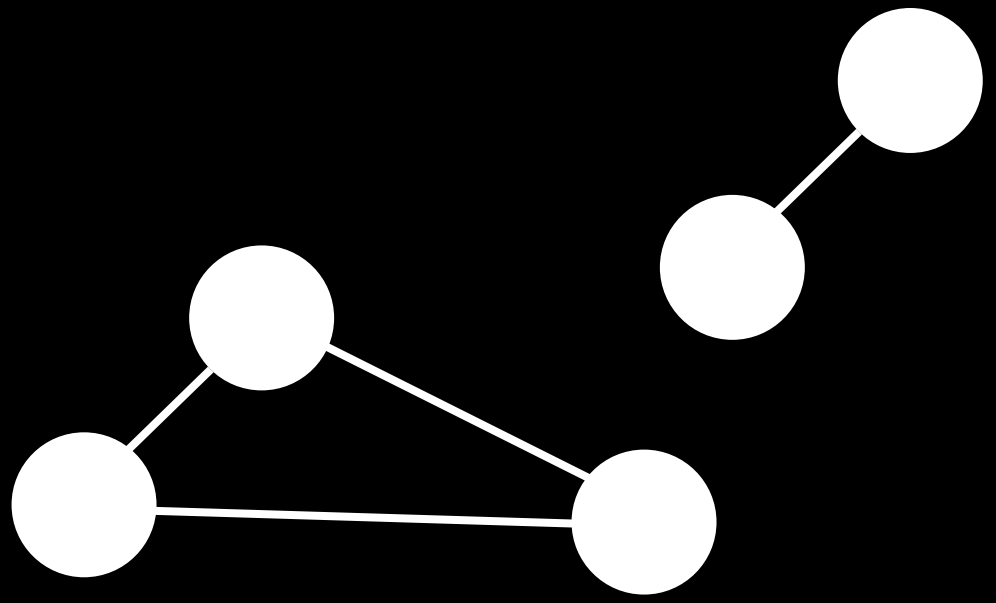


Unweighted



Weighted

Today



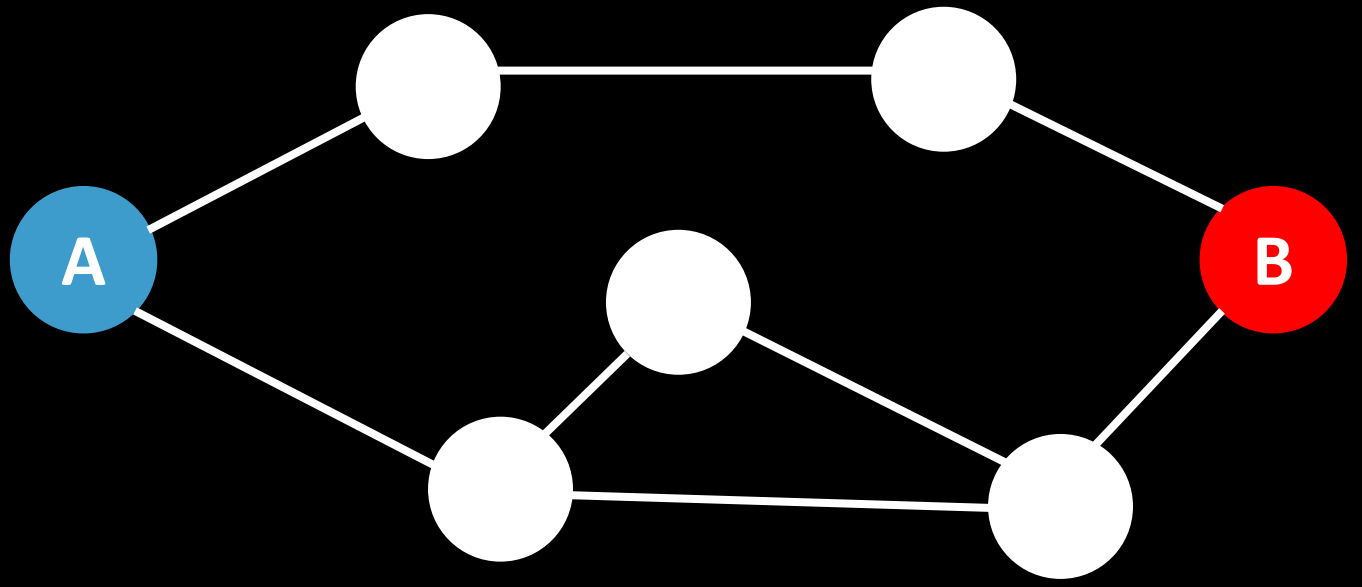
Undirected + Unweighted

Could be Cyclic

Could be Disconnected

Graphs

- Consider an arbitrary undirected, unweighted graph
- Pick 2 arbitrary vertices on the graph: **A** and **B**



Graphs

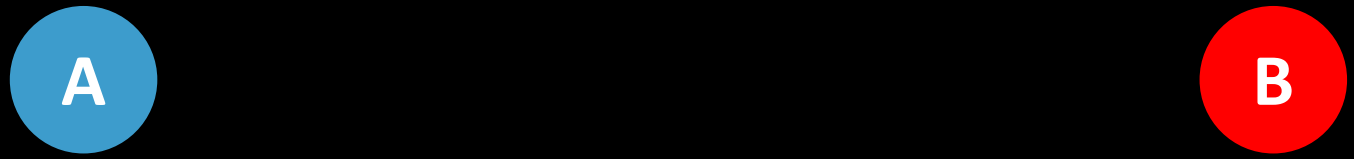
- Q: For an arbitrary graph, are we guaranteed that there is a **path** between nodes **A** and **B**?

Graphs

- Q: For an arbitrary graph, are we guaranteed that there is a **path** between nodes **A** and **B**?
- A: No, we are not

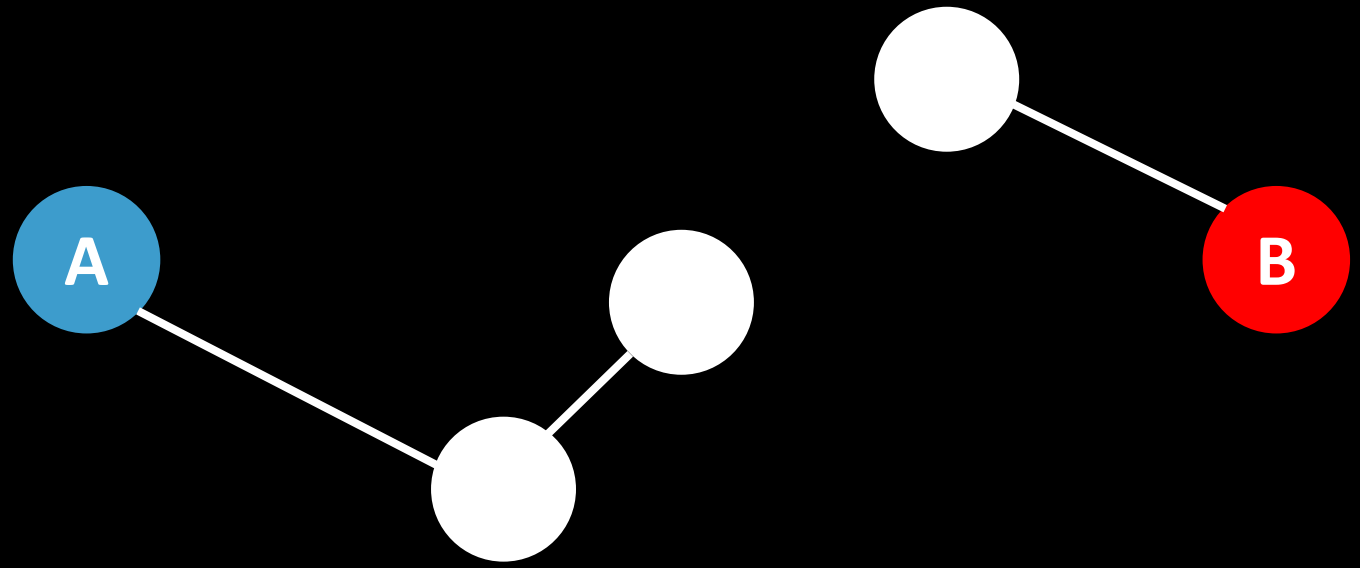
Graphs

- Q: For an arbitrary graph, are we guaranteed that there is a **path** between nodes **A** and **B**?
- A: No, we are not



Graphs

- Q: For an arbitrary graph, are we guaranteed that there is a **path** between nodes **A** and **B**?
- A: No, we are not



Graphs

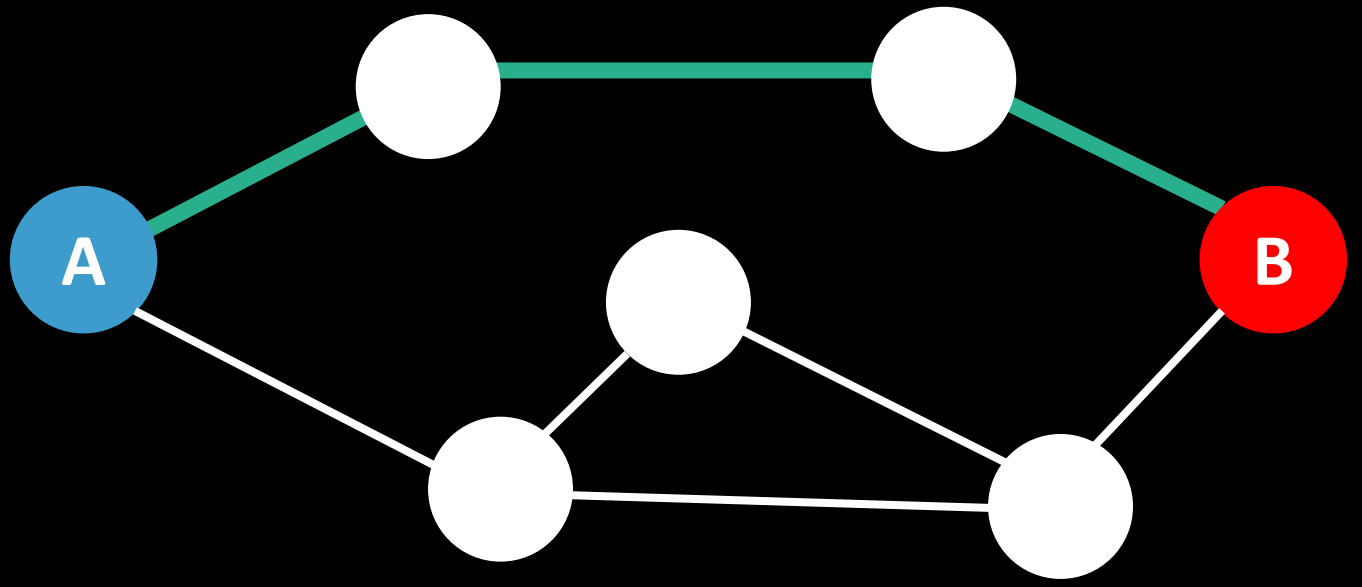
- Q: If a path exists between **A** and **B**, is it unique?

Graphs

- Q: If a path exists between **A** and **B**, is it unique?
- A: No.

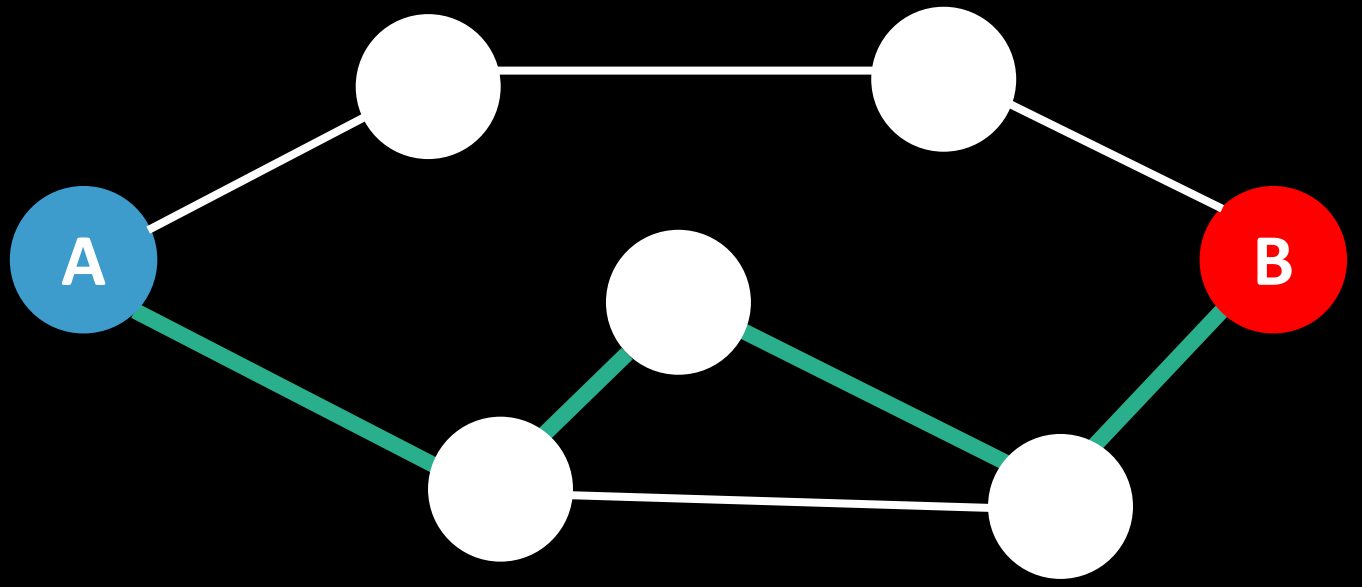
Graphs

- Q: If a path exists between **A** and **B**, is it unique?
- A: No.



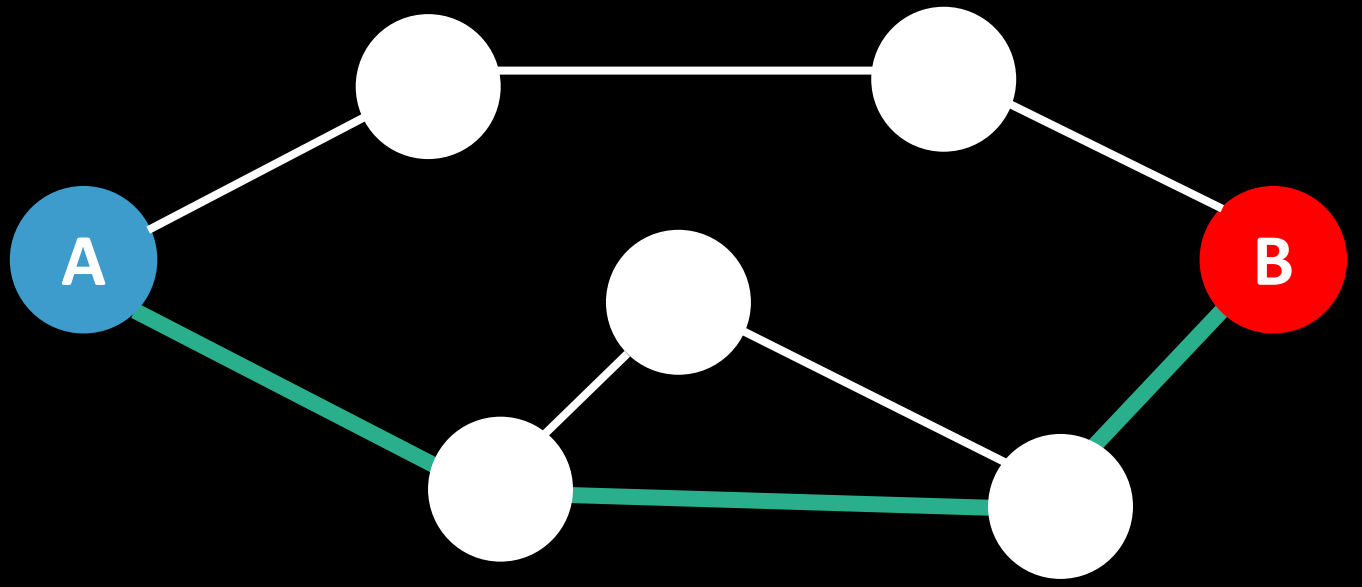
Graphs

- Q: If a path exists between **A** and **B**, is it unique?
- A: No.



Graphs

- Q: If a path exists between **A** and **B**, is it unique?
- A: No.



Graphs

Interested in finding…

- Whether a path exists from A → B
- A path from A → B with **minimal path length**

Definitions

For an undirected, unweighted graph:

- Path length: for a path from node A to node B, the **path length** is the number of edges the path is comprised of.
  - If the path is denoted by $\mathrm{P} = \{e_1, e_2, \ldots, e_n\}$, its length is $|P| = n$
- Level: The minimum number of edges between a node and the *starting node*
  - i.e. the length of the shortest path between a node and the starting node

Definitions

For an undirected, unweighted graph:

- Path length: for a path from node A to node B, the **path length** is the number of edges the path is comprised of.
  - If the path is denoted by $\mathrm{P} = \{e_1, e_2, \dots, e_n\}$, its length is $|P| = n$
- Level: The minimum number of edges between a node and the *starting node*
  - i.e. the length of the shortest path between a node and the starting node
  - Q: What is the starting node?

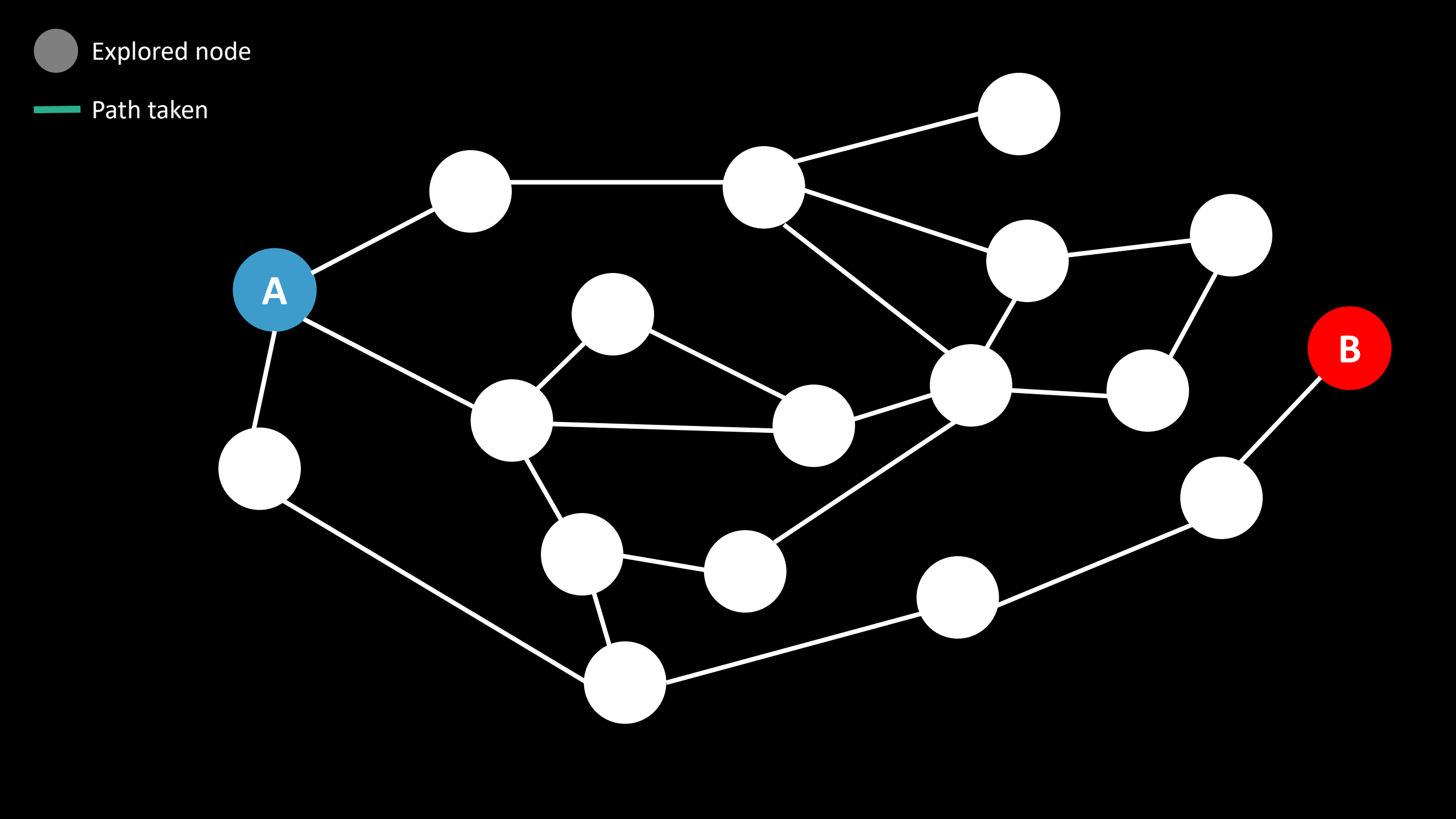
Definitions

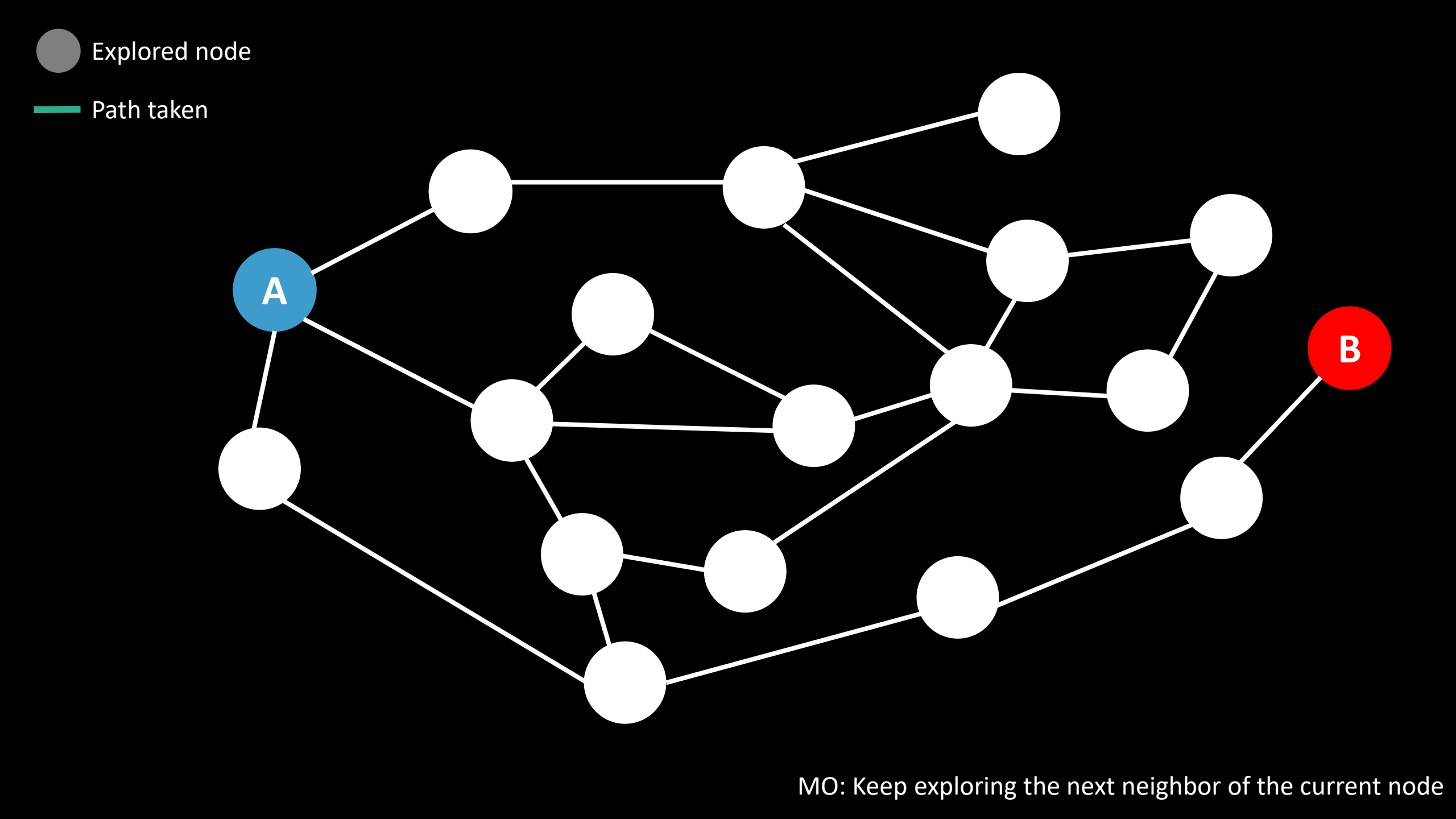
For an undirected, unweighted graph:

- Path length: for a path from node A to node B, the **path length** is the number of edges the path is comprised of.
  - If the path is denoted by $\mathrm{P} = \{e_1, e_2, \dots, e_n\}$, its length is $|P| = n$
- Level: The minimum number of edges between a node and the *starting node*
  - i.e. the length of the shortest path between a node and the starting node
  - Q: What is the starting node?
  - A: You choose, based on the context

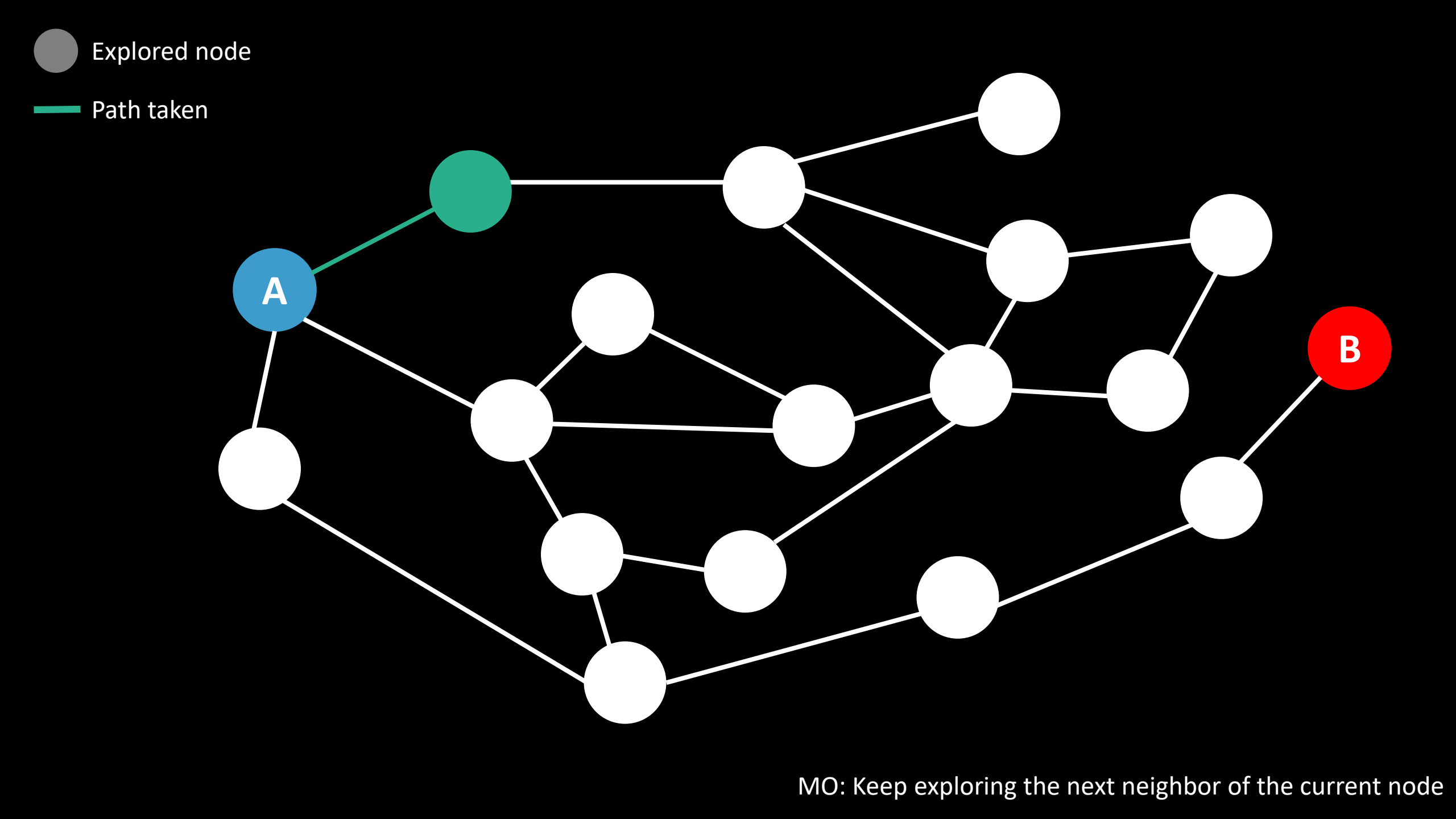
Path-finding

- Does a path exist from A→B?
- Keep track of:
  - Path taken
  - Nodes explored

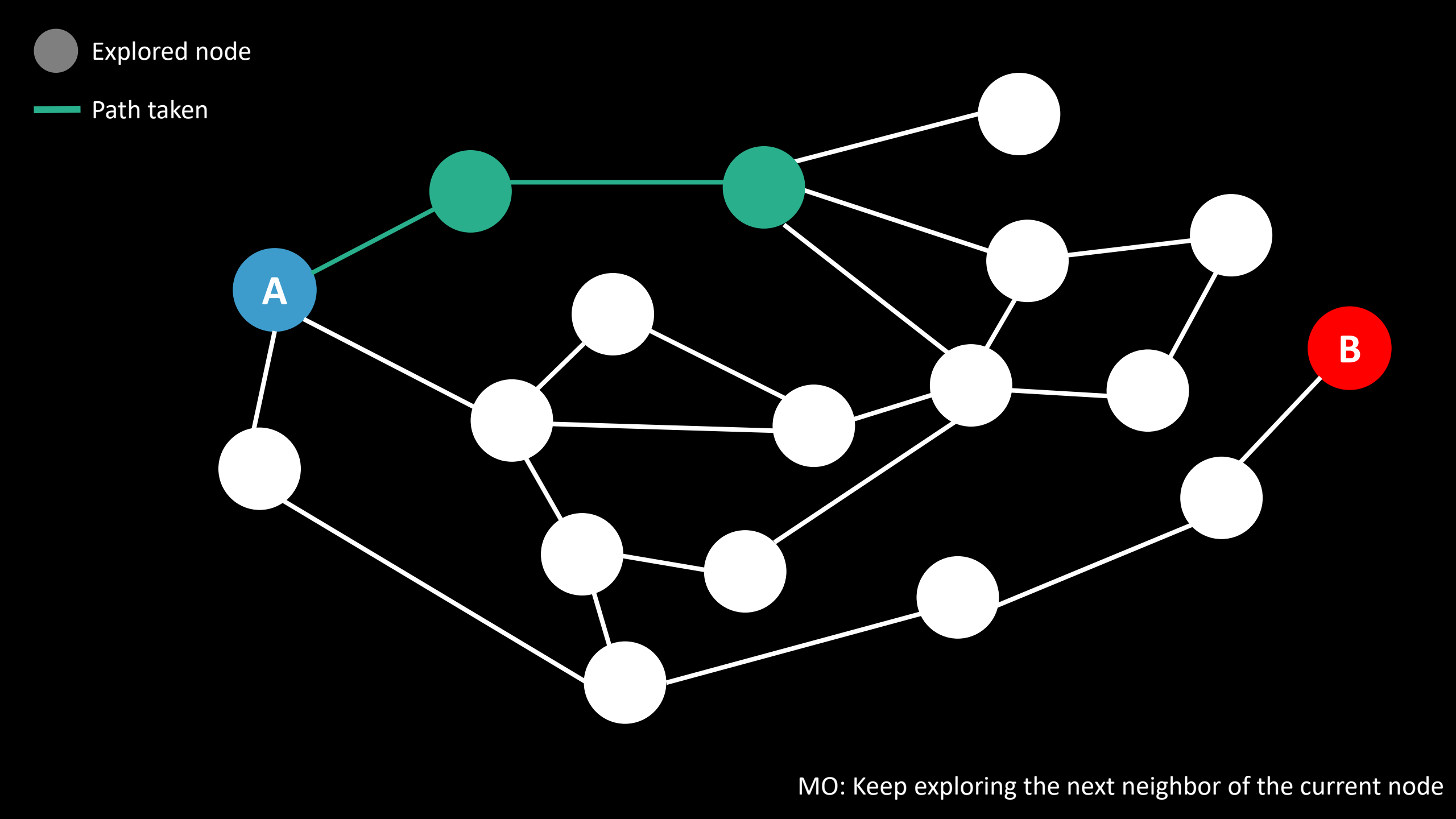

Explored node
Path taken
A
B



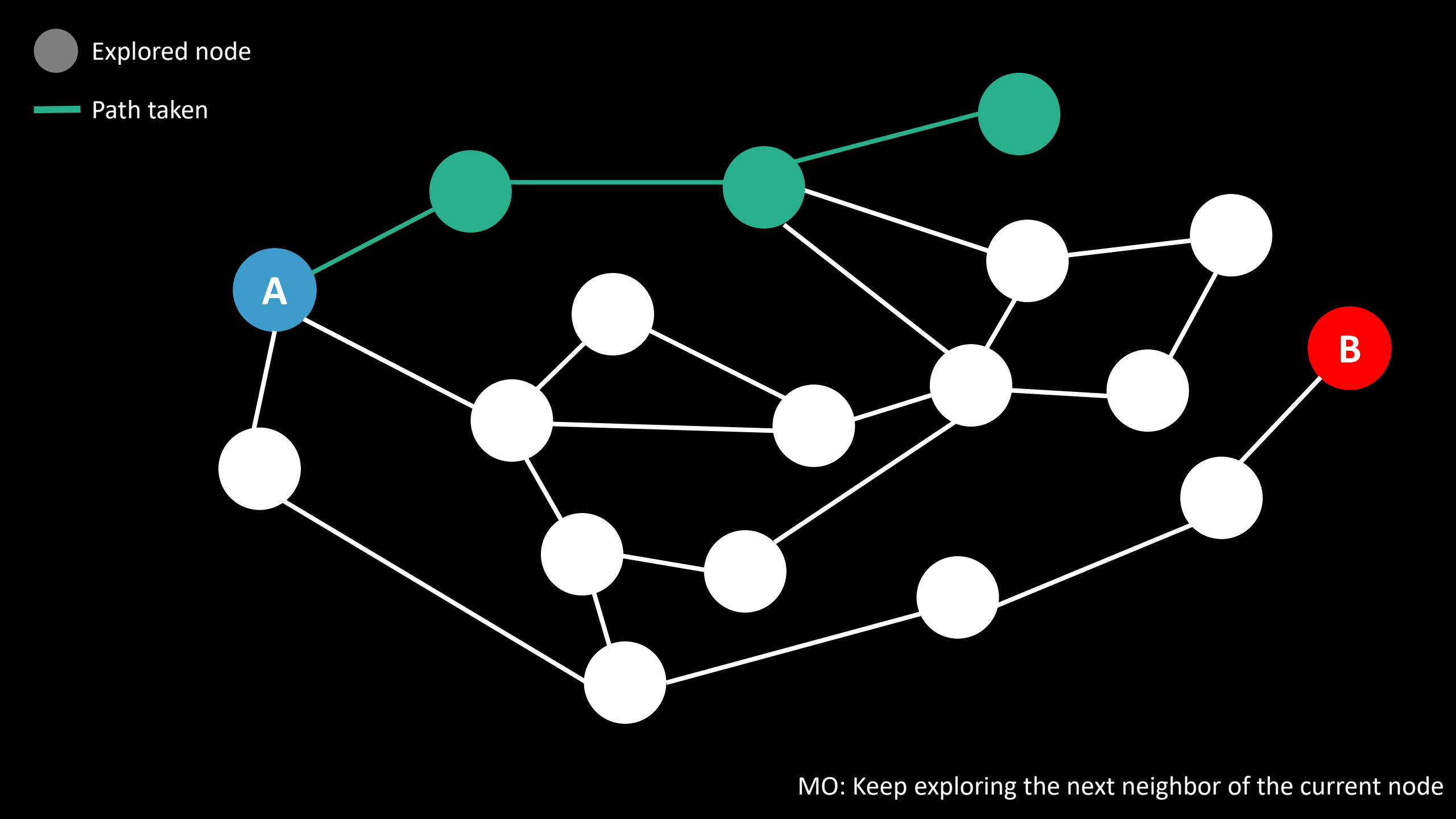
MO: Keep exploring the next neighbor of the current node



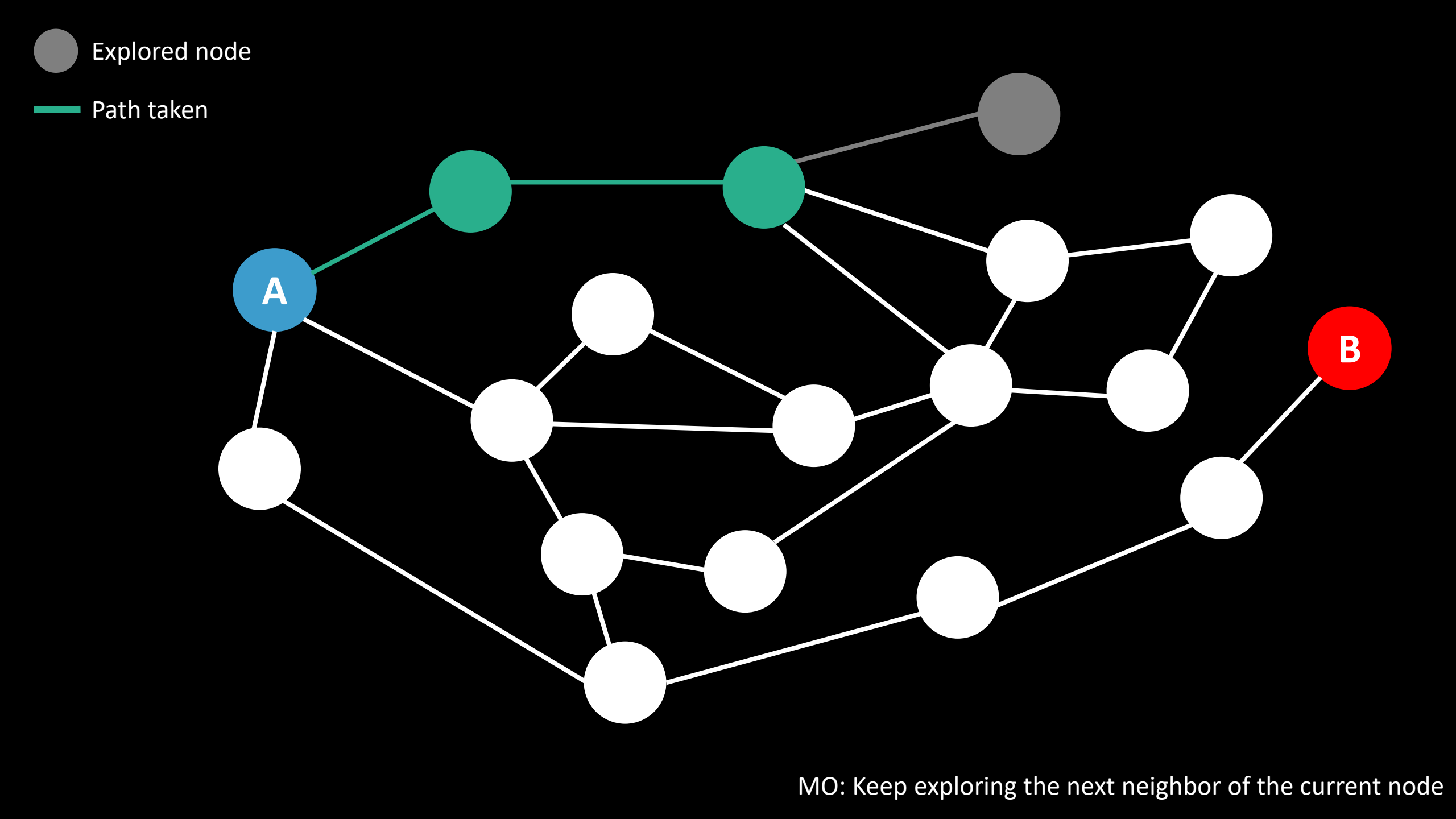
MO: Keep exploring the next neighbor of the current node



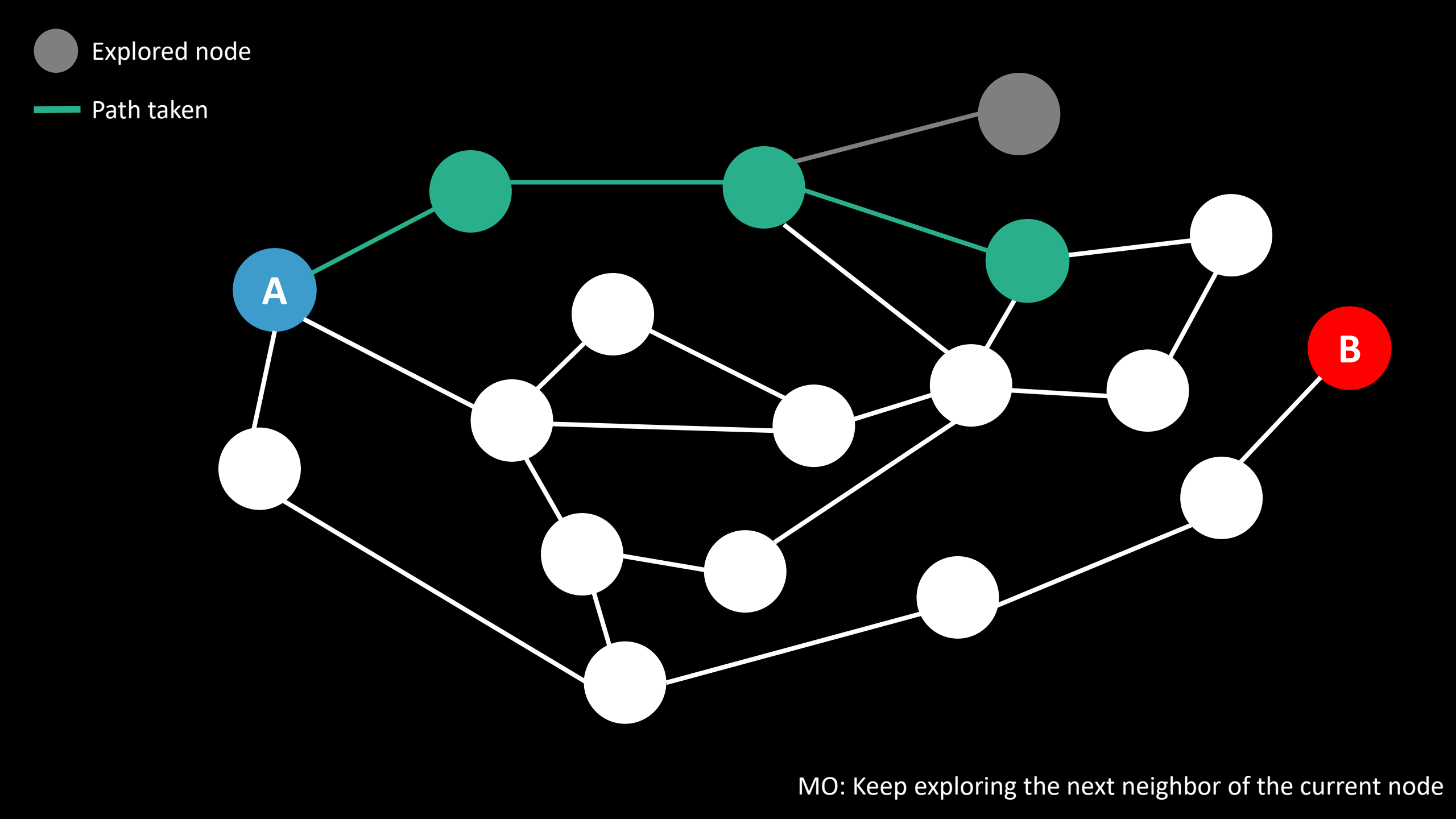
MO: Keep exploring the next neighbor of the current node



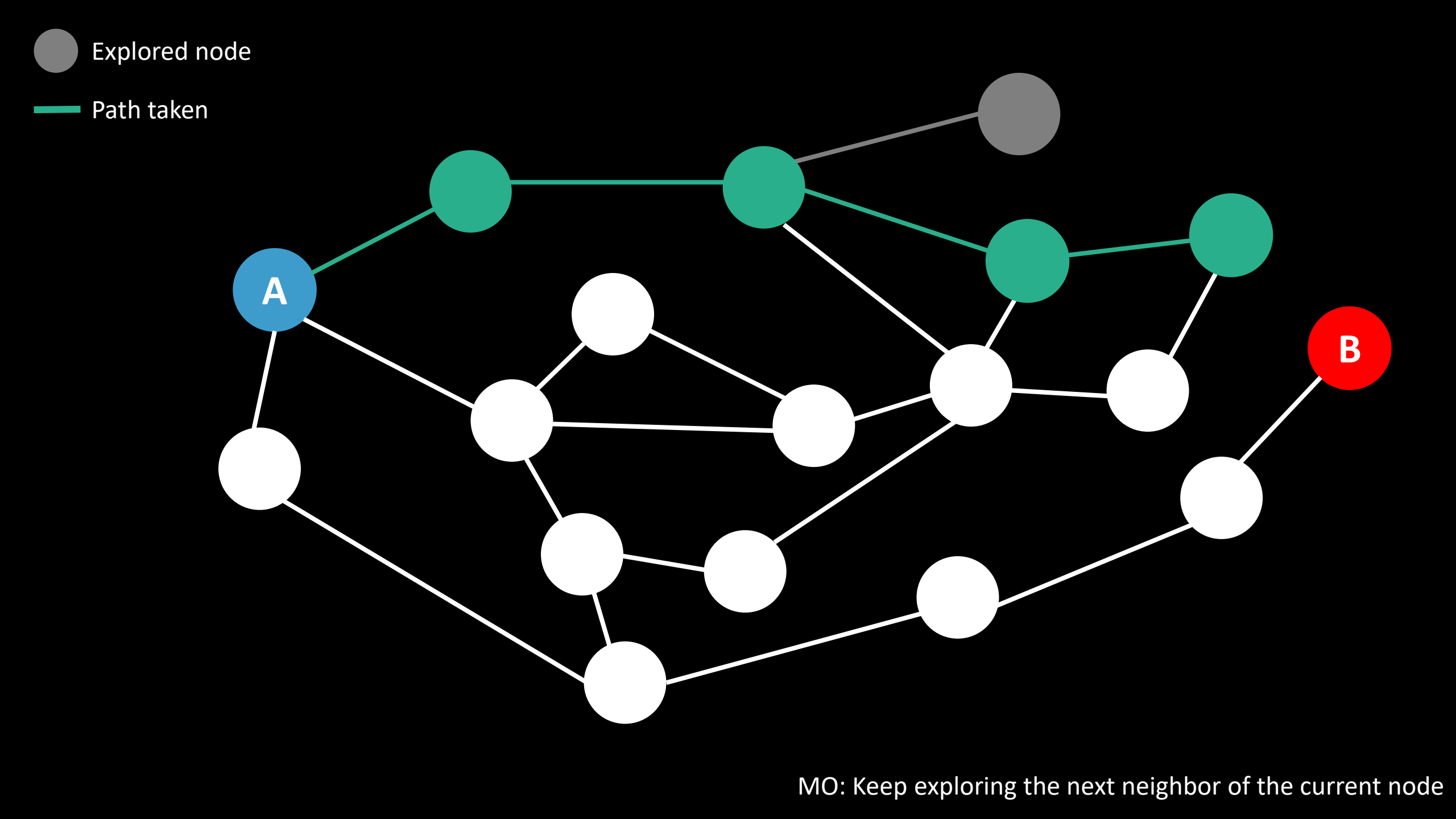
MO: Keep exploring the next neighbor of the current node



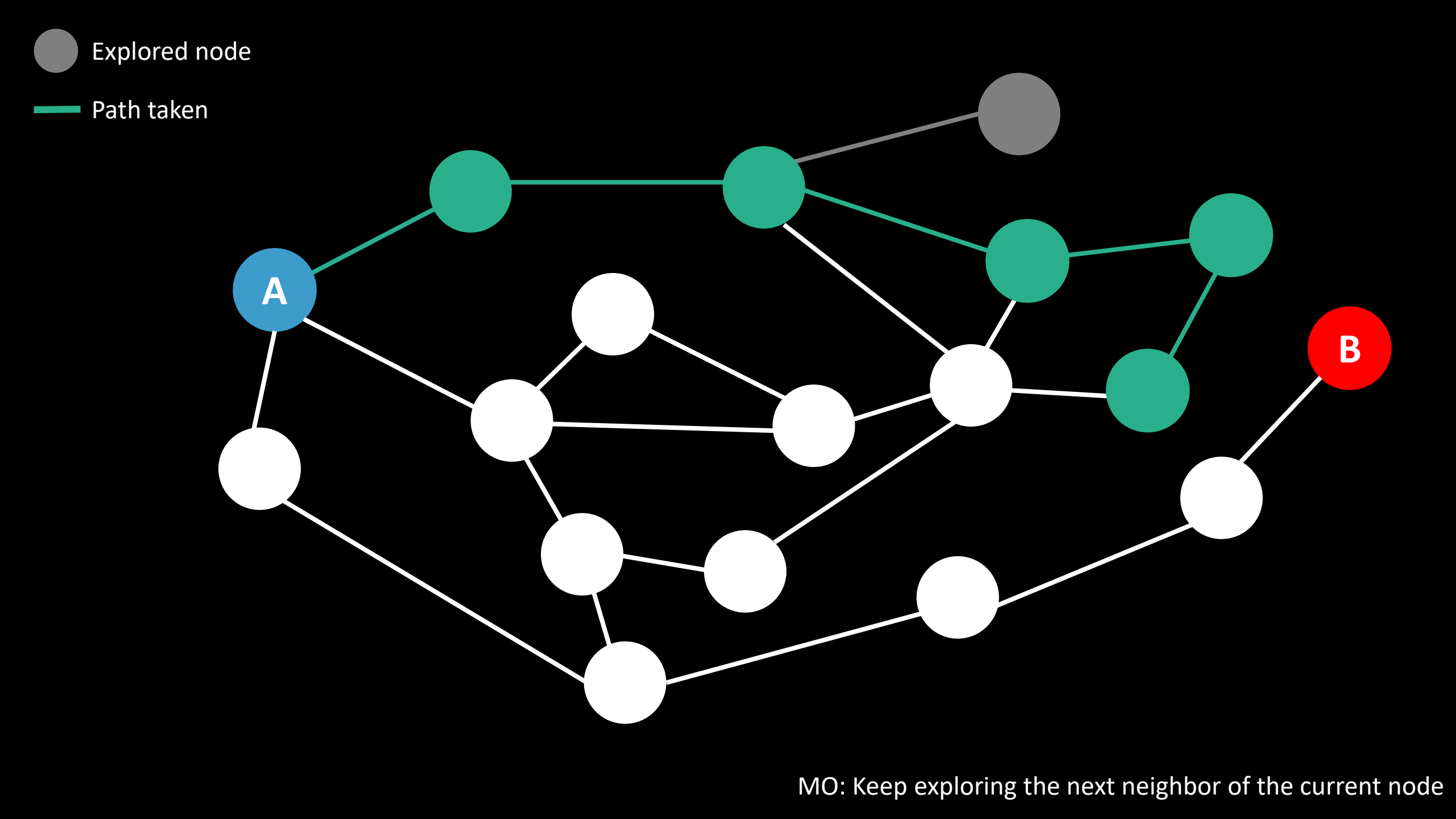
MO: Keep exploring the next neighbor of the current node



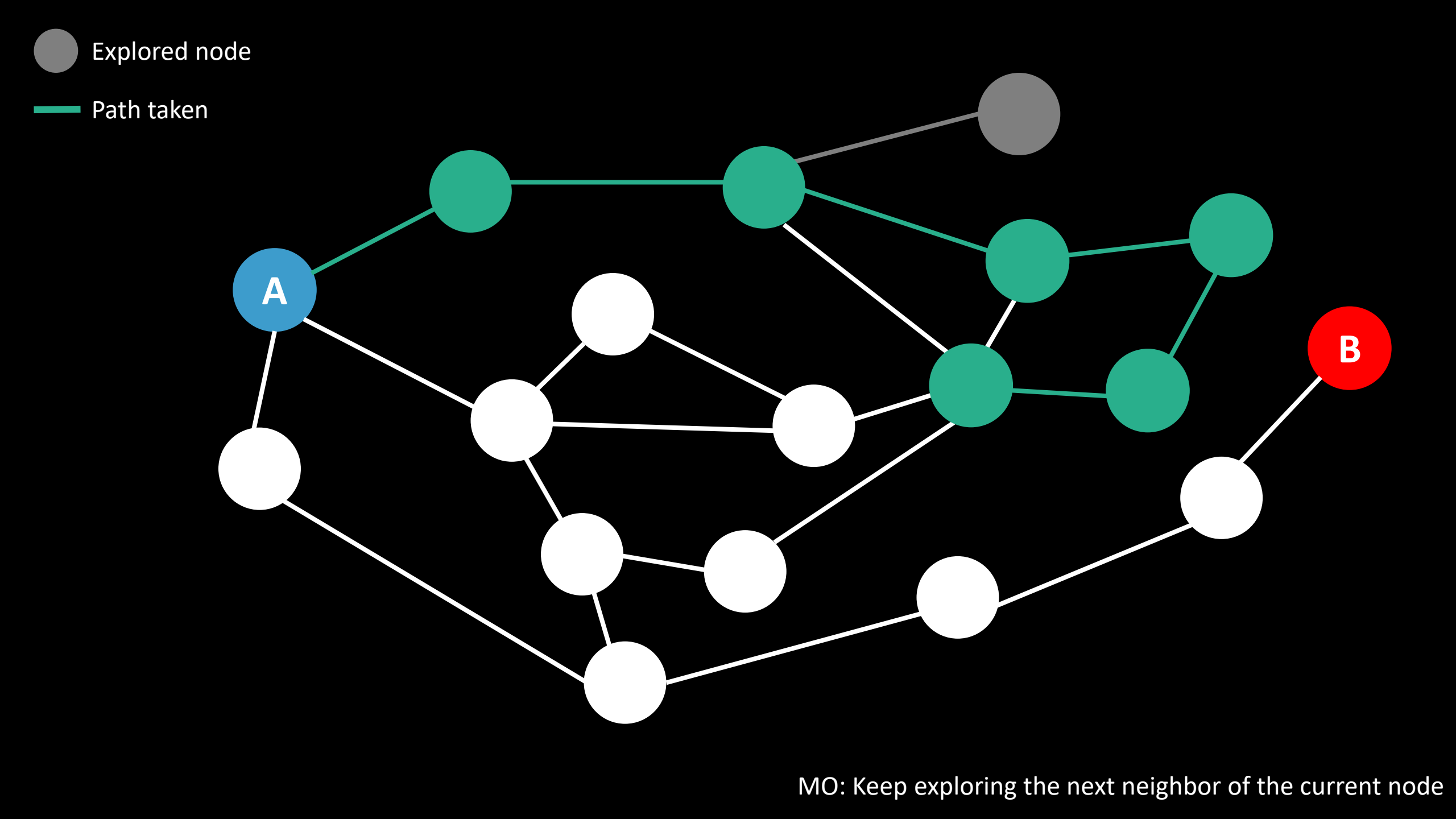
MO: Keep exploring the next neighbor of the current node



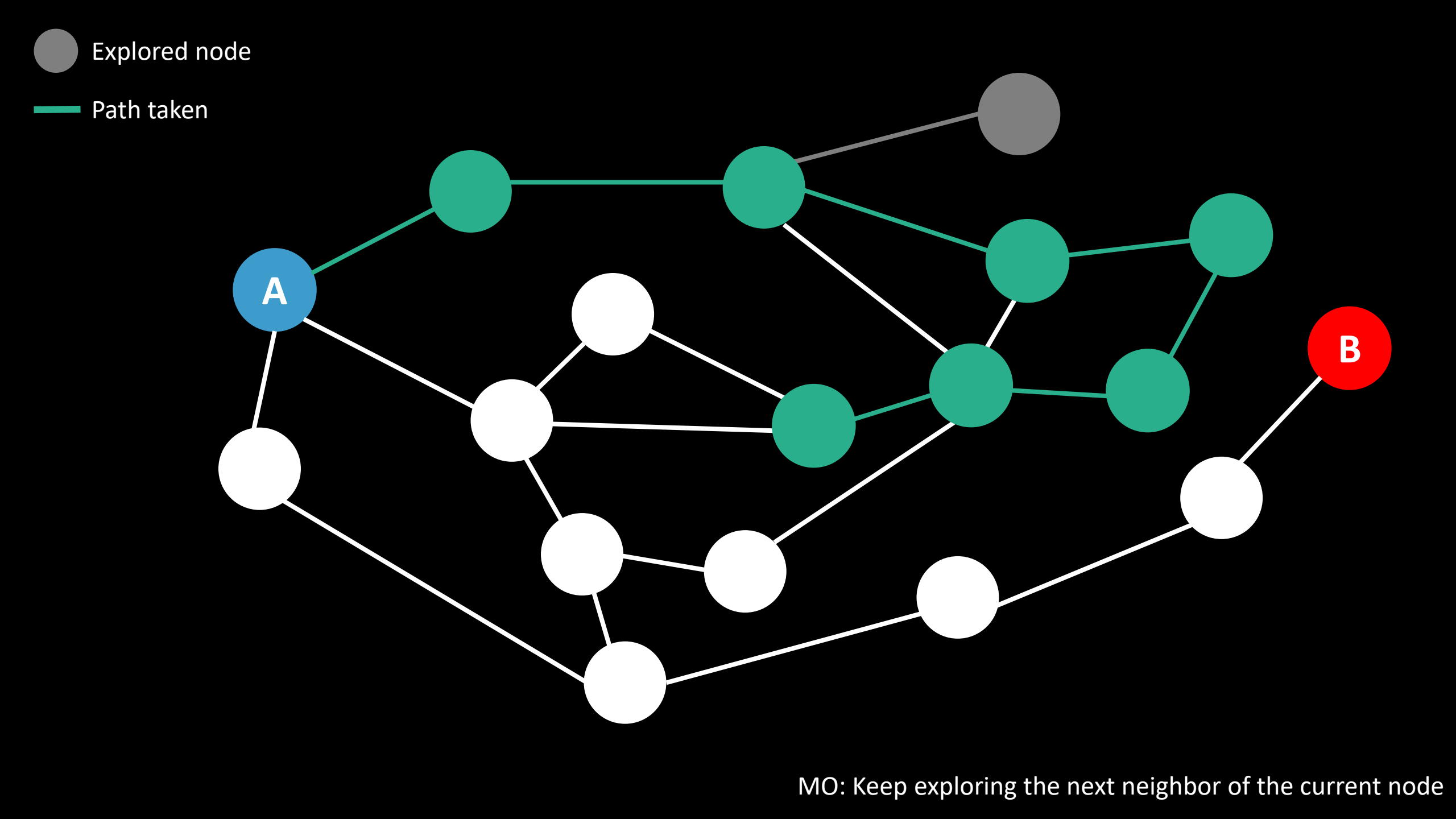
MO: Keep exploring the next neighbor of the current node



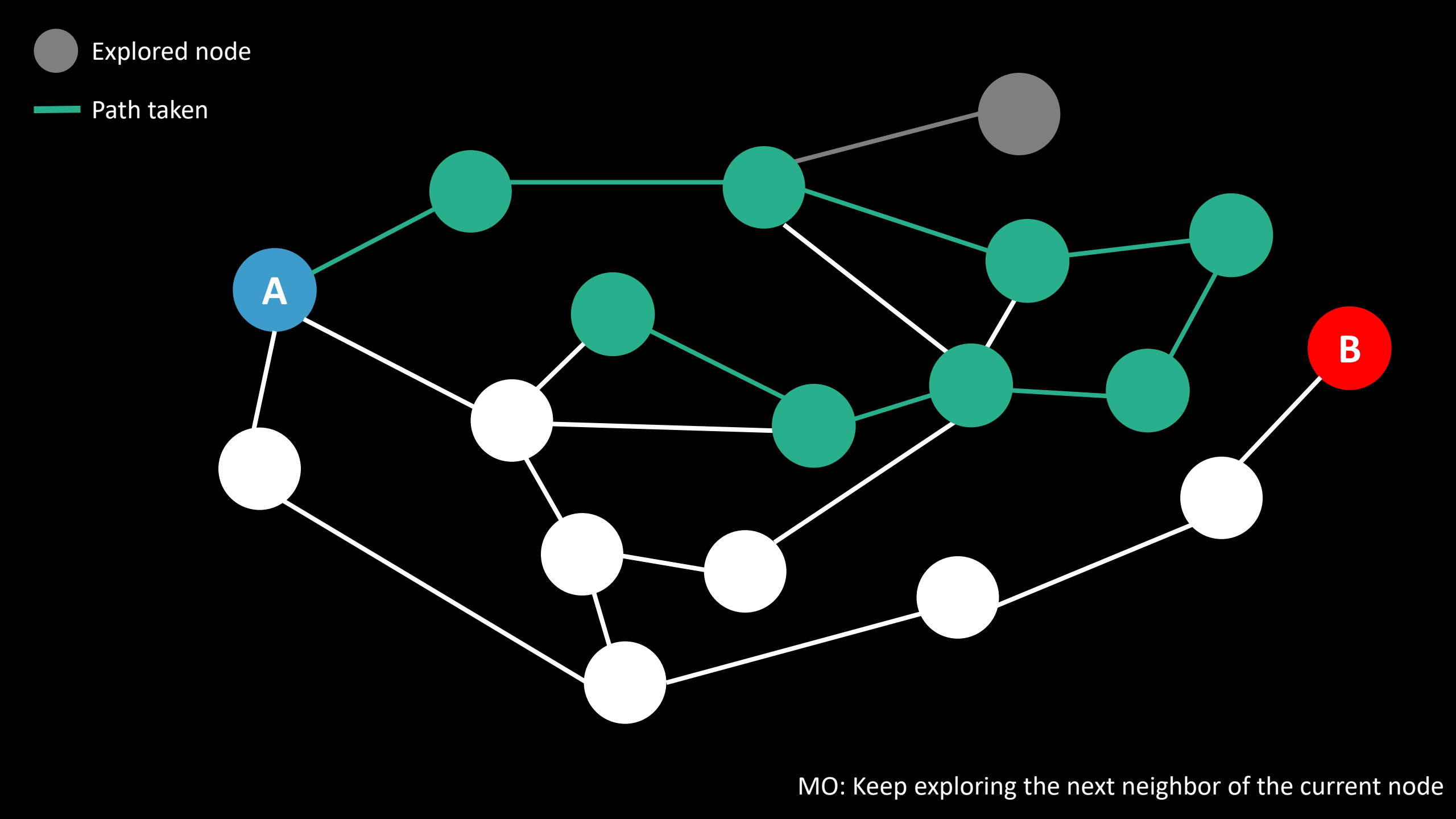
MO: Keep exploring the next neighbor of the current node



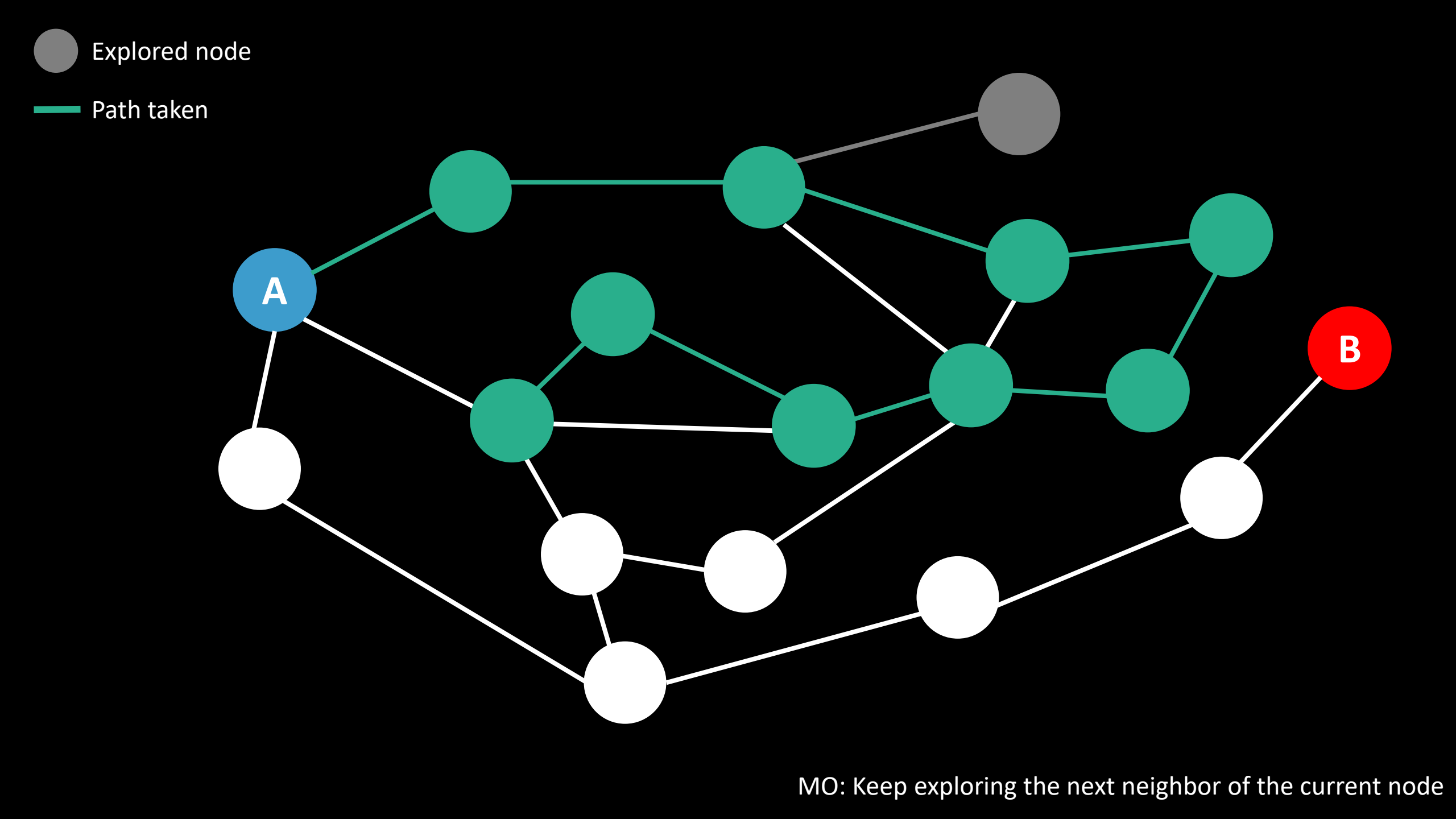
MO: Keep exploring the next neighbor of the current node



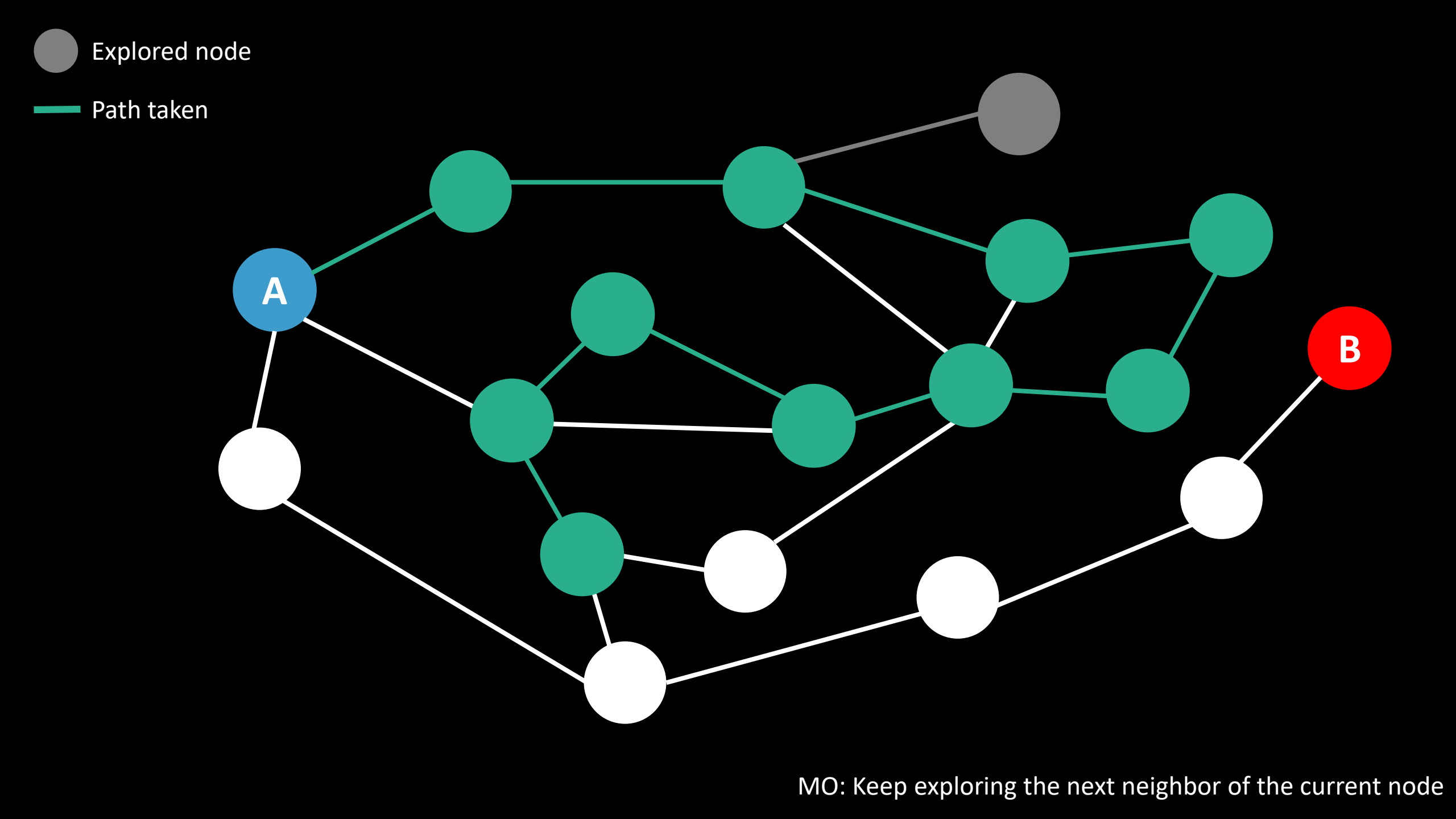
MO: Keep exploring the next neighbor of the current node



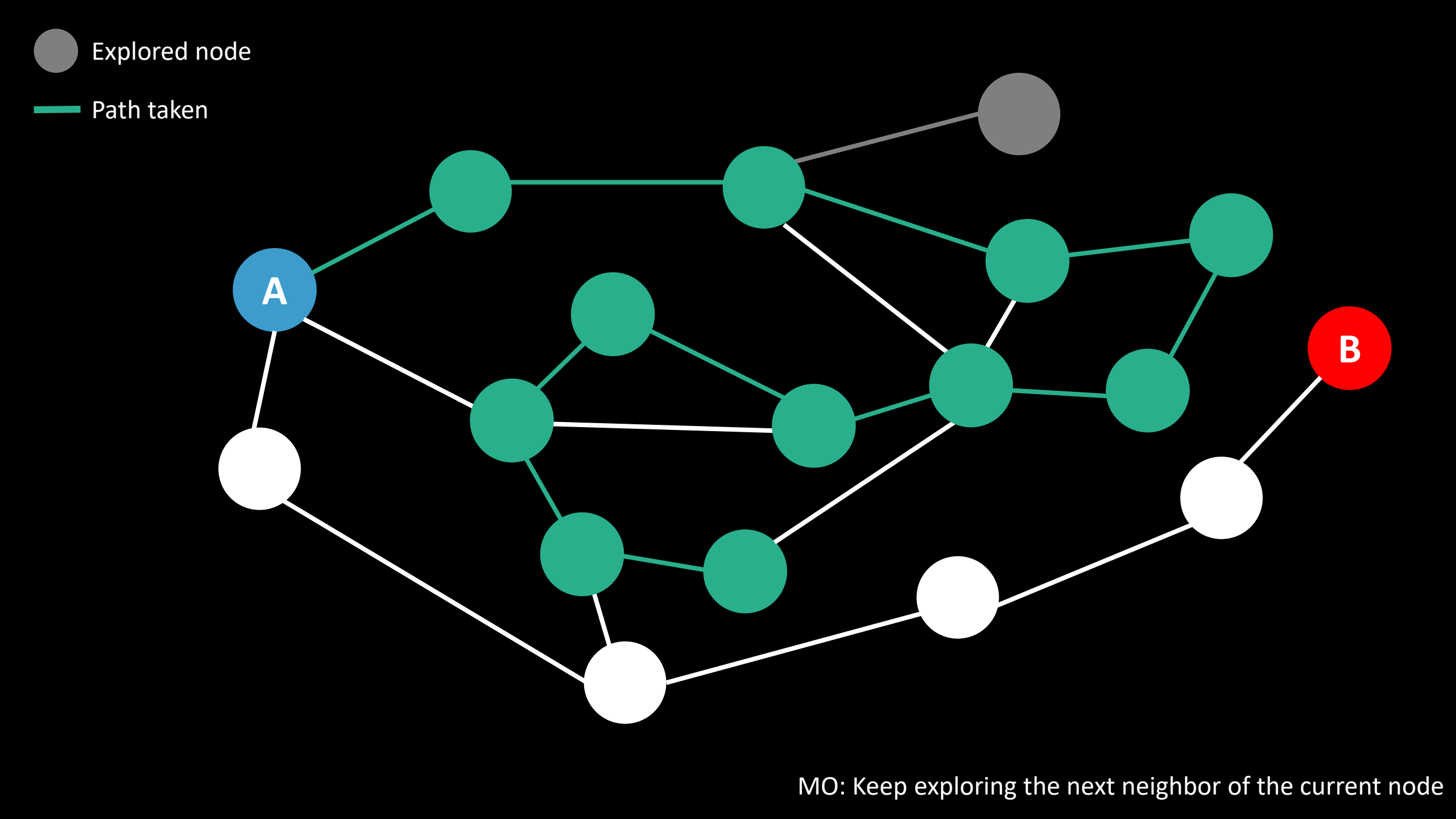
MO: Keep exploring the next neighbor of the current node



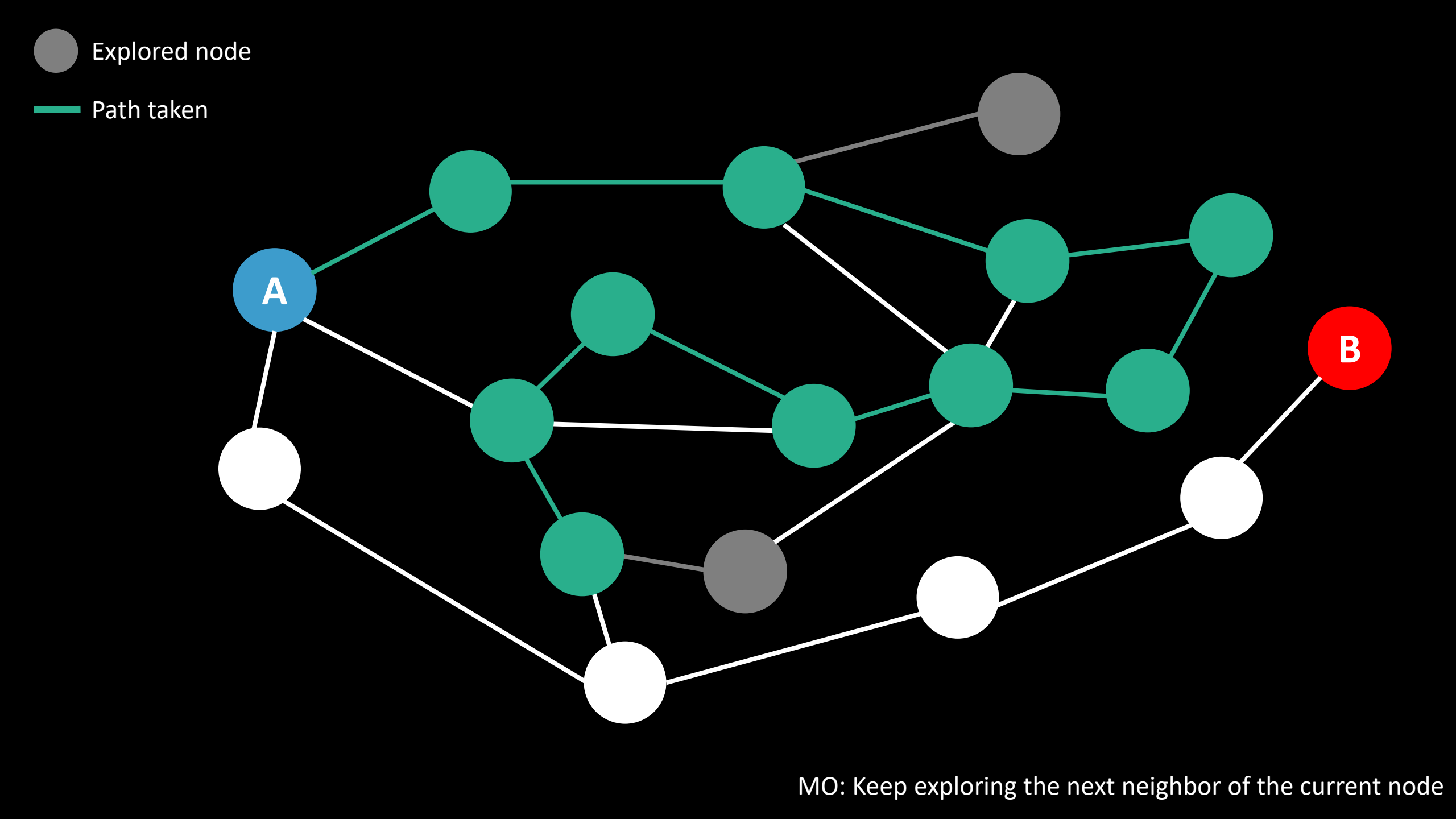
MO: Keep exploring the next neighbor of the current node



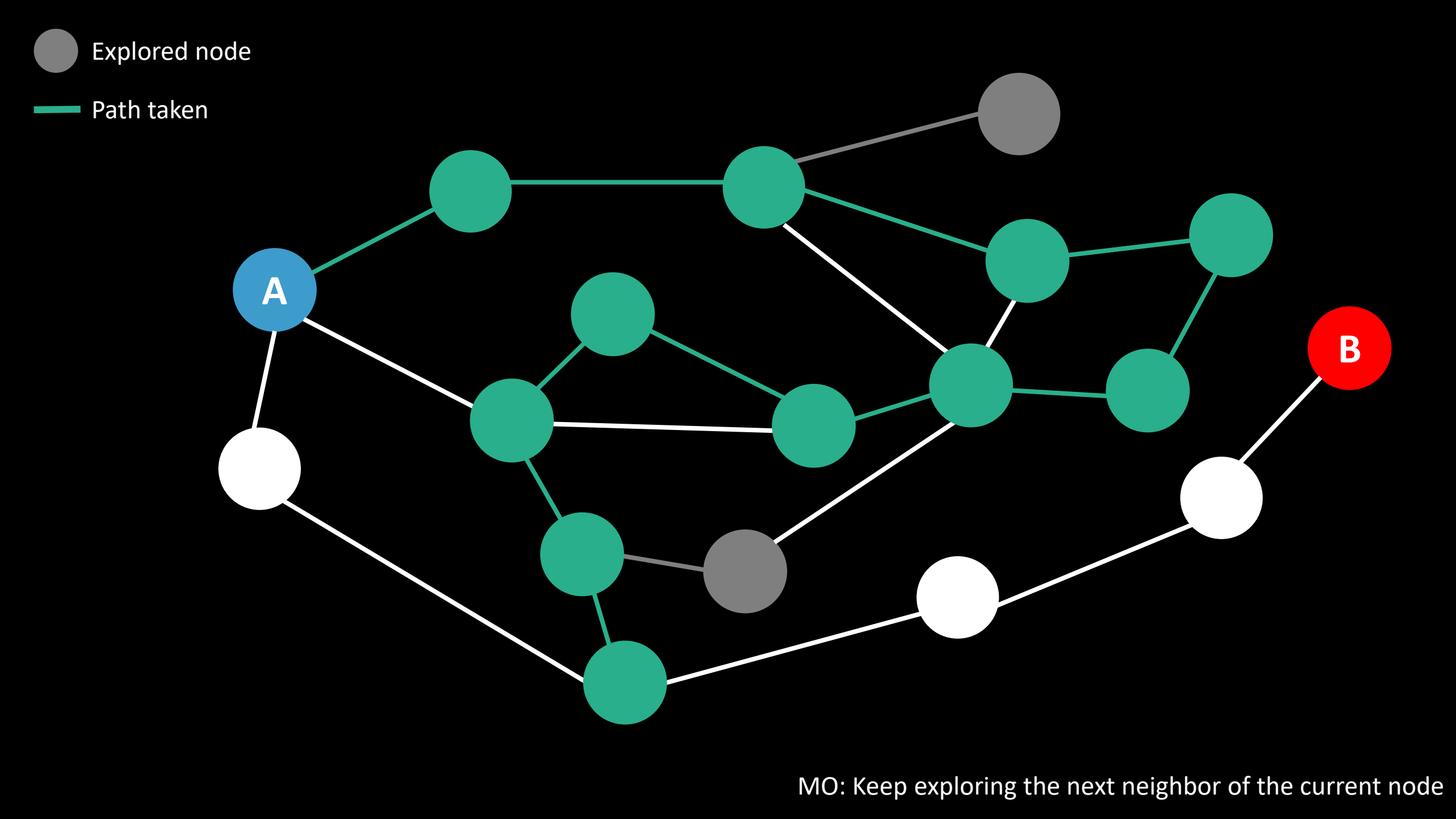
MO: Keep exploring the next neighbor of the current node



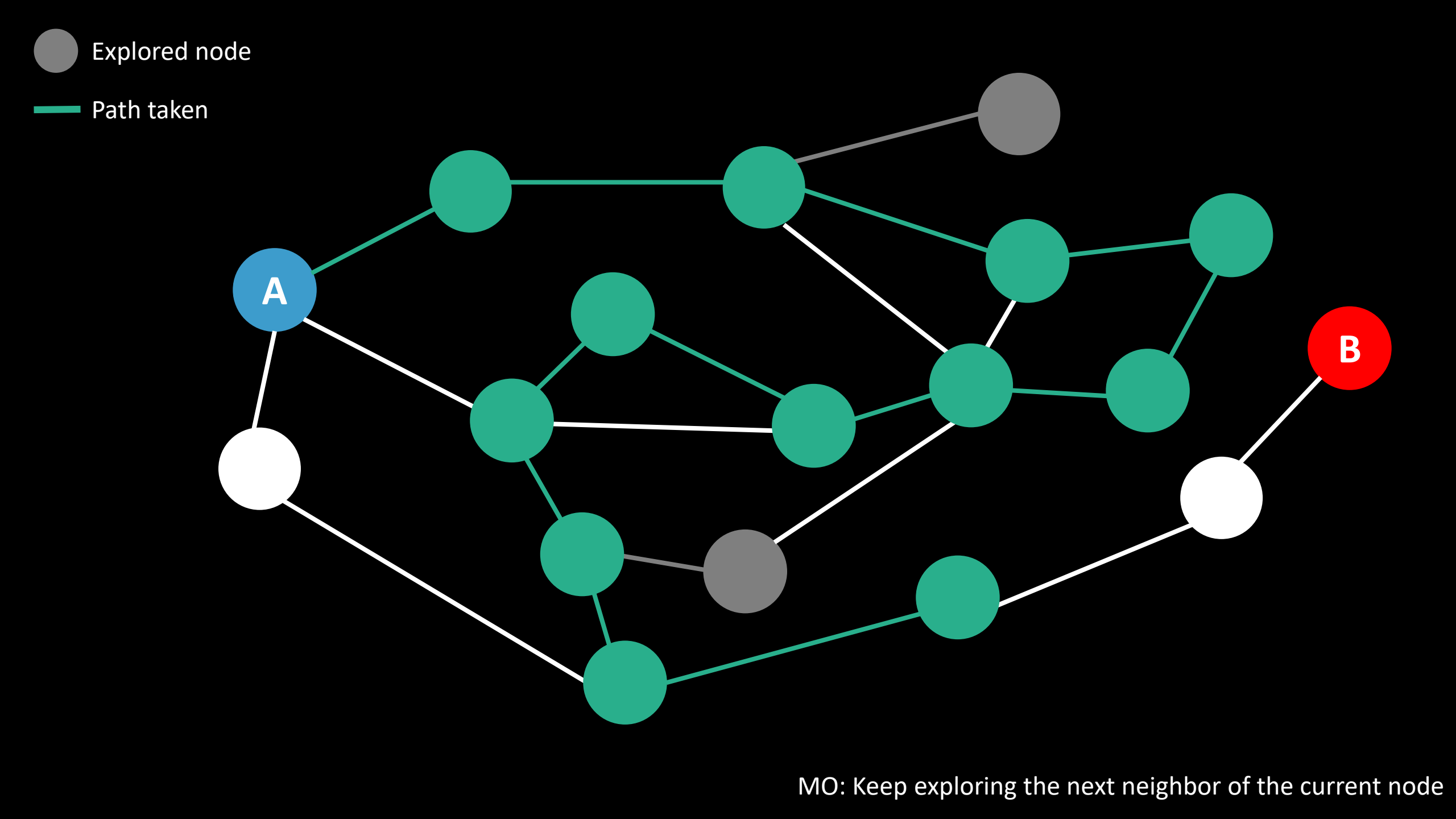
MO: Keep exploring the next neighbor of the current node



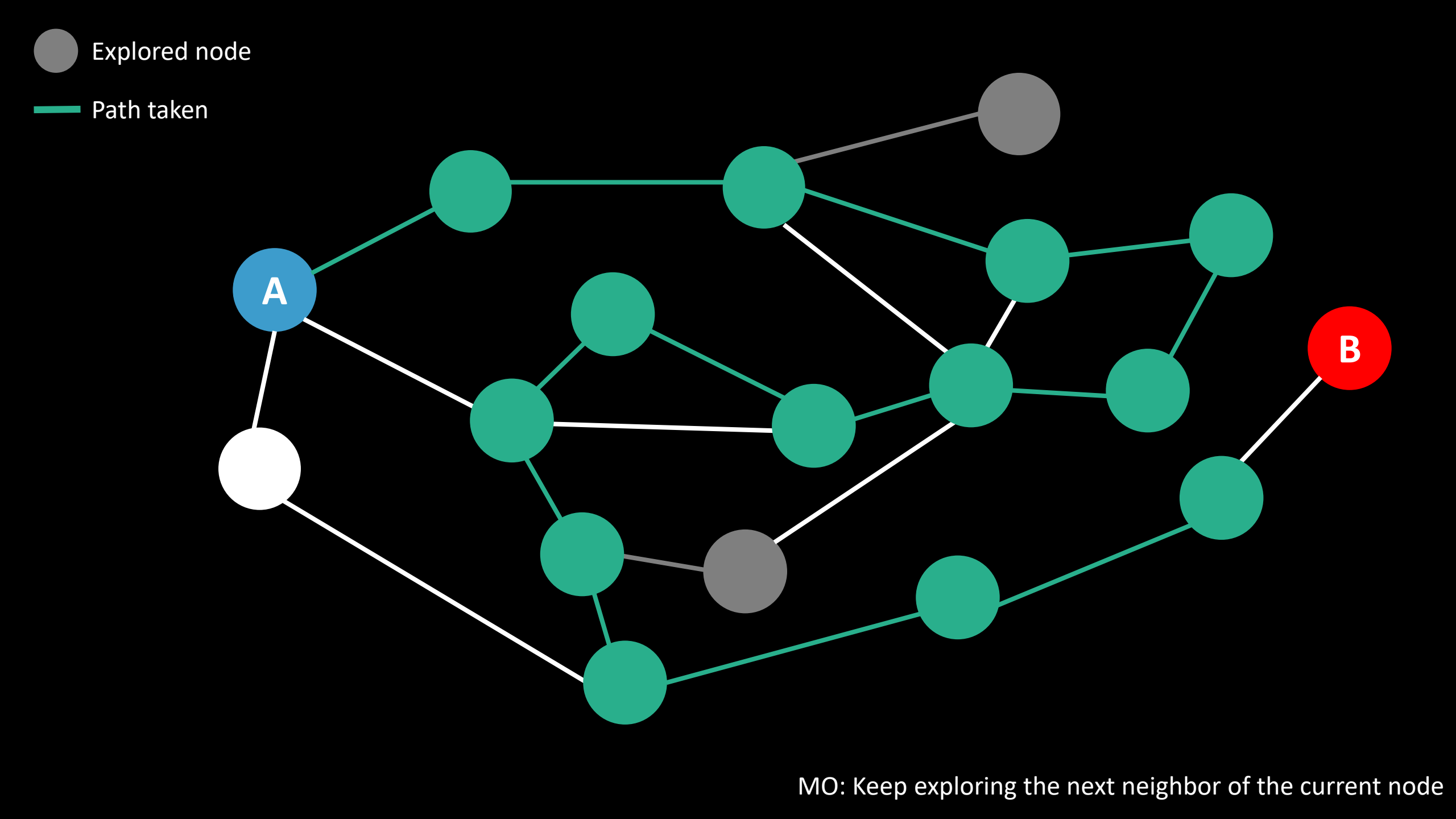
MO: Keep exploring the next neighbor of the current node



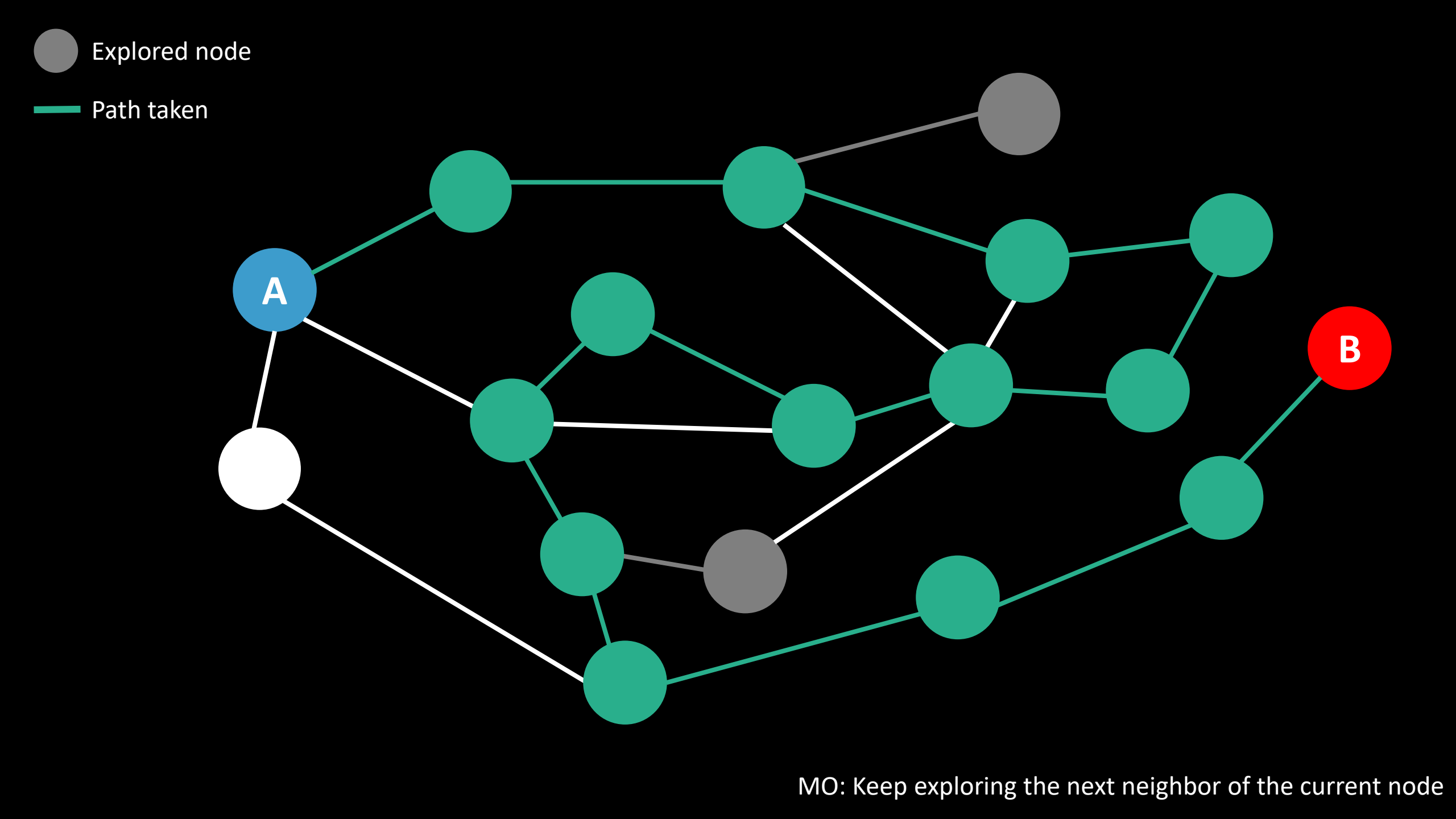
MO: Keep exploring the next neighbor of the current node



MO: Keep exploring the next neighbor of the current node



MO: Keep exploring the next neighbor of the current node



MO: Keep exploring the next neighbor of the current node

Path-finding

- Does a path exist from A→B?
- Keep track of:
  - Path taken
  - Nodes explored: do not re-explore previously explored nodes

Path-finding

- Does a path exist from A→B?
- Keep track of:
  - Path taken
  - Nodes explored: do not re-explore previously explored nodes
- Identified that a path exists through Depth-First Search (DFS)

Depth-First Search

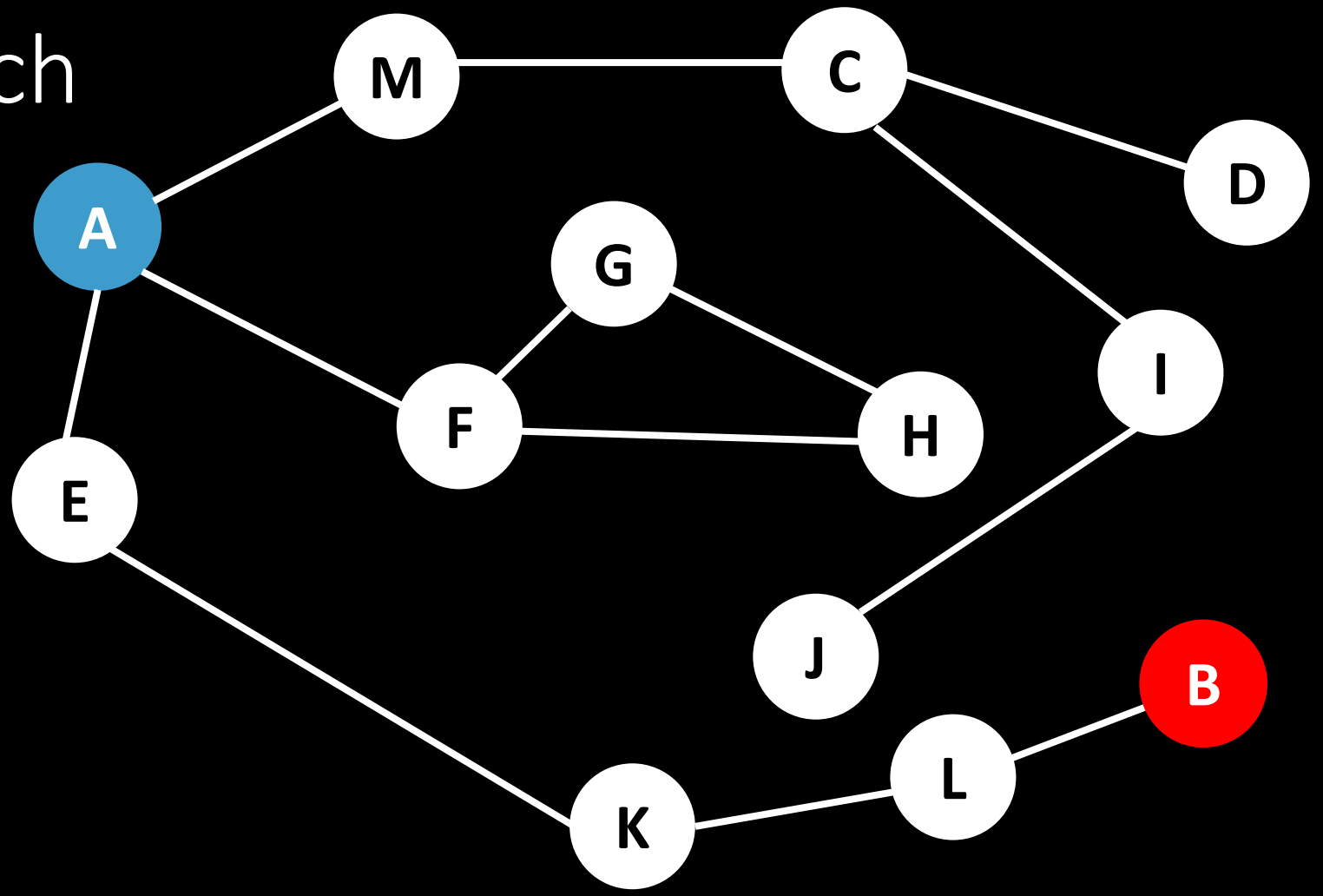
Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (push onto stack)
3. **Pop** a node off the stack
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left on the stack

Depth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (push onto stack)
3. **Pop** a node off the stack
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left on the stack

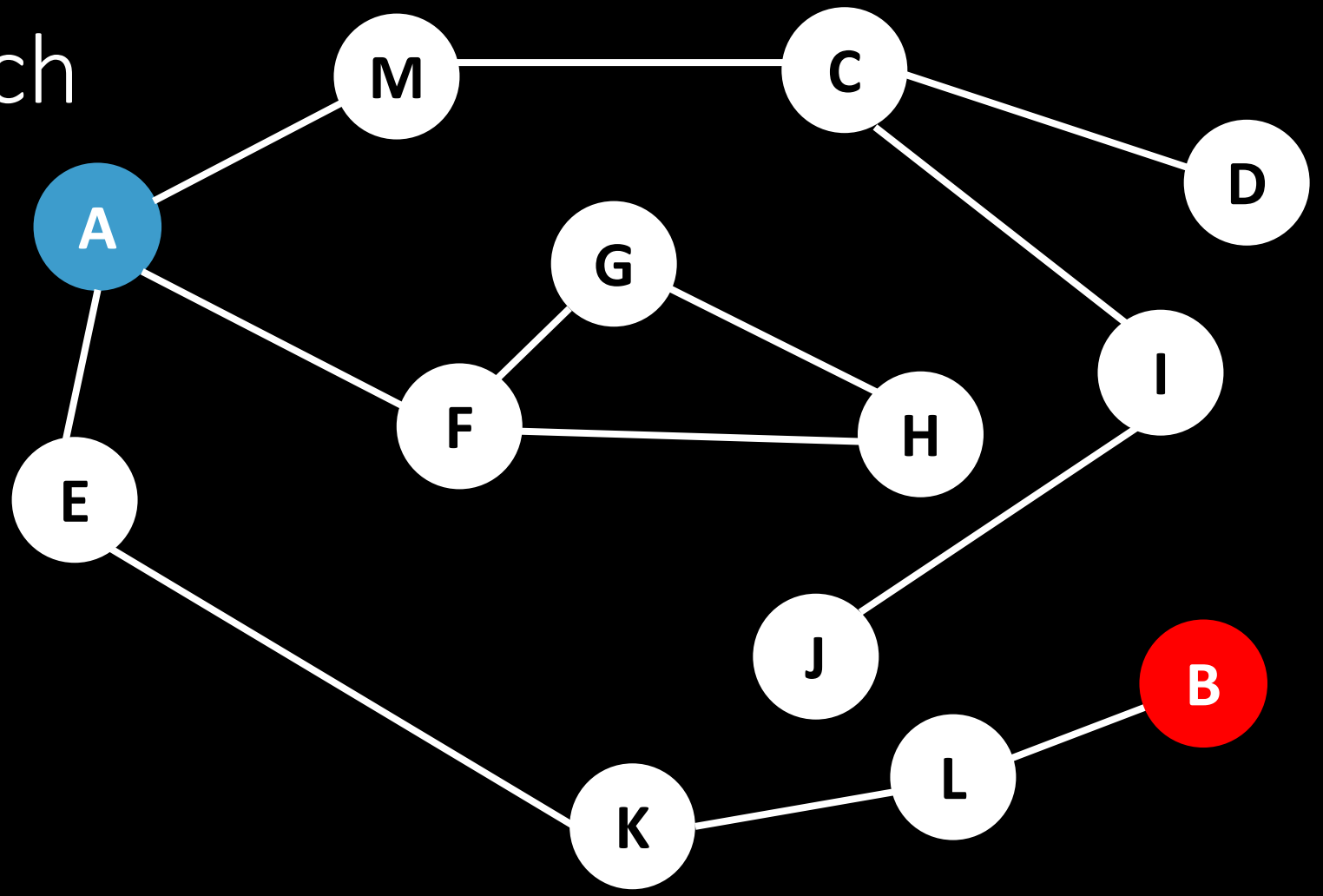


Search Frontier

Depth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (push onto stack)
3. **Pop** a node off the stack
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left on the stack



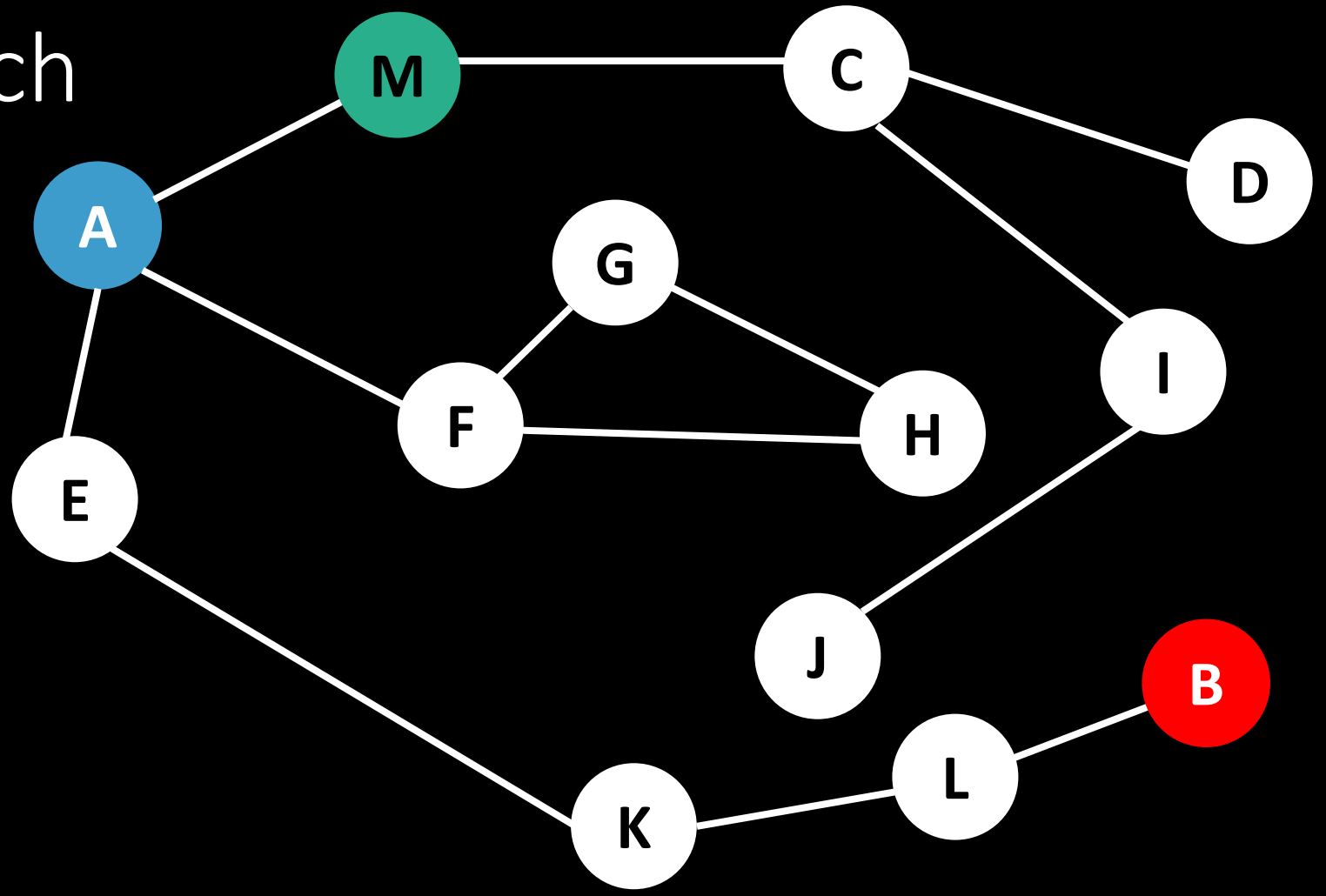
Search Frontier

~~A~~	M	F	E				

Depth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (push onto stack)
3. **Pop** a node off the stack
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left on the stack



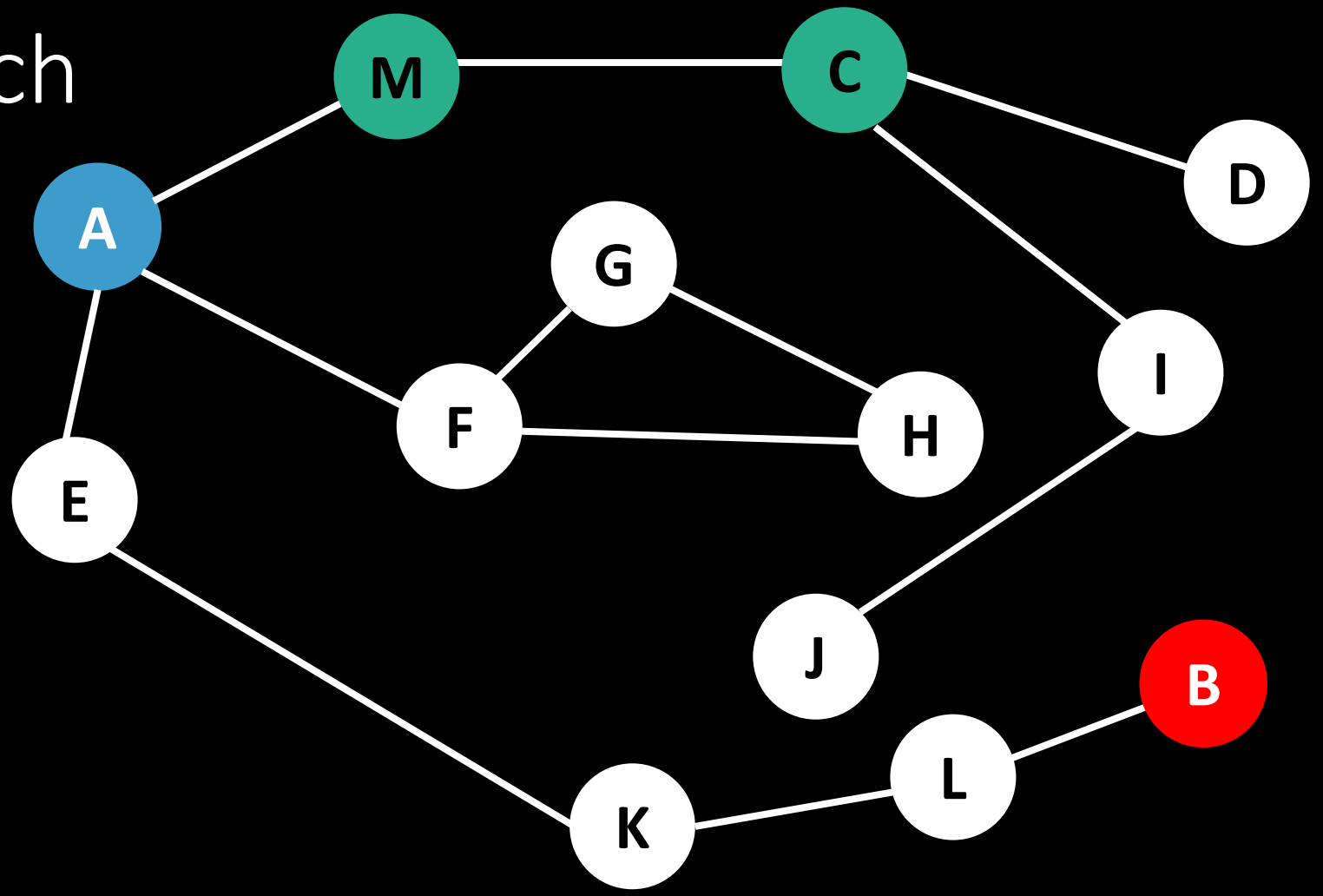
Search Frontier

A	M	C	F	E			

Depth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (push onto stack)
3. **Pop** a node off the stack
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left on the stack



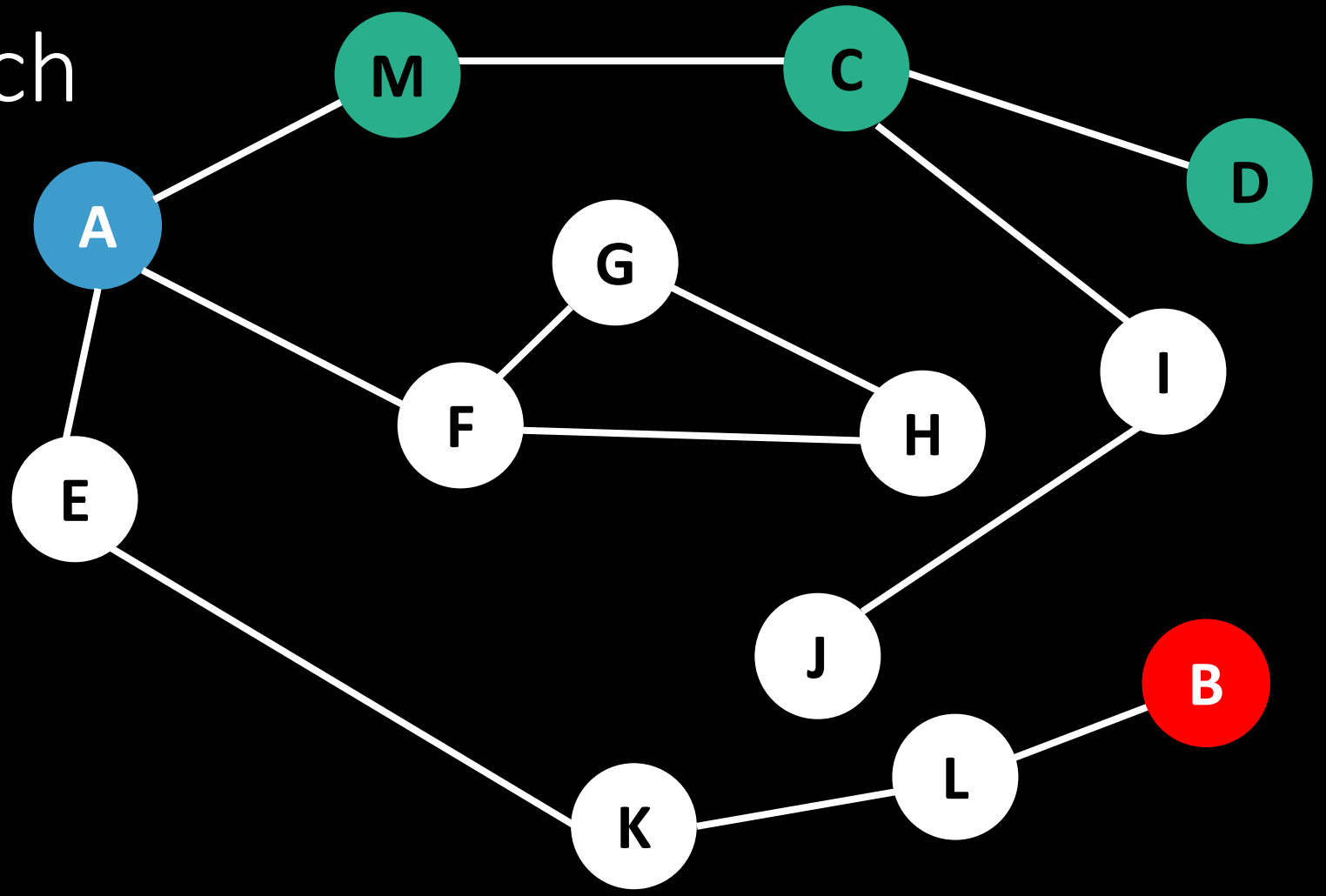
Search Frontier

A	M	C	D	I	F	E	

Depth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (push onto stack)
3. **Pop** a node off the stack
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left on the stack



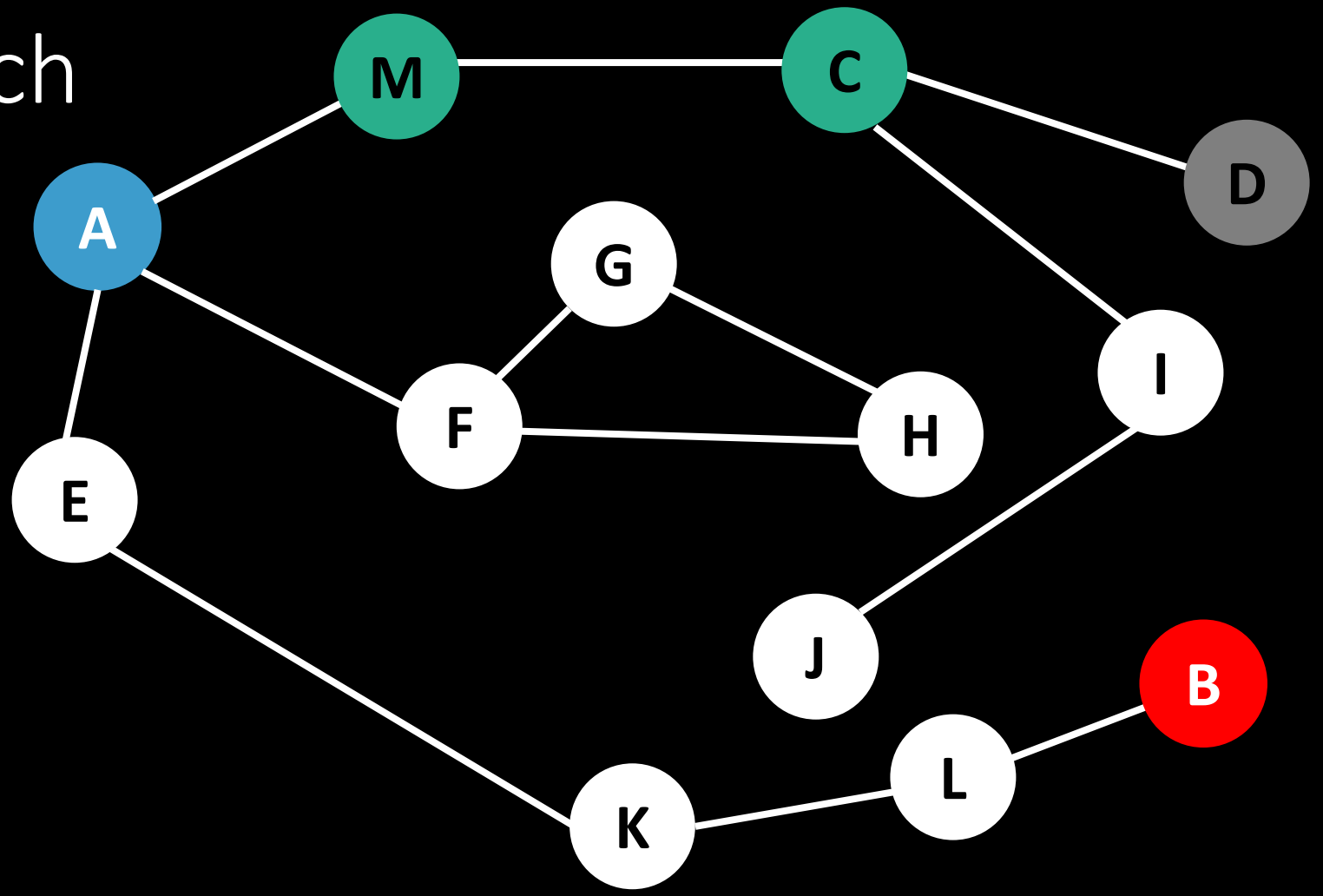
Search Frontier

A	M	C	D	I	F	E	

Depth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (push onto stack)
3. **Pop** a node off the stack
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left on the stack



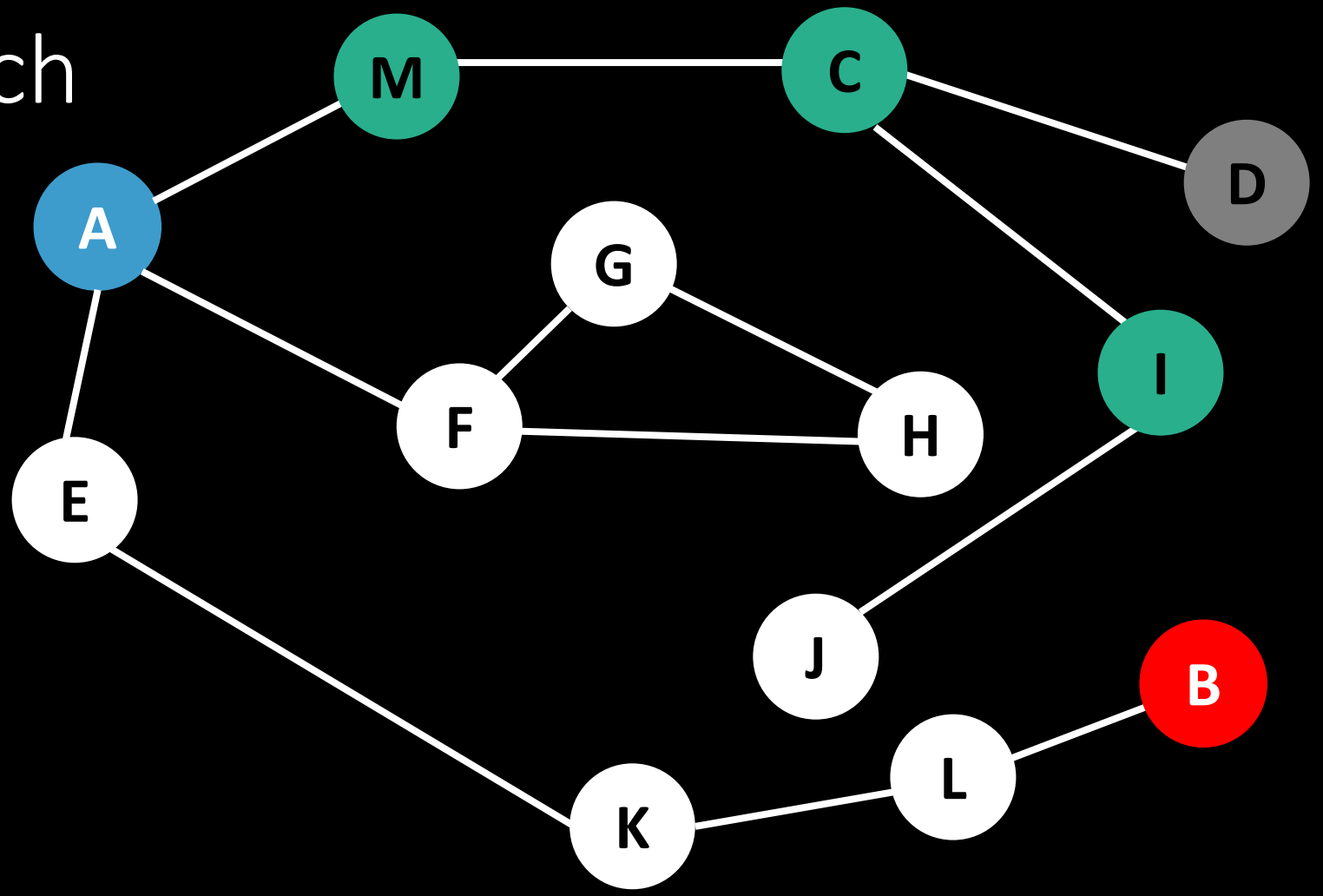
Search Frontier

A	M	C	D	I	F	E	

Depth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (push onto stack)
3. **Pop** a node off the stack
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left on the stack



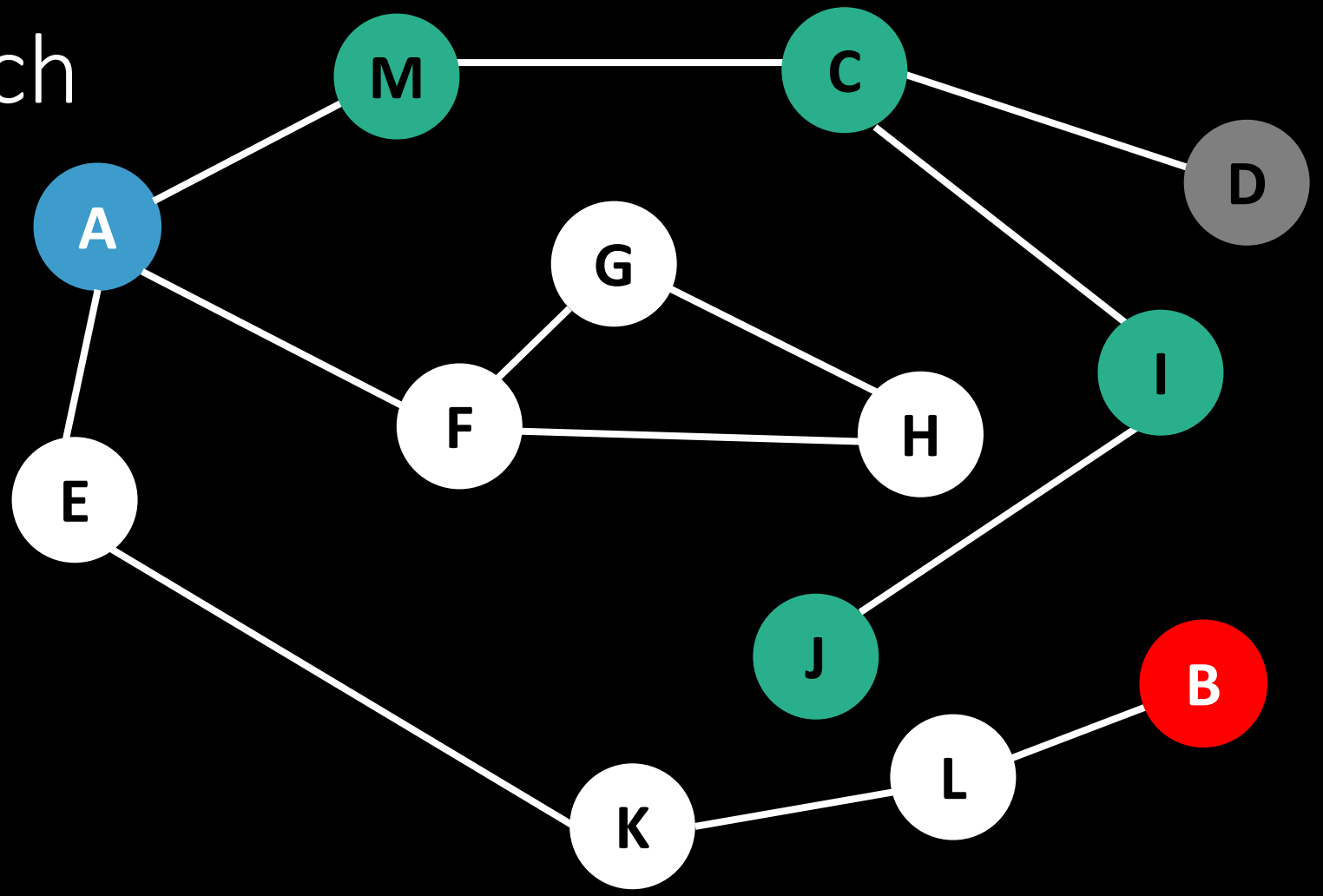
Search Frontier

A	M	C	D	I	J	F	E

Depth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (push onto stack)
3. **Pop** a node off the stack
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left on the stack



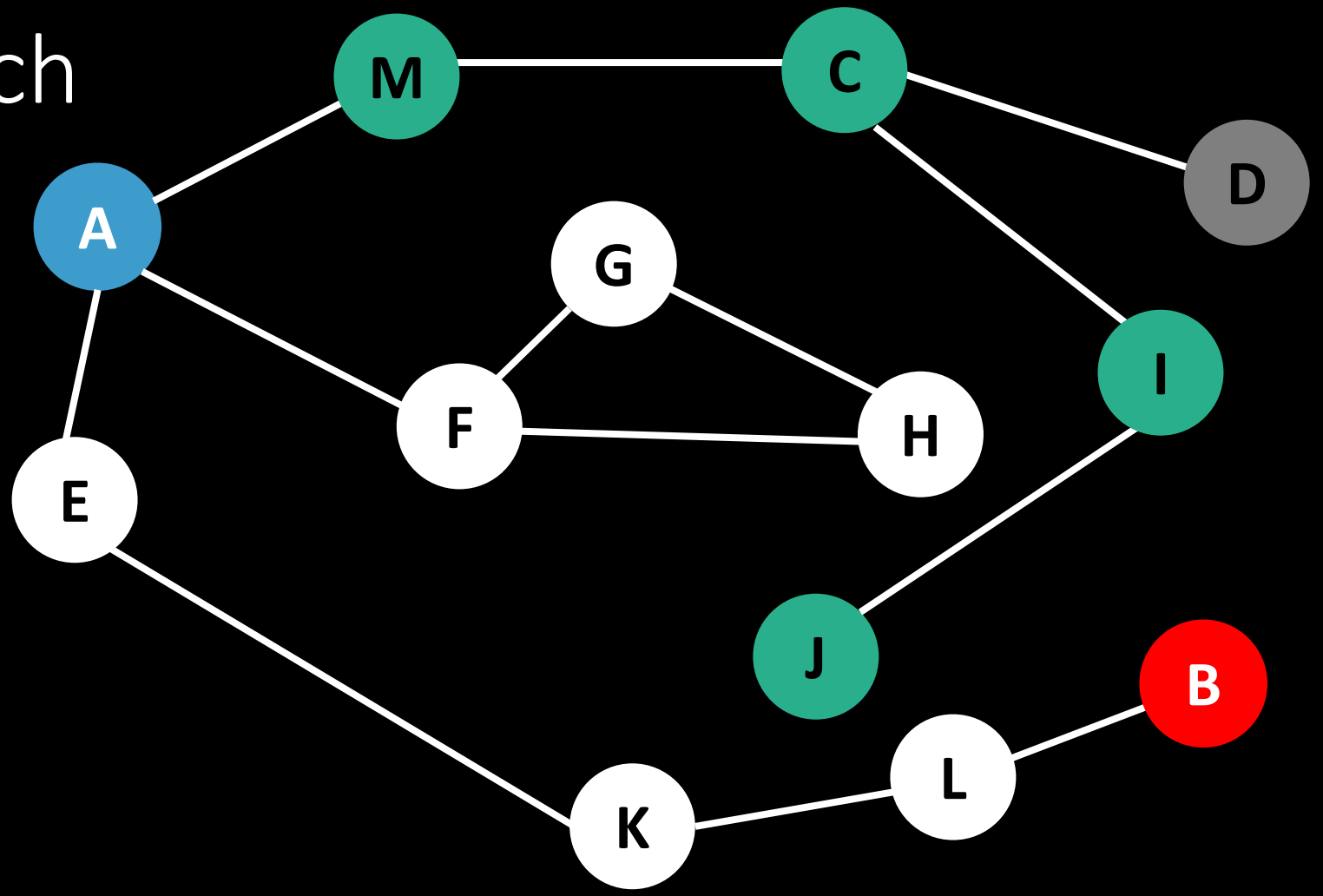
Search Frontier

A	M	C	D	I	J	F	E

Depth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (push onto stack)
3. **Pop** a node off the stack
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left on the stack



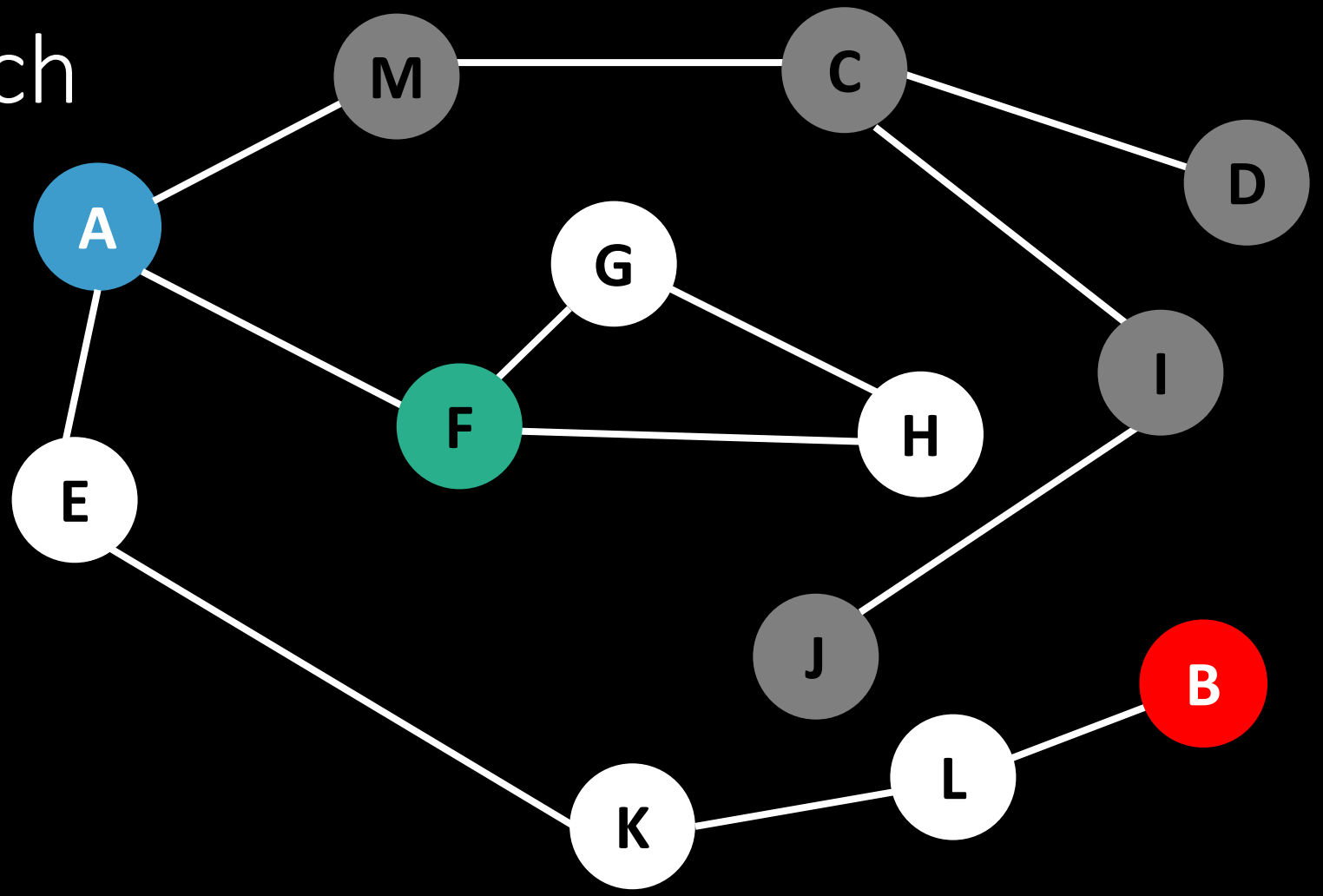
Search Frontier

F	E						

Depth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (push onto stack)
3. **Pop** a node off the stack
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left on the stack



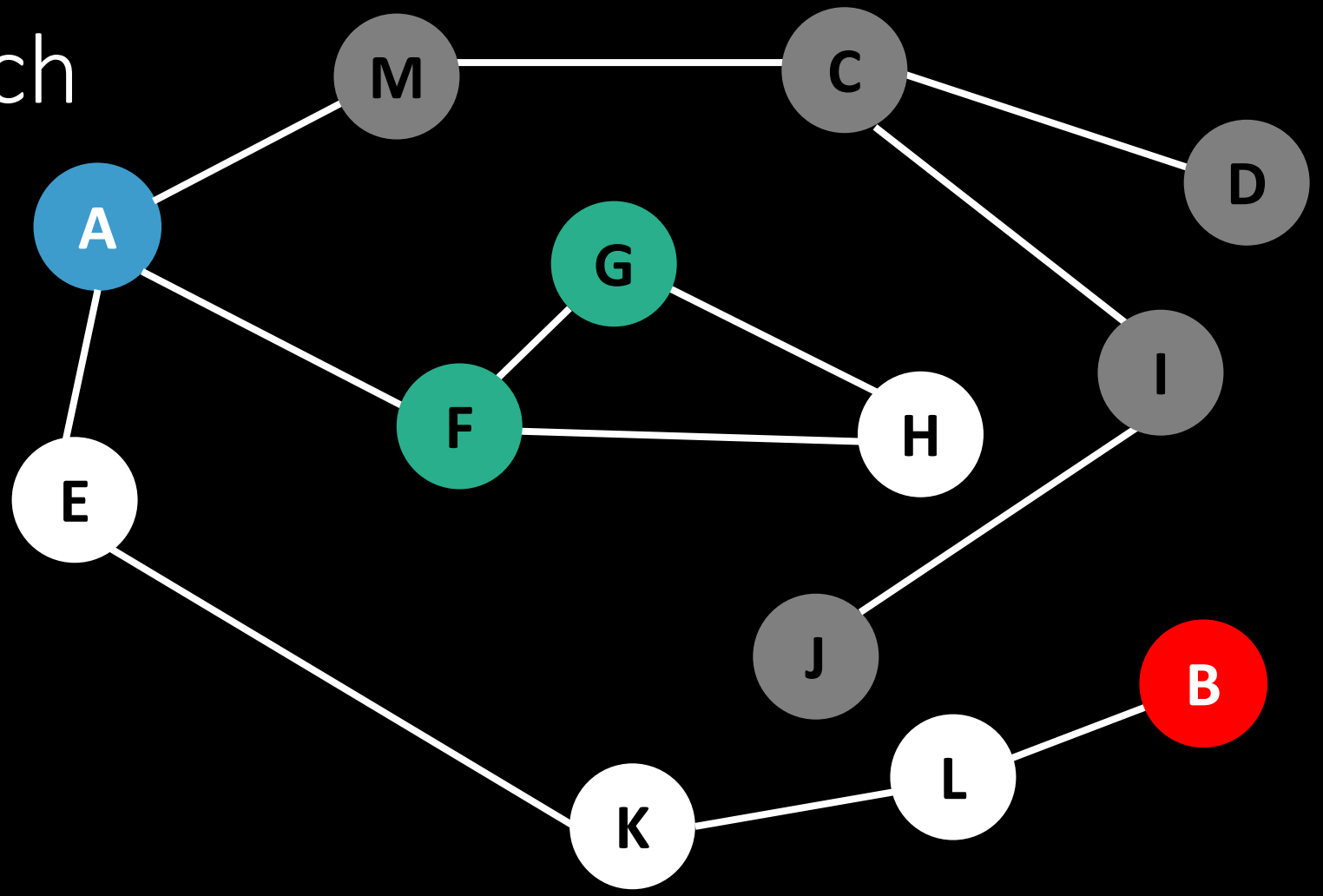
Search Frontier

~~F~~	G	H	E				

Depth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (push onto stack)
3. **Pop** a node off the stack
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left on the stack



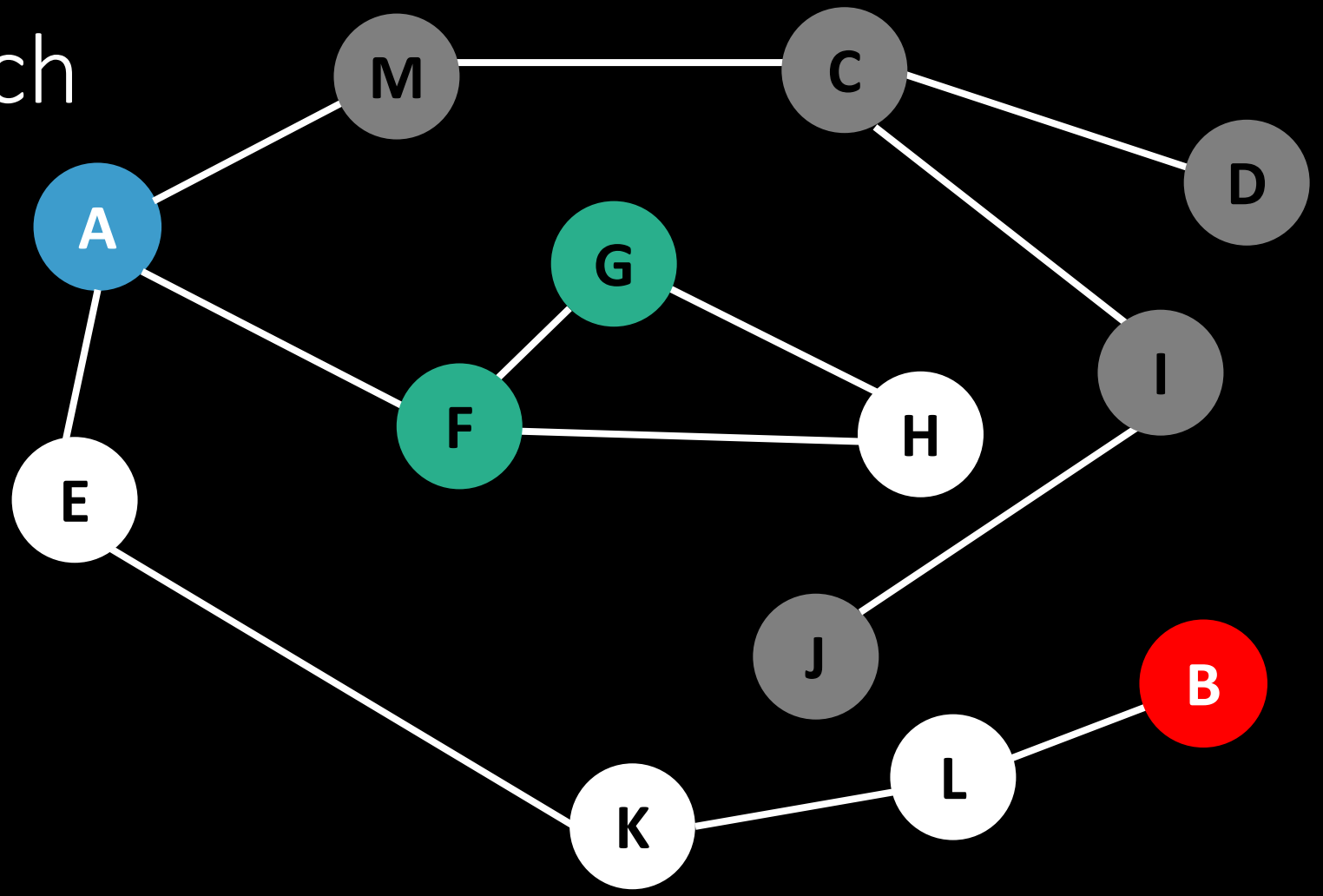
Search Frontier

F	G	H	H	E			

Depth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (push onto stack)
3. **Pop** a node off the stack
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left on the stack



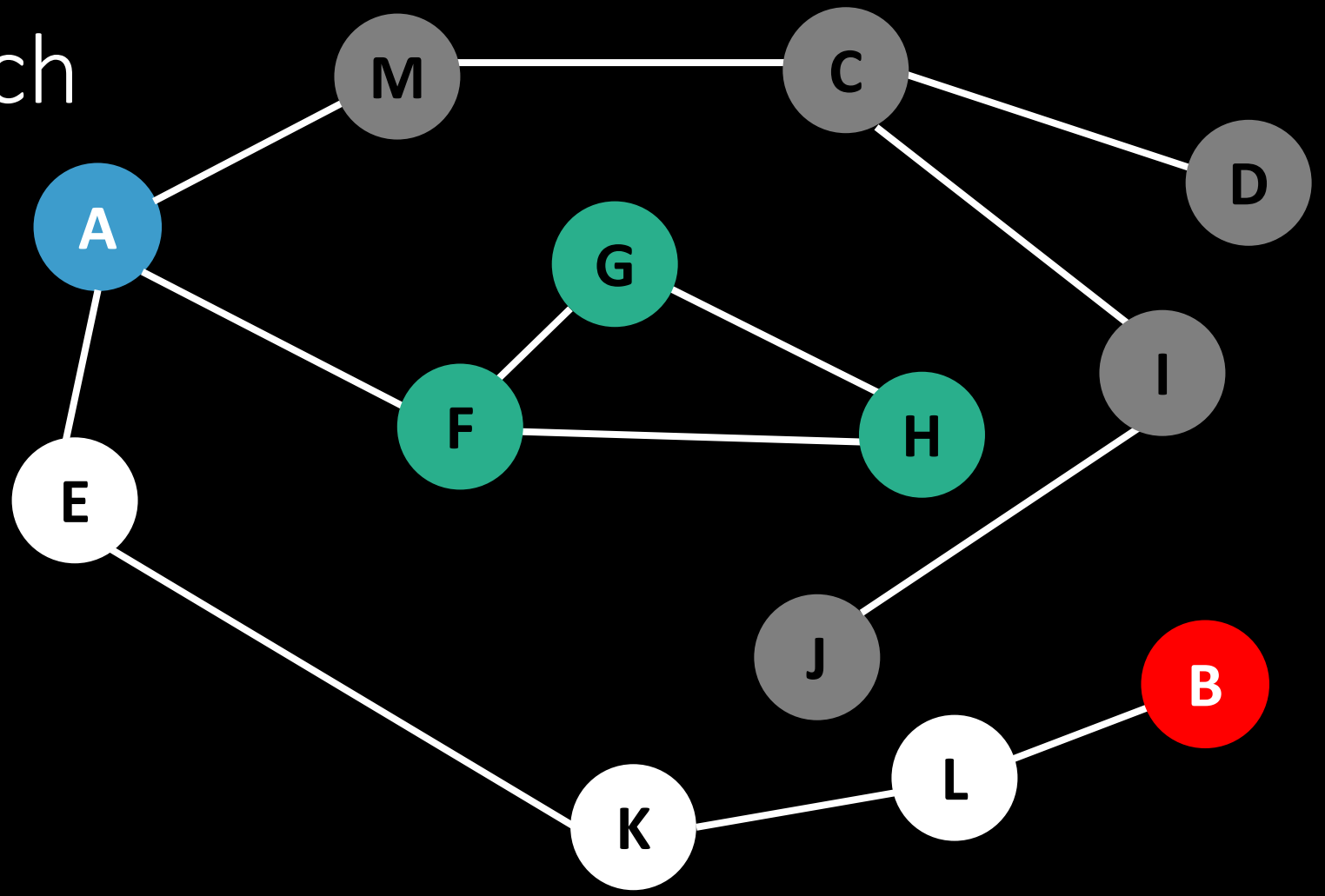
Search Frontier

F	G	H	H	E			

Depth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (push onto stack)
3. **Pop** a node off the stack
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left on the stack

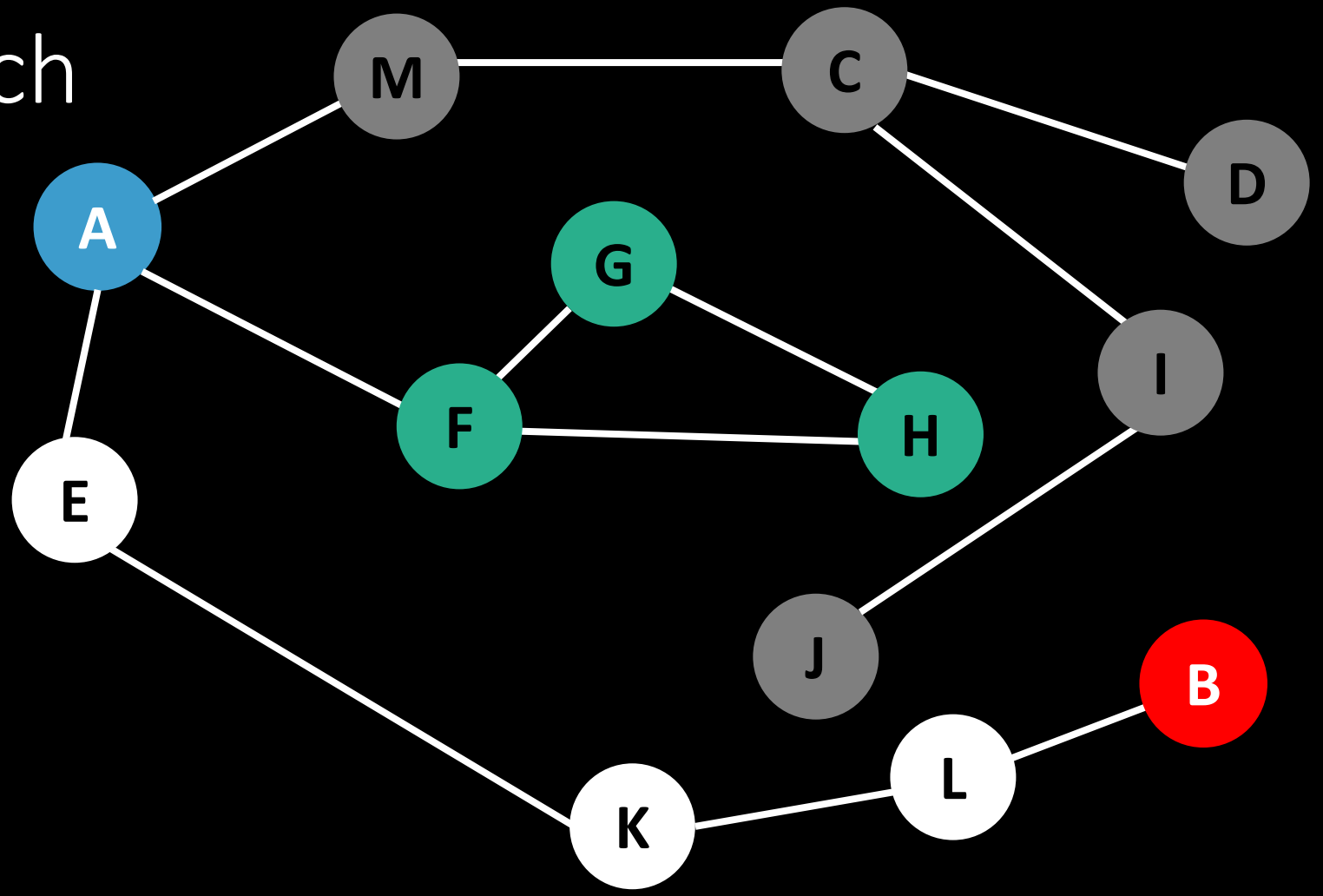


Search Frontier

Depth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (push onto stack)
3. **Pop** a node off the stack
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left on the stack

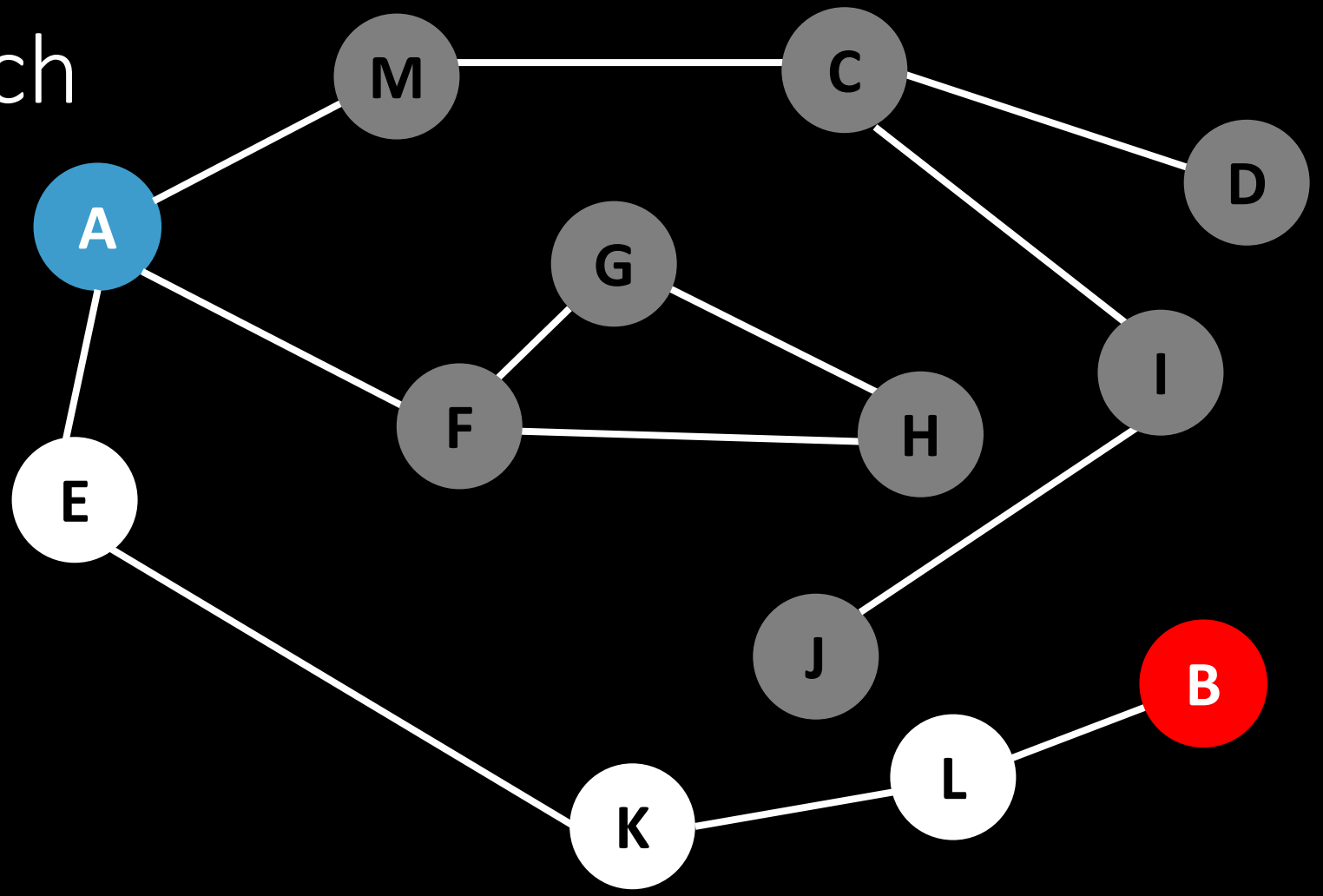


Search Frontier

Depth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (push onto stack)
3. **Pop** a node off the stack
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left on the stack

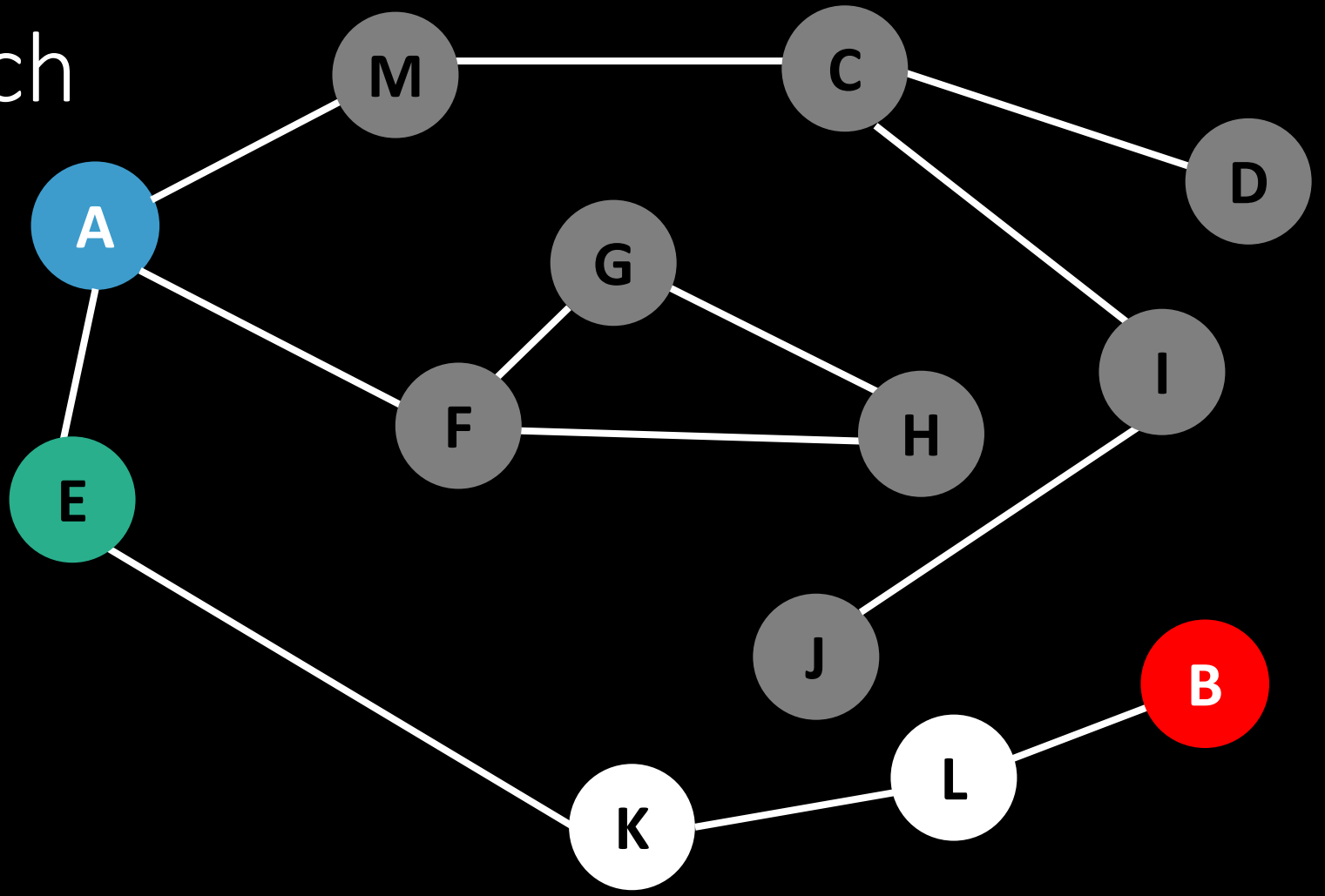


Search Frontier

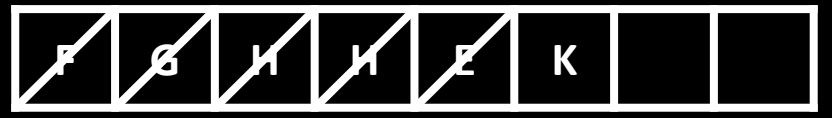
Depth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (push onto stack)
3. **Pop** a node off the stack
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left on the stack



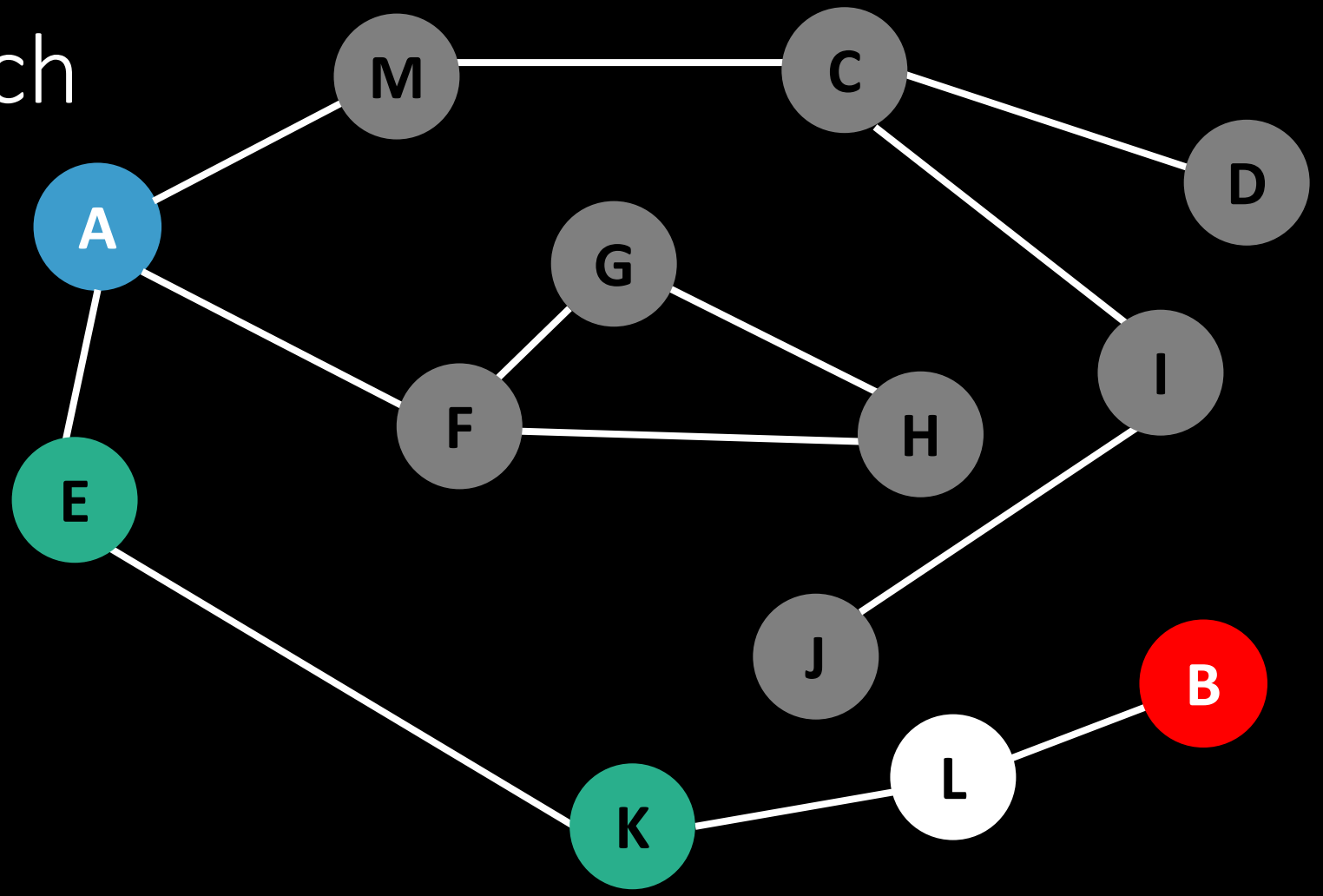
Search Frontier



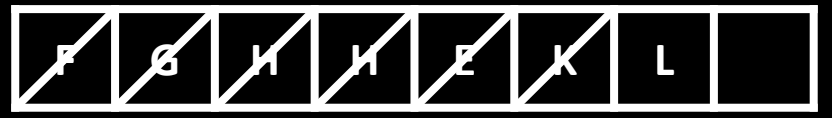
Depth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (push onto stack)
3. **Pop** a node off the stack
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left on the stack



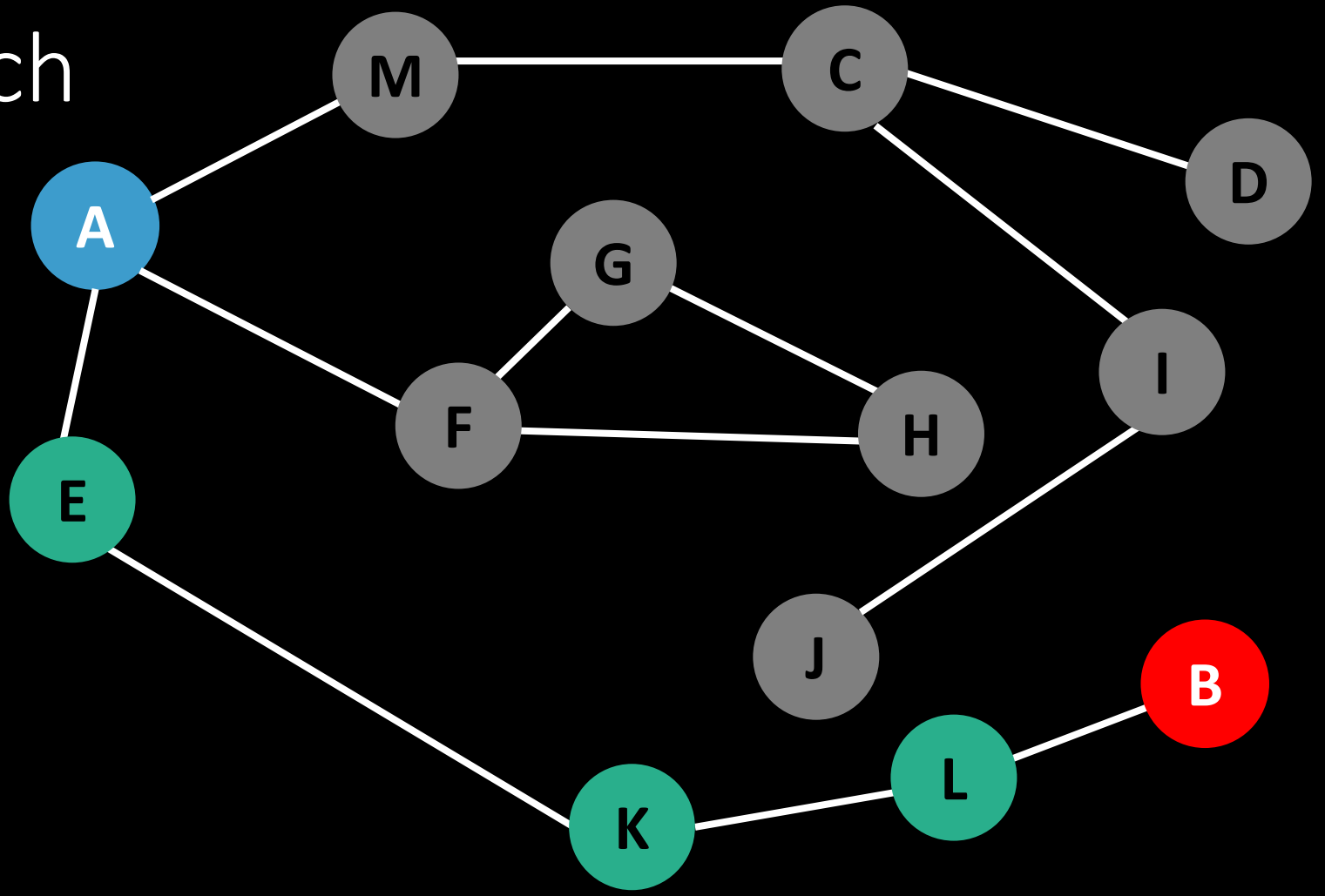
Search Frontier



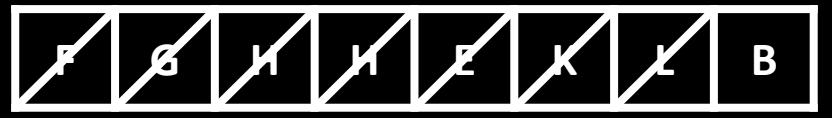
Depth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (push onto stack)
3. **Pop** a node off the stack
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left on the stack



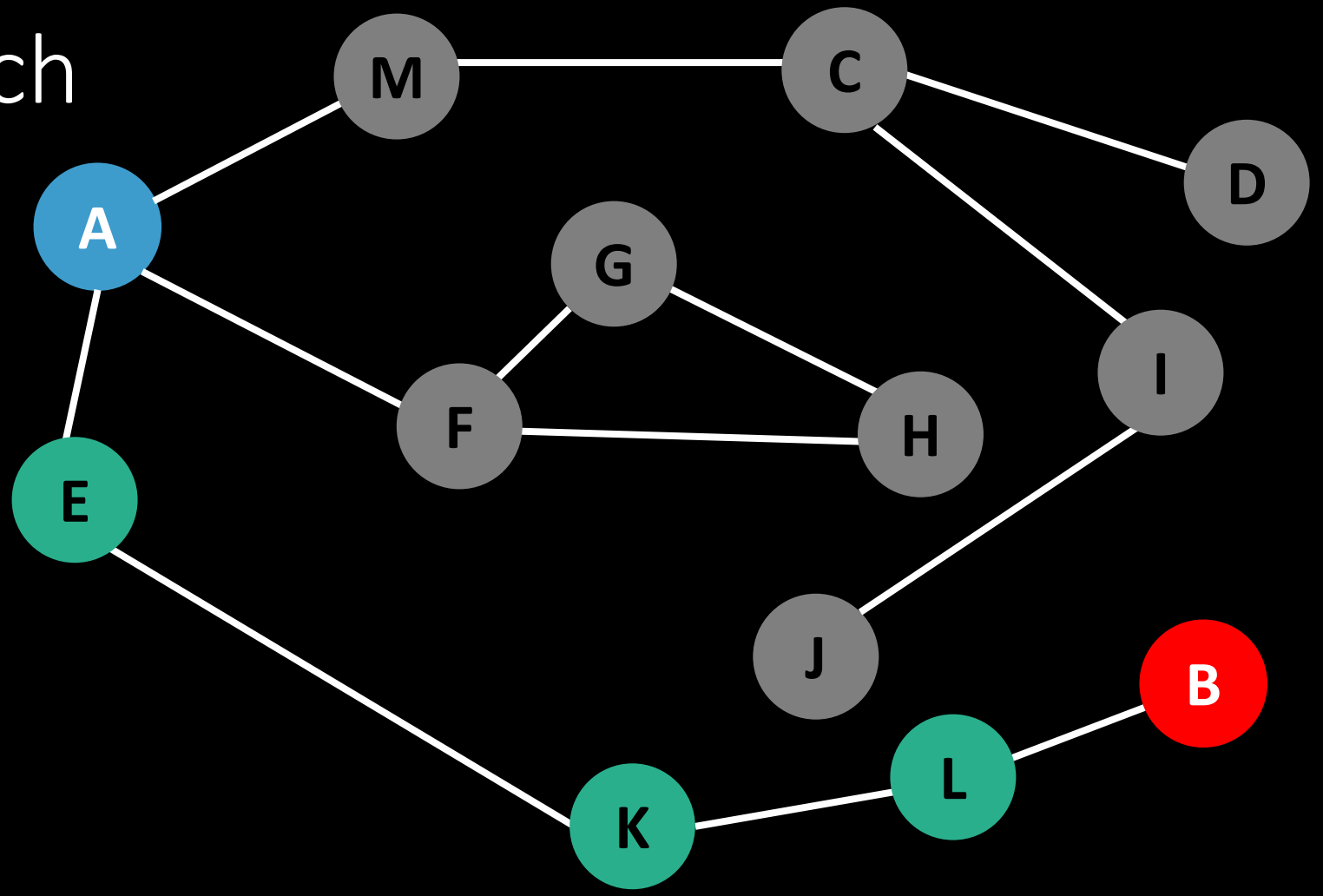
Search Frontier



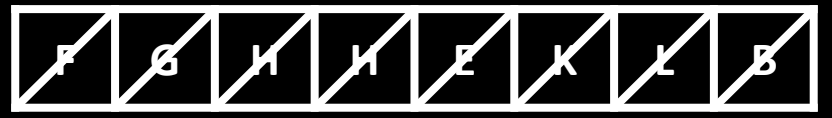
Depth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (push onto stack)
3. **Pop** a node off the stack
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left on the stack



Search Frontier



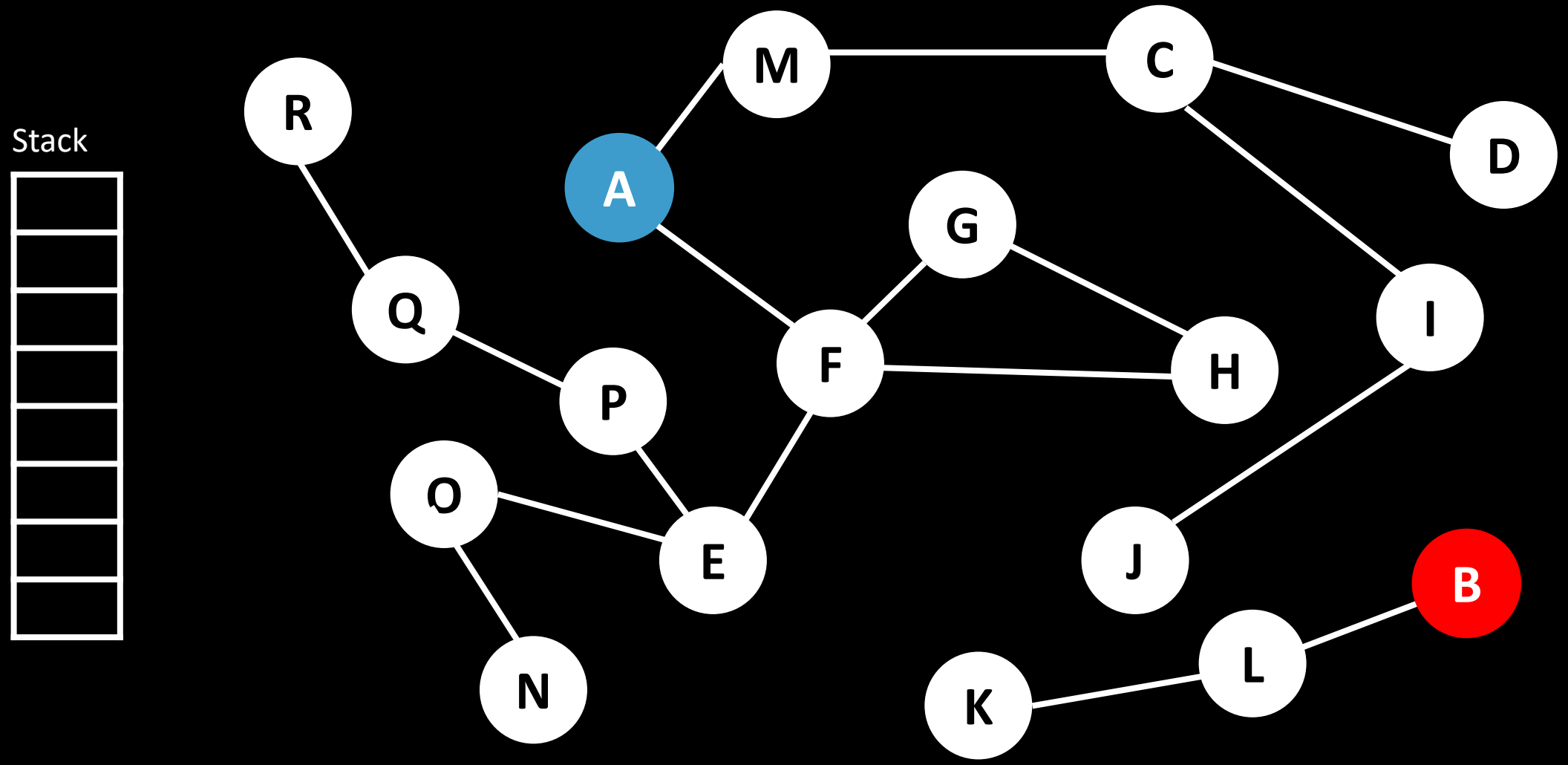
Depth-First Search

- Expands depth-first
- Does **not** give us a shortest path between two nodes
- Finds a path if one exists

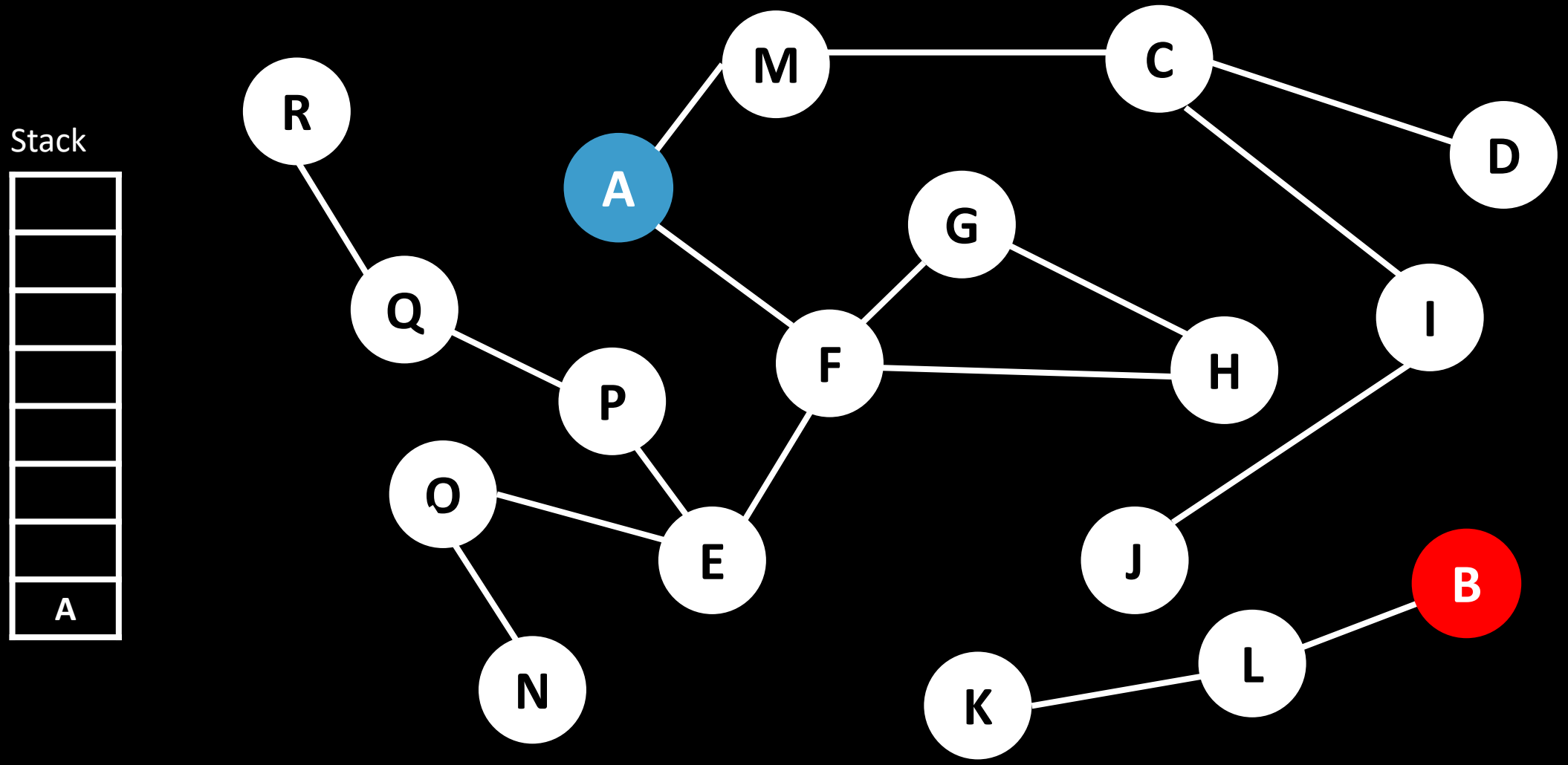
Depth-First Search

- Expands depth-first
- Does **not** give us a shortest path between two nodes
- Finds a path if one exists
- Can be used for maze solving
  - We just want a path, any path, who cares about the length?
  - https://www.youtube.com/watch?v=Kv8SRv4HH1g
- There are other cool applications of DFS which we may get to later in the course

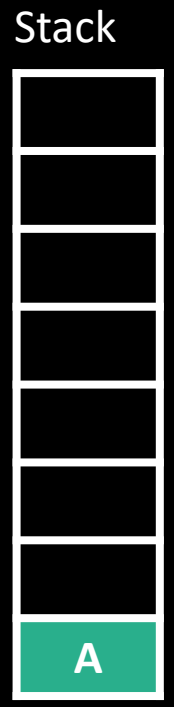
Another DFS Example

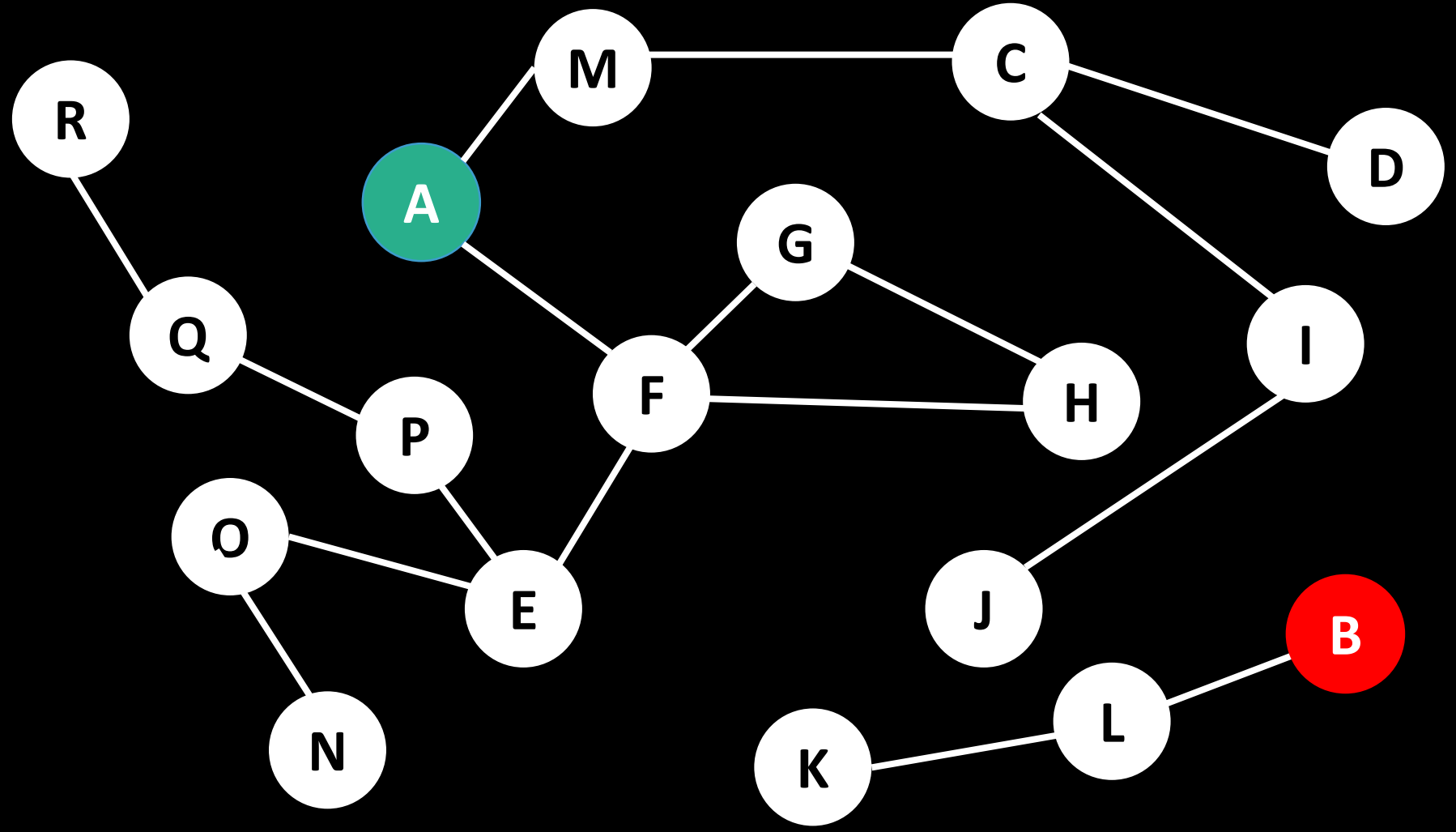


Another DFS Example

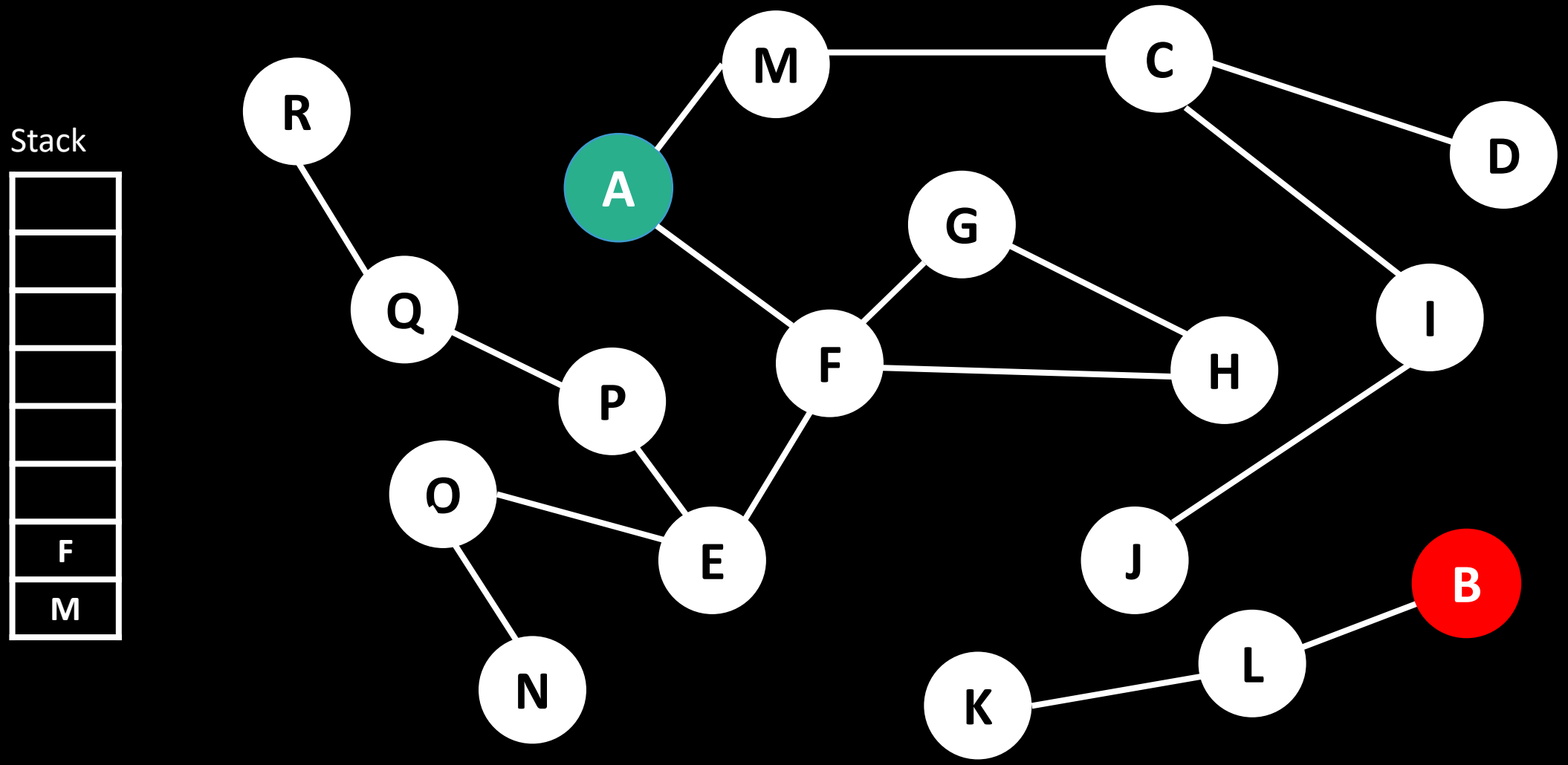


Another DFS Example

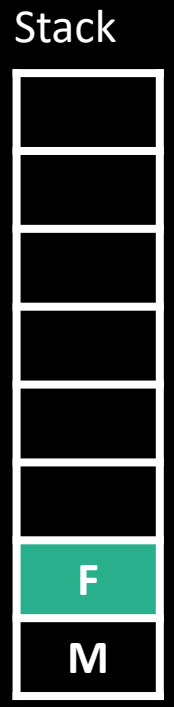


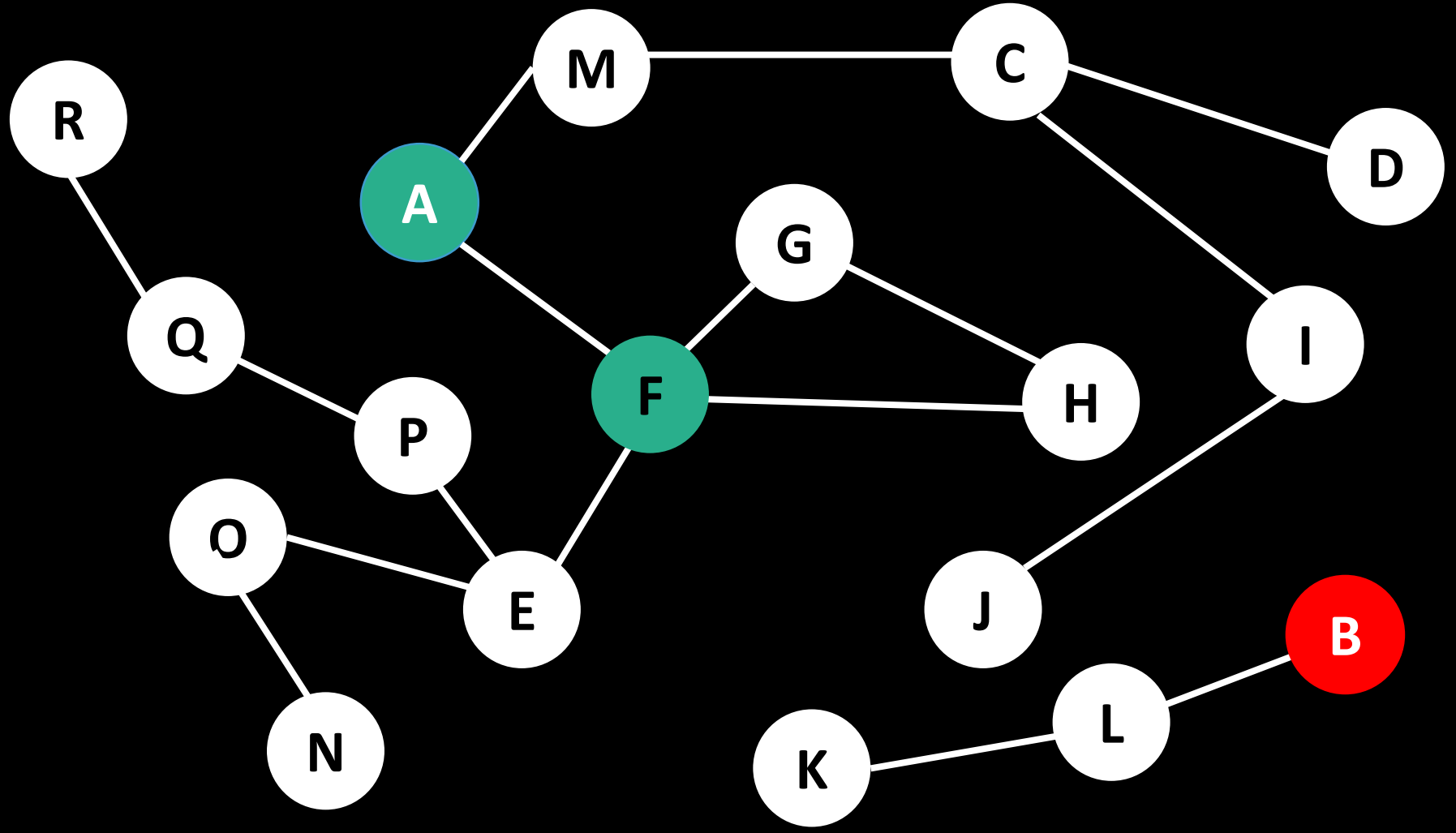


Another DFS Example

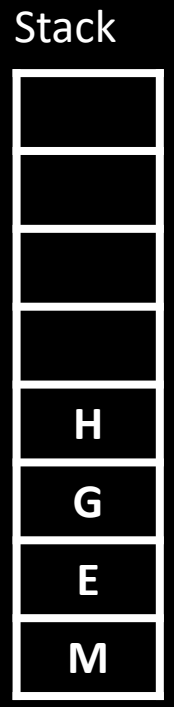


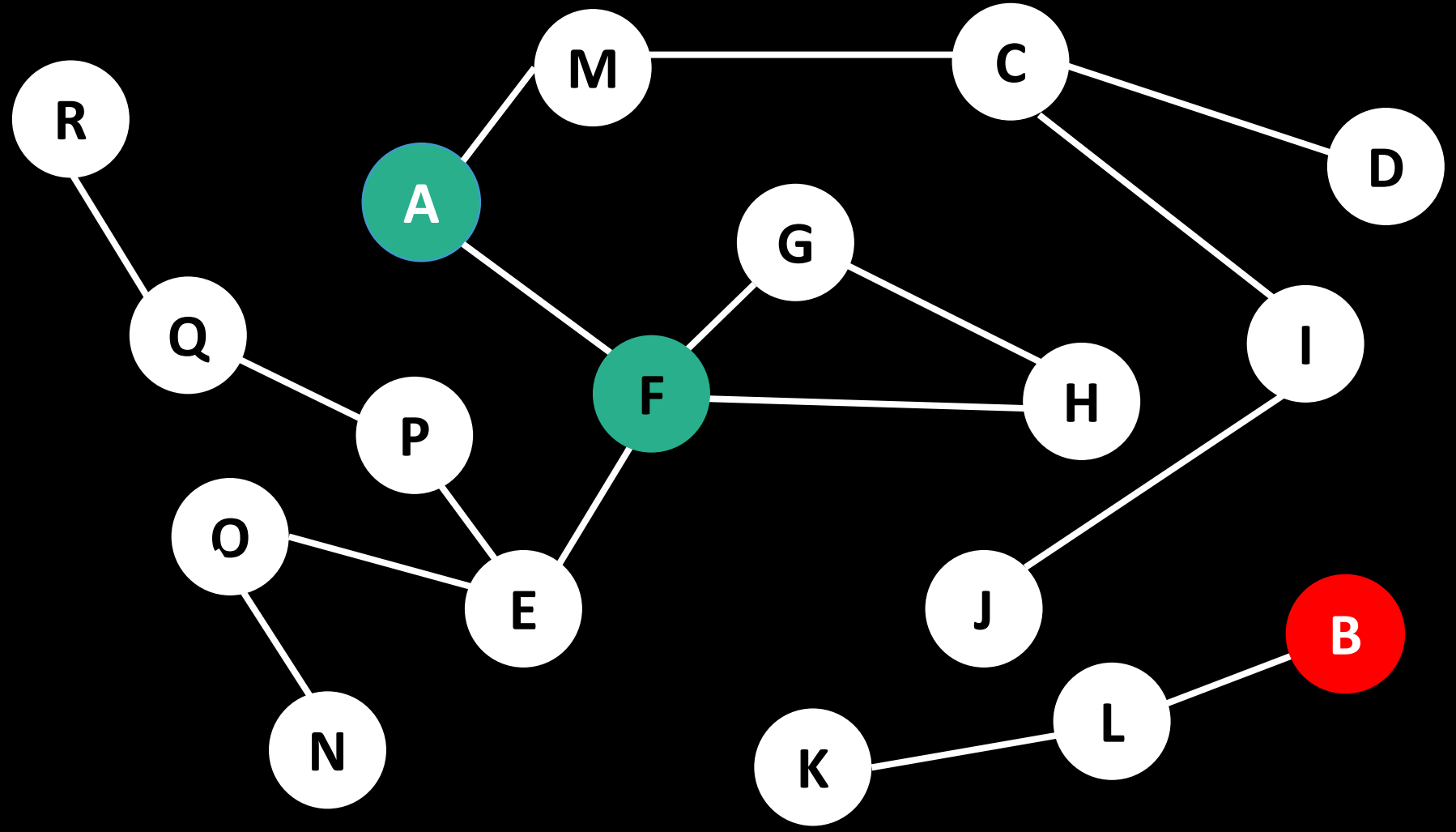
Another DFS Example



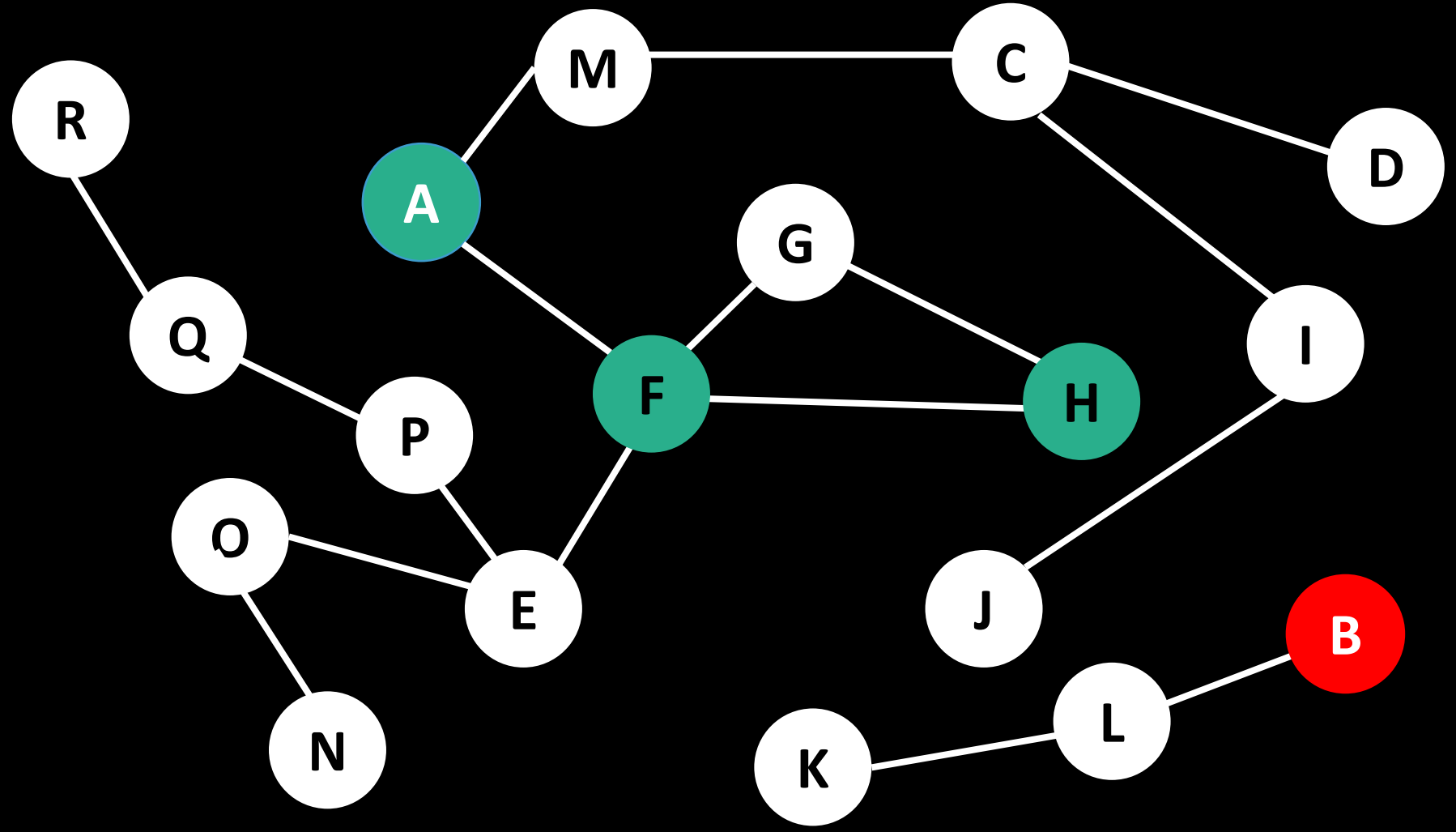


Another DFS Example

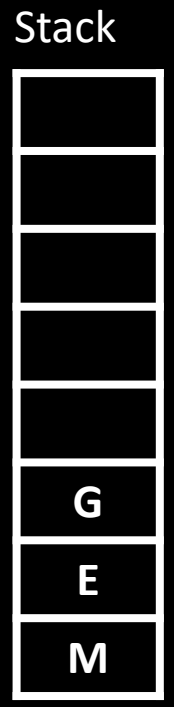


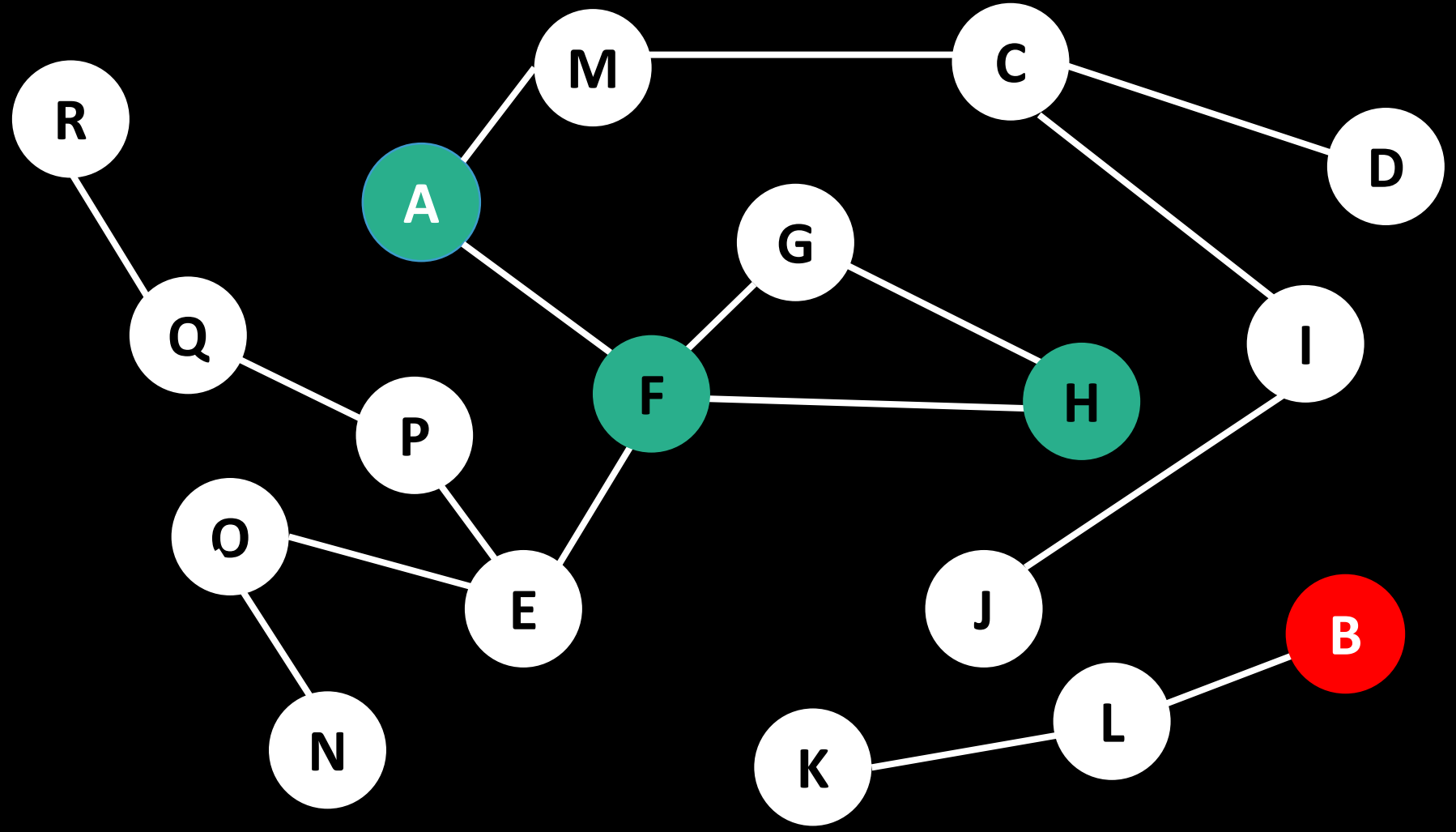


Another DFS Example

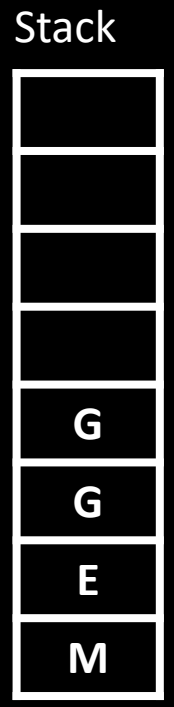


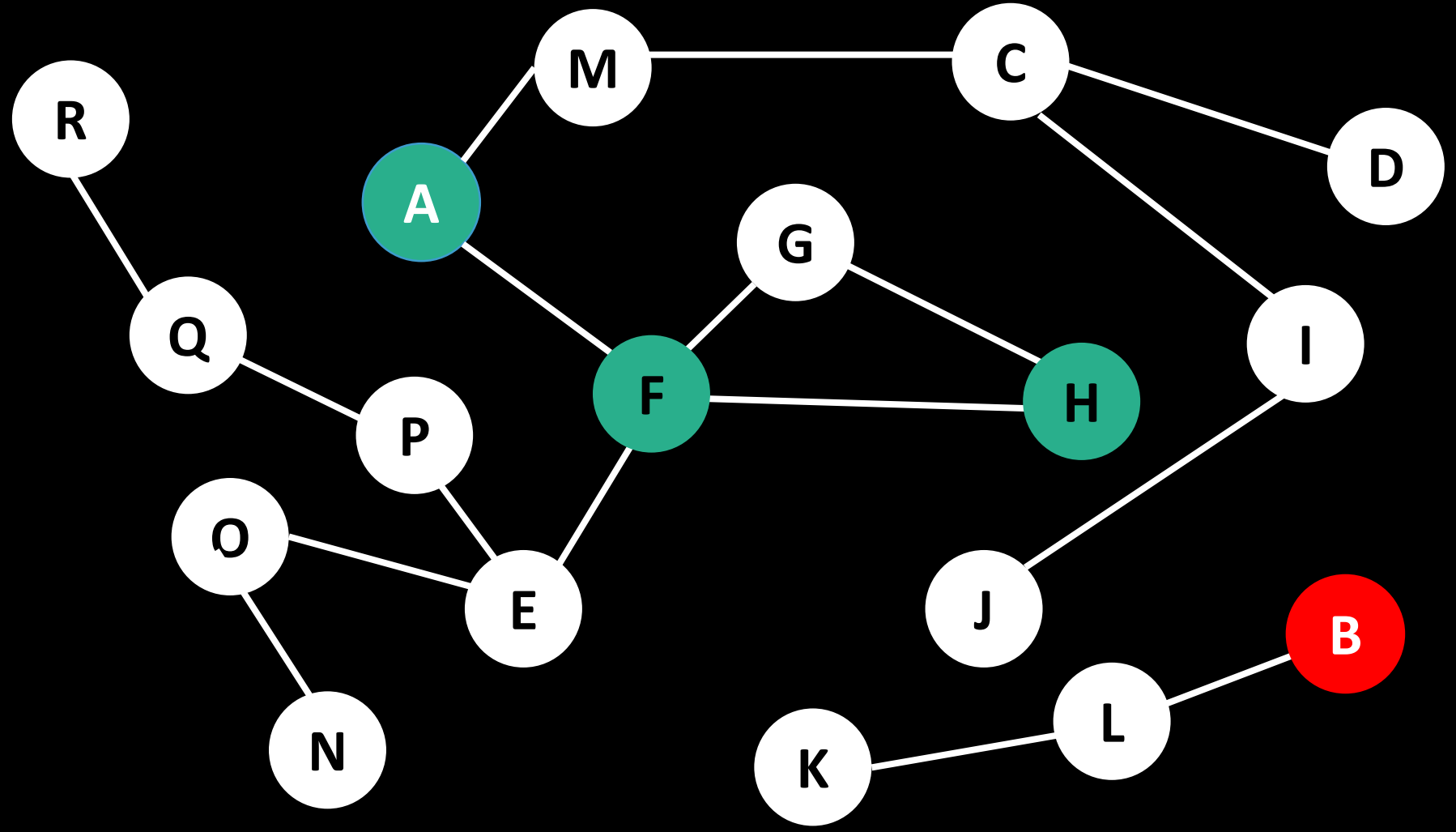
Another DFS Example



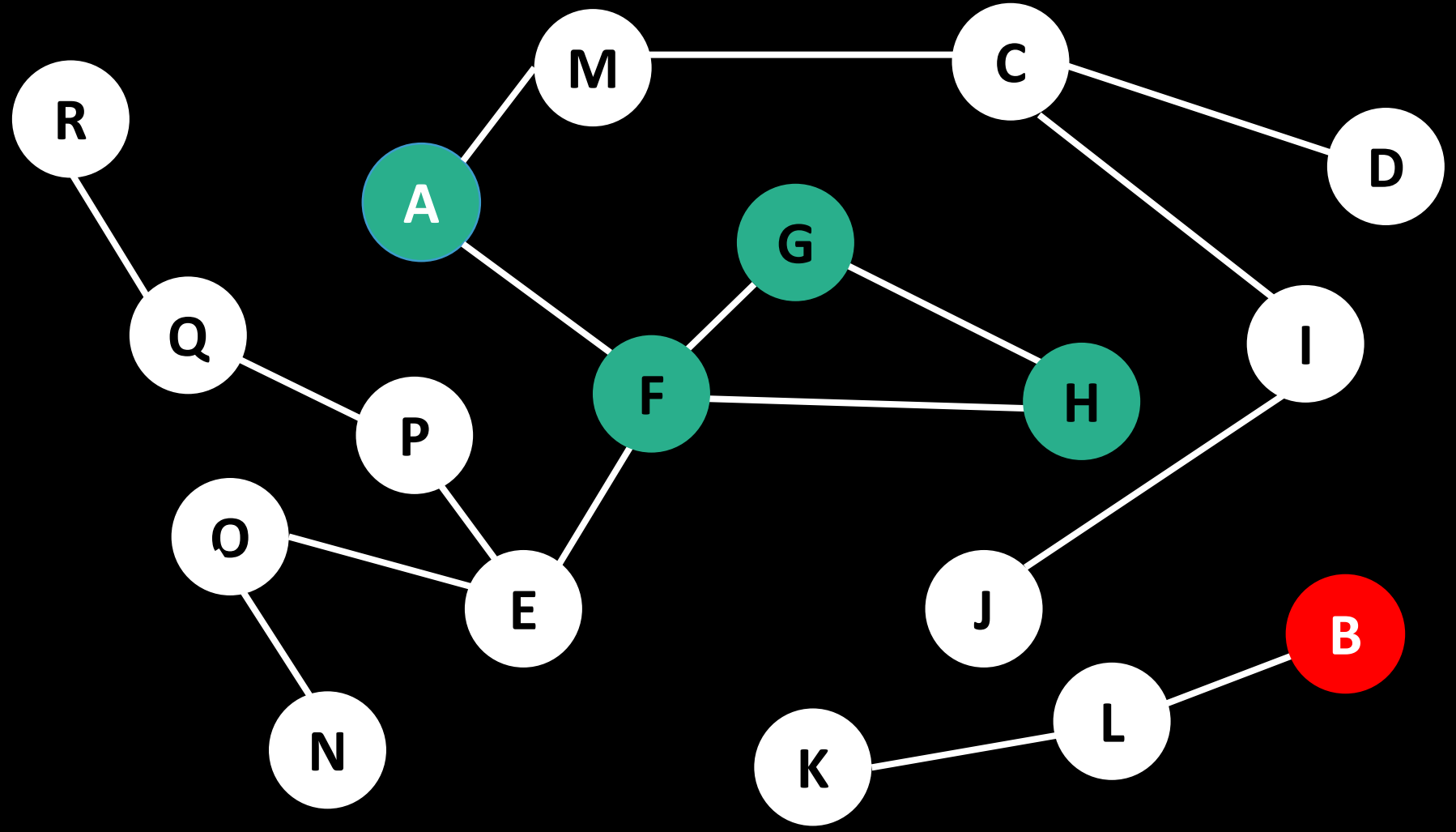


Another DFS Example

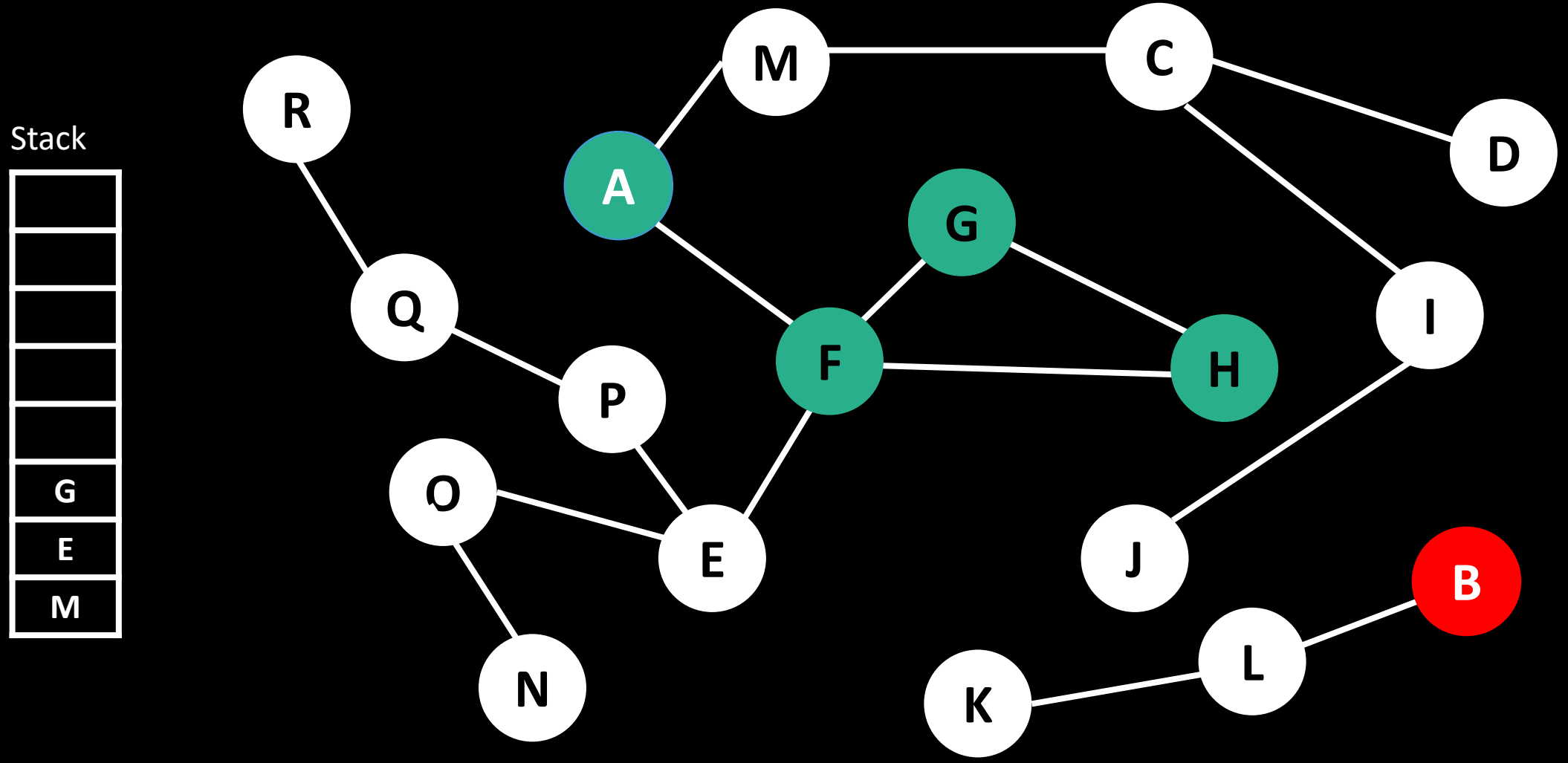




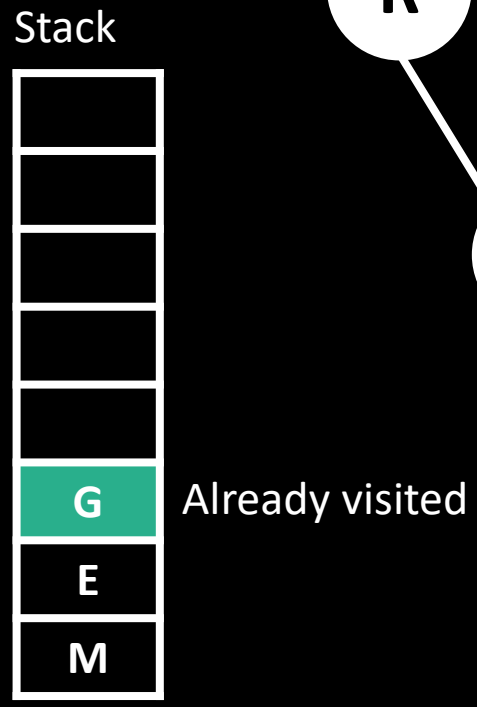
Another DFS Example



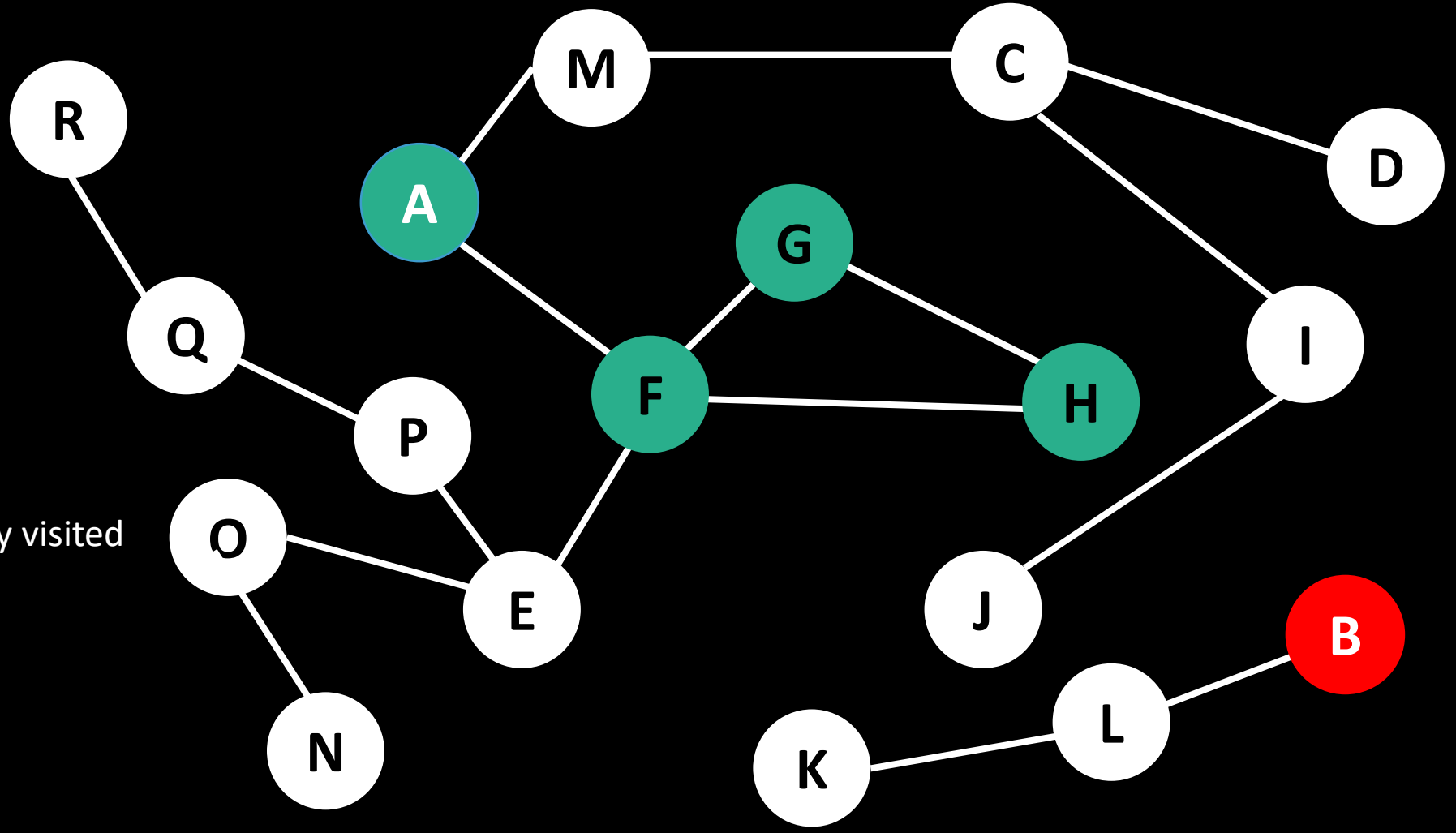
Another DFS Example



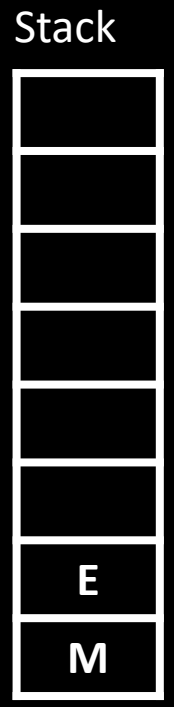
Another DFS Example

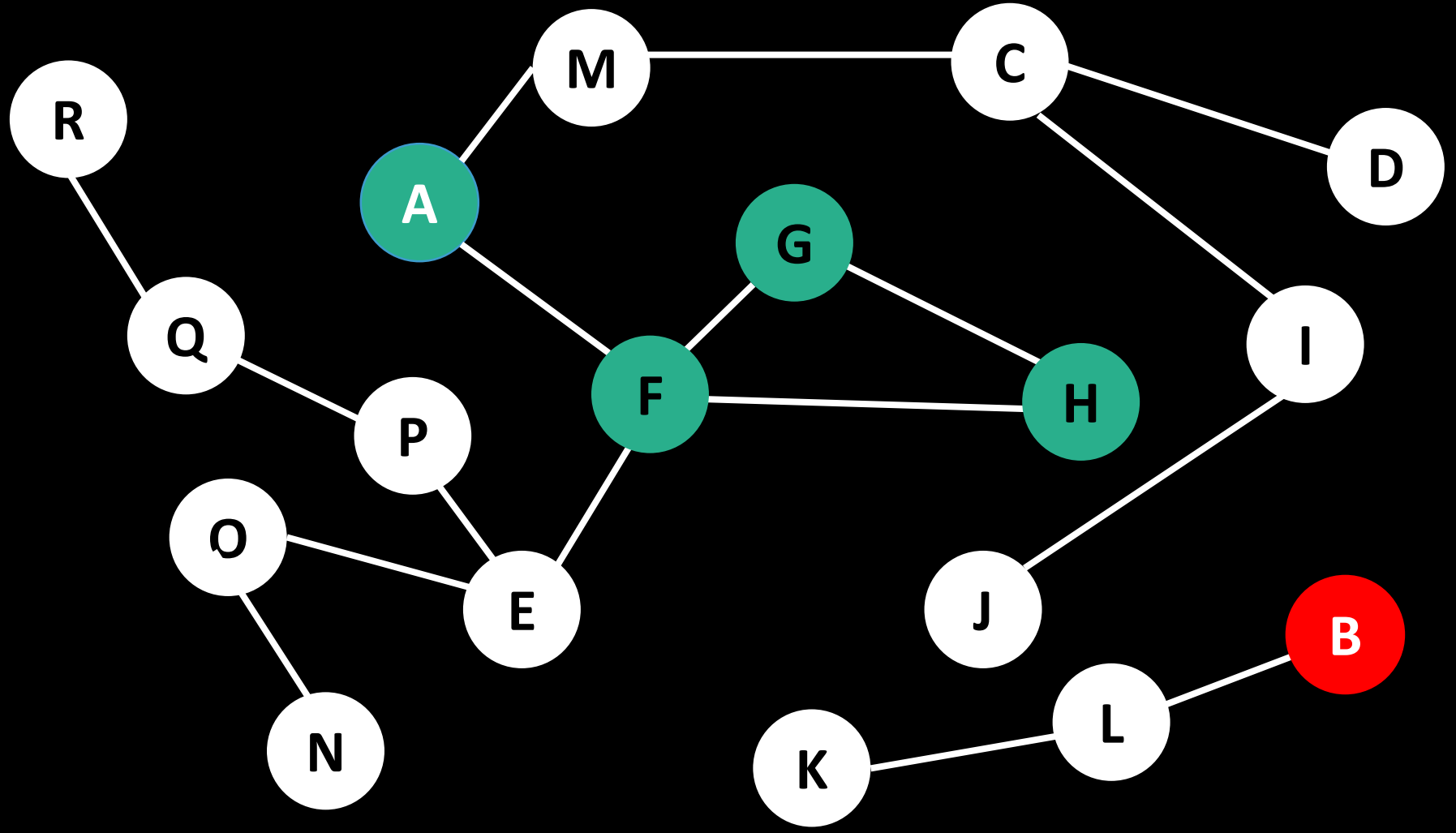


Already visited

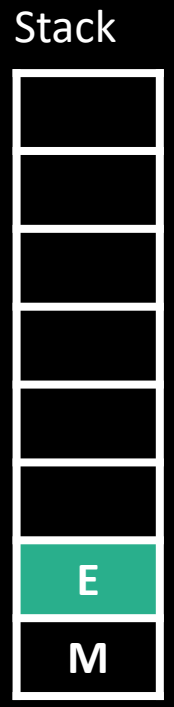


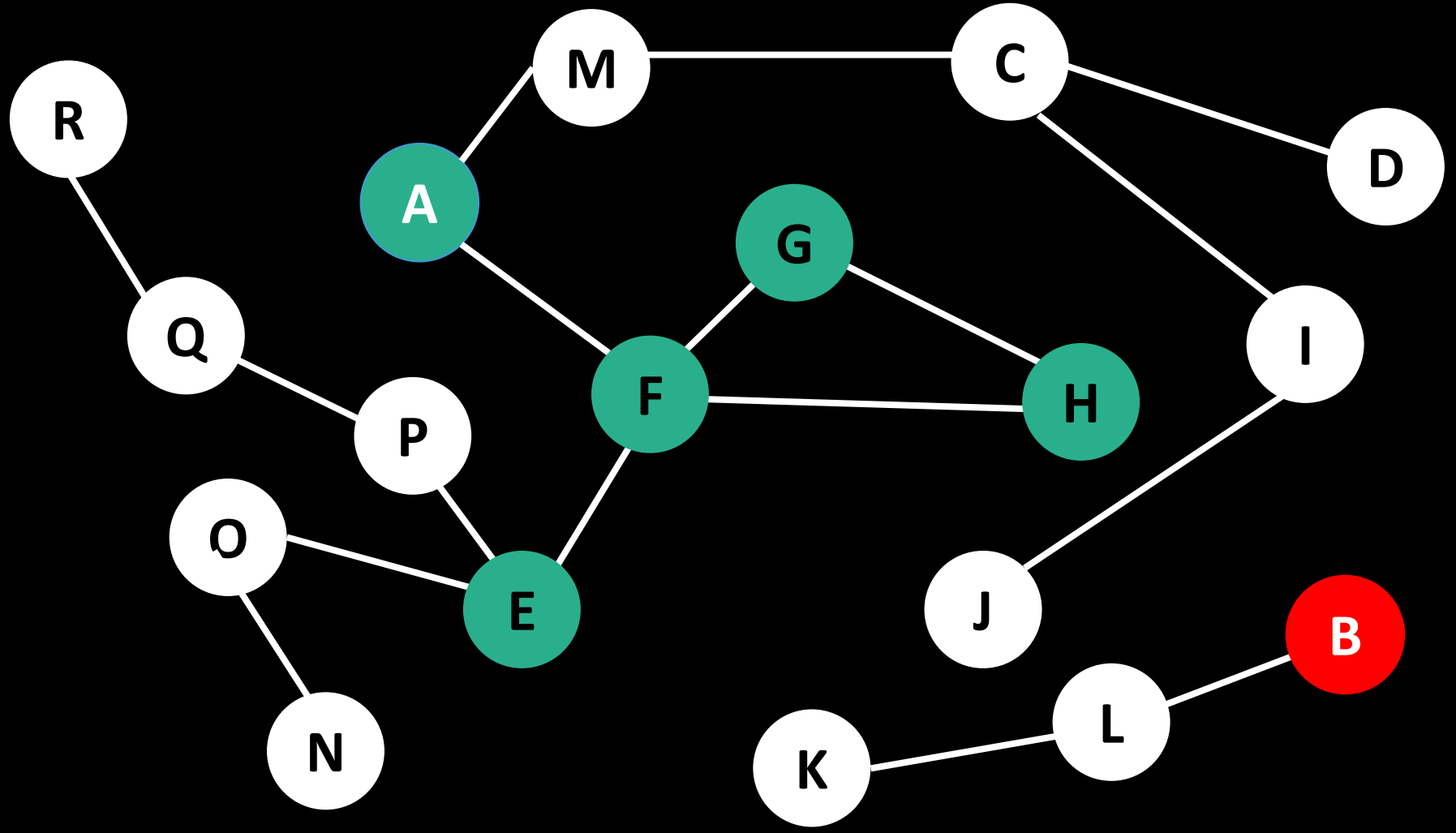
Another DFS Example



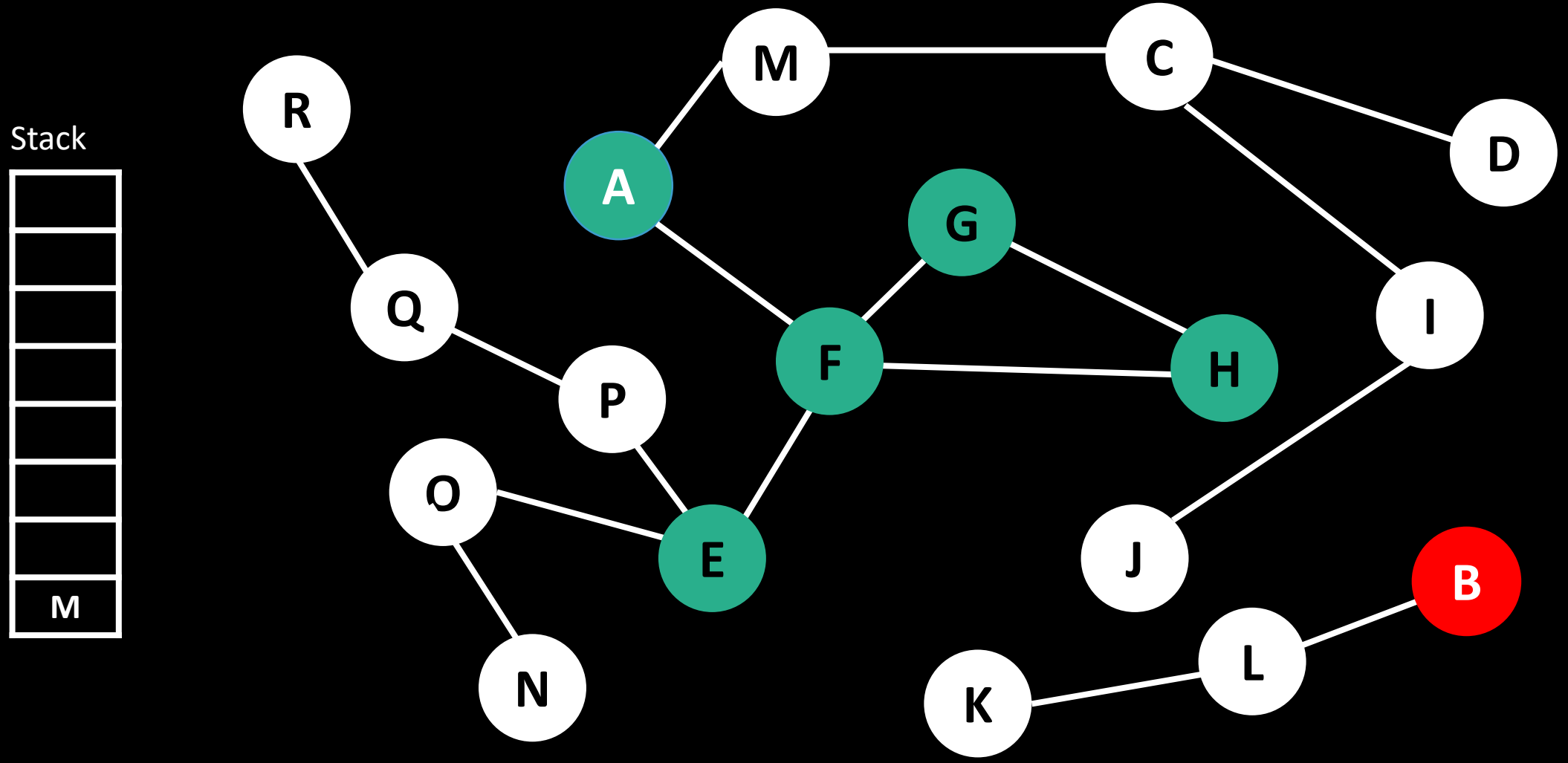


Another DFS Example

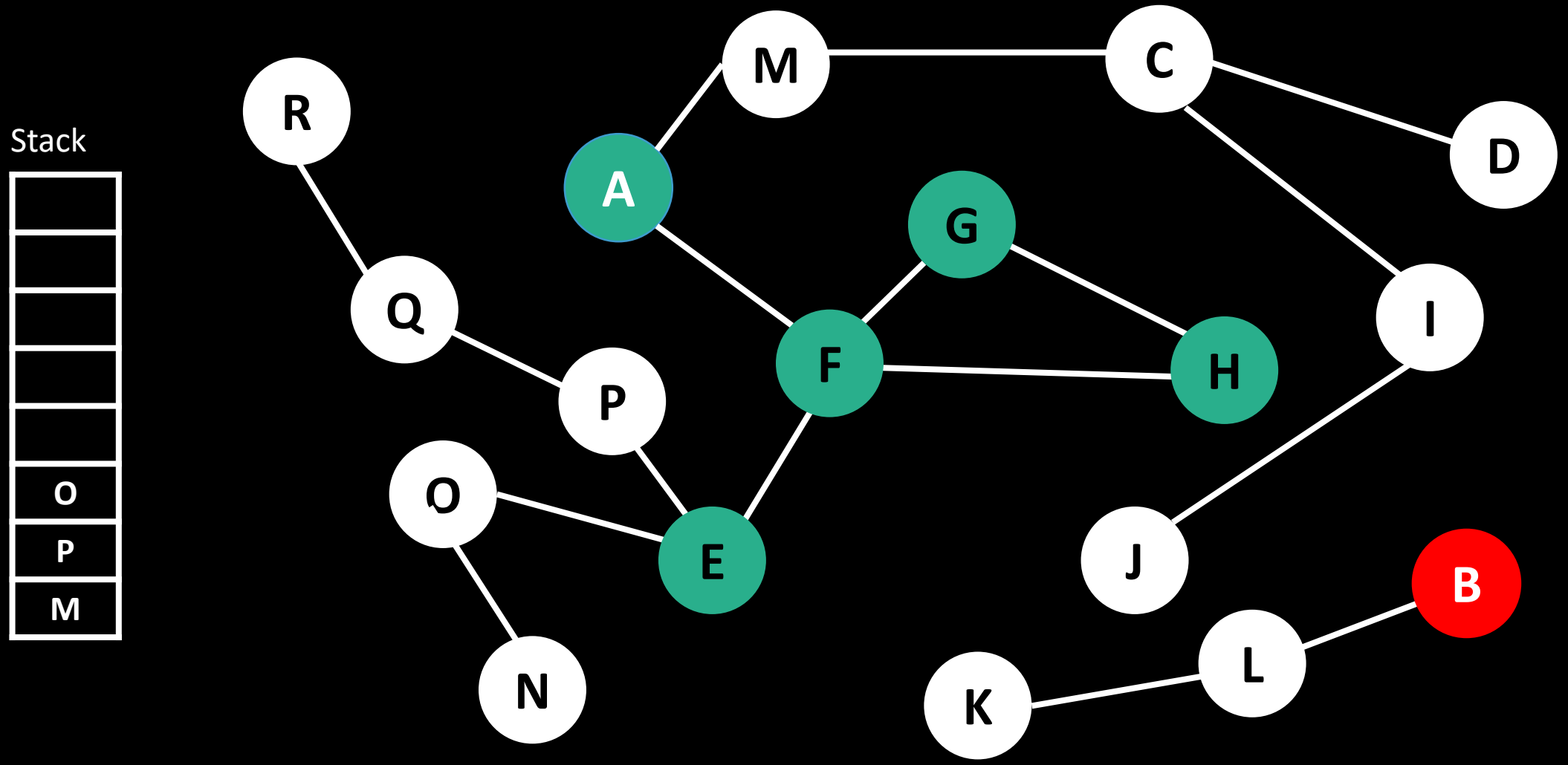




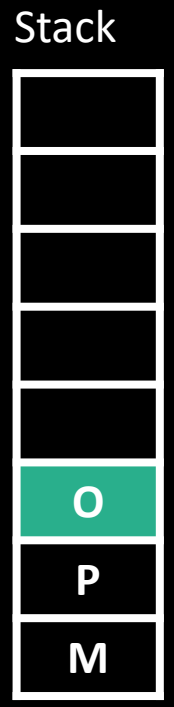
Another DFS Example

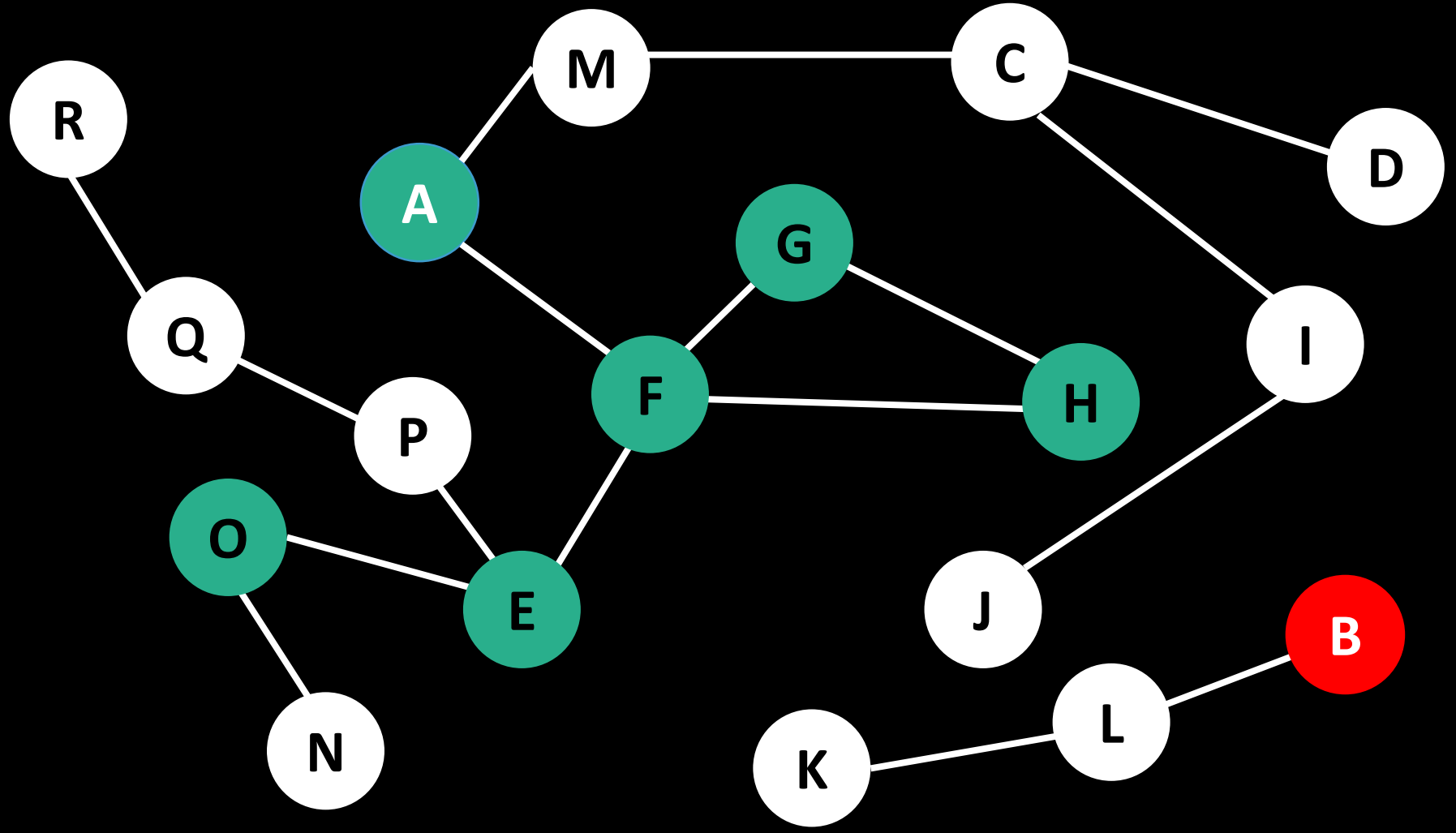


Another DFS Example

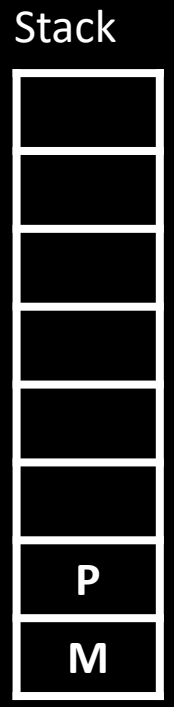


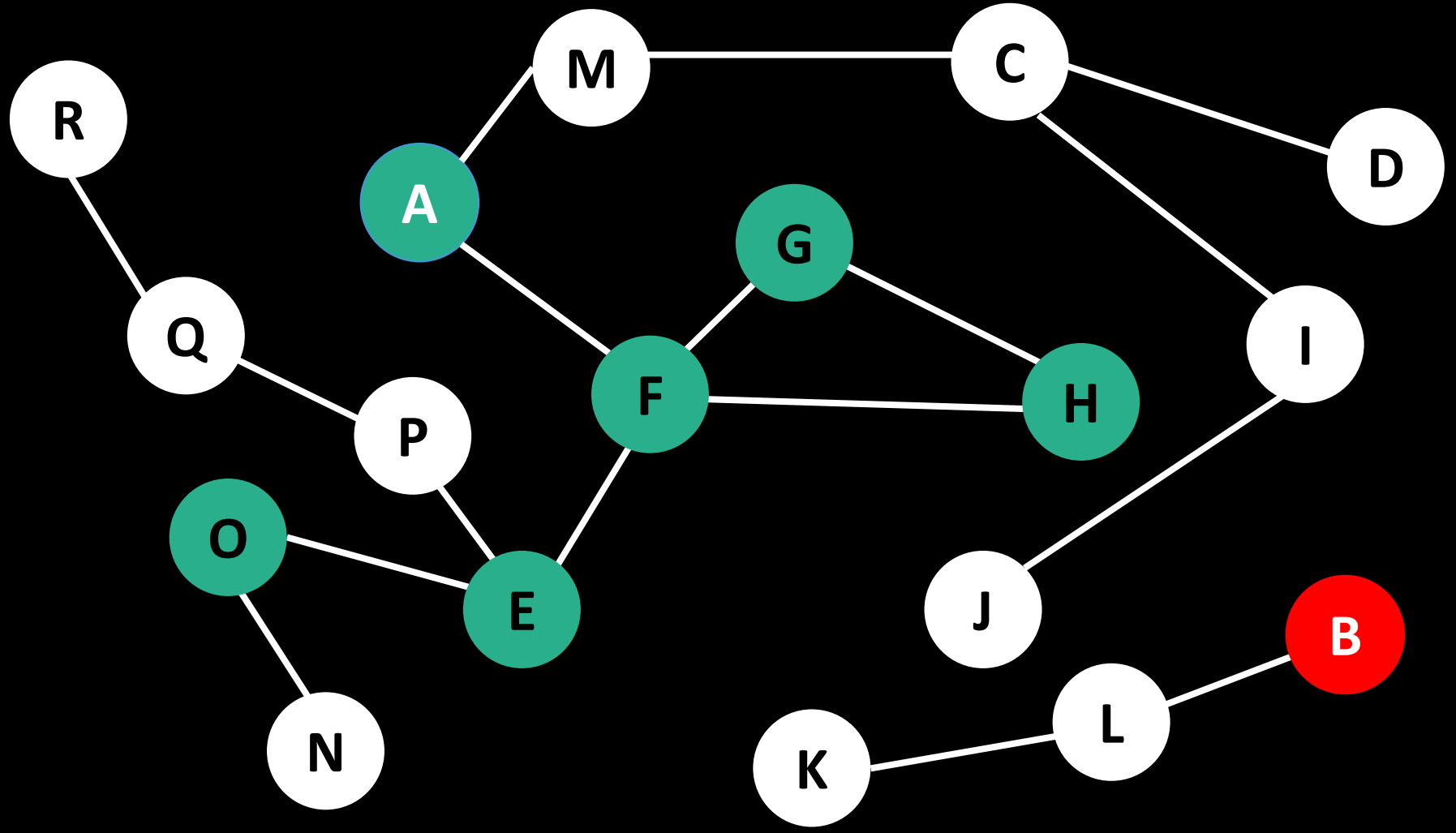
Another DFS Example



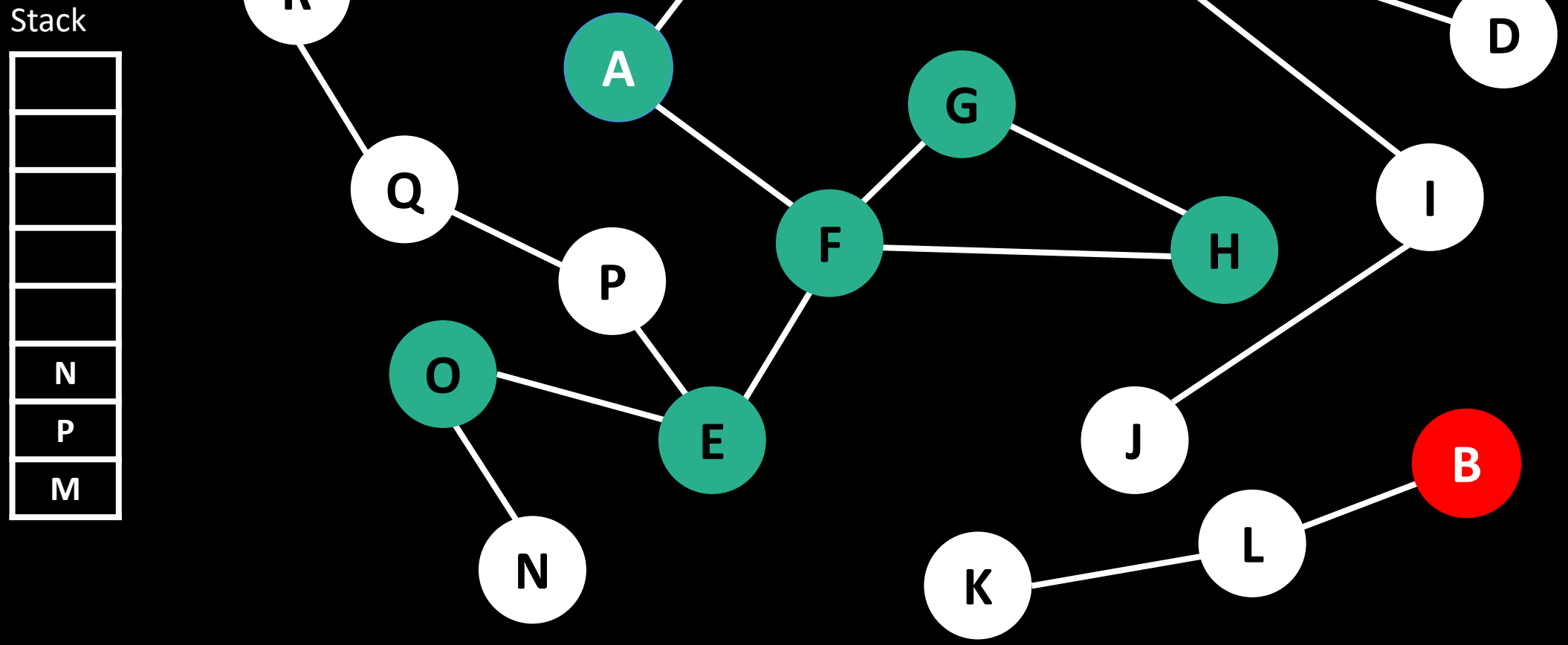


Another DFS Example

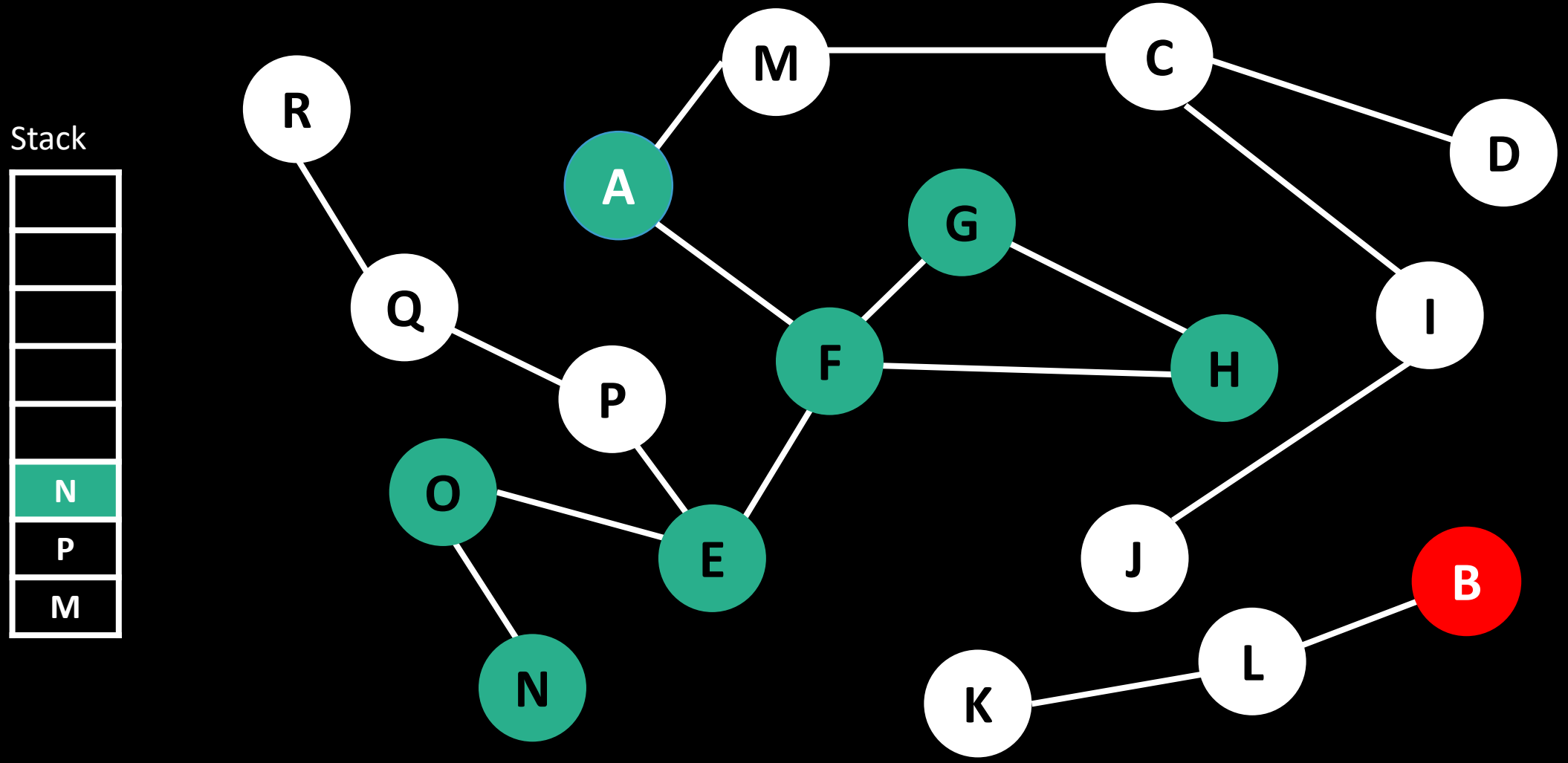




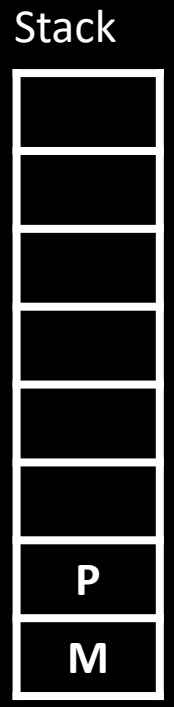
Another DFS Example

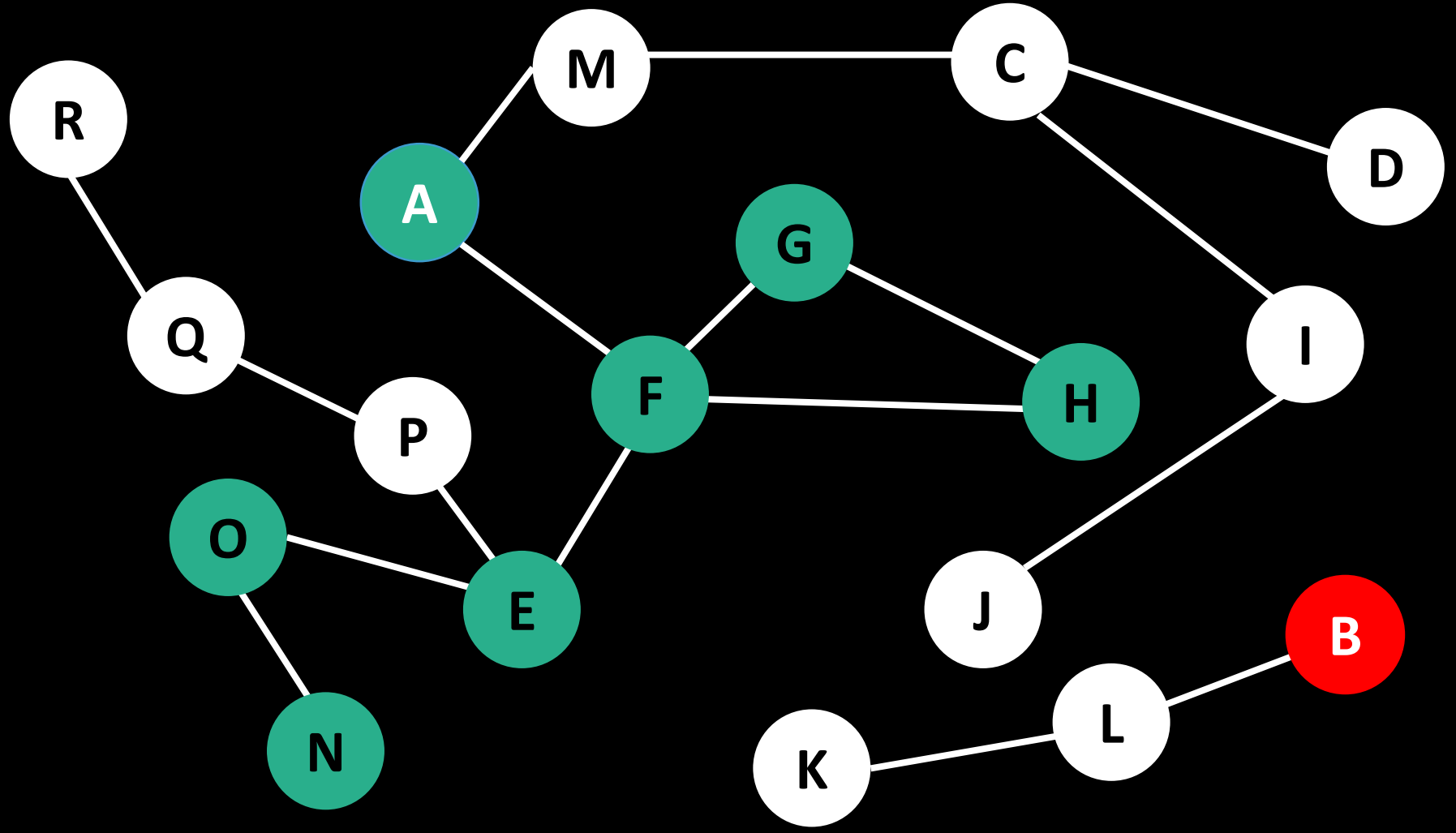


Another DFS Example

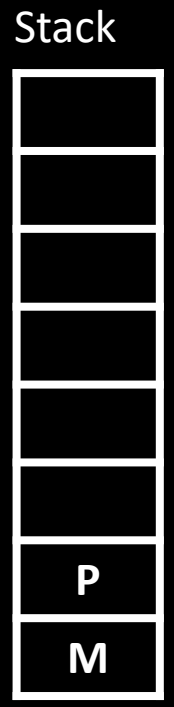


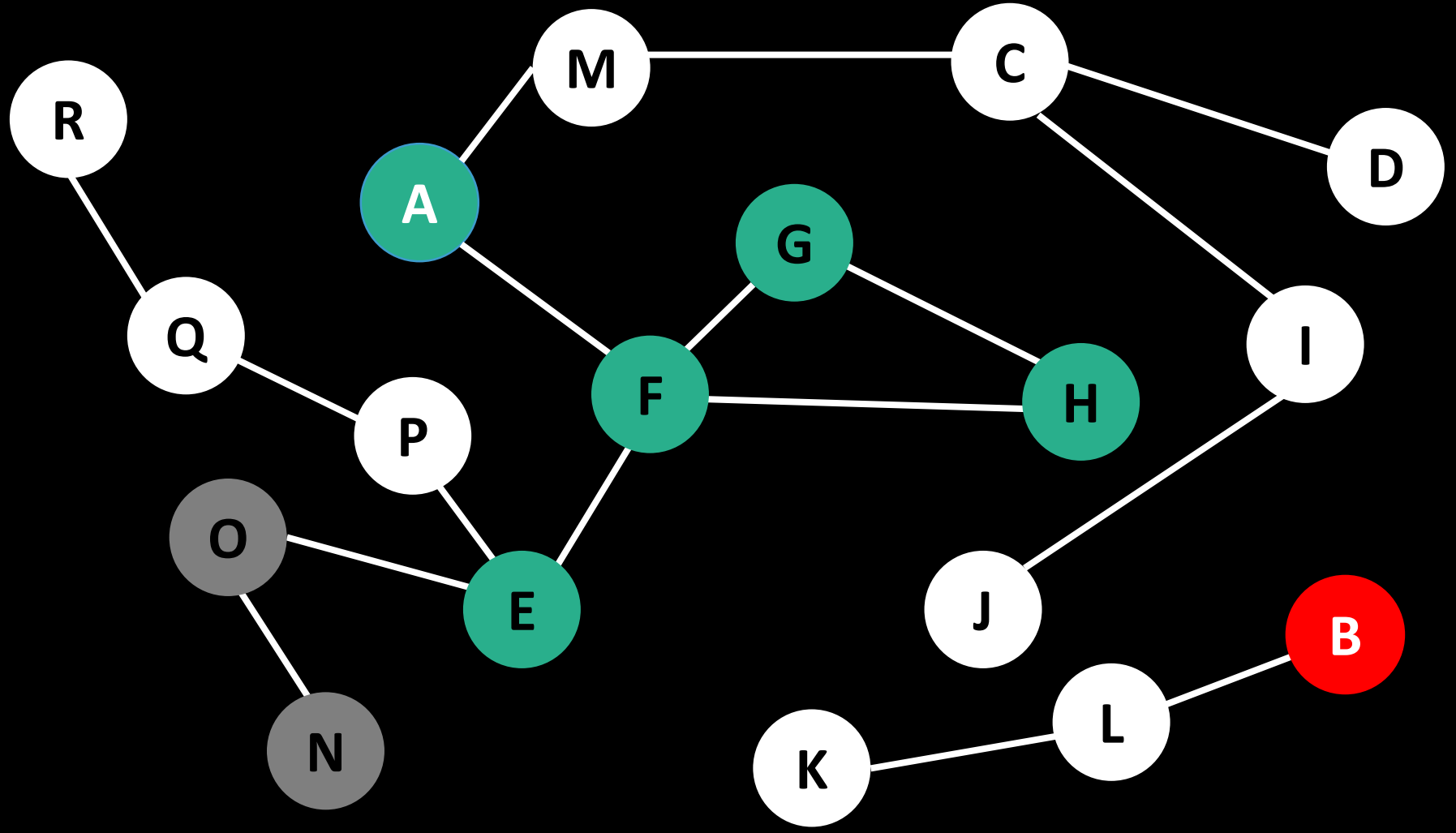
Another DFS Example



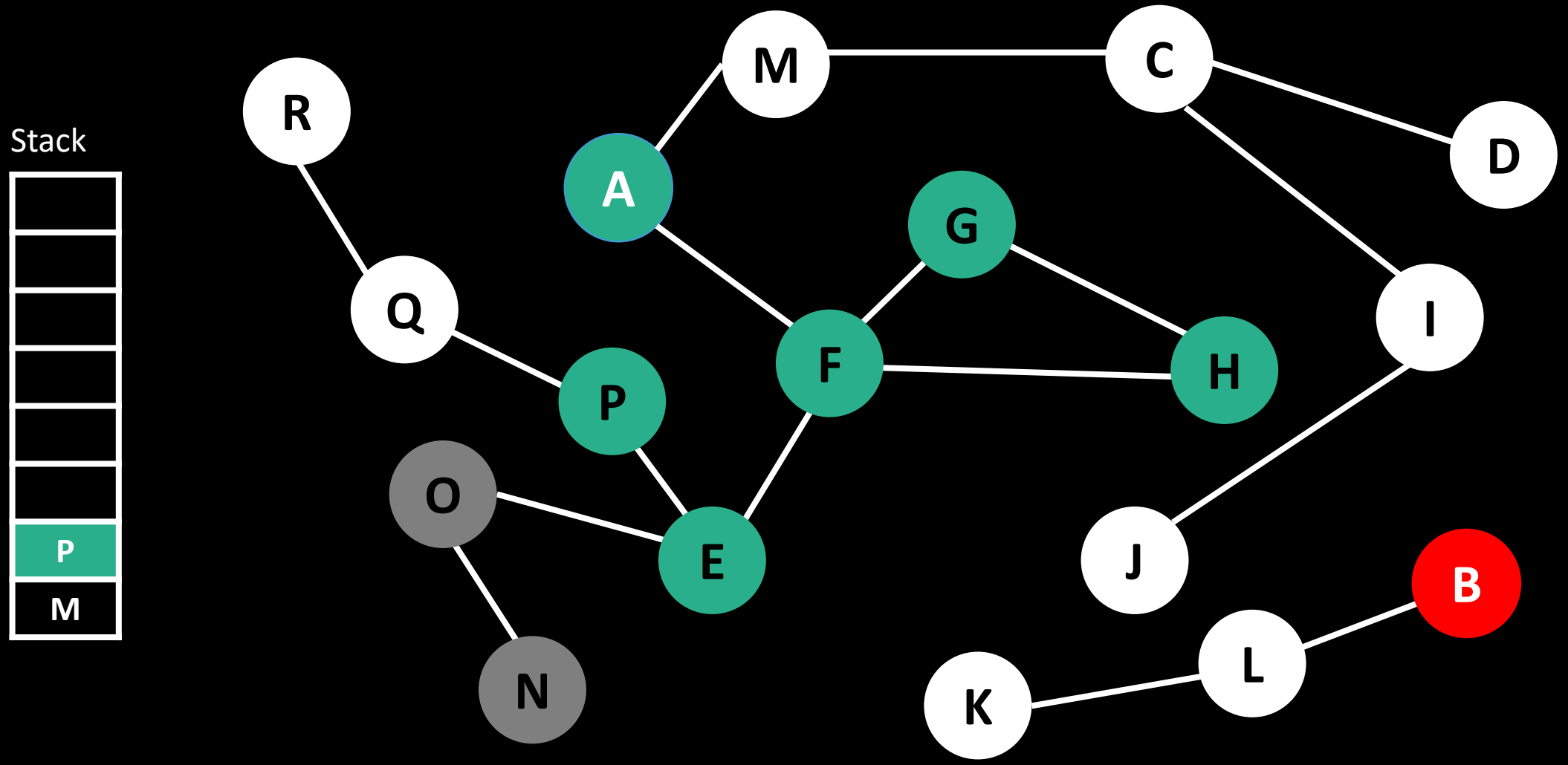


Another DFS Example

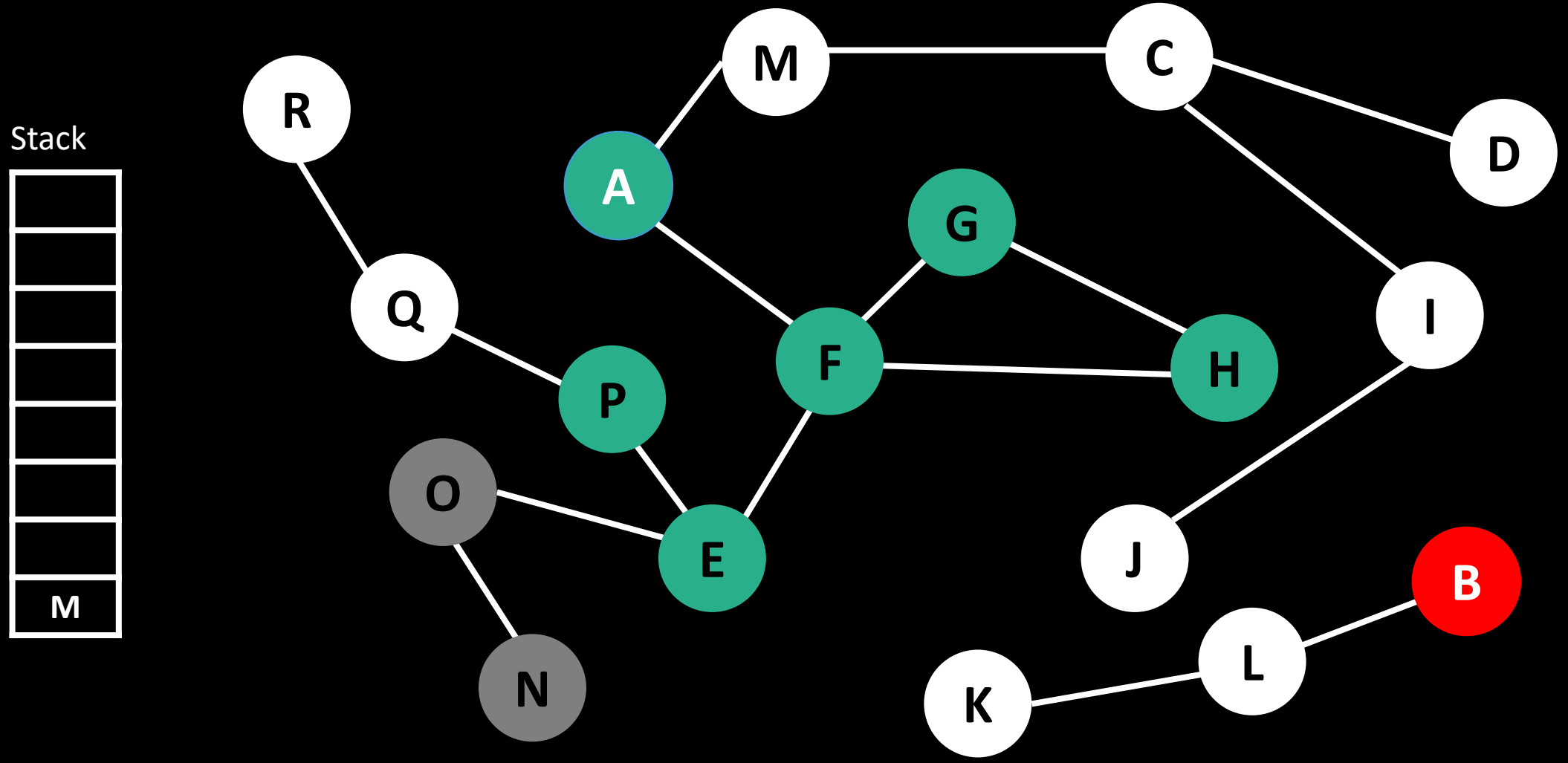




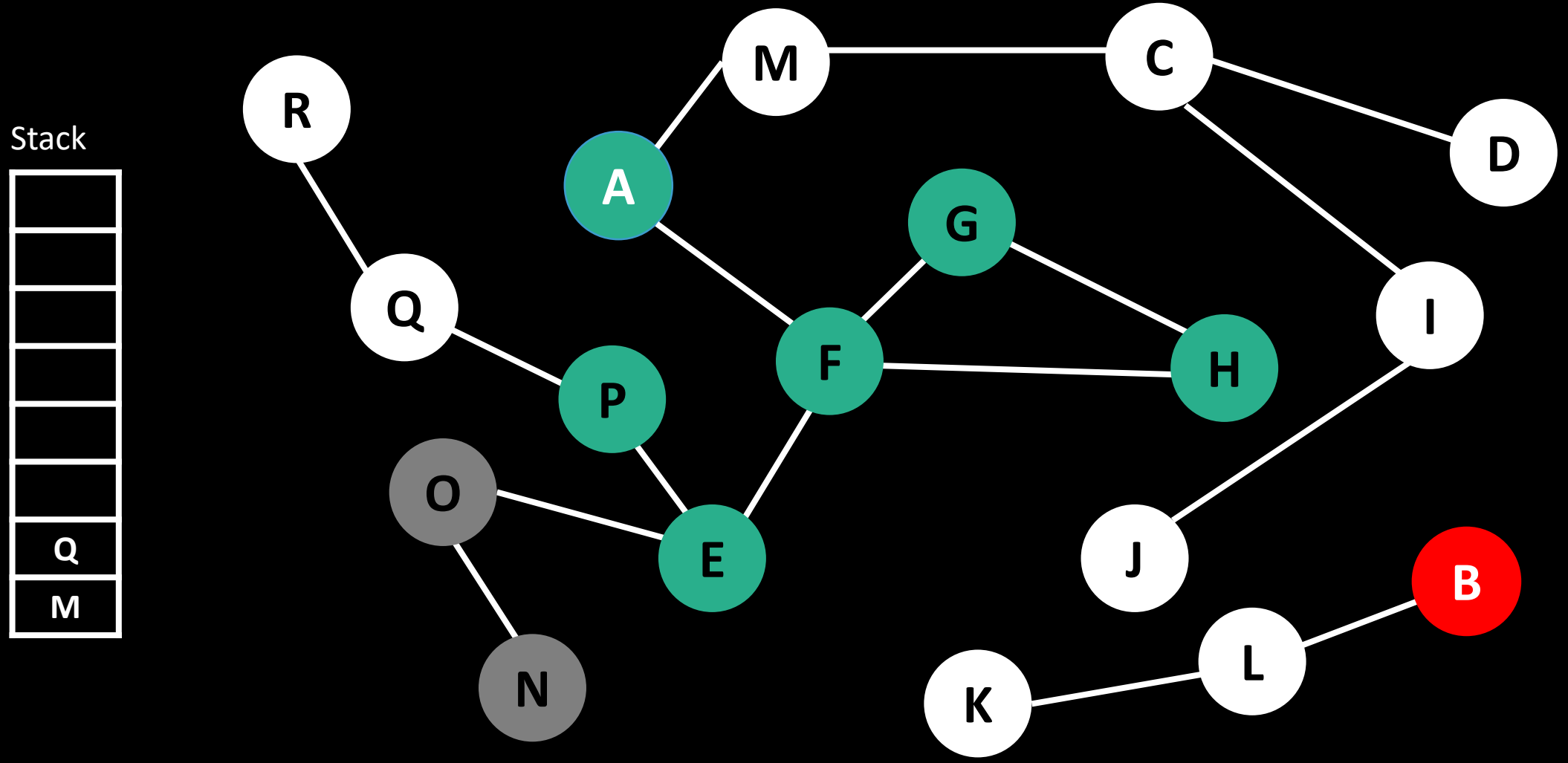
Another DFS Example



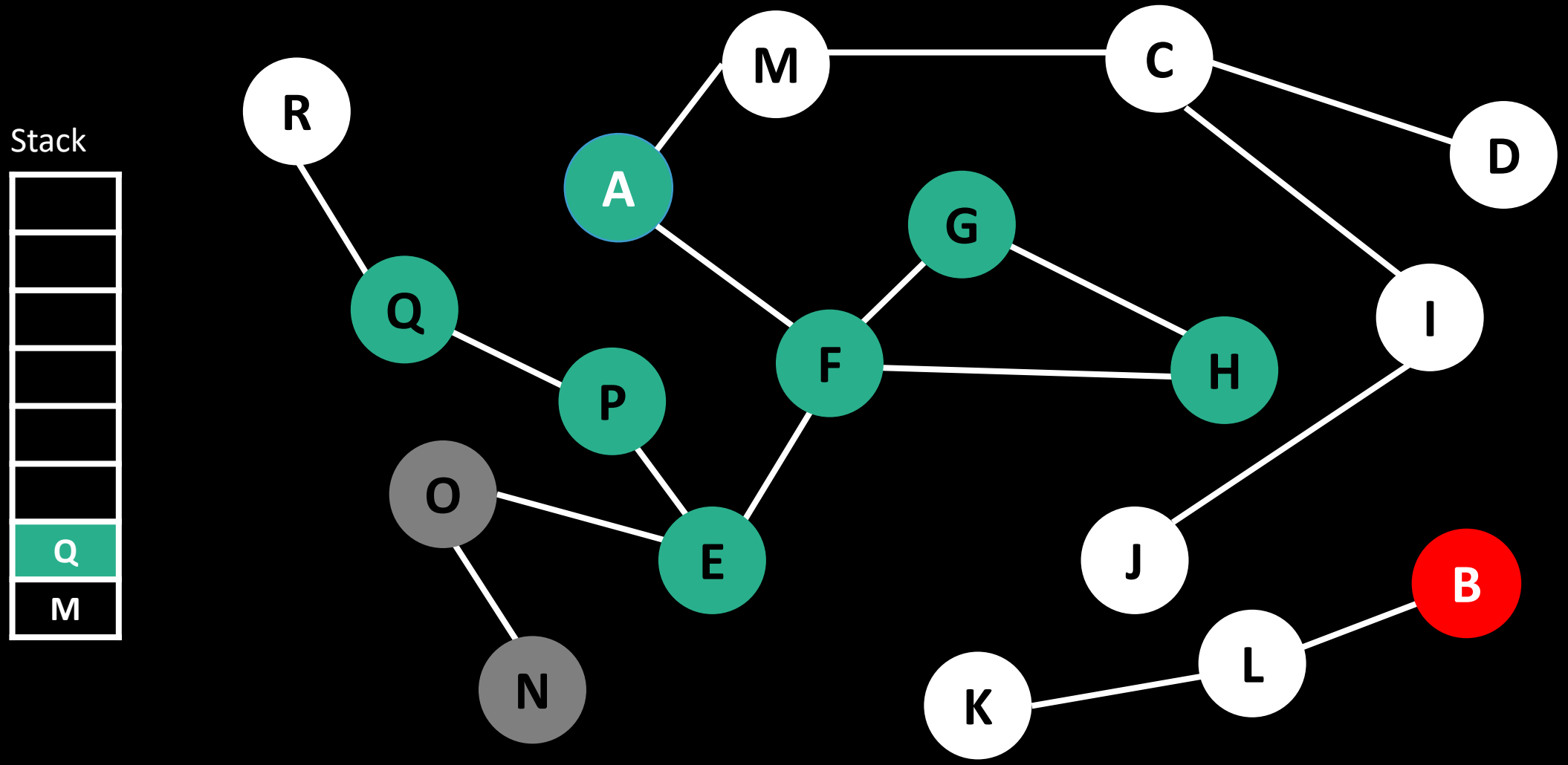
Another DFS Example



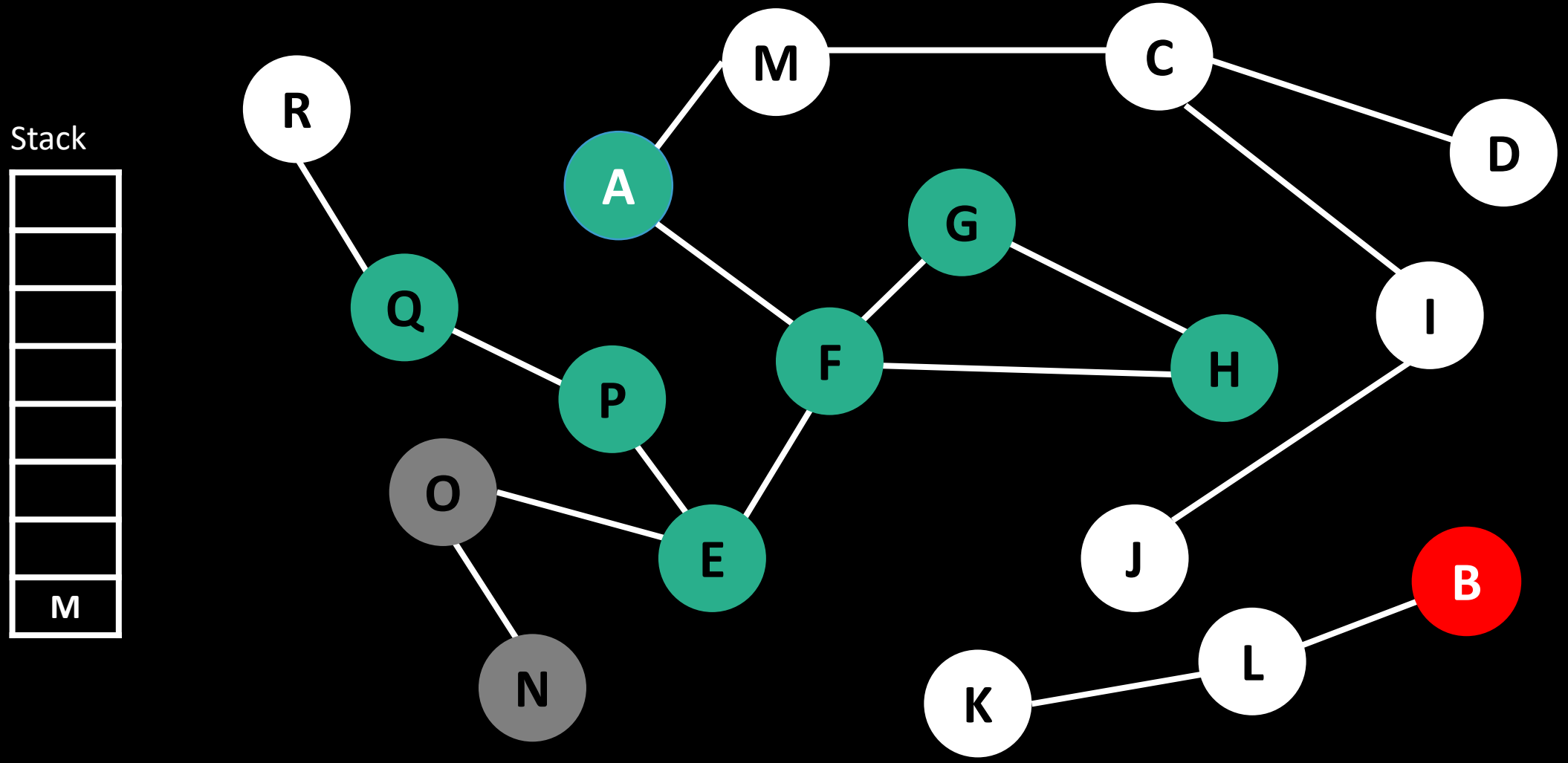
Another DFS Example



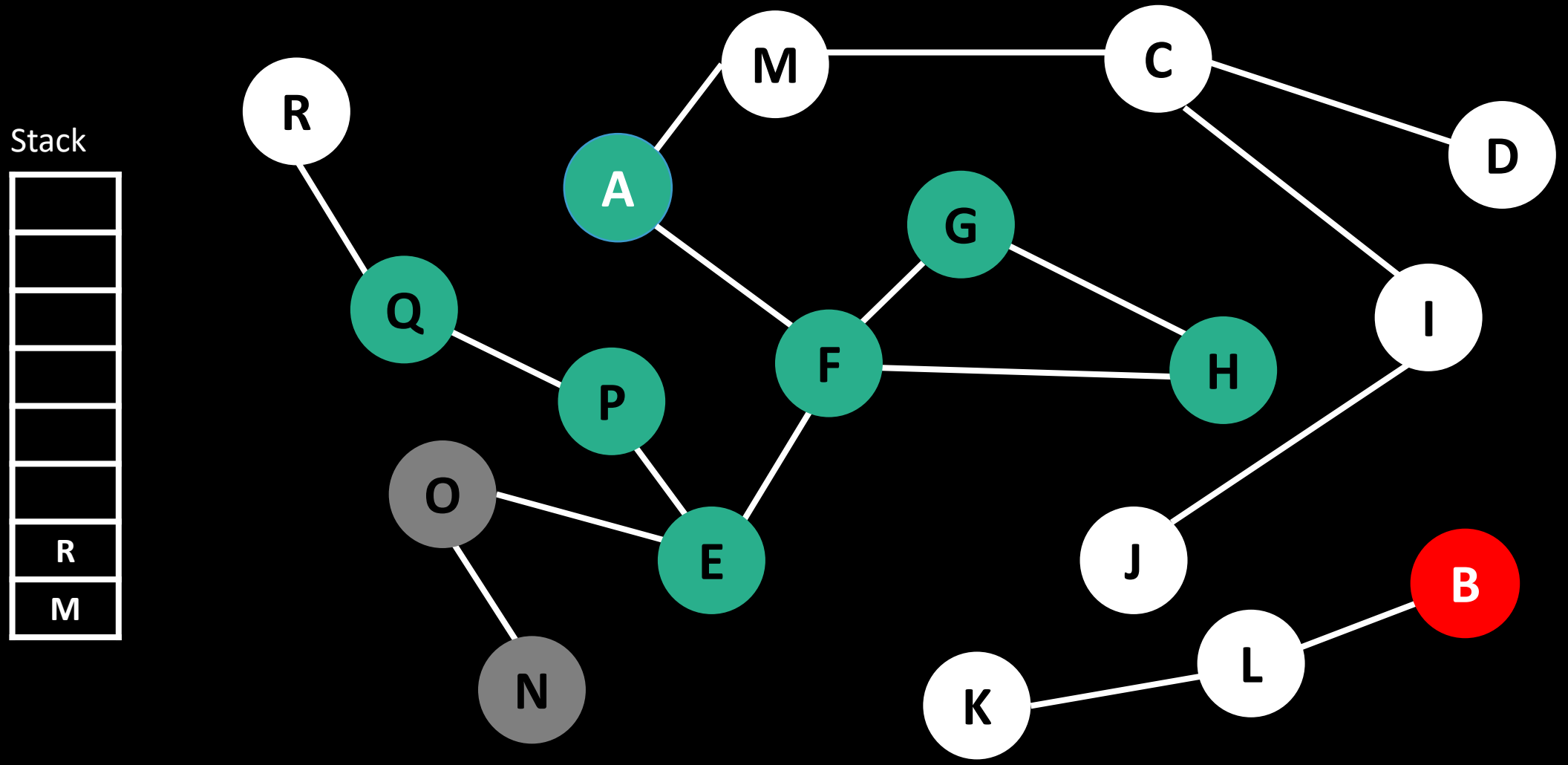
Another DFS Example



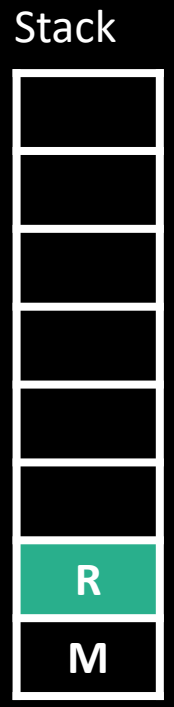
Another DFS Example

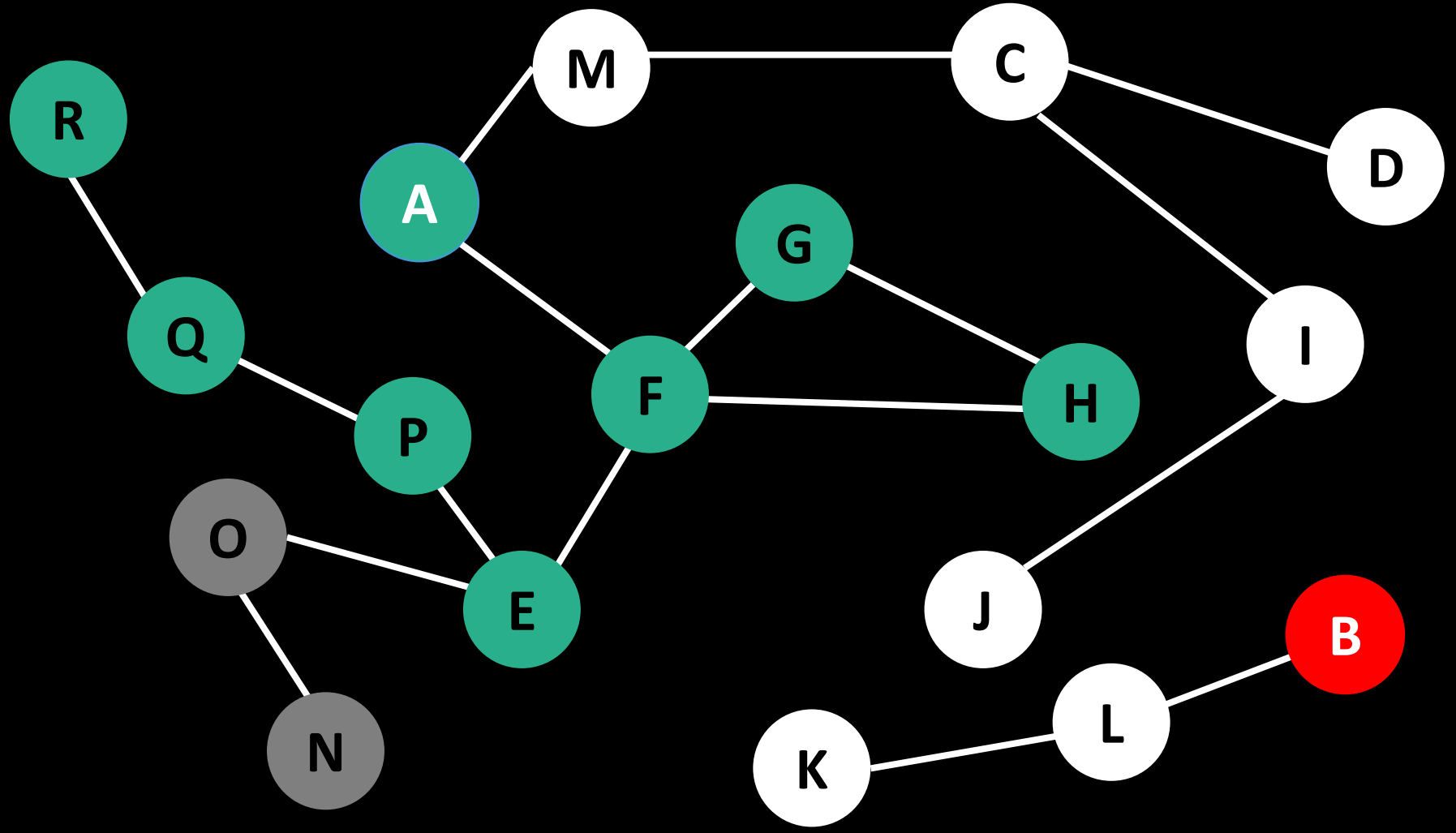


Another DFS Example

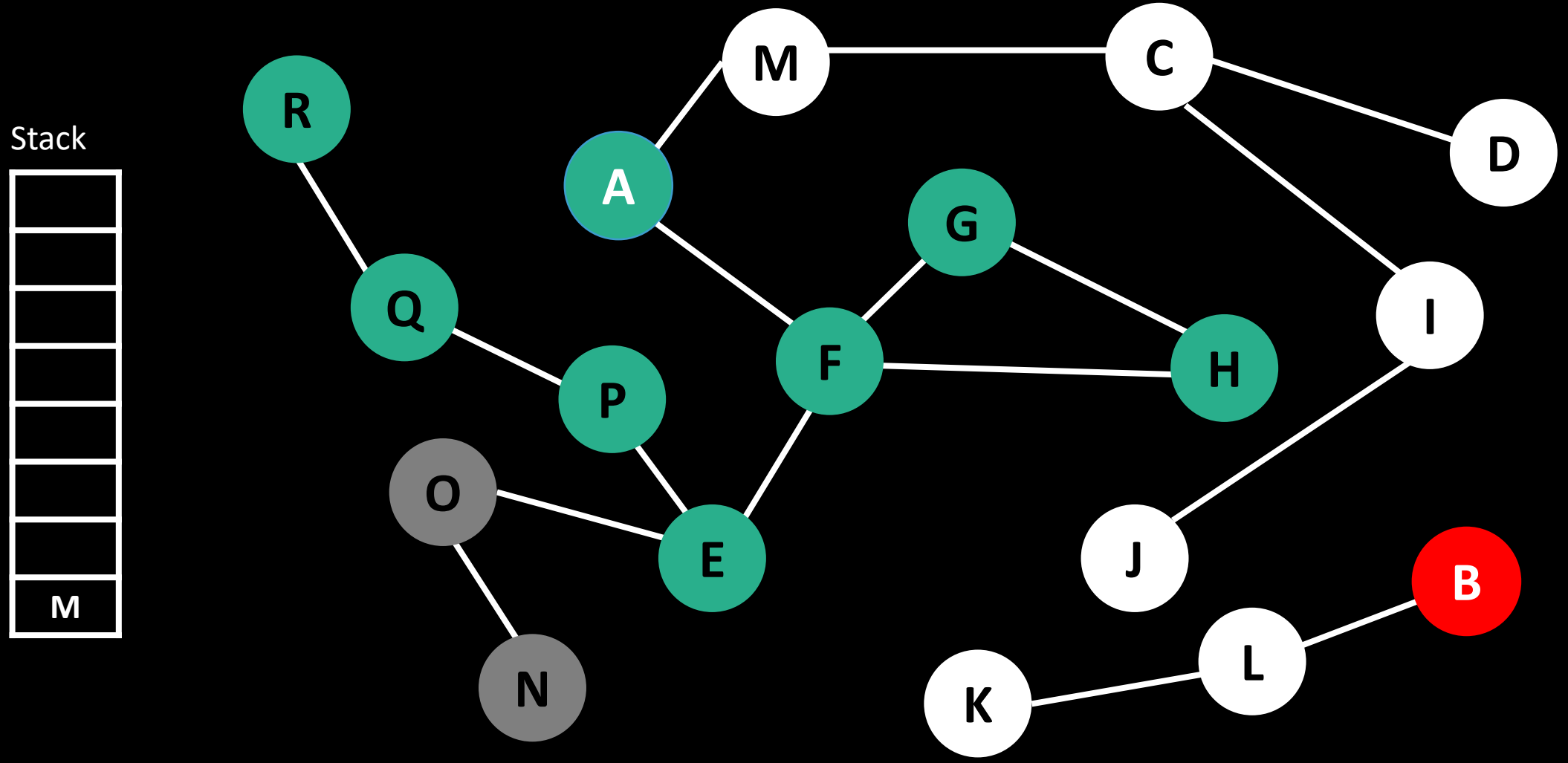


Another DFS Example

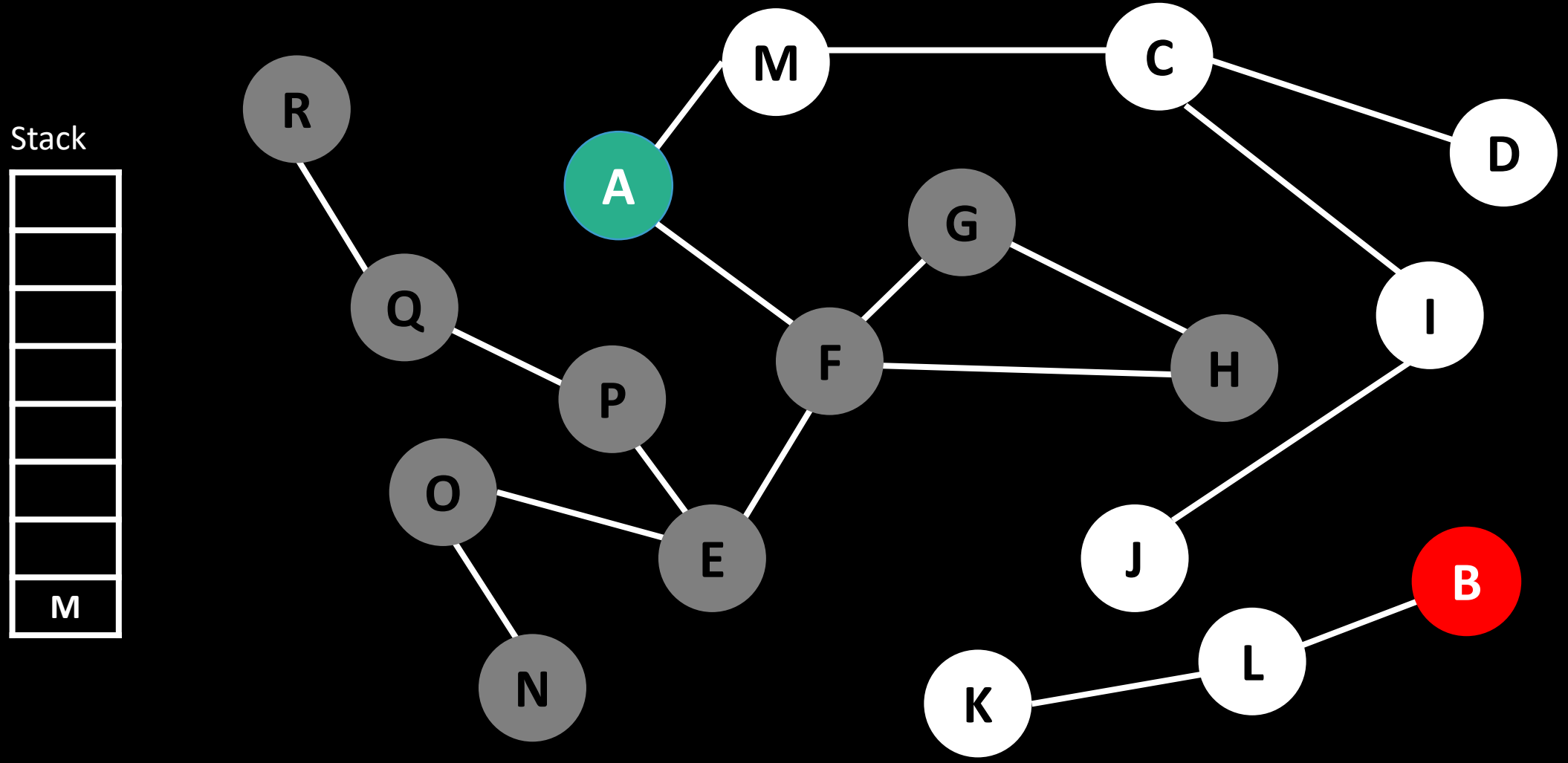




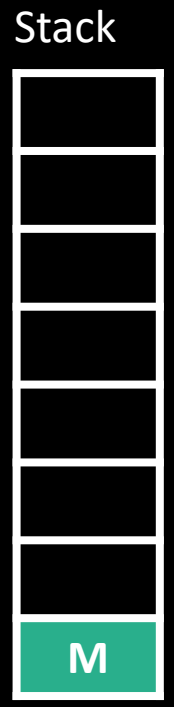
Another DFS Example

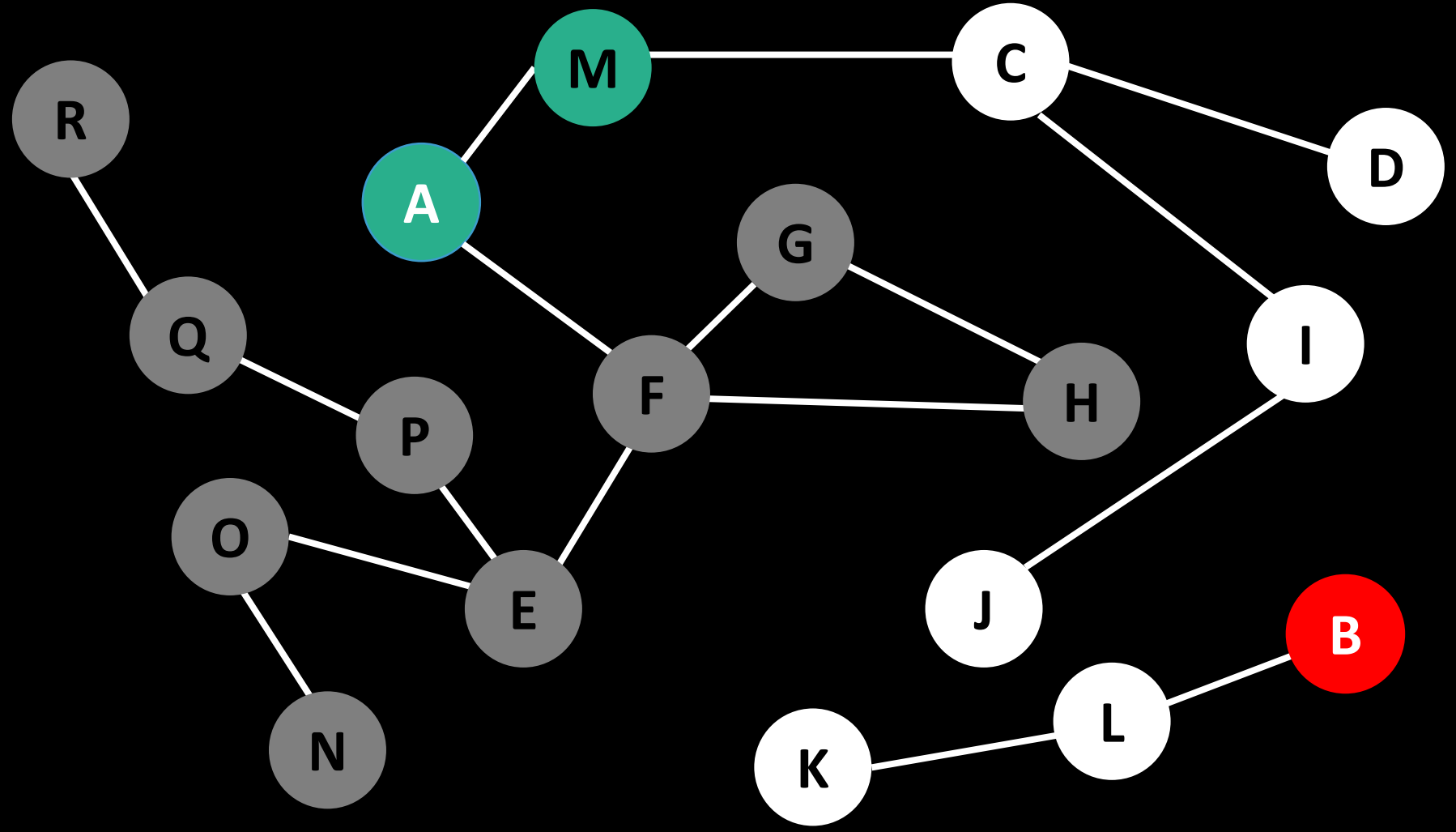


Another DFS Example

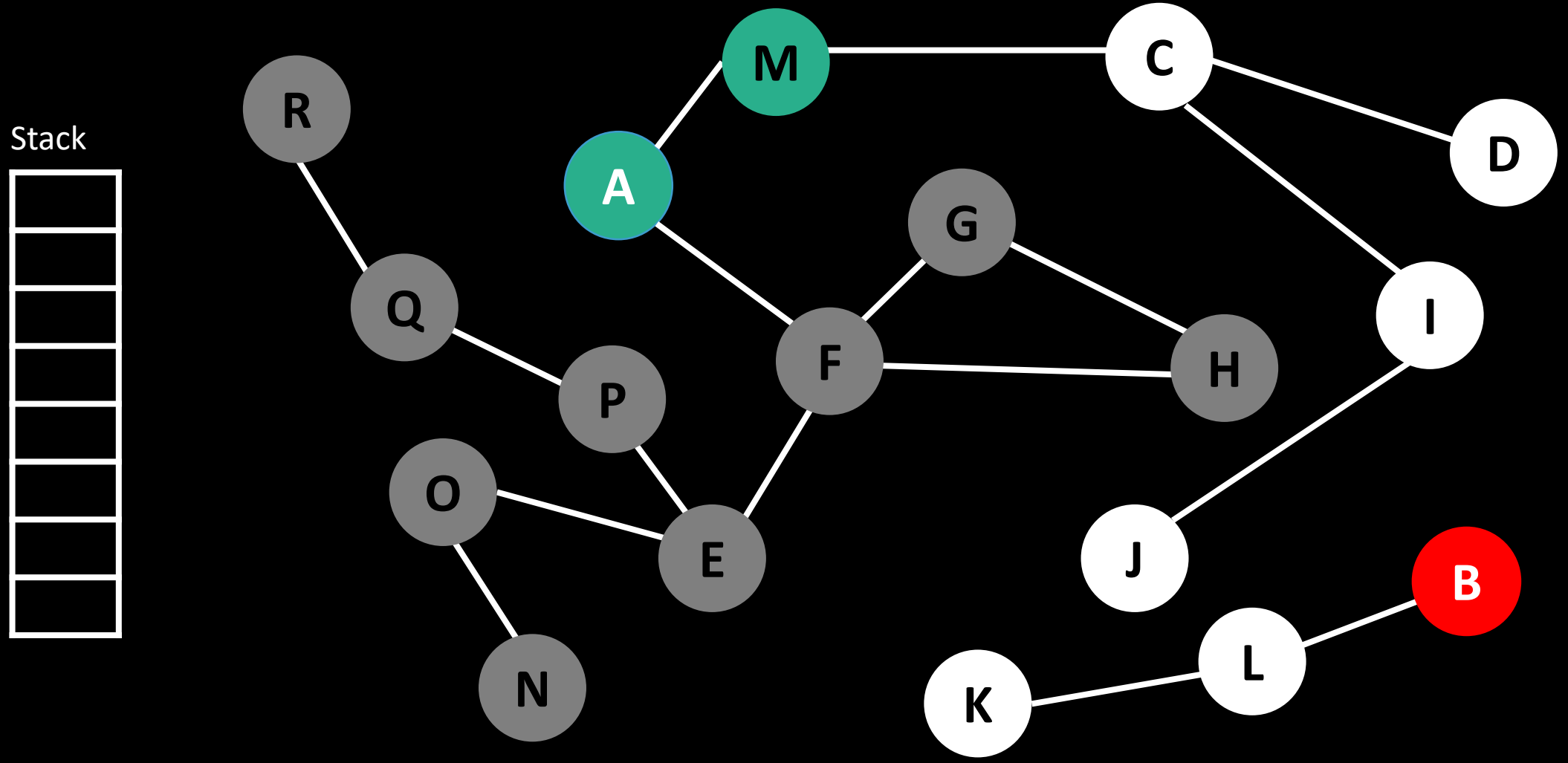


Another DFS Example

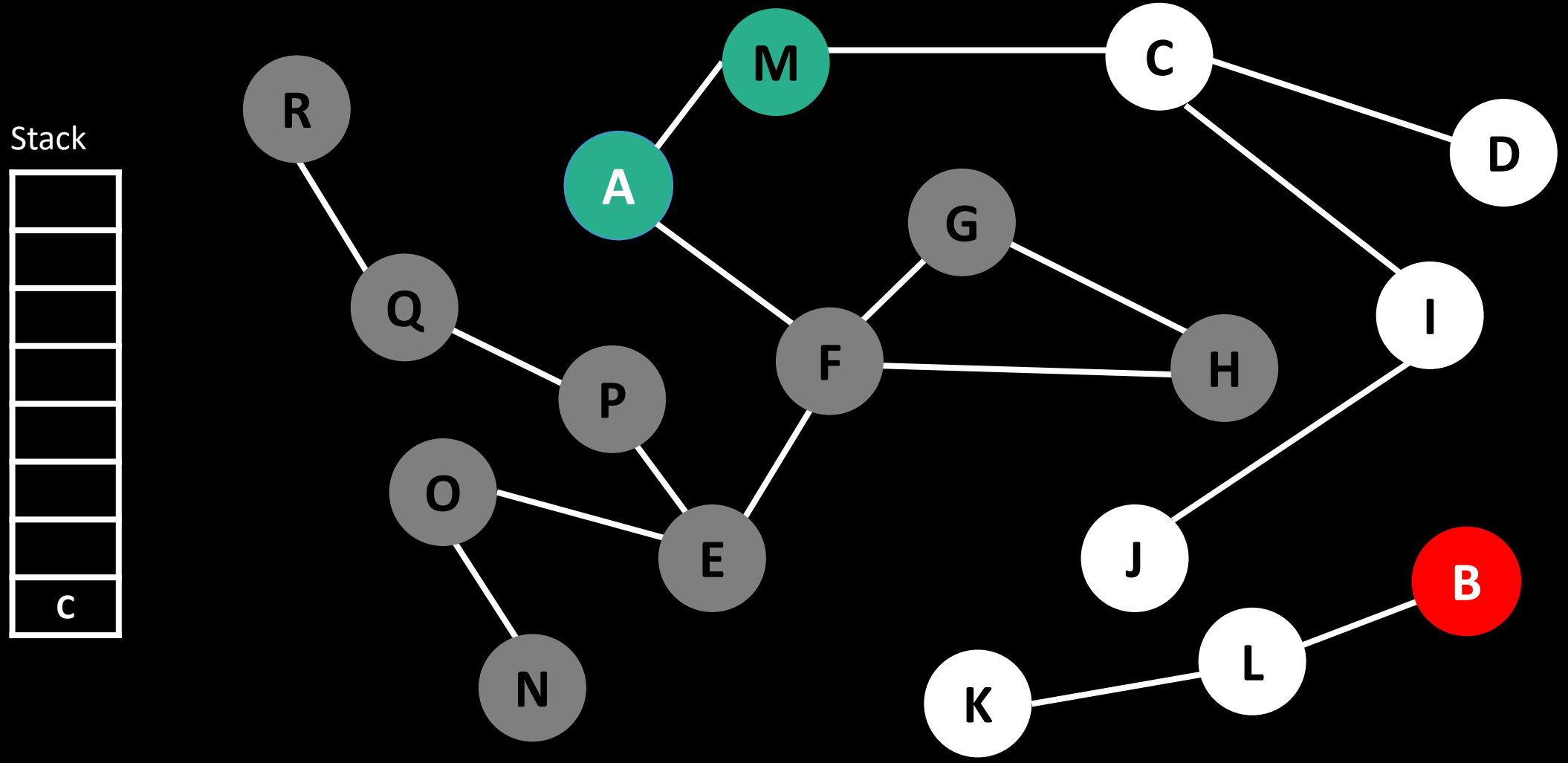




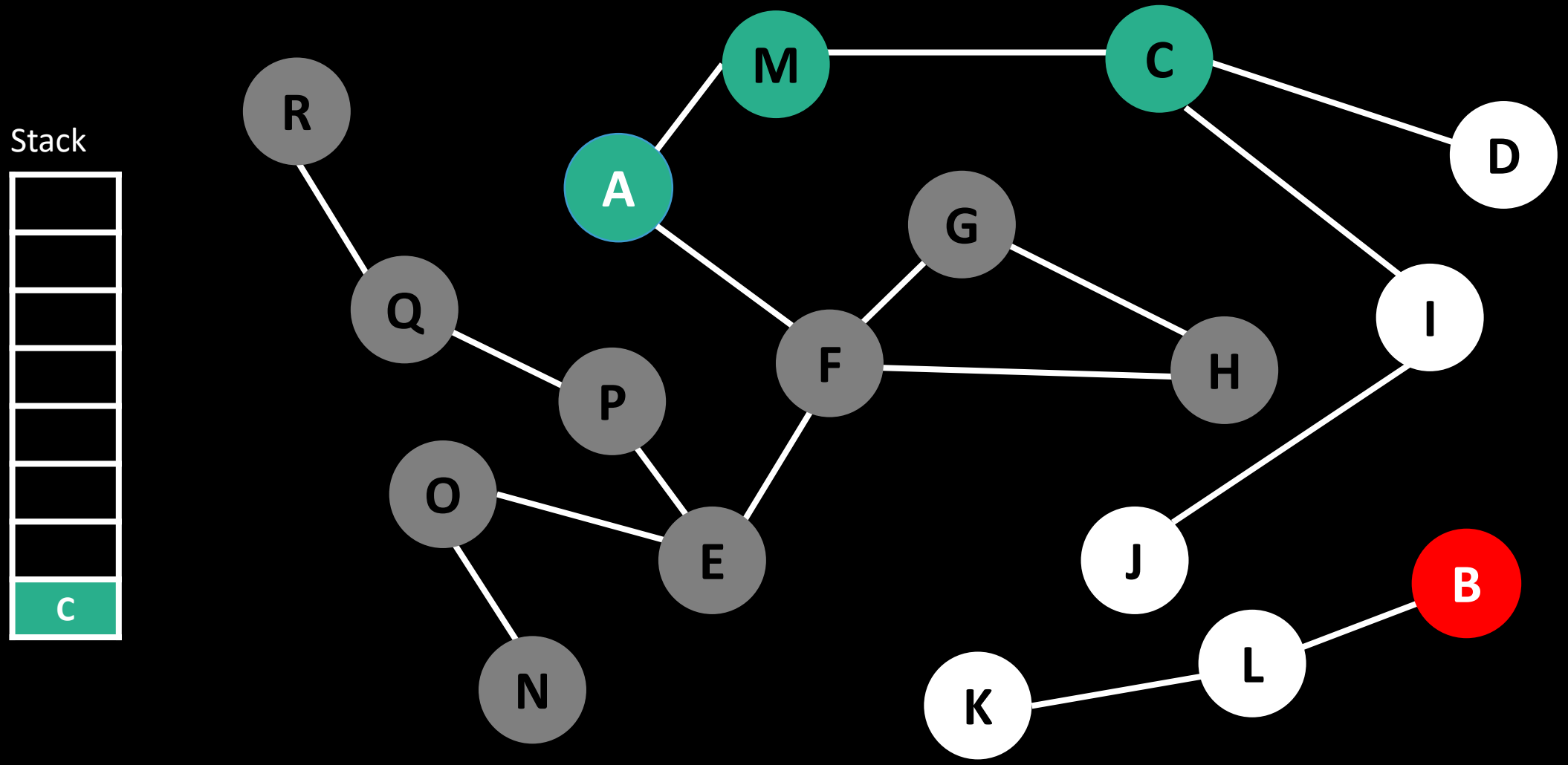
Another DFS Example



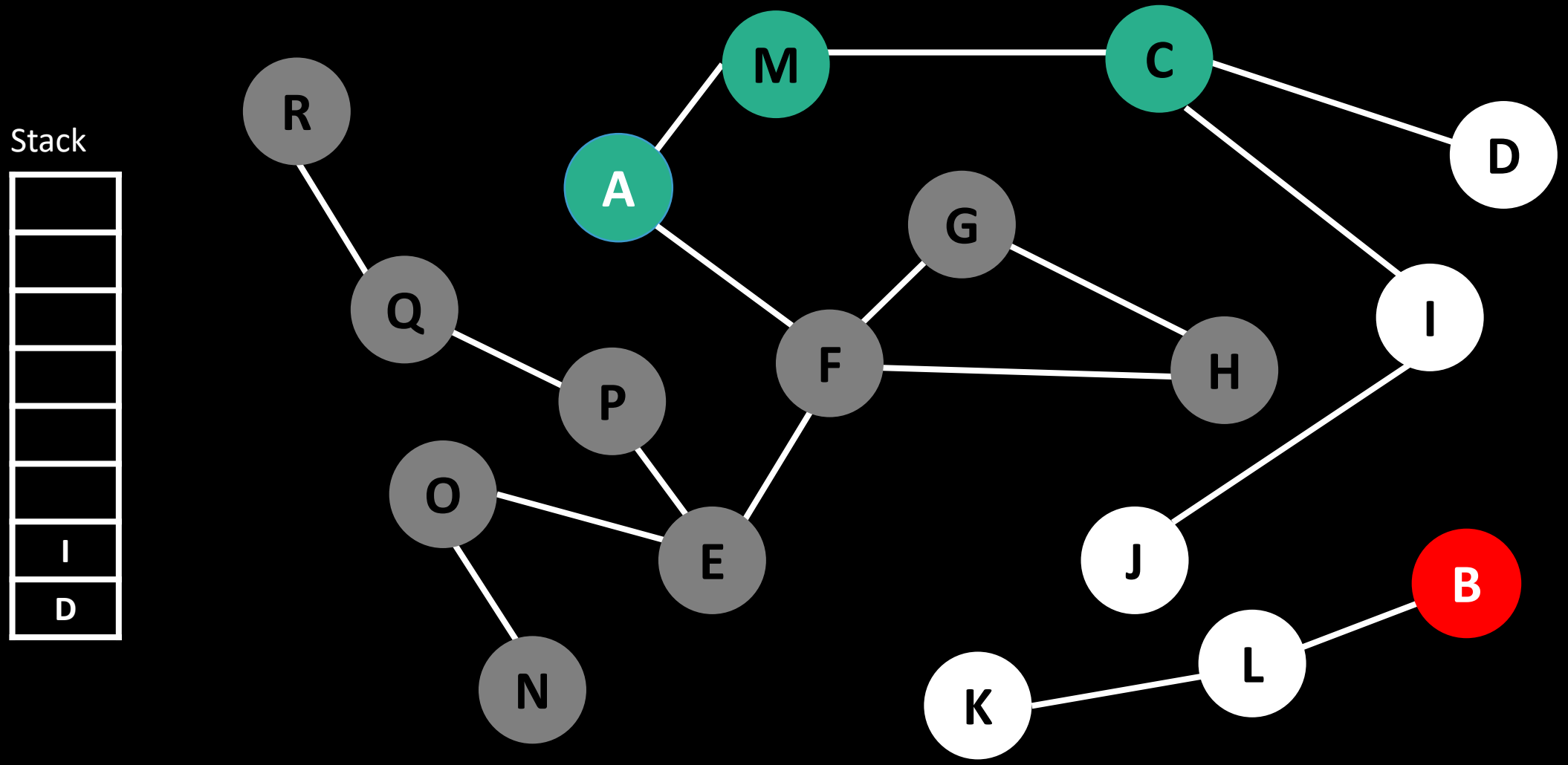
Another DFS Example



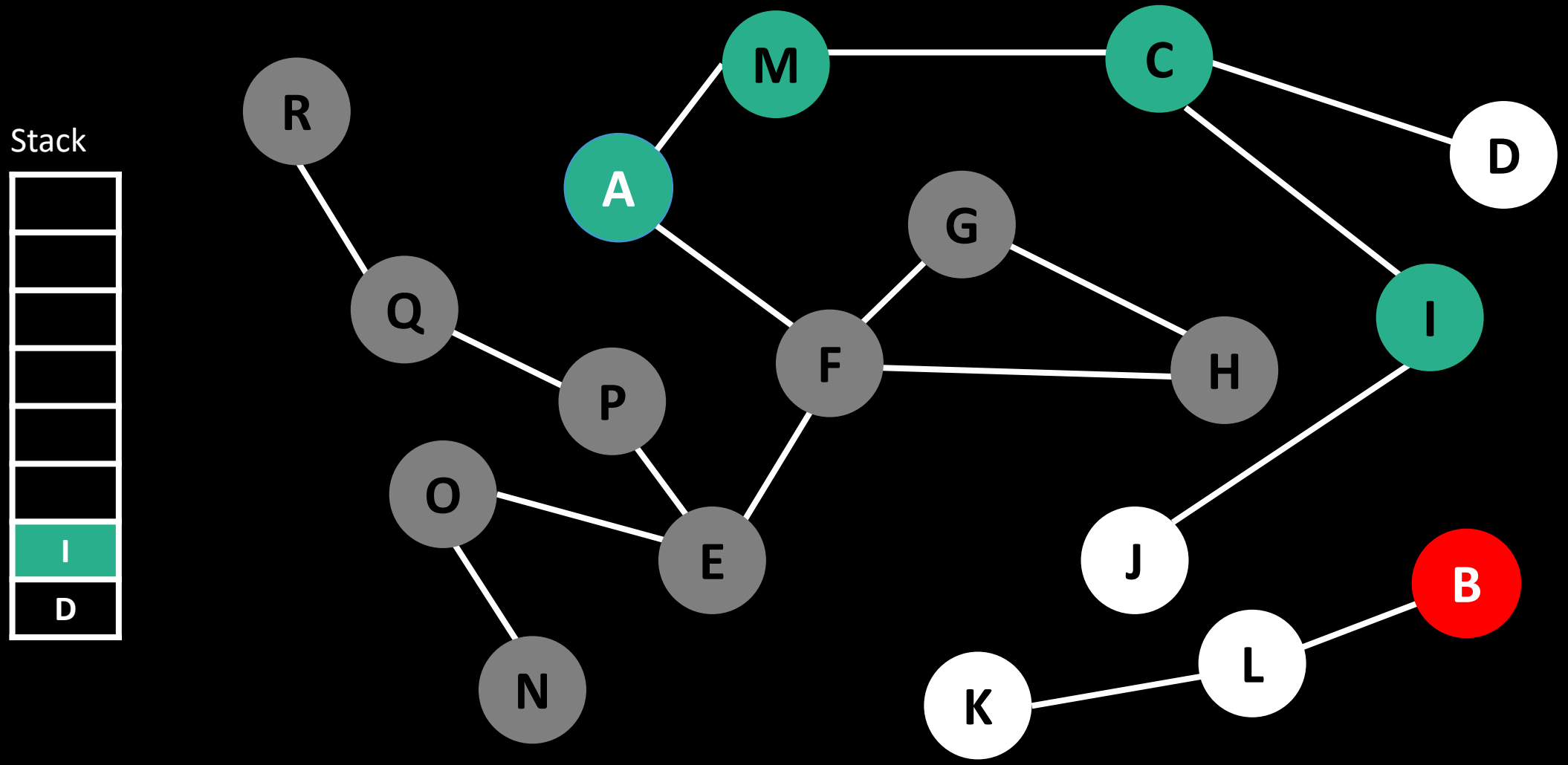
Another DFS Example



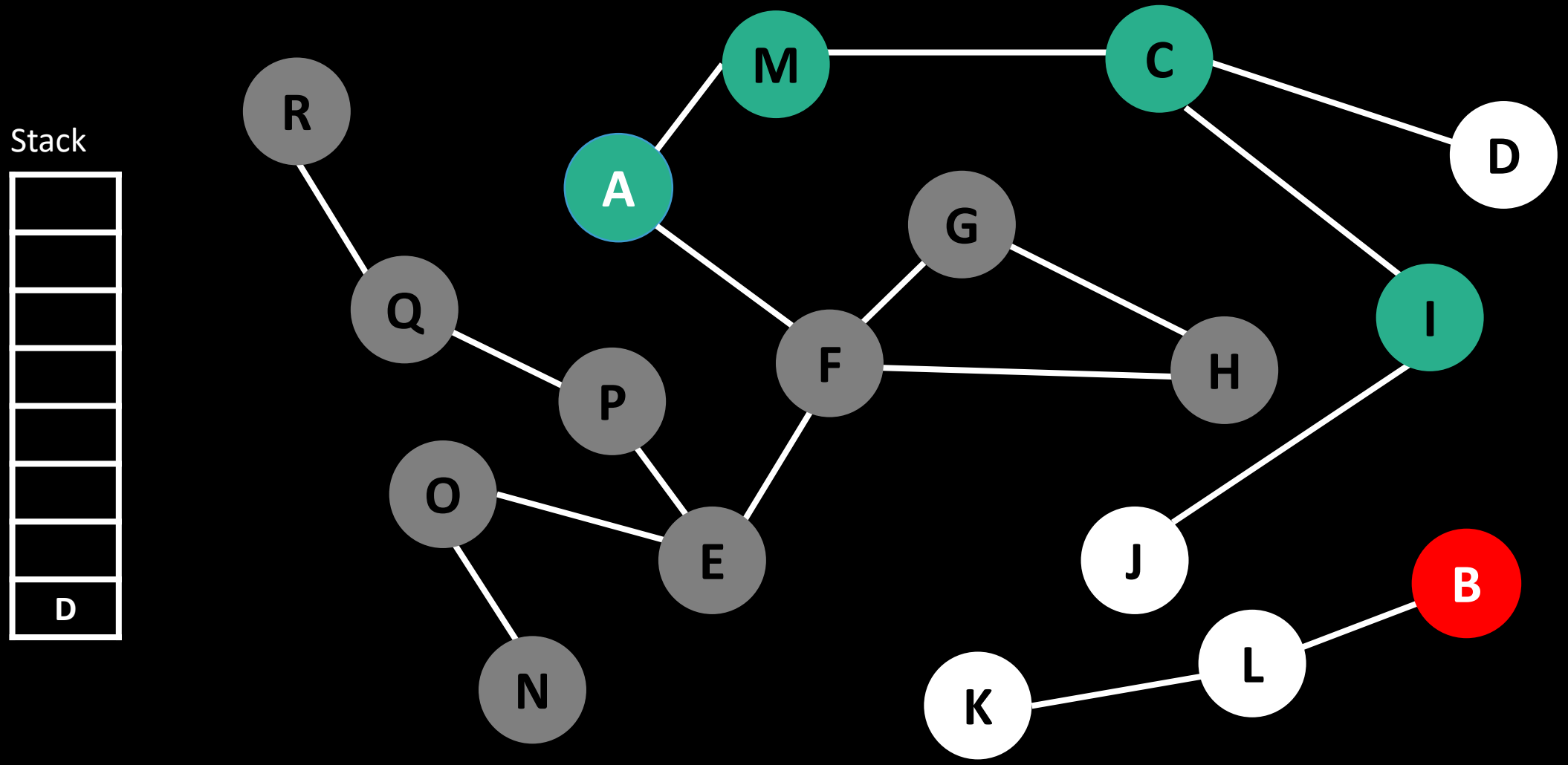
Another DFS Example



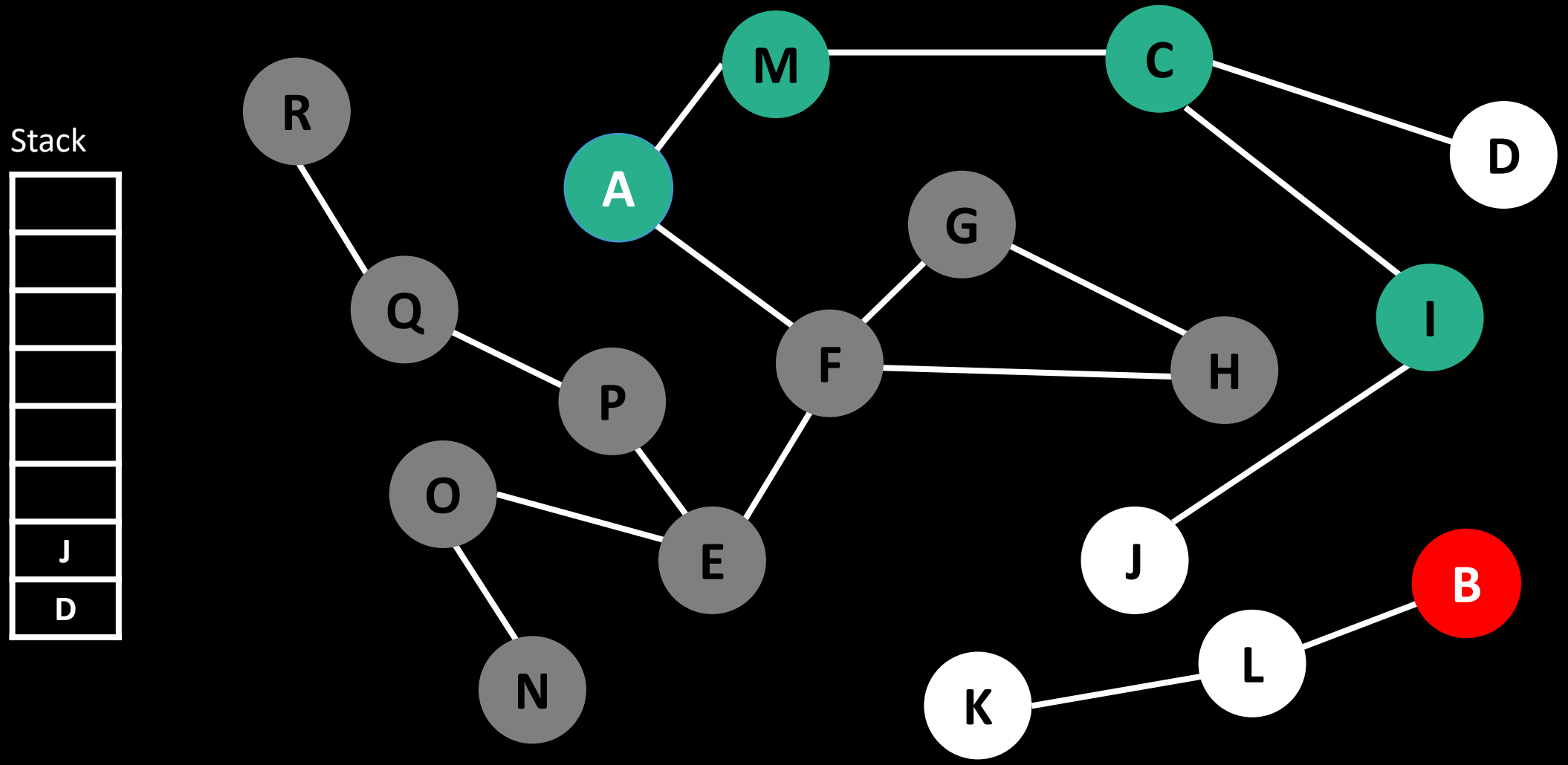
Another DFS Example



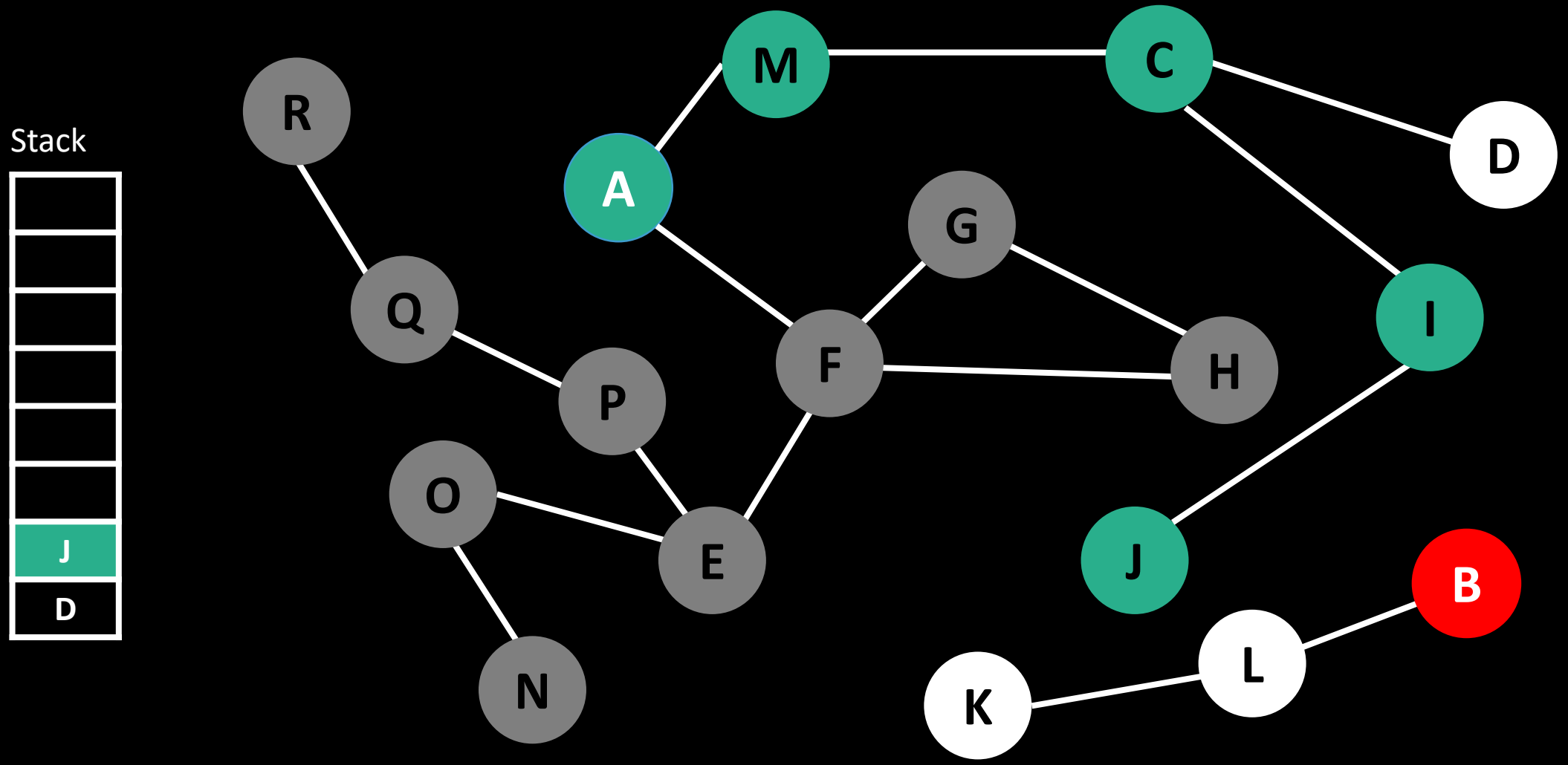
Another DFS Example



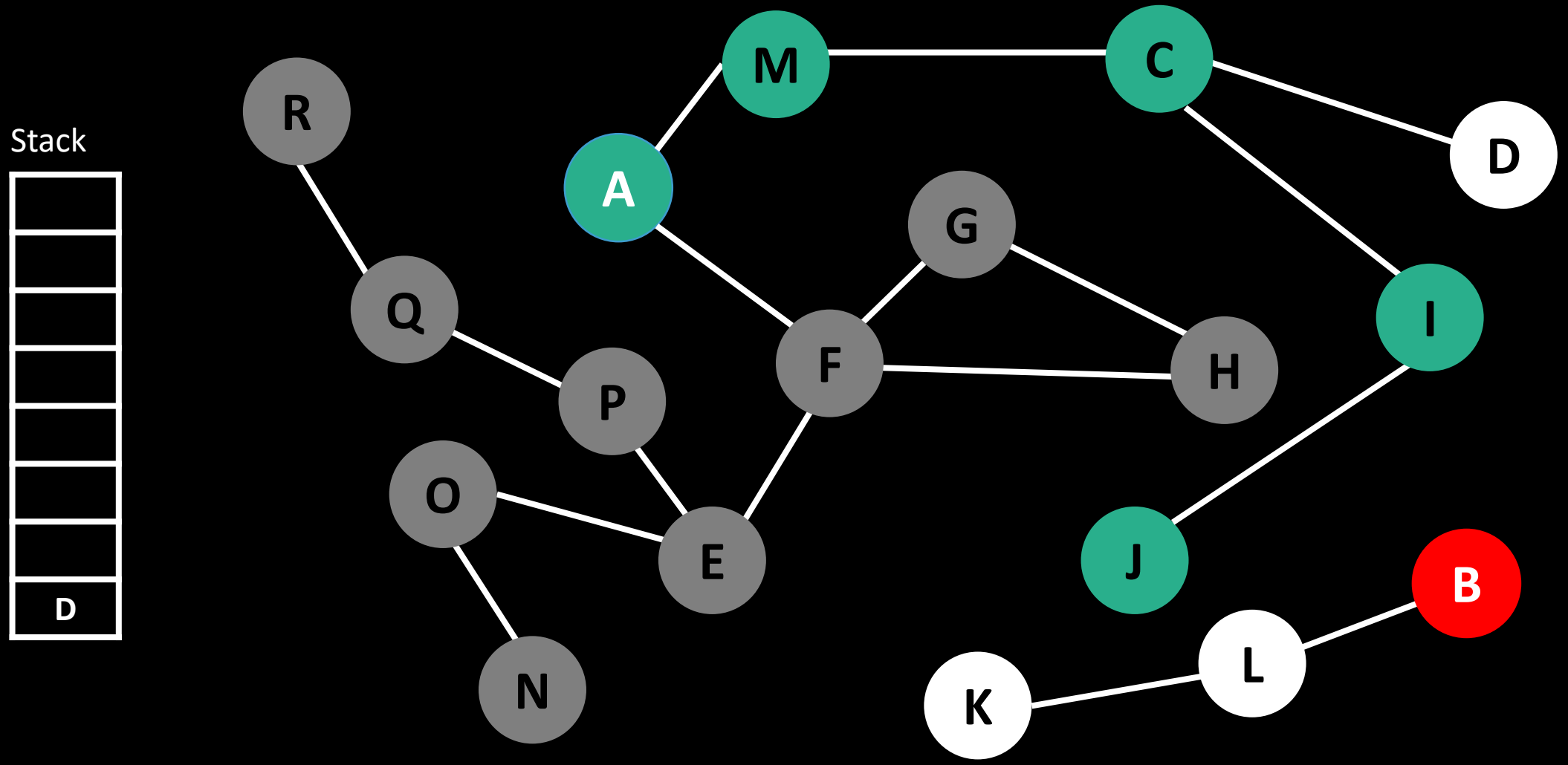
Another DFS Example



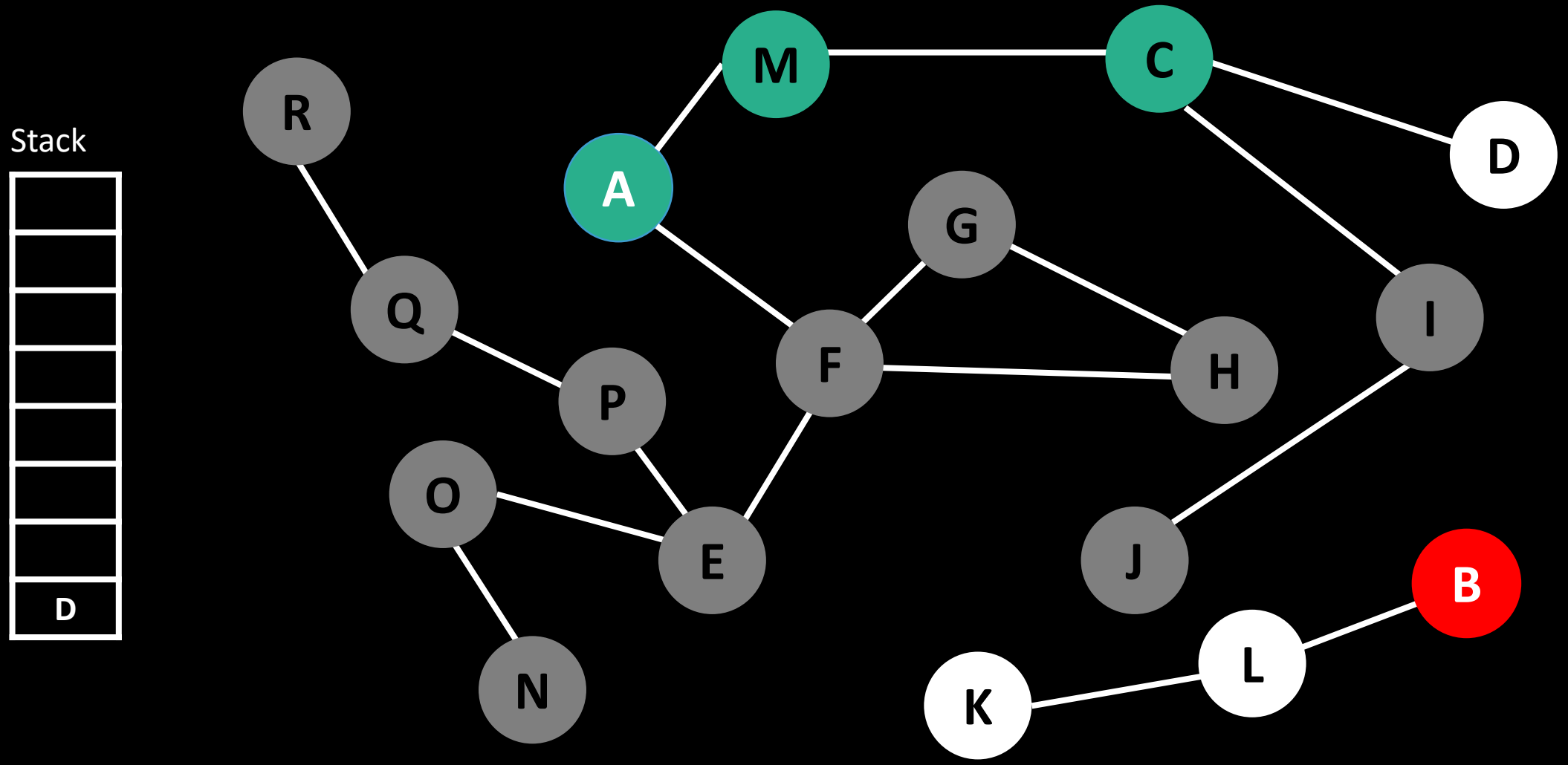
Another DFS Example



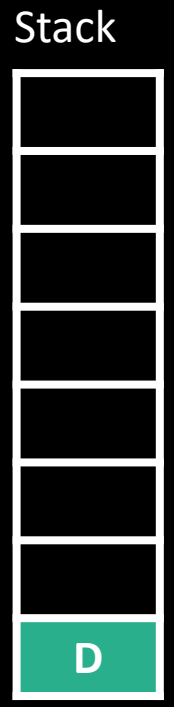
Another DFS Example

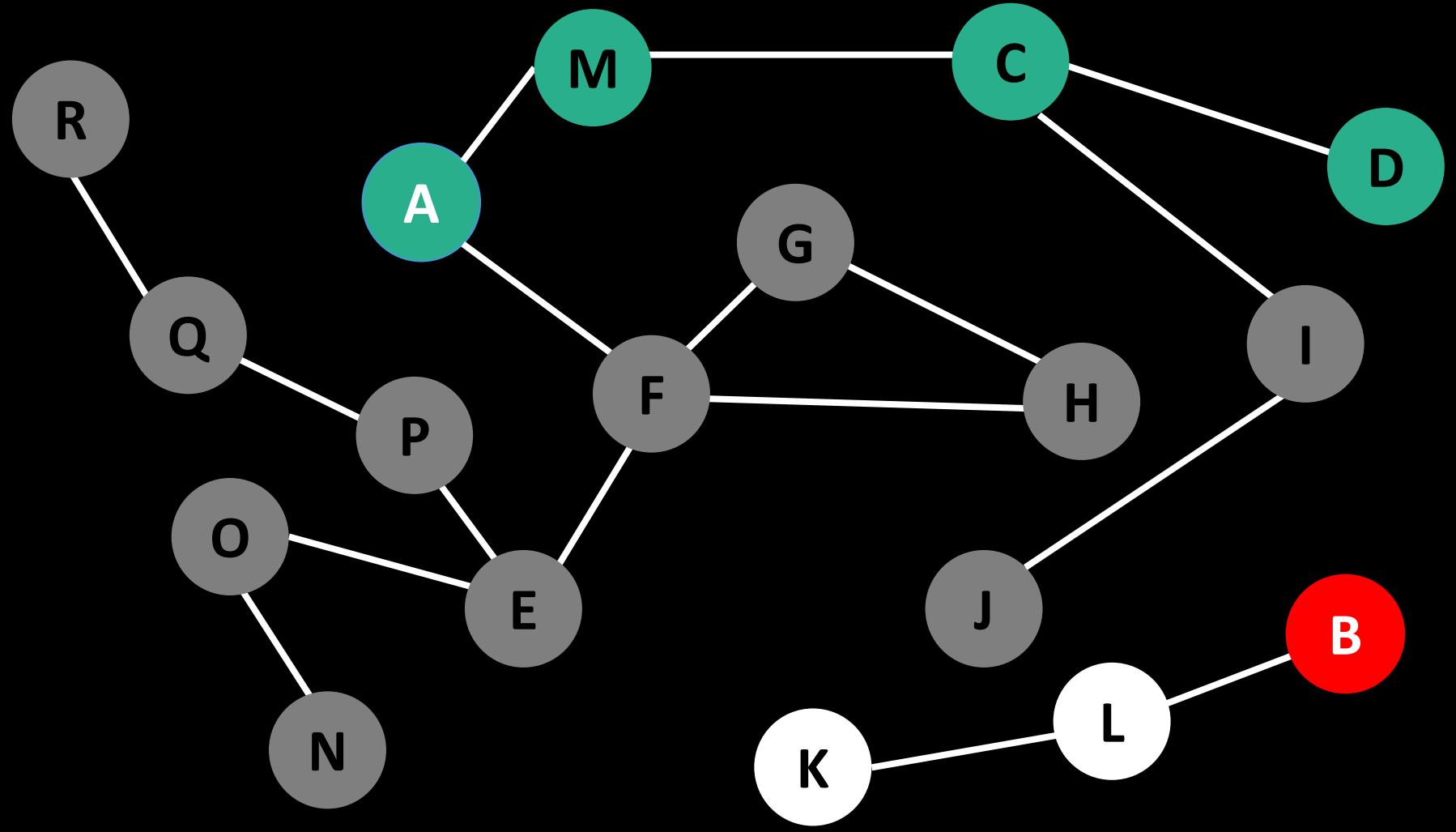


Another DFS Example

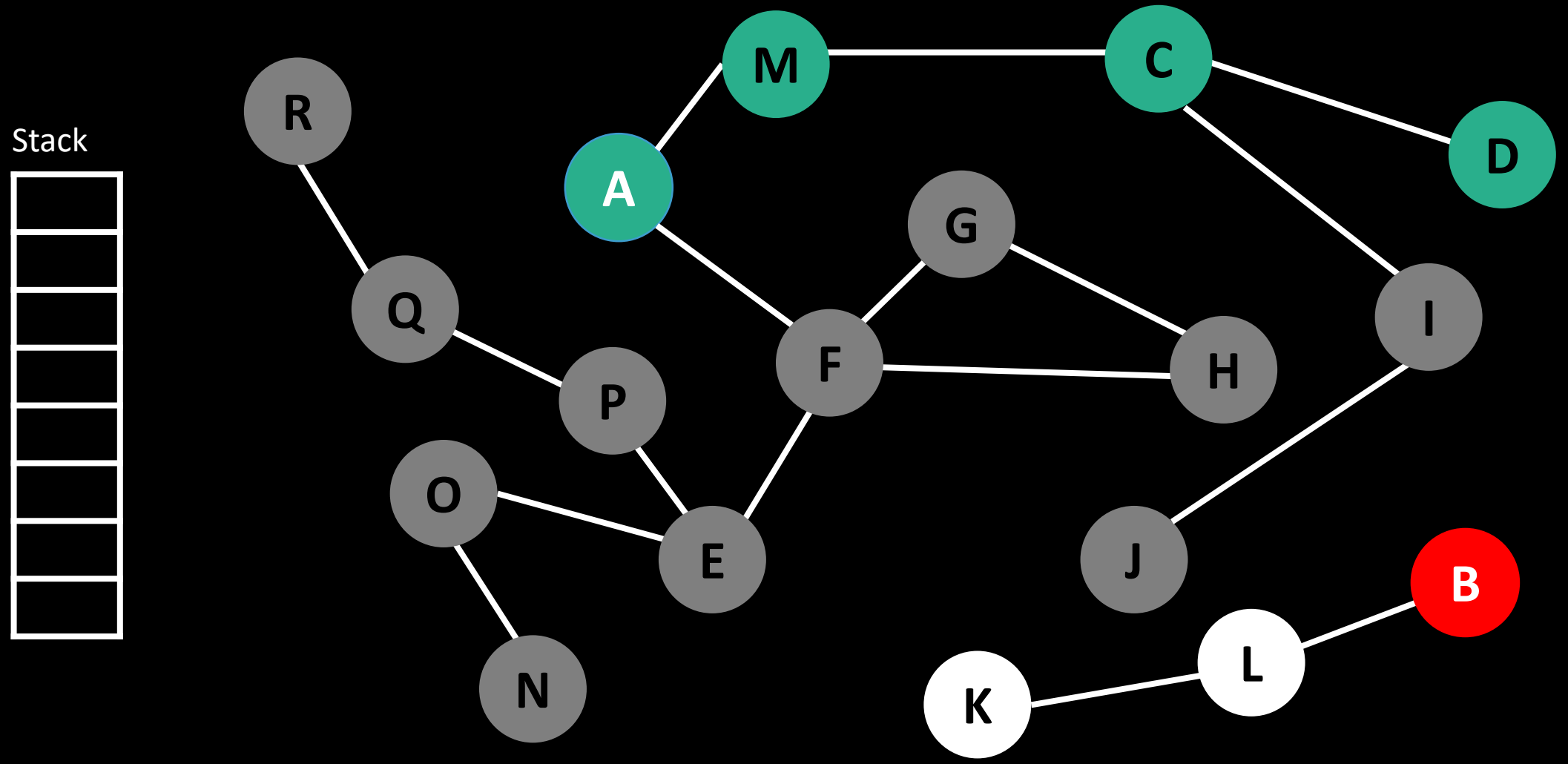


Another DFS Example

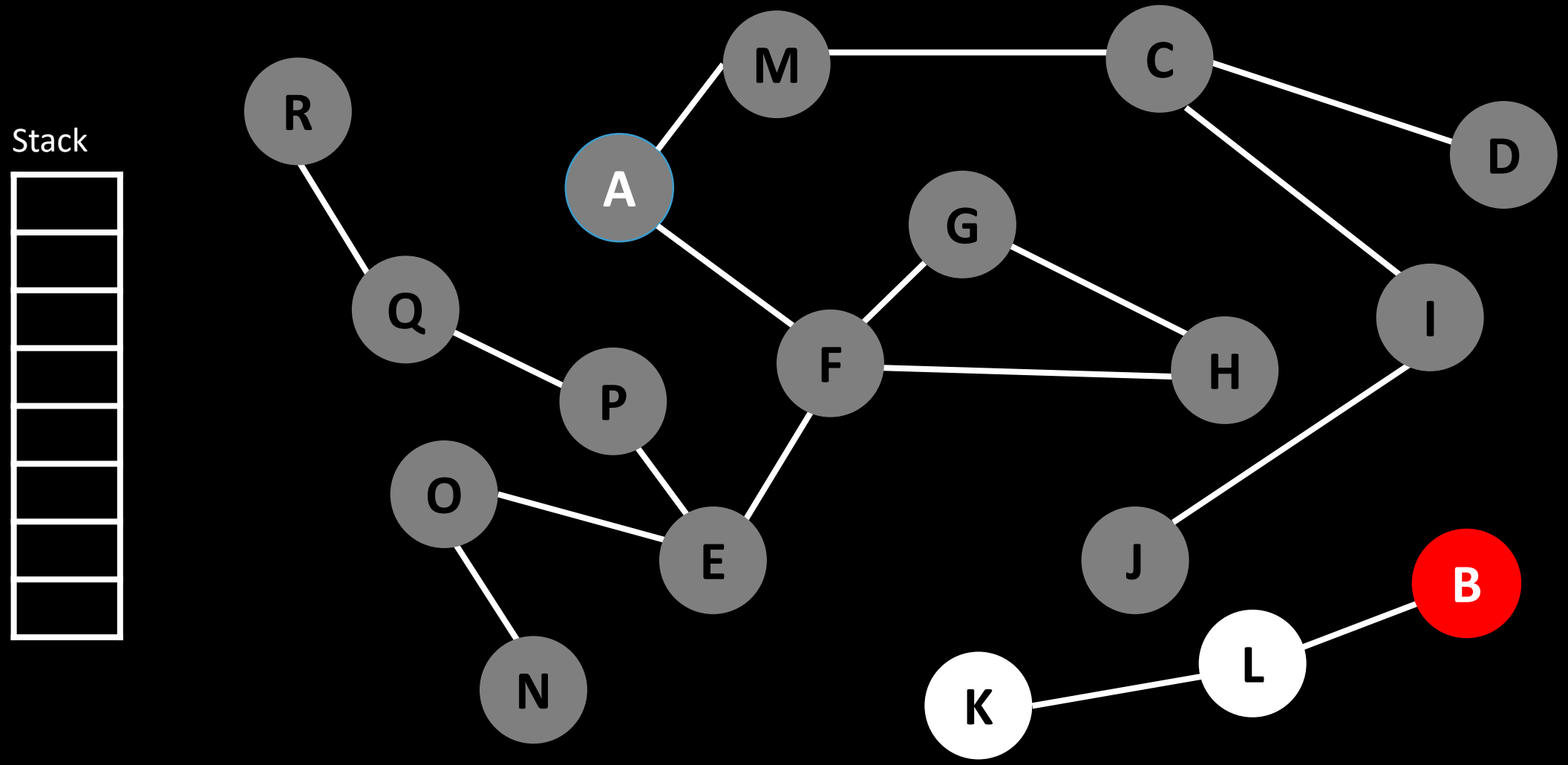




Another DFS Example

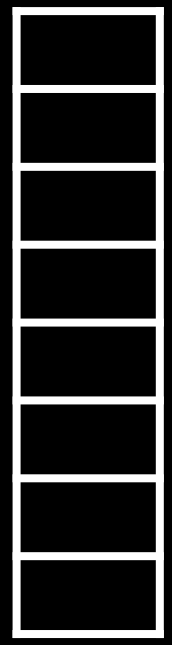


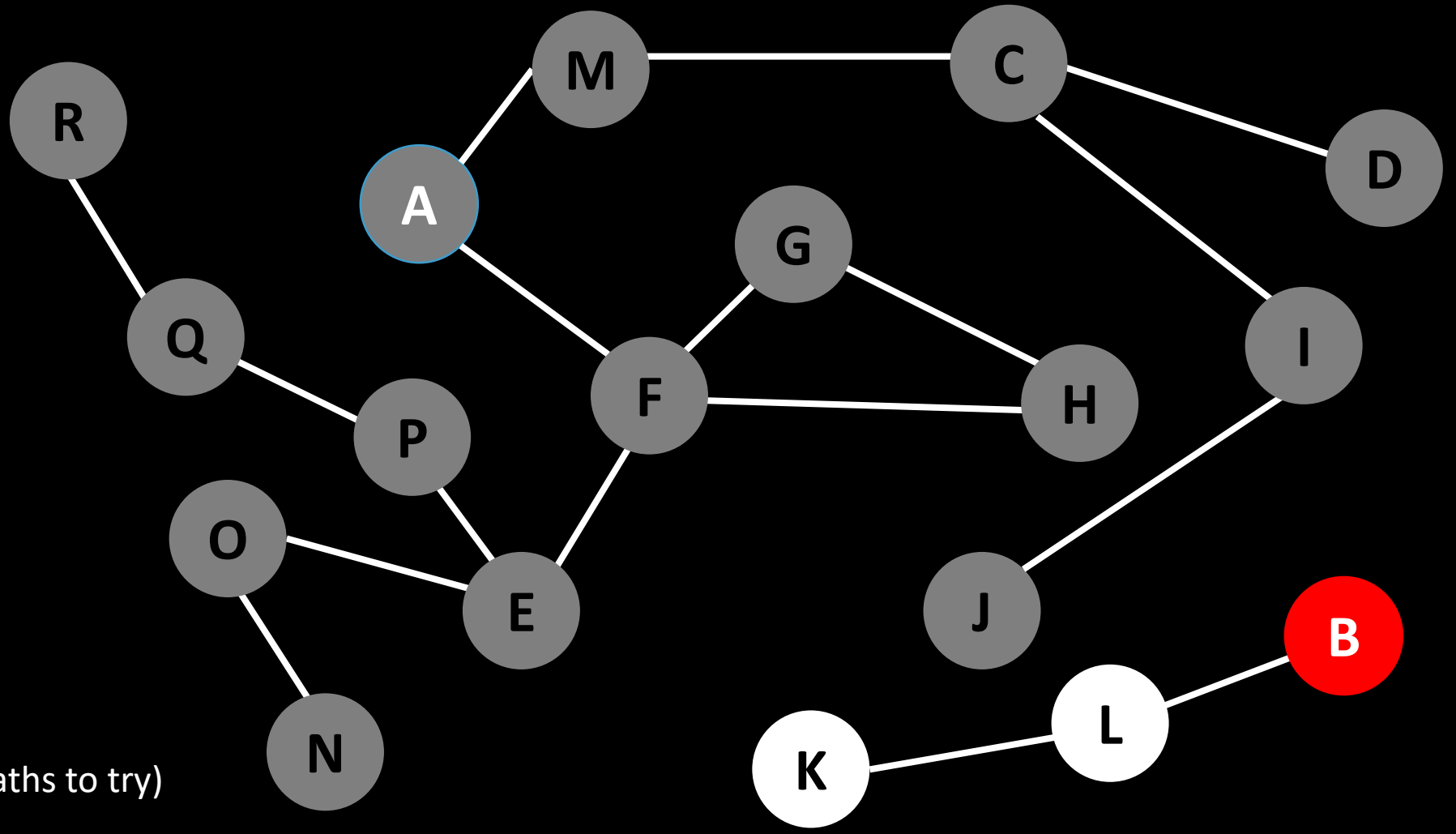
Another DFS Example



Another DFS Example

Stack





Stack is empty (i.e. no more paths to try)
We did not find B
Thus, no path exists from A → B

Depth-First Search PseudoCode

```
stack = []
stack.push(starting_node)

while (stack not empty) and (target not found):
        node = stack.pop()
        if node is visited:
                skip
        if node = target:
                target = found
        for all unvisited neighbors of node:
                stack.push(neighbor)
        node.visited = True
```

Depth-First Search PseudoCode

```
stack = []
stack.push(starting_node)

while (stack not empty) and (target not found):
        node = stack.pop()
        if node is visited:
                skip
        if node = target:
                target = found
        for all unvisited neighbors of node:
                neighbor.prev = node
                stack.push(neighbor)
        node.visited = True
```

Depth-First Search PseudoCode

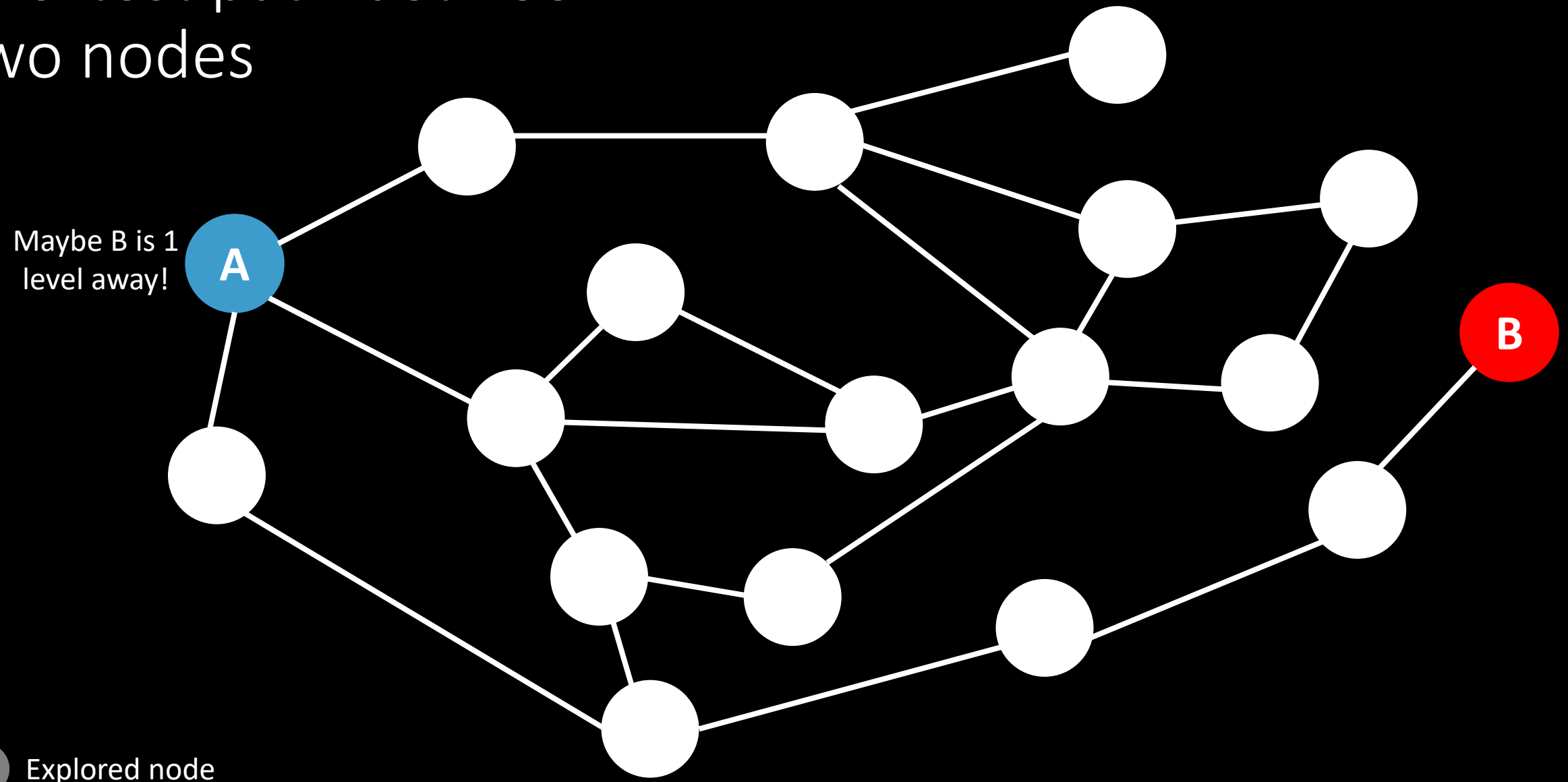
```
stack = []
stack.push(starting_node)

while (stack not empty) and (target not found):
        node = stack.pop()
        if node is visited:
                skip
        if node = target:
                target = found
        for all unvisited neighbors of node:
                neighbor.prev = node
                stack.push(neighbor)
        node.visited = True
```

Retrieving the path, if one is found

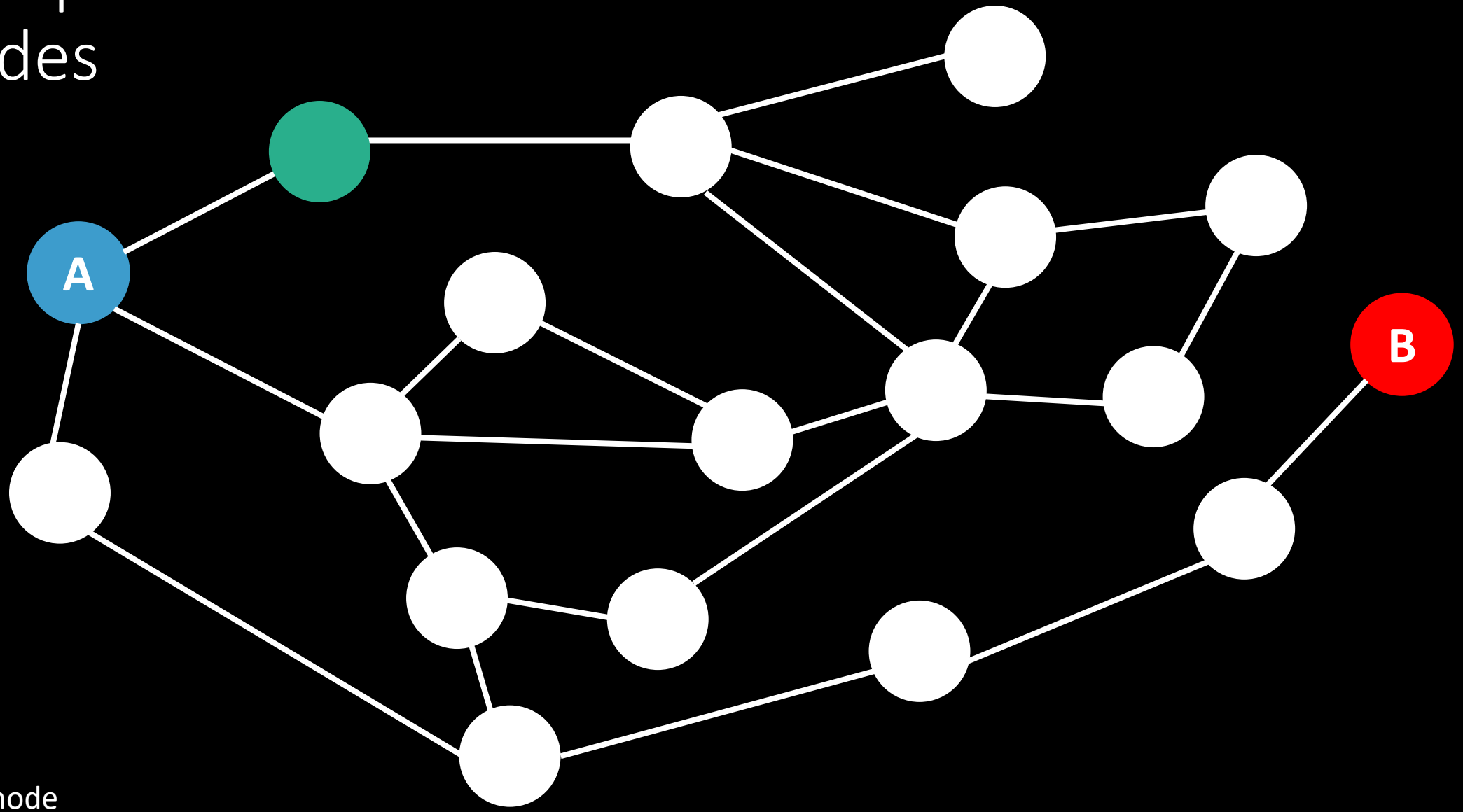
```
node = target
path = [node]
while node is not starting_node:
        path.push(node.prev)
        node = node.prev
```

Shortest path between two nodes



MO: Finish exploring current level before deepening the search

Shortest path between two nodes

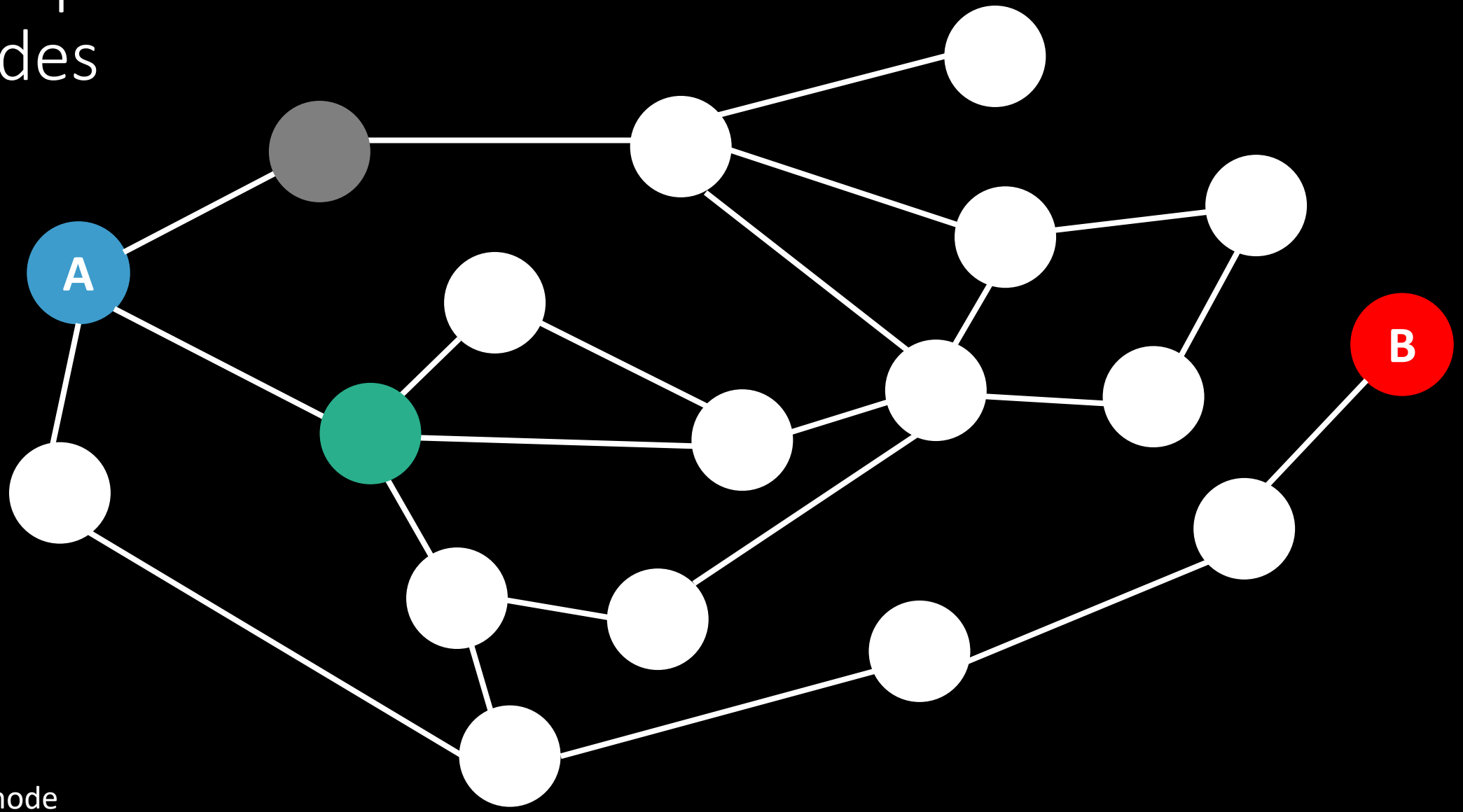


Explored node

Path taken

MO: Finish exploring current level before deepening the search

Shortest path between two nodes

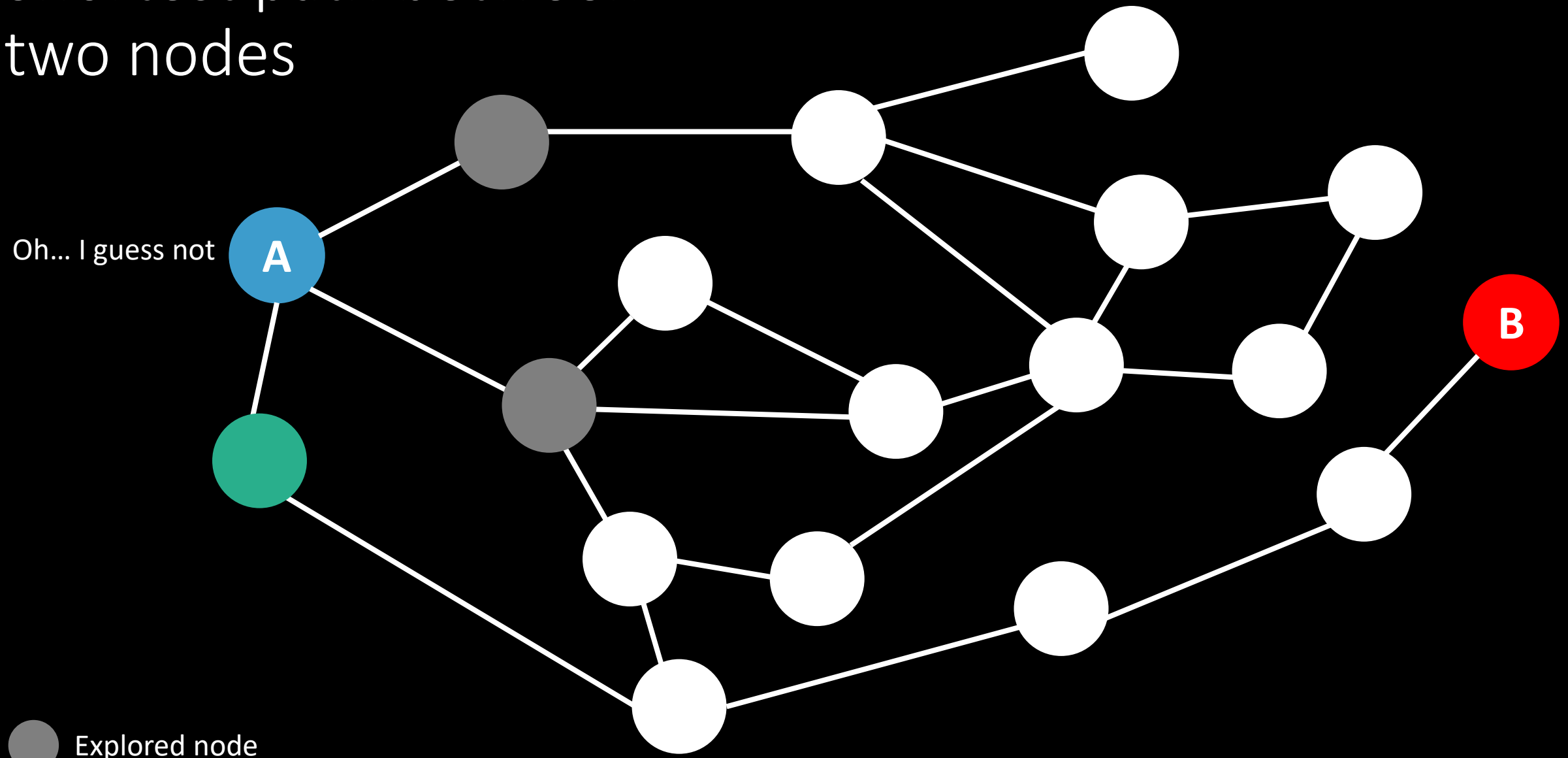


Explored node

Path taken

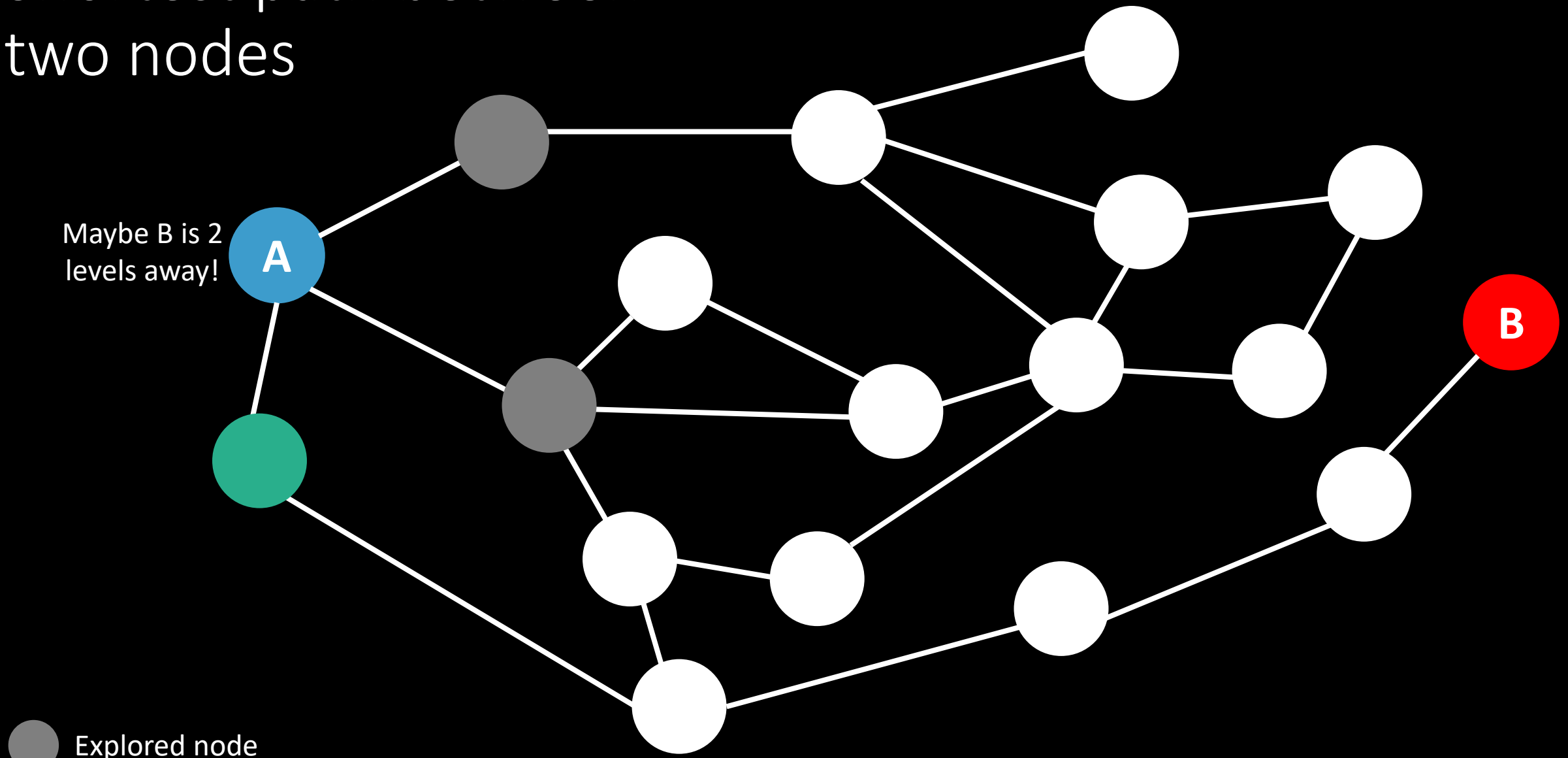
MO: Finish exploring current level before deepening the search

Shortest path between two nodes



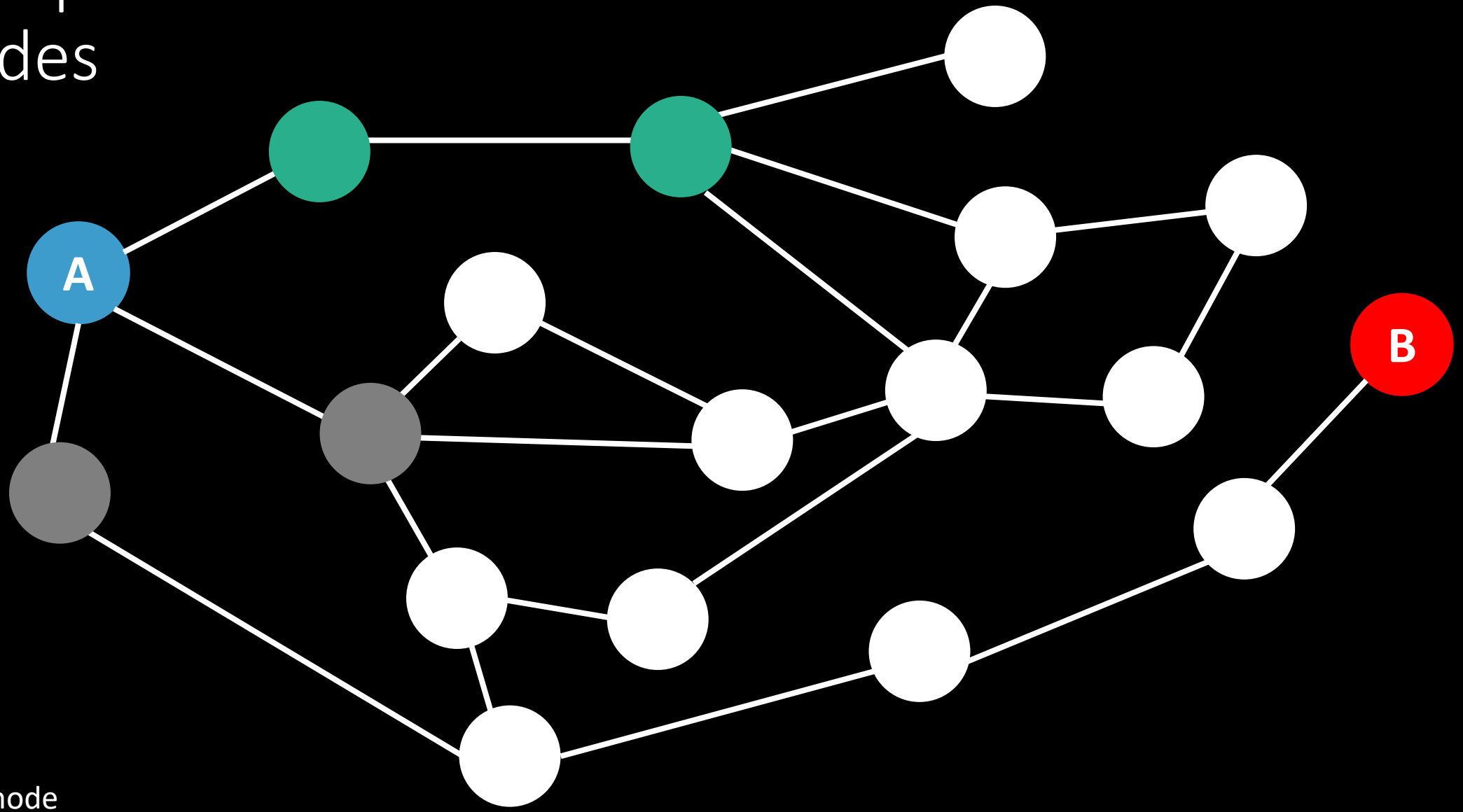
MO: Finish exploring current level before deepening the search

Shortest path between two nodes



MO: Finish exploring current level before deepening the search

Shortest path between two nodes

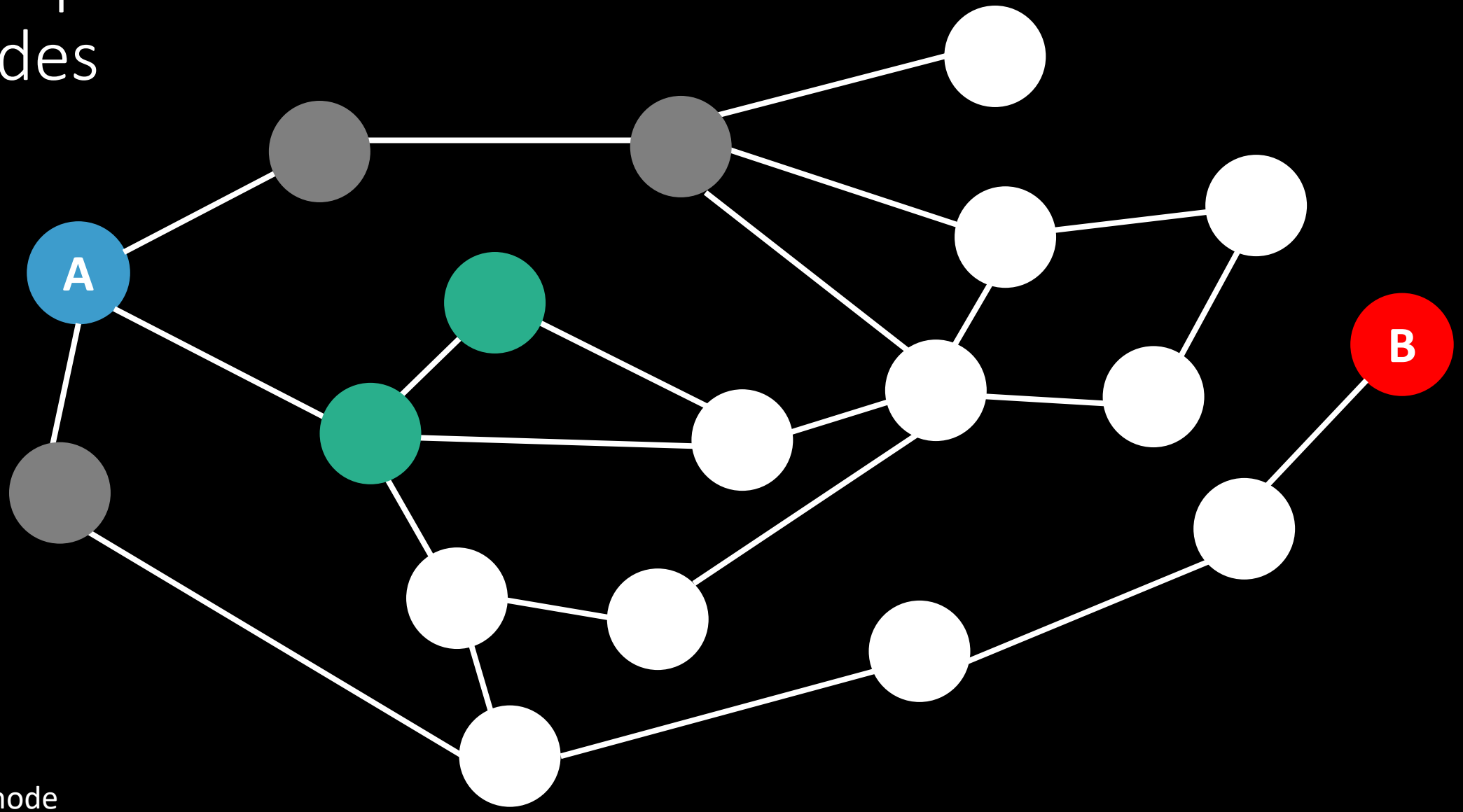


Explored node

Path taken

MO: Finish exploring current level before deepening the search

Shortest path between two nodes

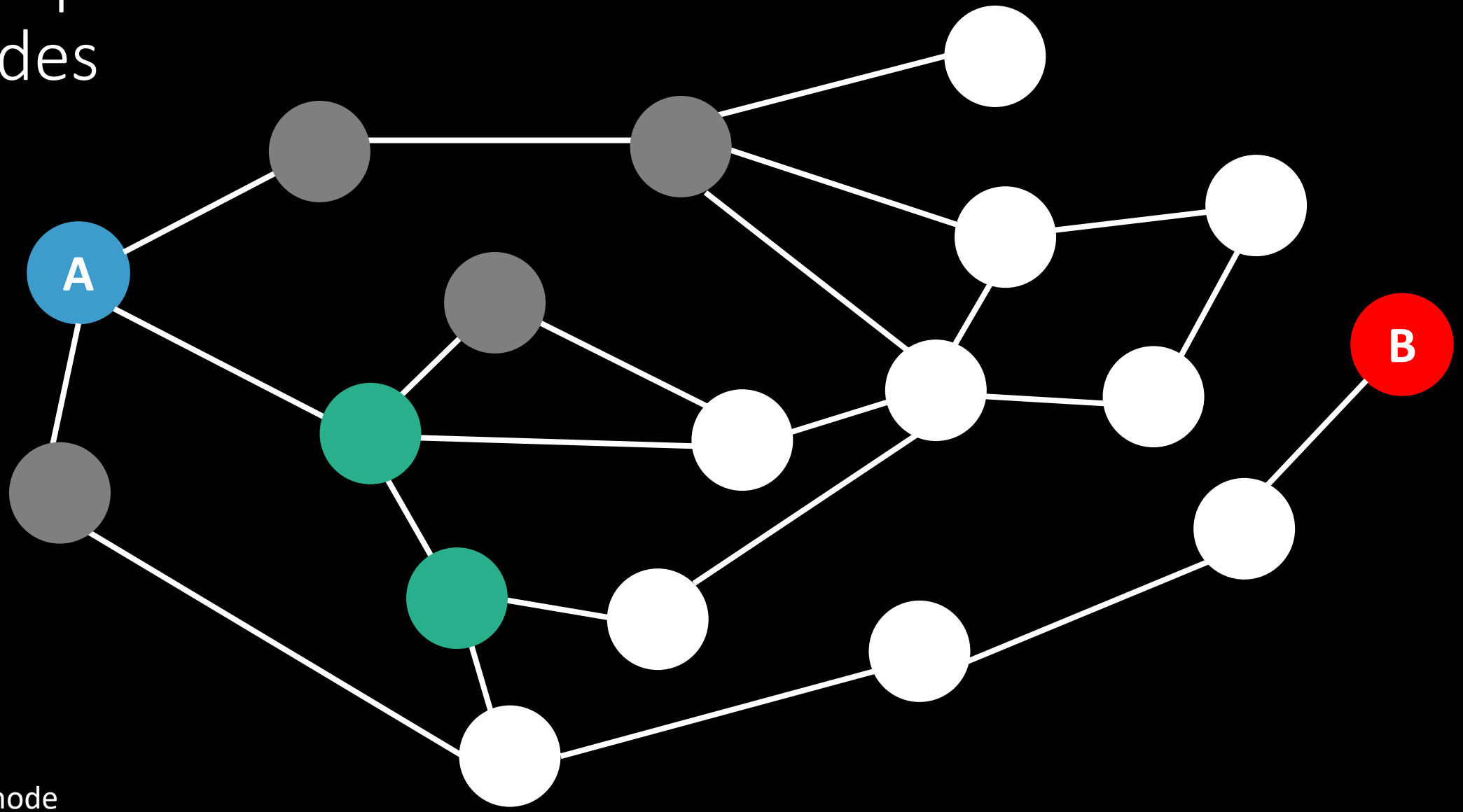


Explored node

Path taken

MO: Finish exploring current level before deepening the search

Shortest path between two nodes

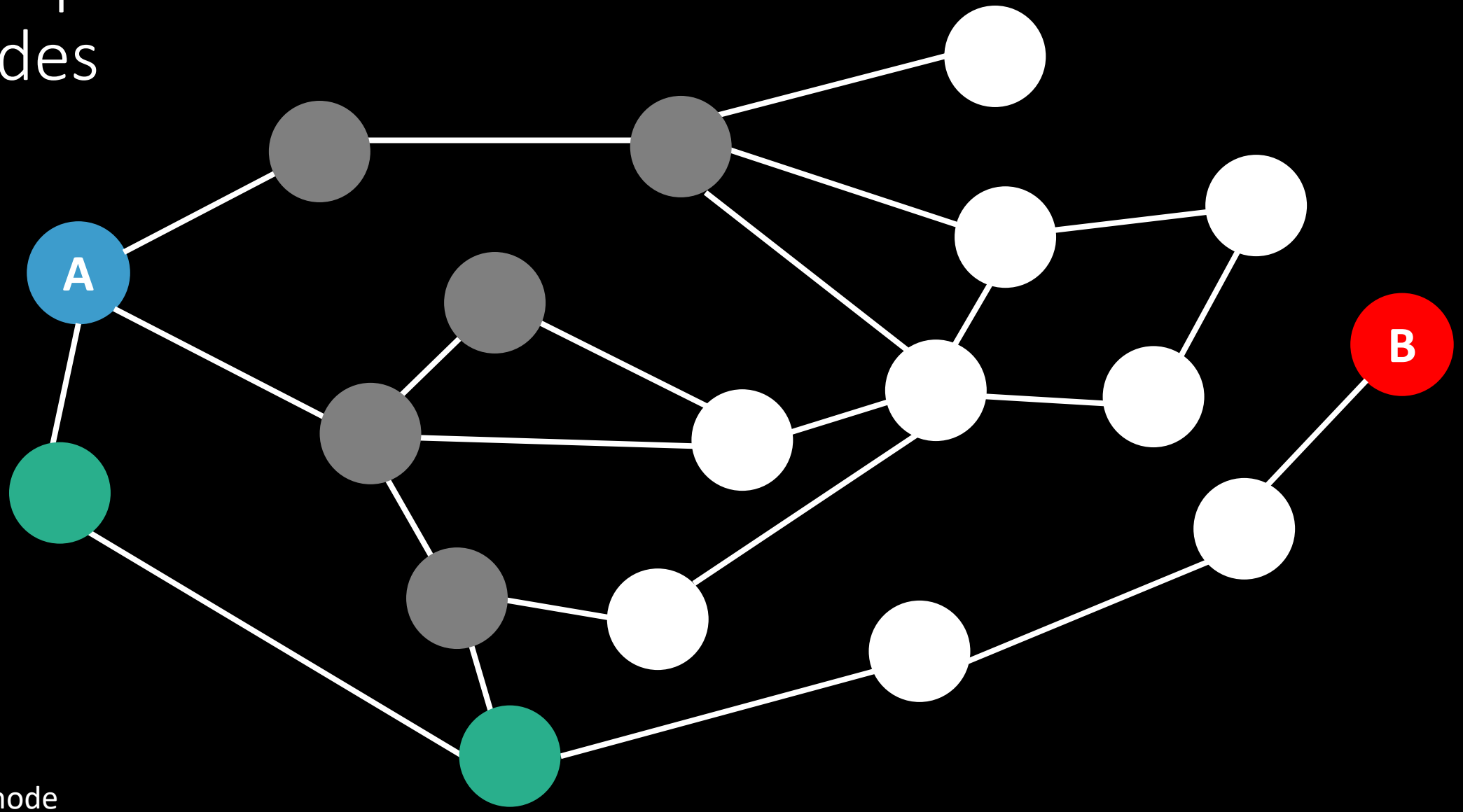


Explored node

Path taken

MO: Finish exploring current level before deepening the search

Shortest path between two nodes

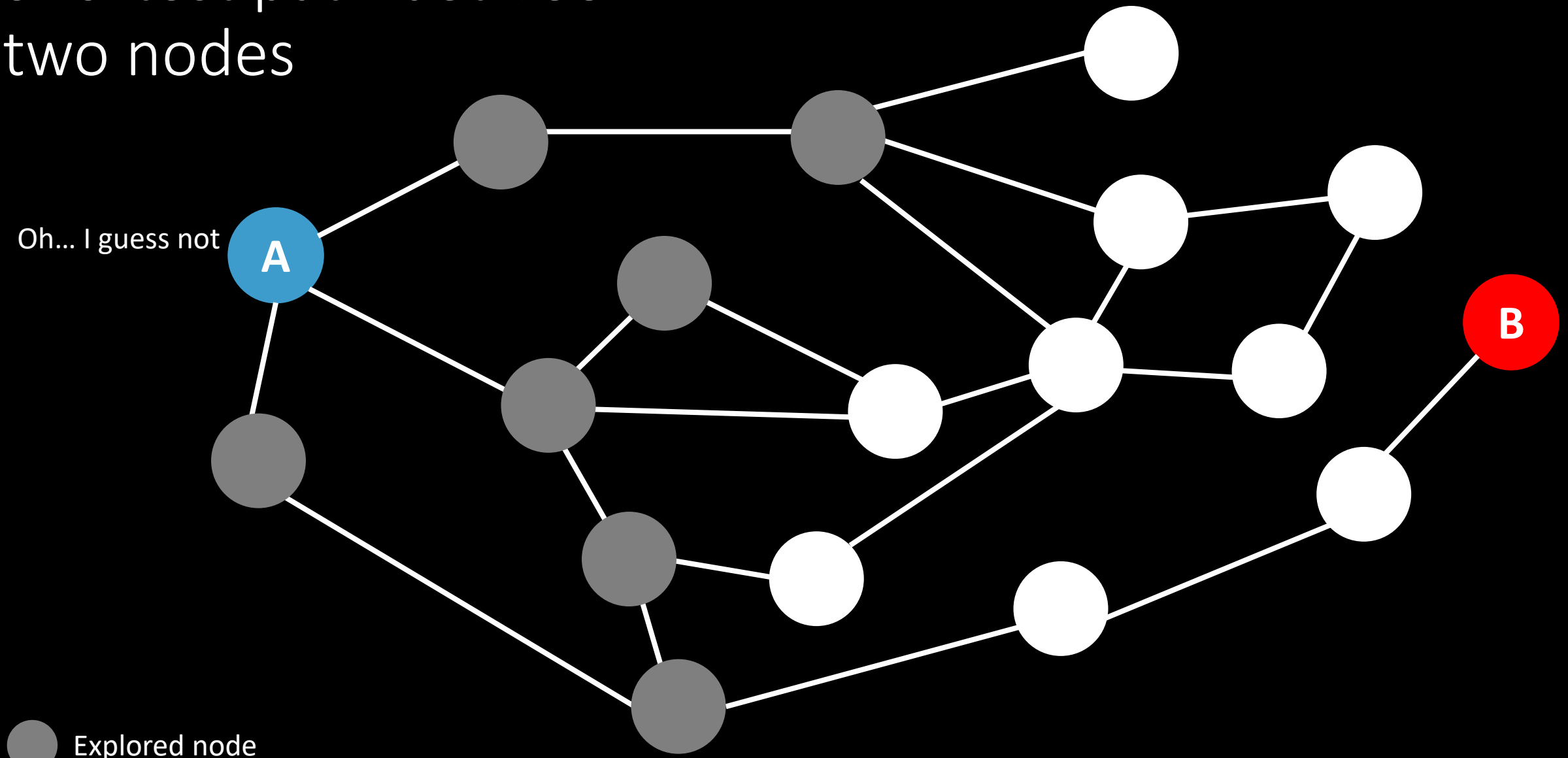


Explored node

Path taken

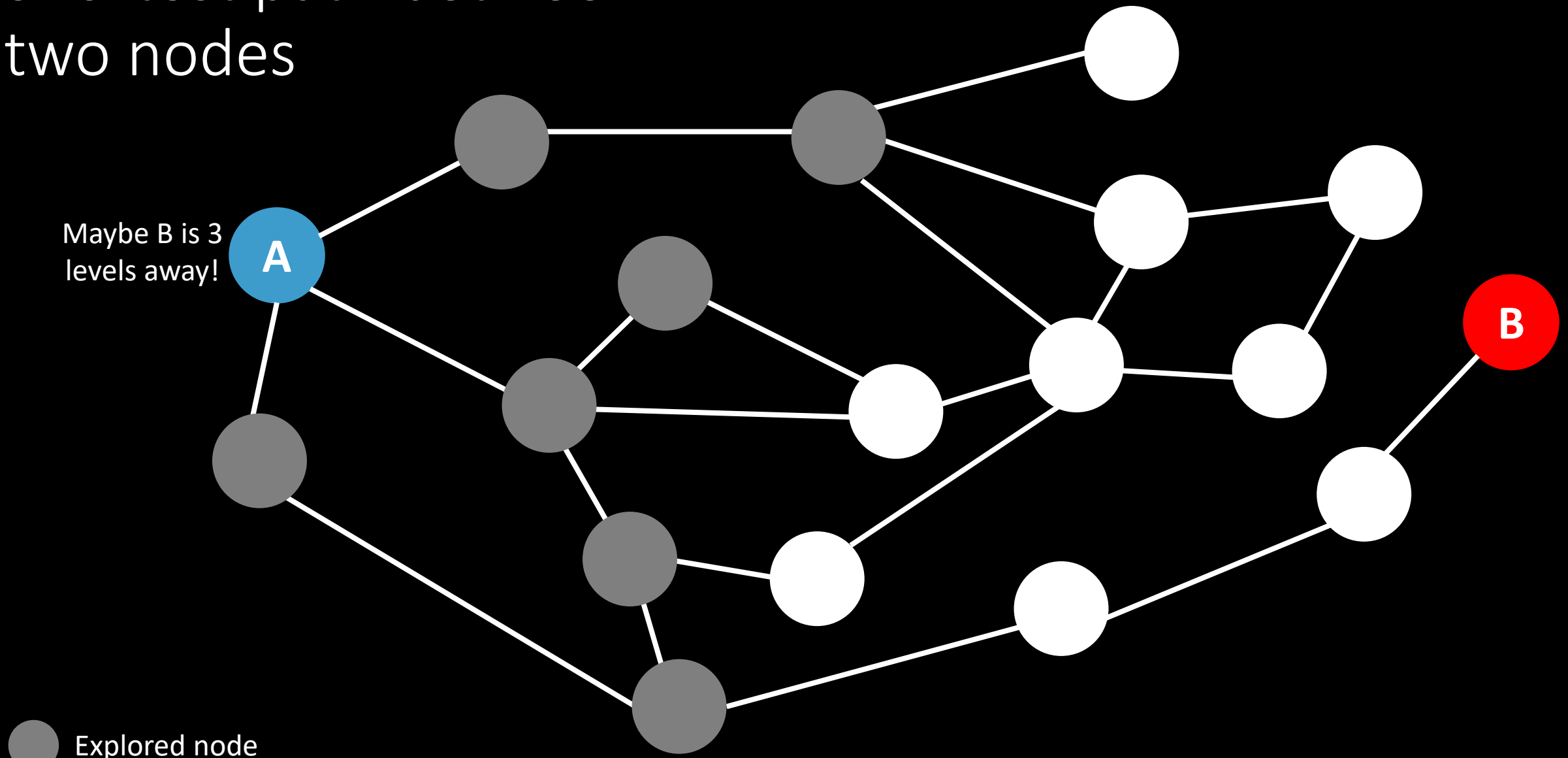
MO: Finish exploring current level before deepening the search

Shortest path between two nodes



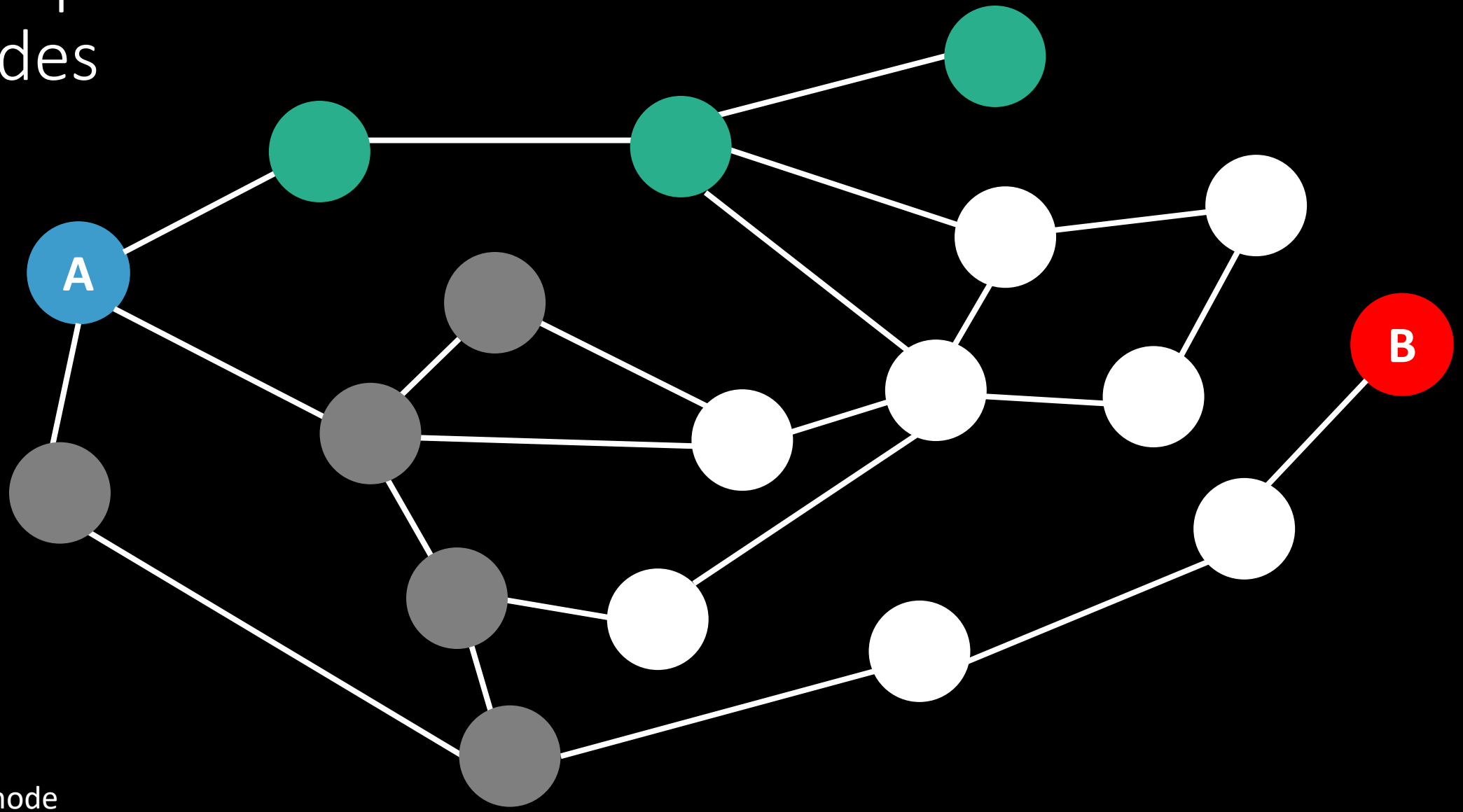
MO: Finish exploring current level before deepening the search

Shortest path between two nodes



MO: Finish exploring current level before deepening the search

Shortest path between two nodes

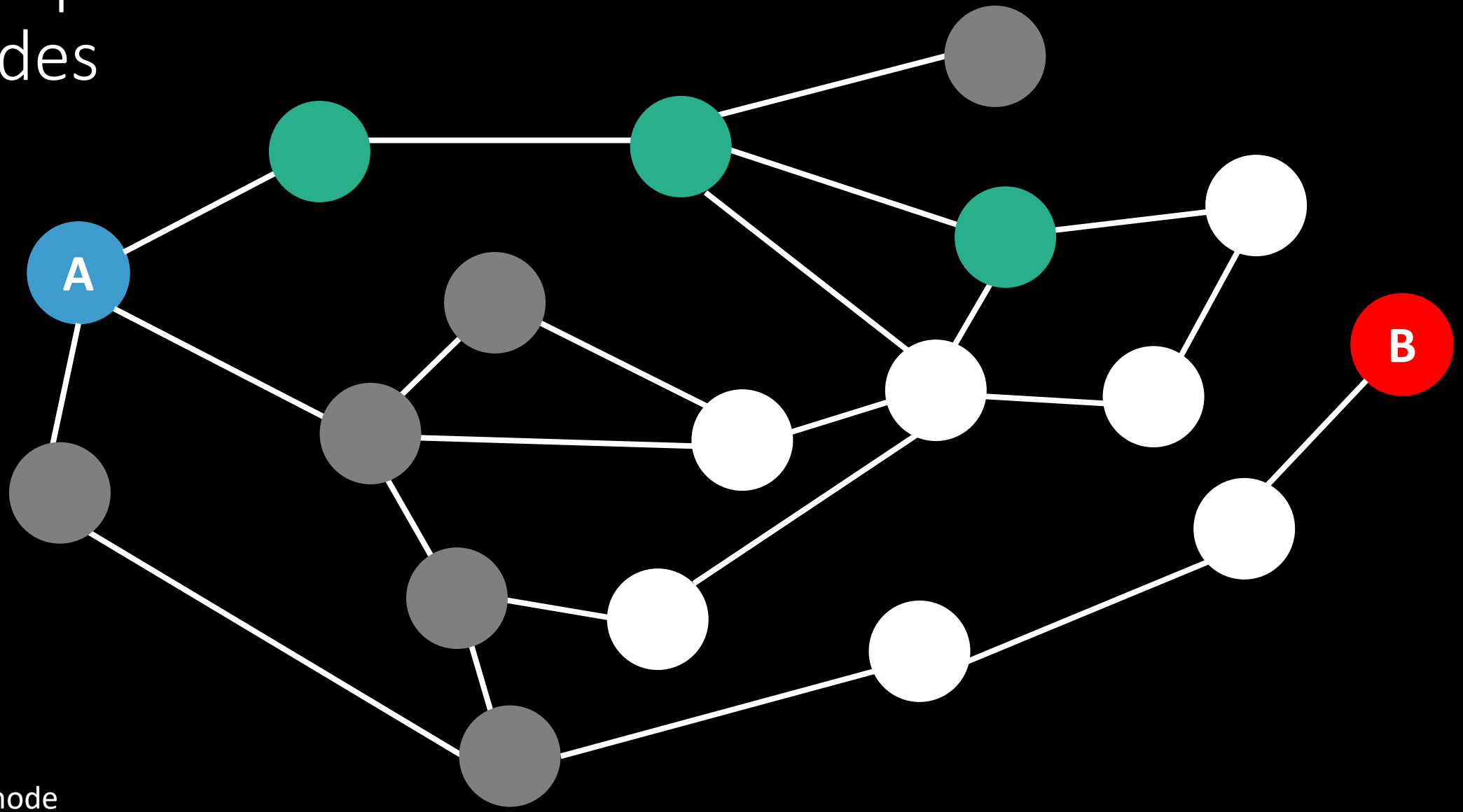


Explored node

Path taken

MO: Finish exploring current level before deepening the search

Shortest path between two nodes

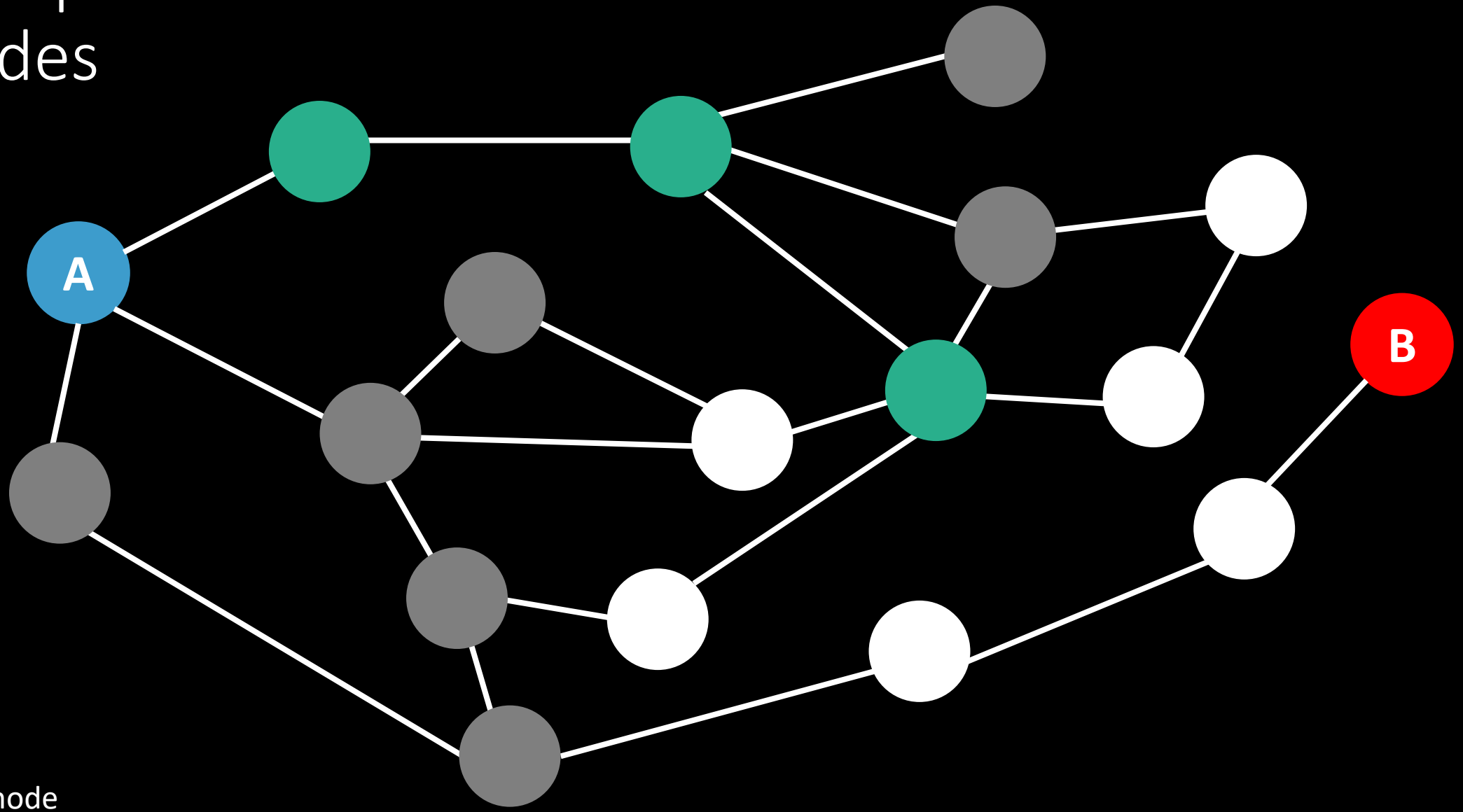


Explored node

Path taken

MO: Finish exploring current level before deepening the search

Shortest path between two nodes

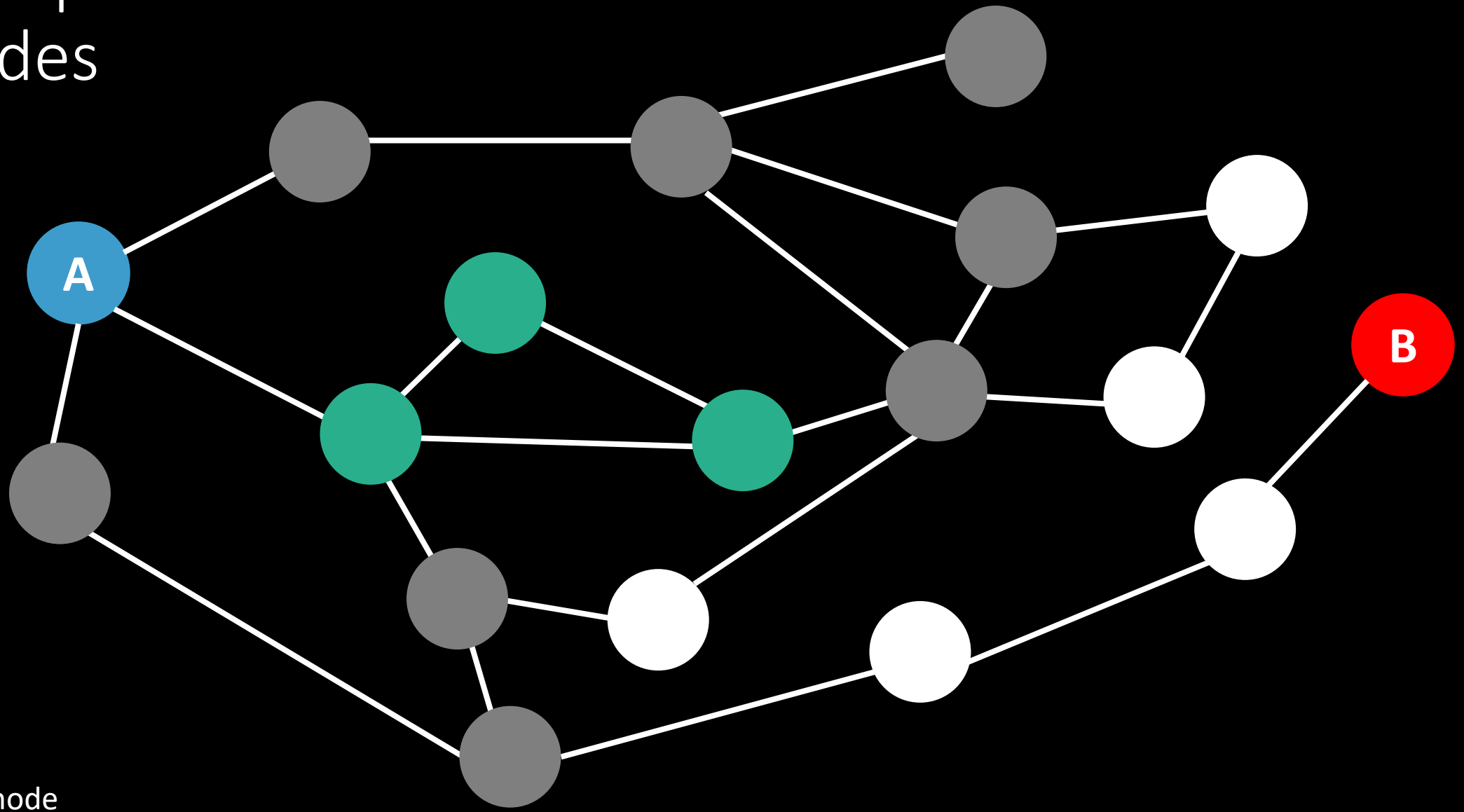


Explored node

Path taken

MO: Finish exploring current level before deepening the search

Shortest path between two nodes

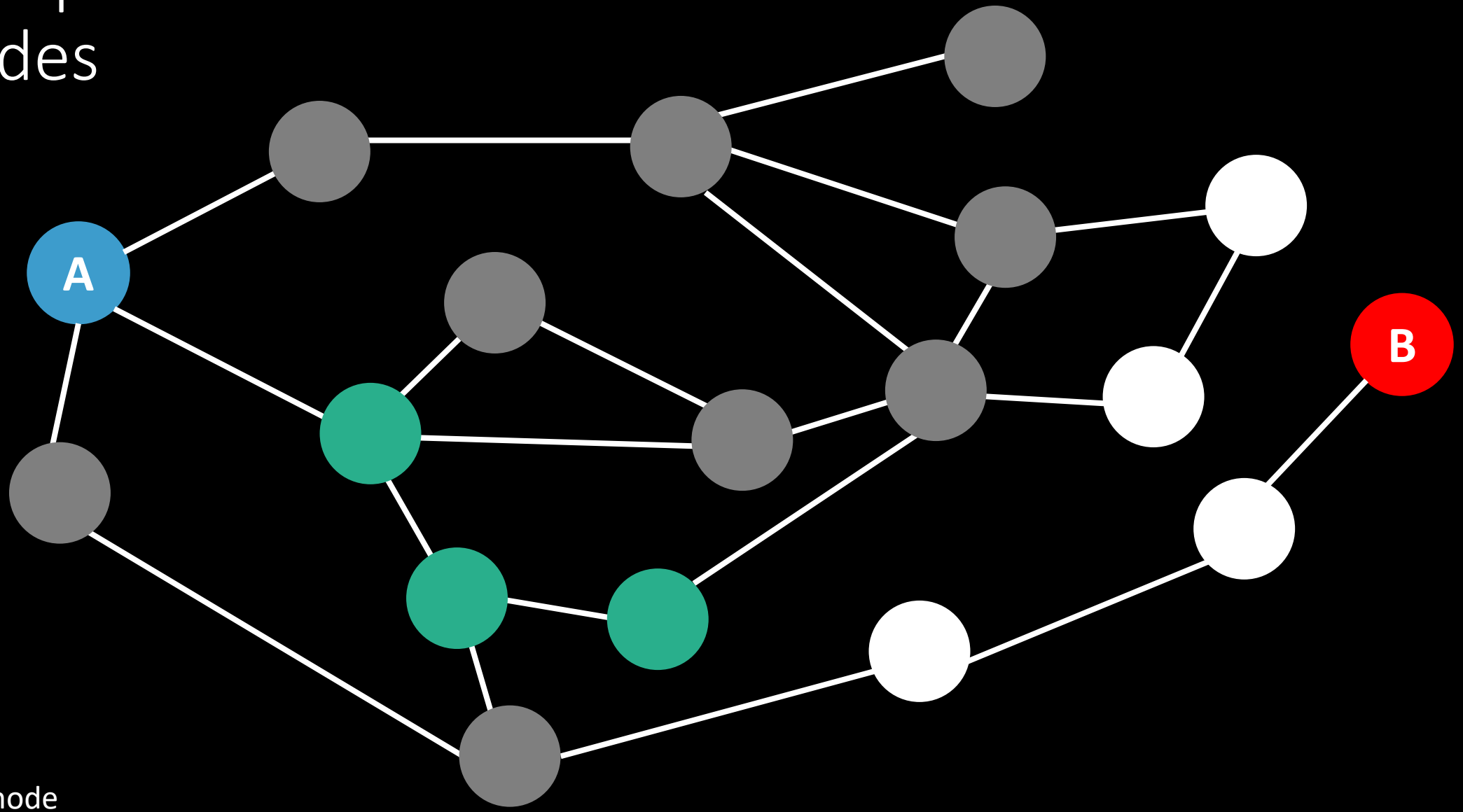


Explored node

Path taken

MO: Finish exploring current level before deepening the search

Shortest path between two nodes

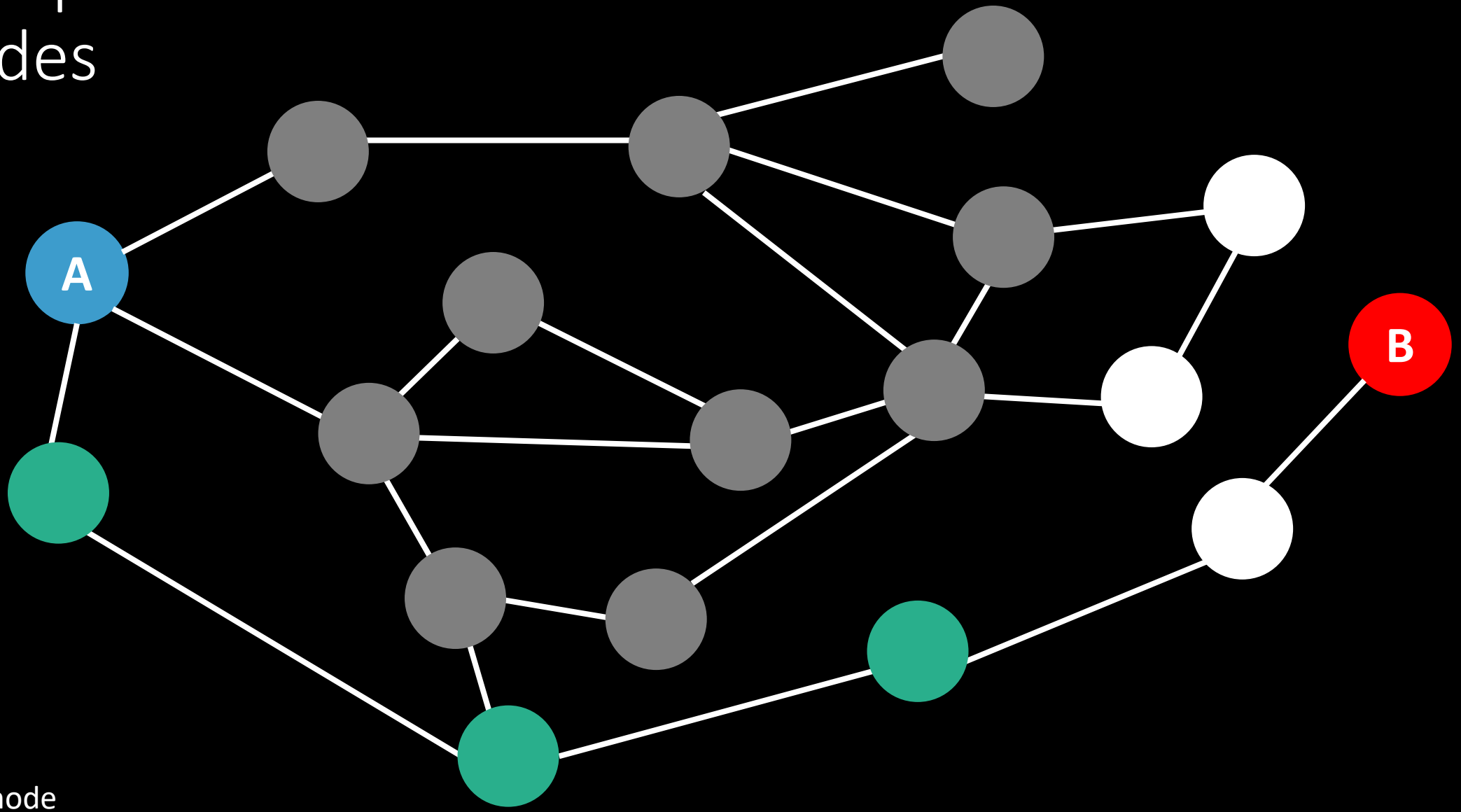


Explored node

Path taken

MO: Finish exploring current level before deepening the search

Shortest path between two nodes

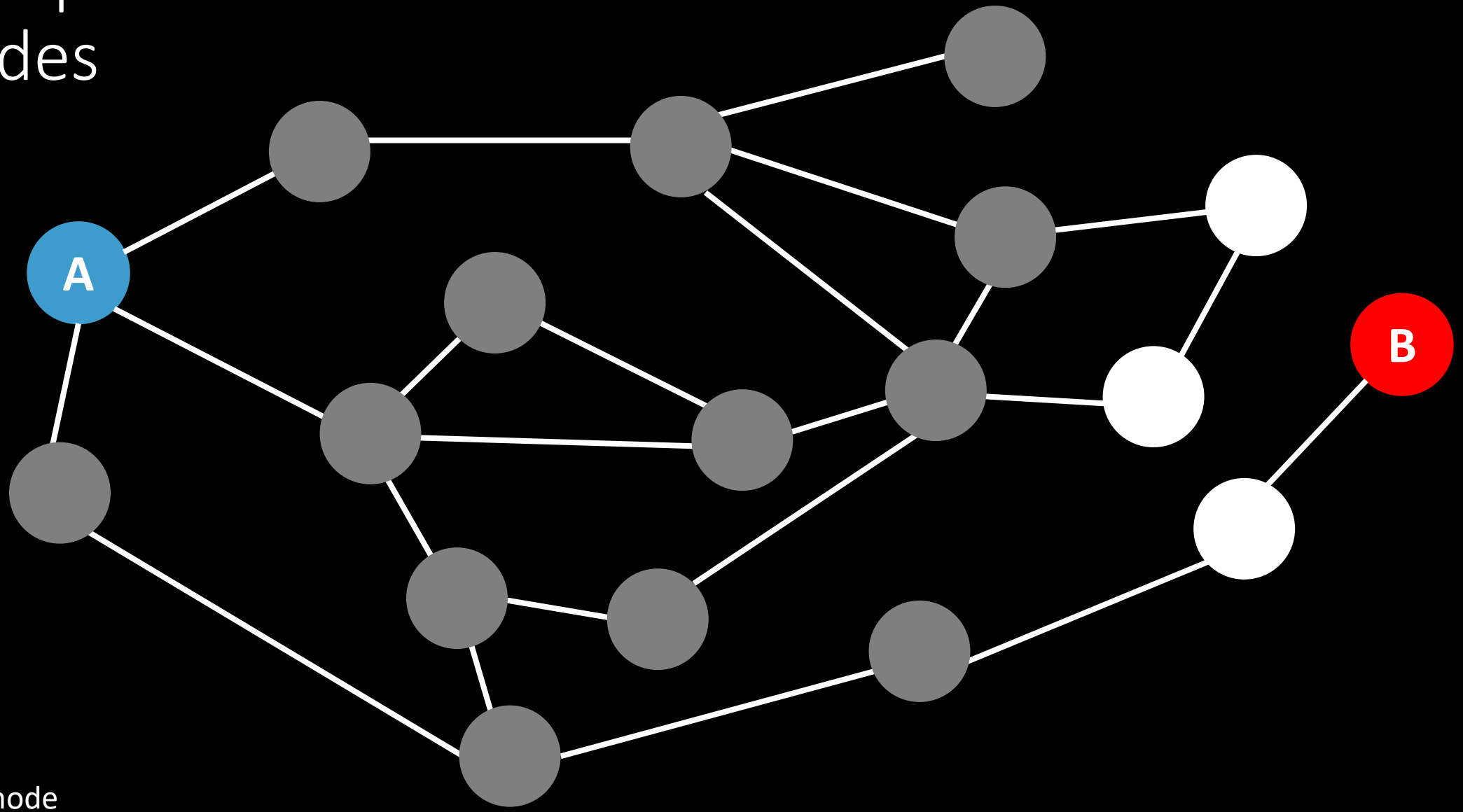


Explored node

Path taken

MO: Finish exploring current level before deepening the search

Shortest path between two nodes

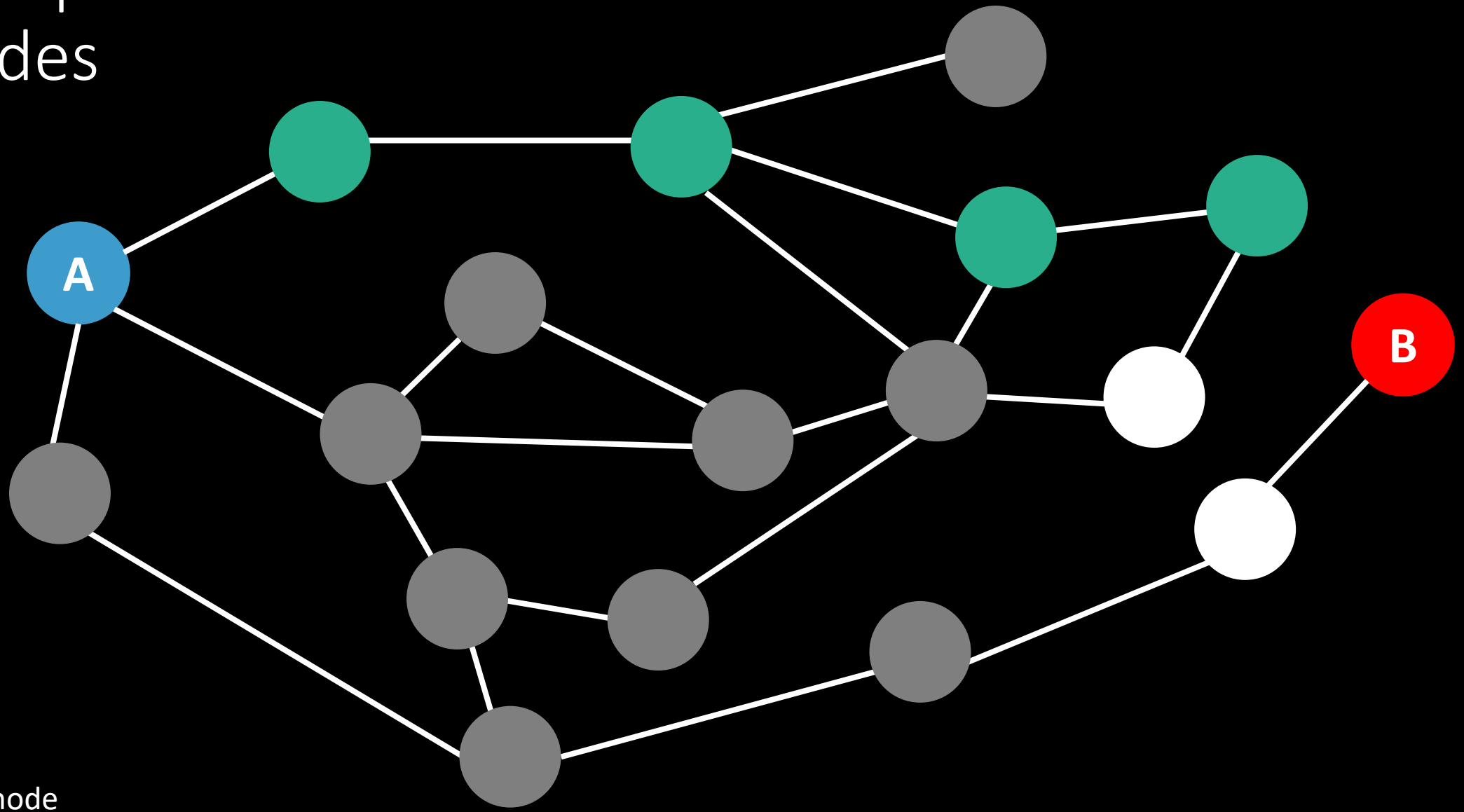


Explored node

Path taken

MO: Finish exploring current level before deepening the search

Shortest path between two nodes

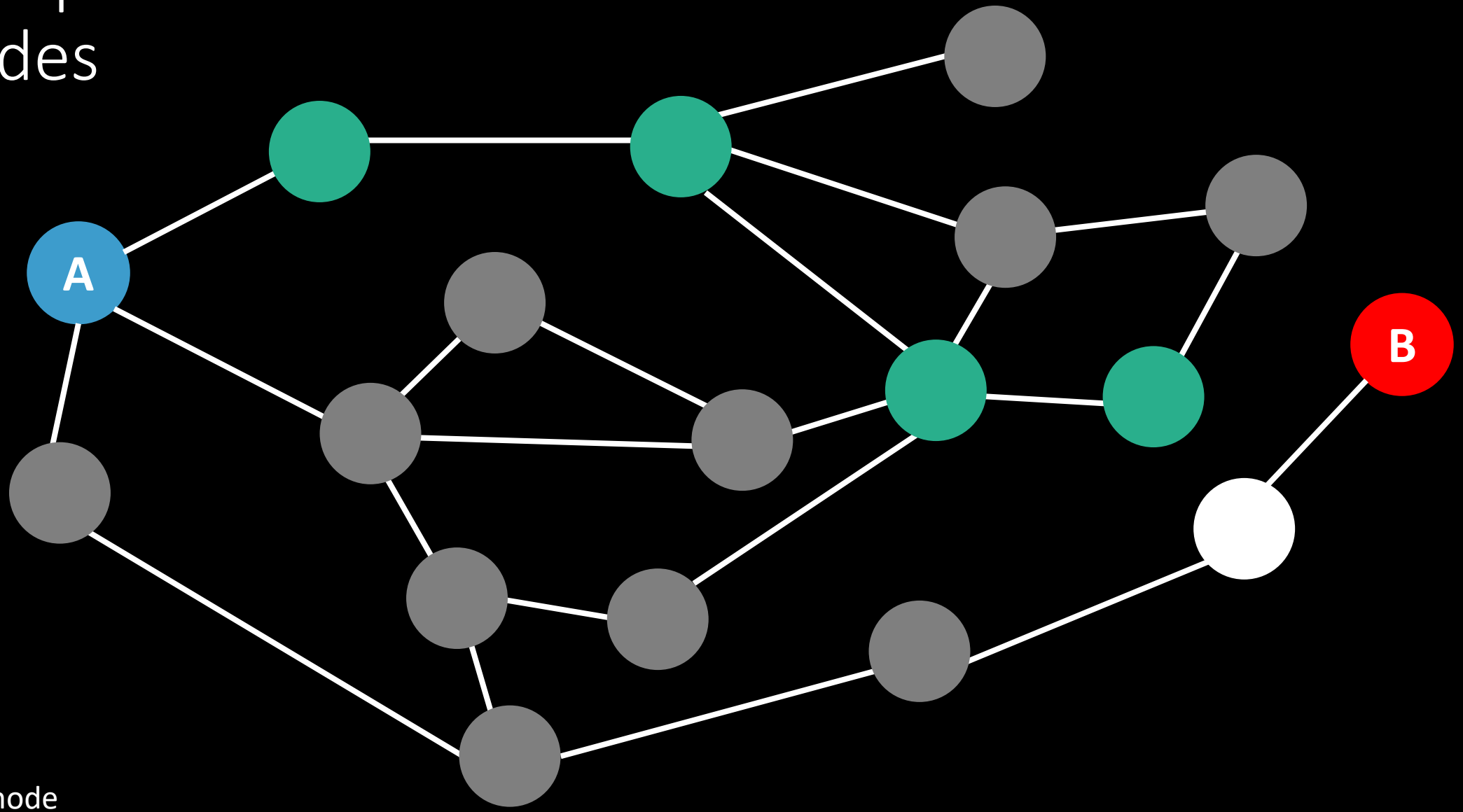


Explored node

Path taken

MO: Finish exploring current level before deepening the search

Shortest path between two nodes

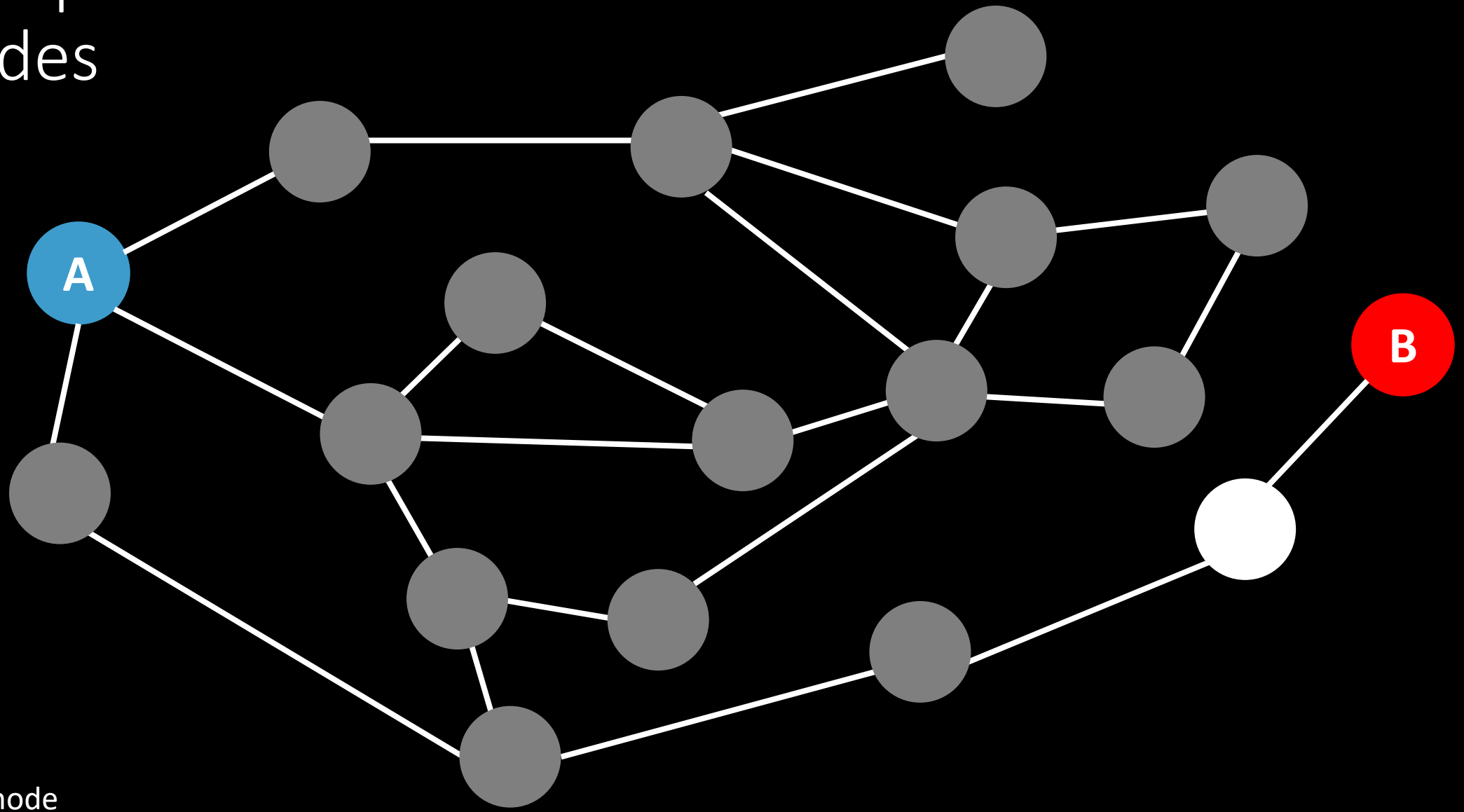


Explored node

Path taken

MO: Finish exploring current level before deepening the search

Shortest path between two nodes

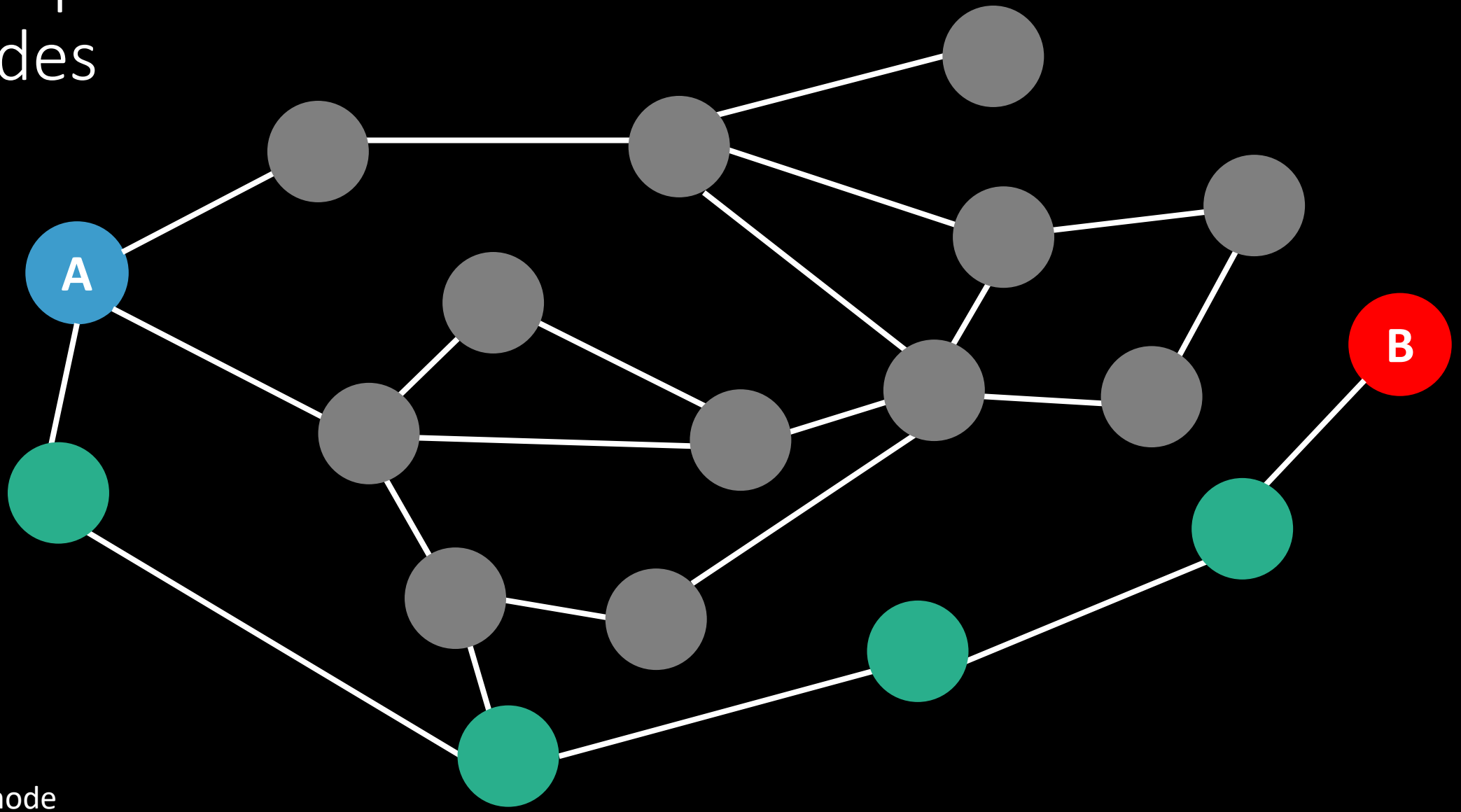


Explored node

Path taken

MO: Finish exploring current level before deepening the search

Shortest path between two nodes

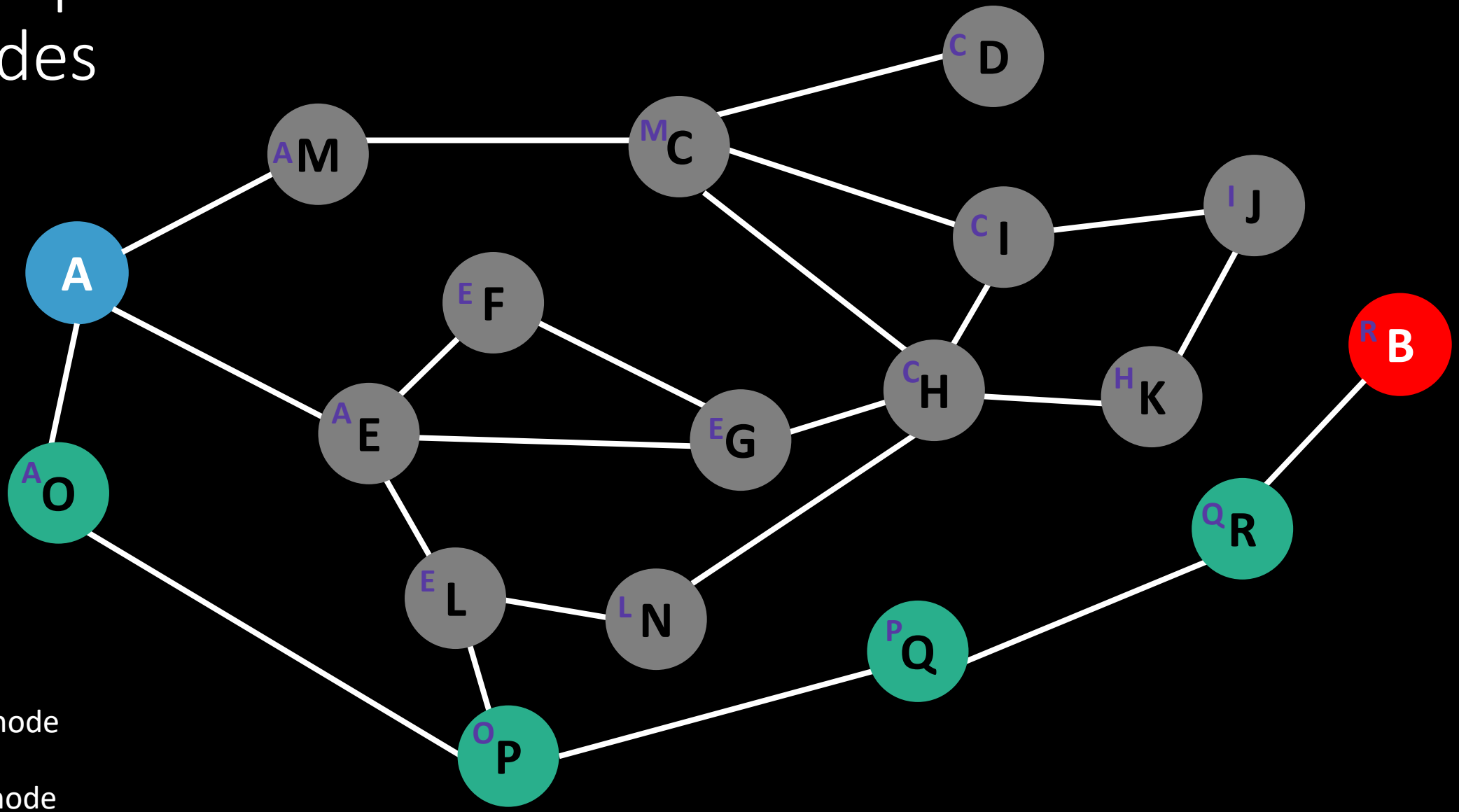


Explored node

Path taken

MO: Finish exploring current level before deepening the search

Shortest path between two nodes



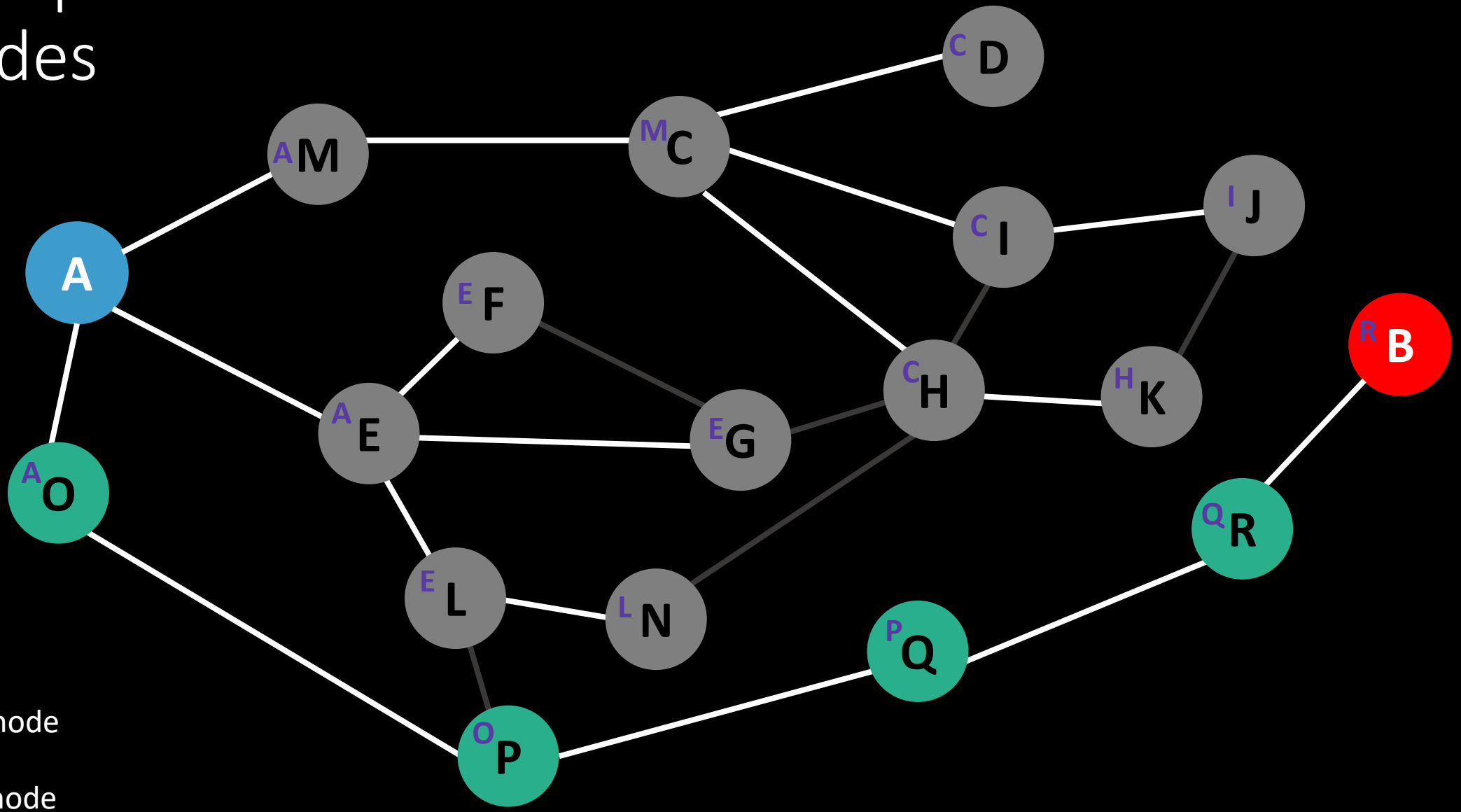
* Previous node

Explored node

Path taken

MO: Finish exploring current level before deepening the search

Shortest path between two nodes



* Previous node

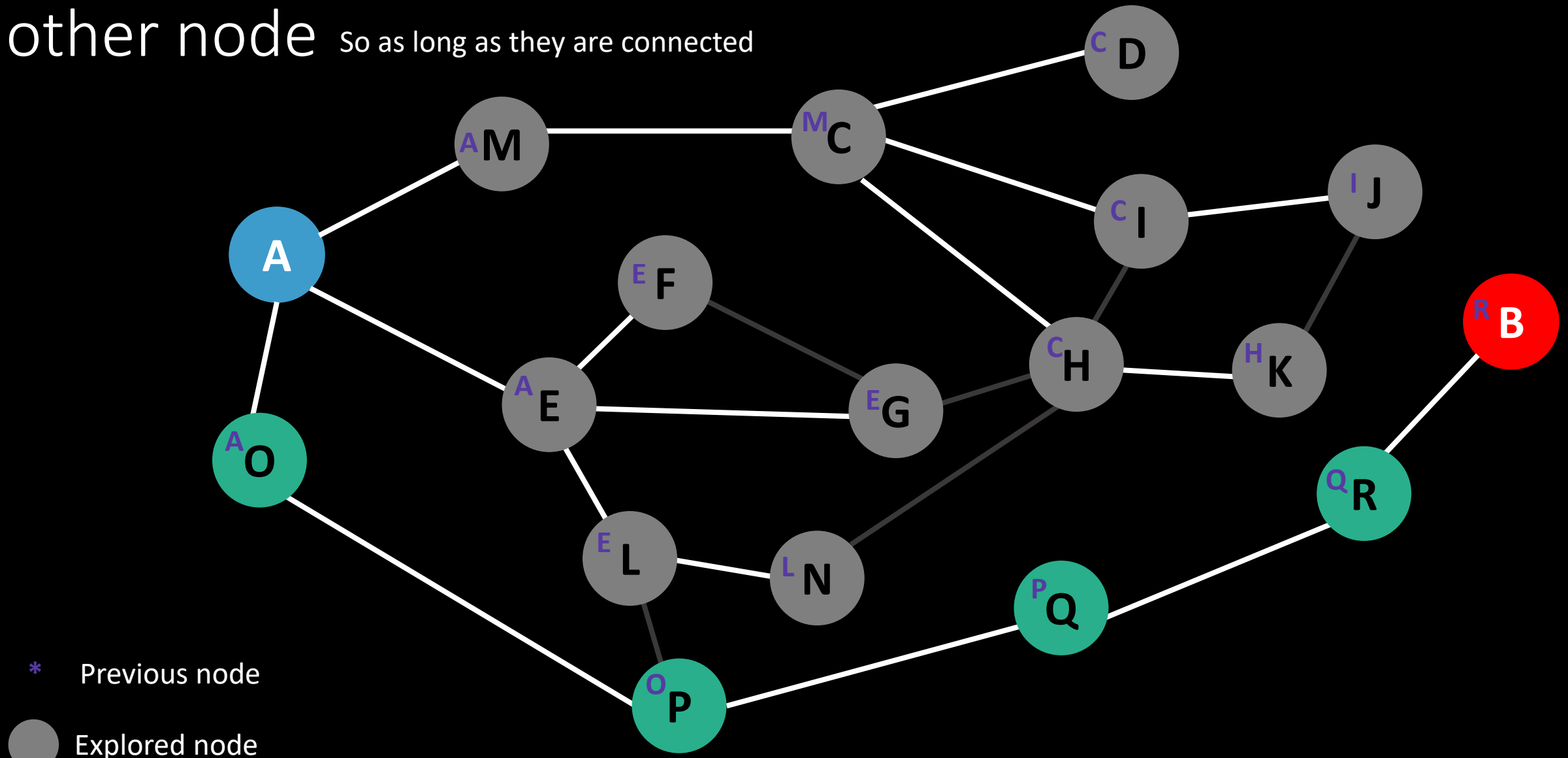
Explored node

Path taken

MO: Finish exploring current level before deepening the search

Shortest path between the starting node and **any** other node

So as long as they are connected

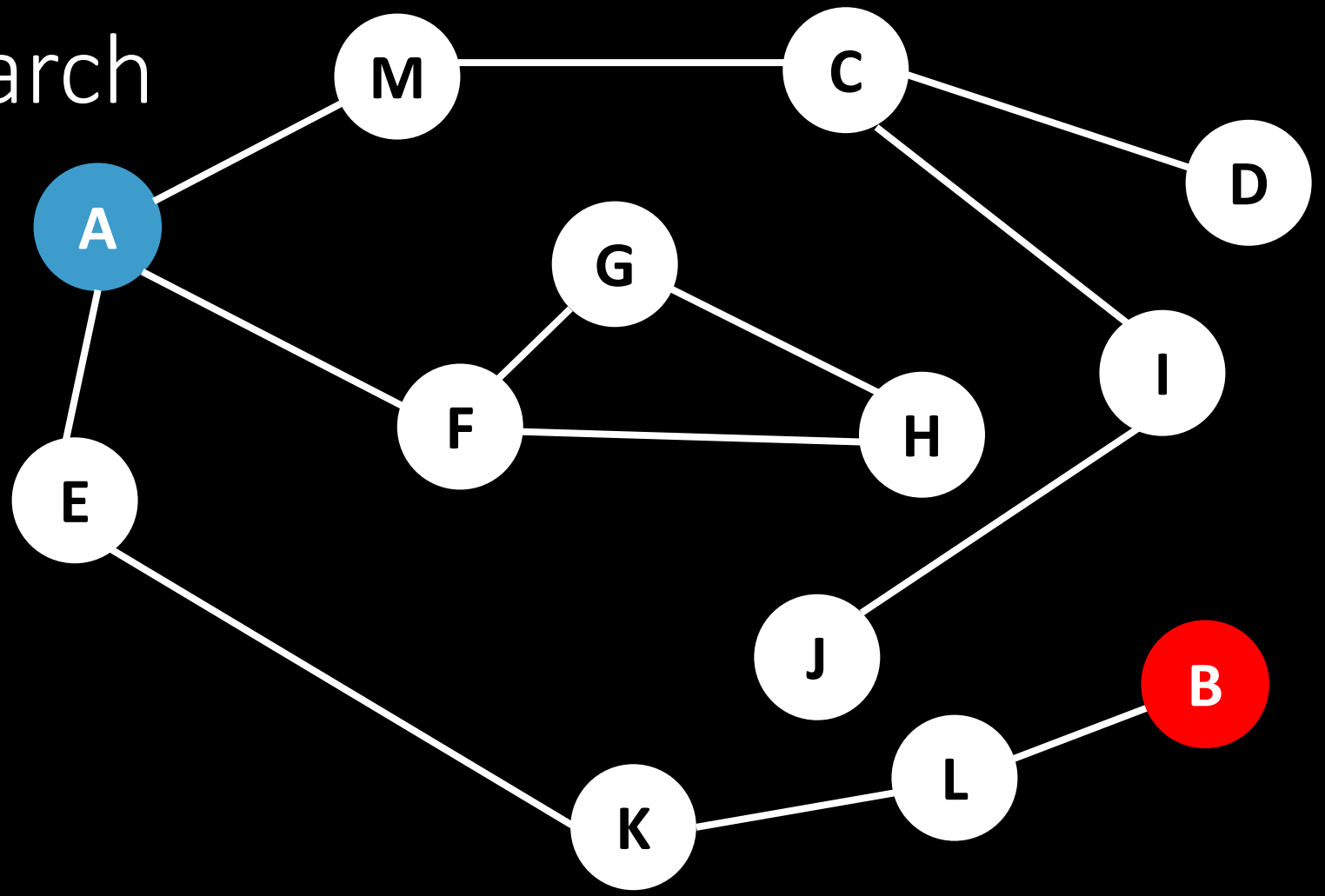


MO: Finish exploring current level before deepening the search

Breadth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (enqueue)
3. **Dequeue** a node from the queue
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left in the queue



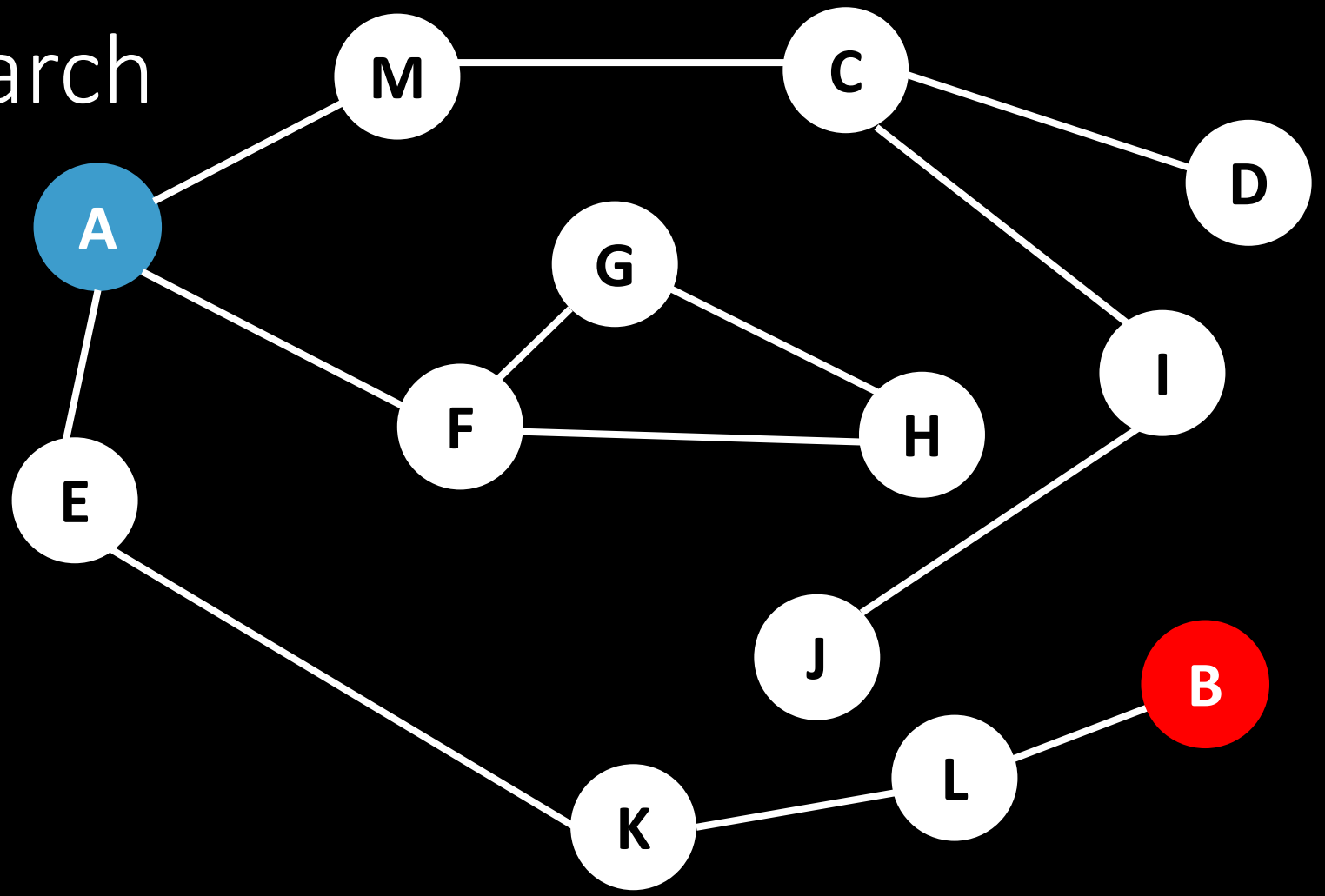
Search Frontier

A							

Breadth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (enqueue)
3. **Dequeue** a node from the queue
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left in the queue



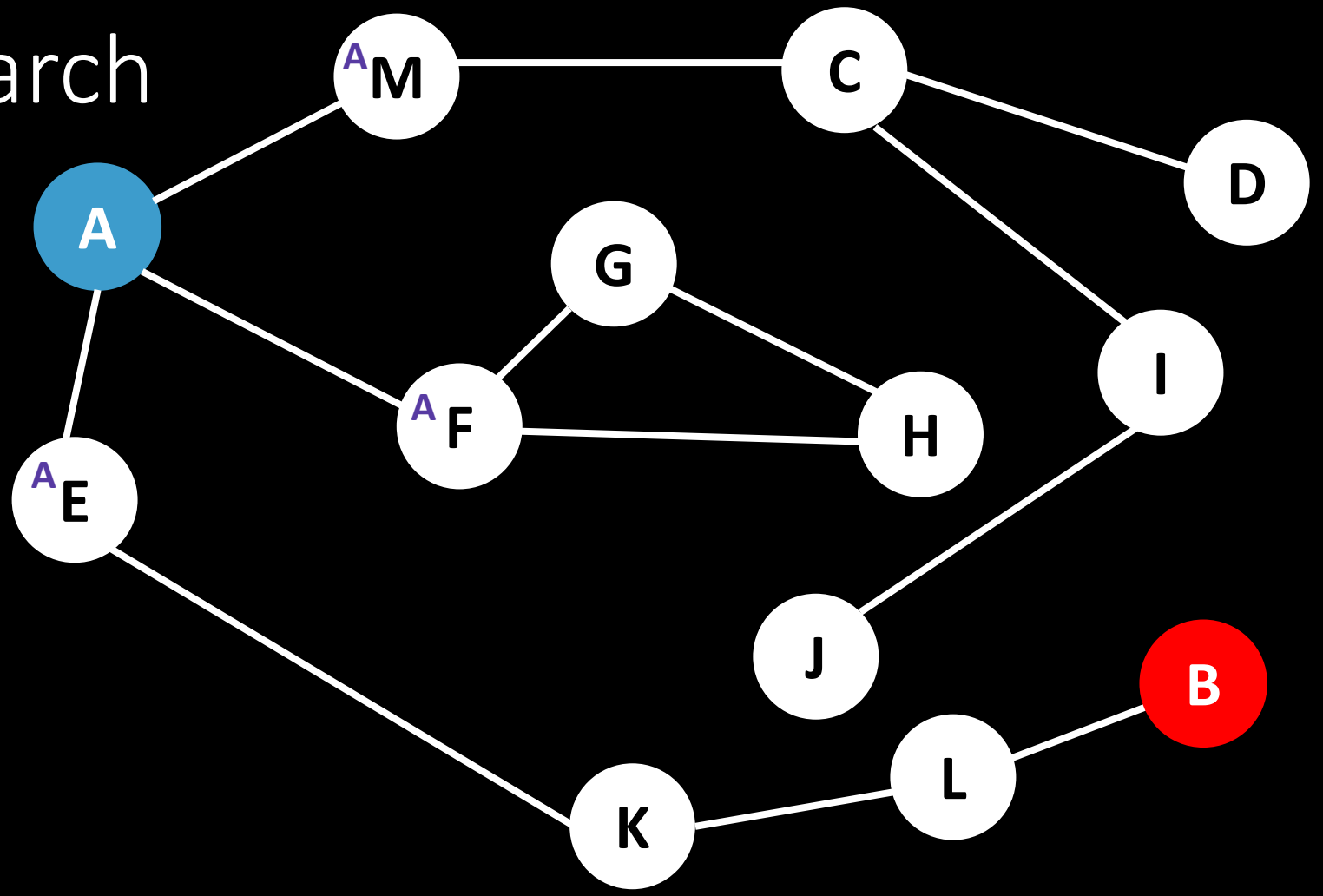
Search Frontier

~~A~~	M	F	E				

Breadth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (enqueue)
3. **Dequeue** a node from the queue
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left in the queue



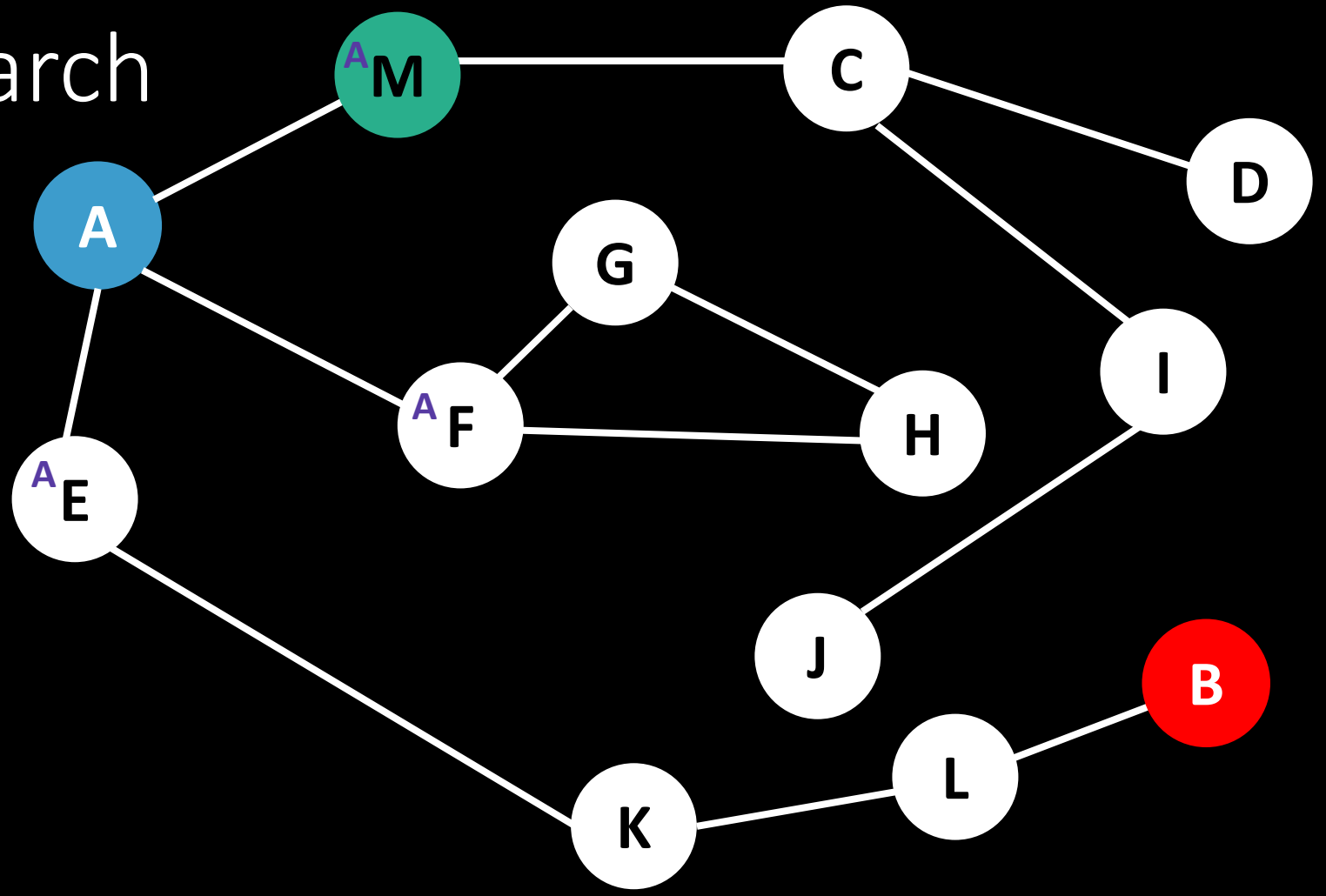
Search Frontier

~~A~~	M	F	E				

Breadth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (enqueue)
3. **Dequeue** a node from the queue
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left in the queue



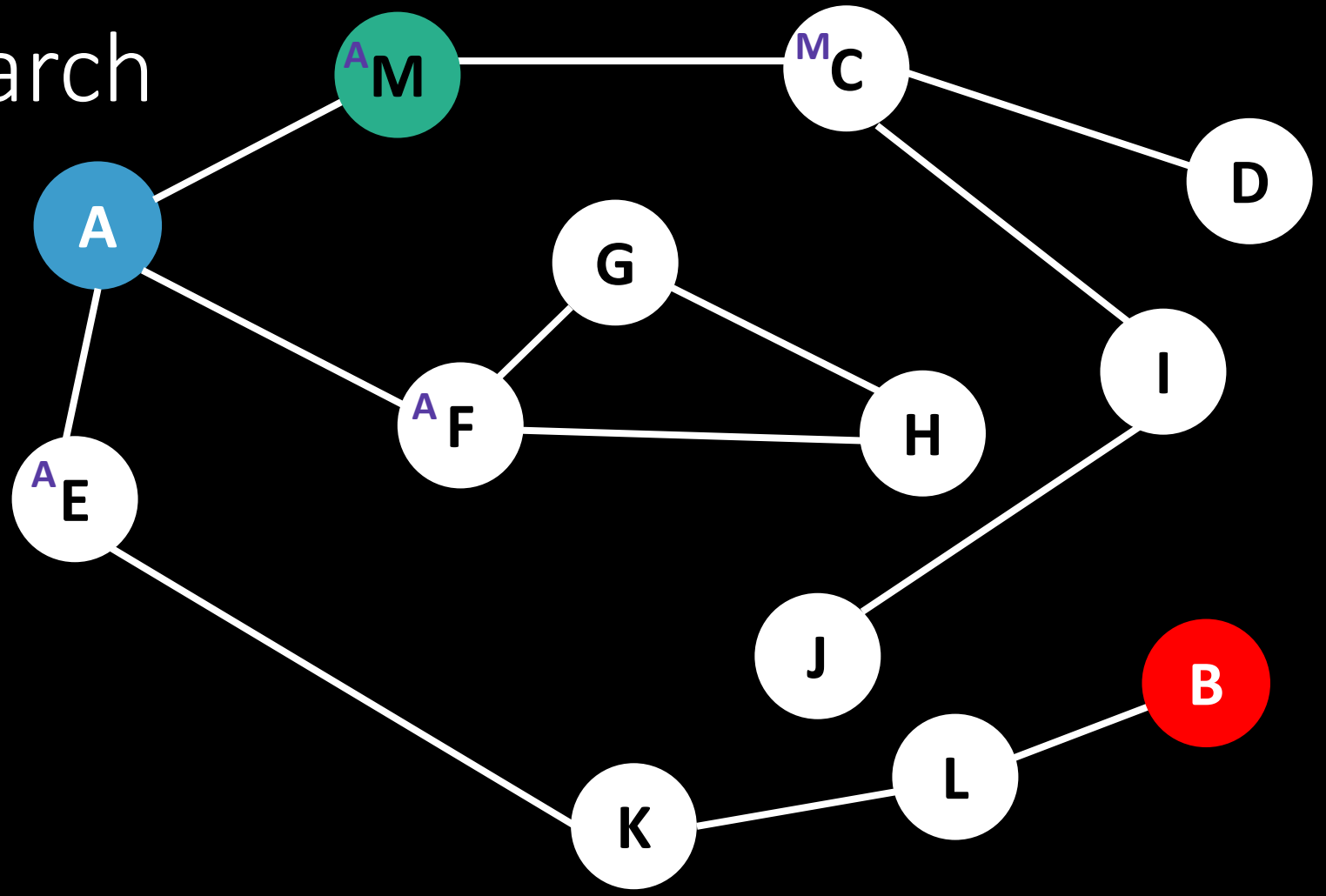
Search Frontier

A	M	F	E	C			

Breadth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (enqueue)
3. **Dequeue** a node from the queue
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left in the queue



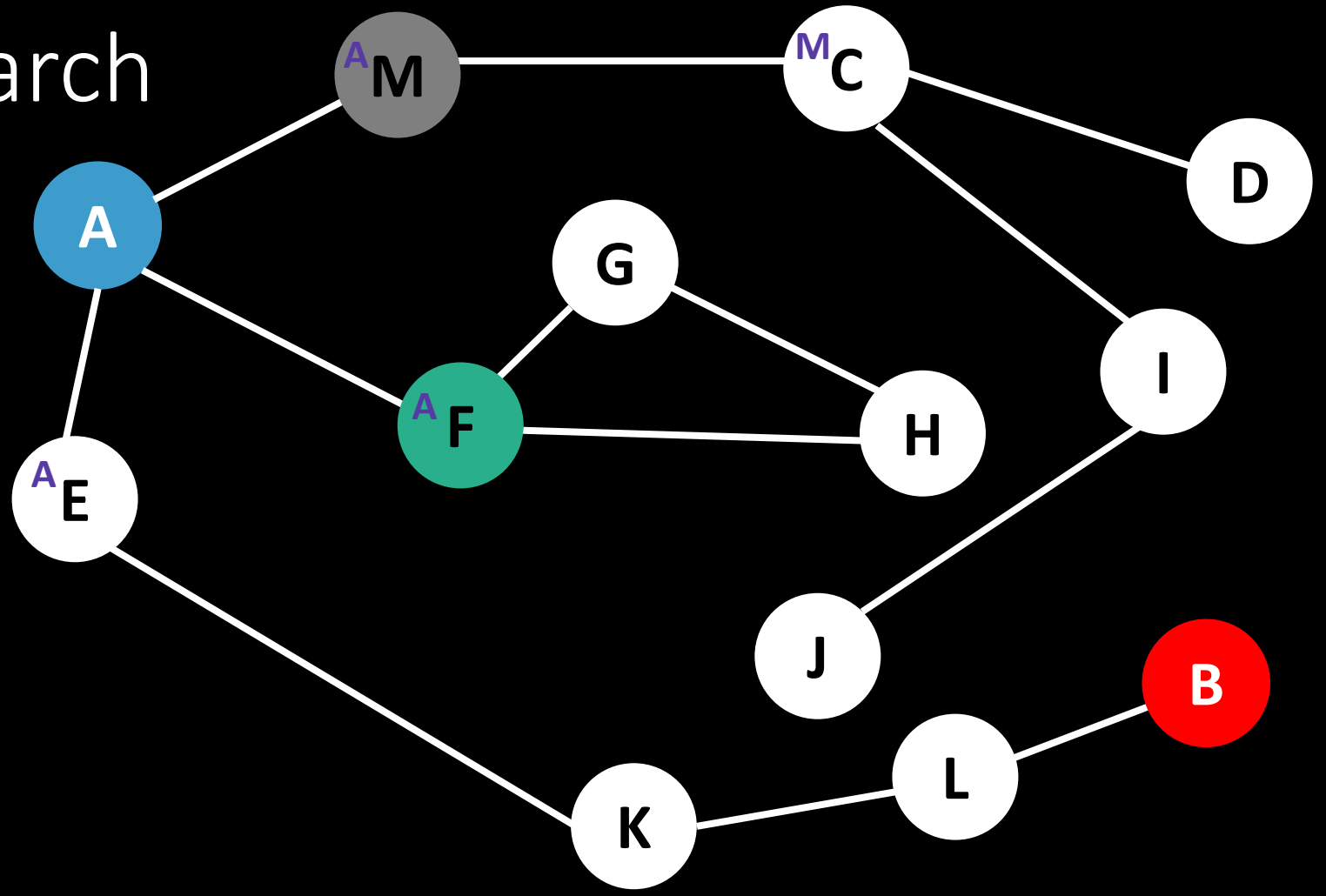
Search Frontier

A	M	F	E	C			

Breadth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (enqueue)
3. **Dequeue** a node from the queue
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left in the queue



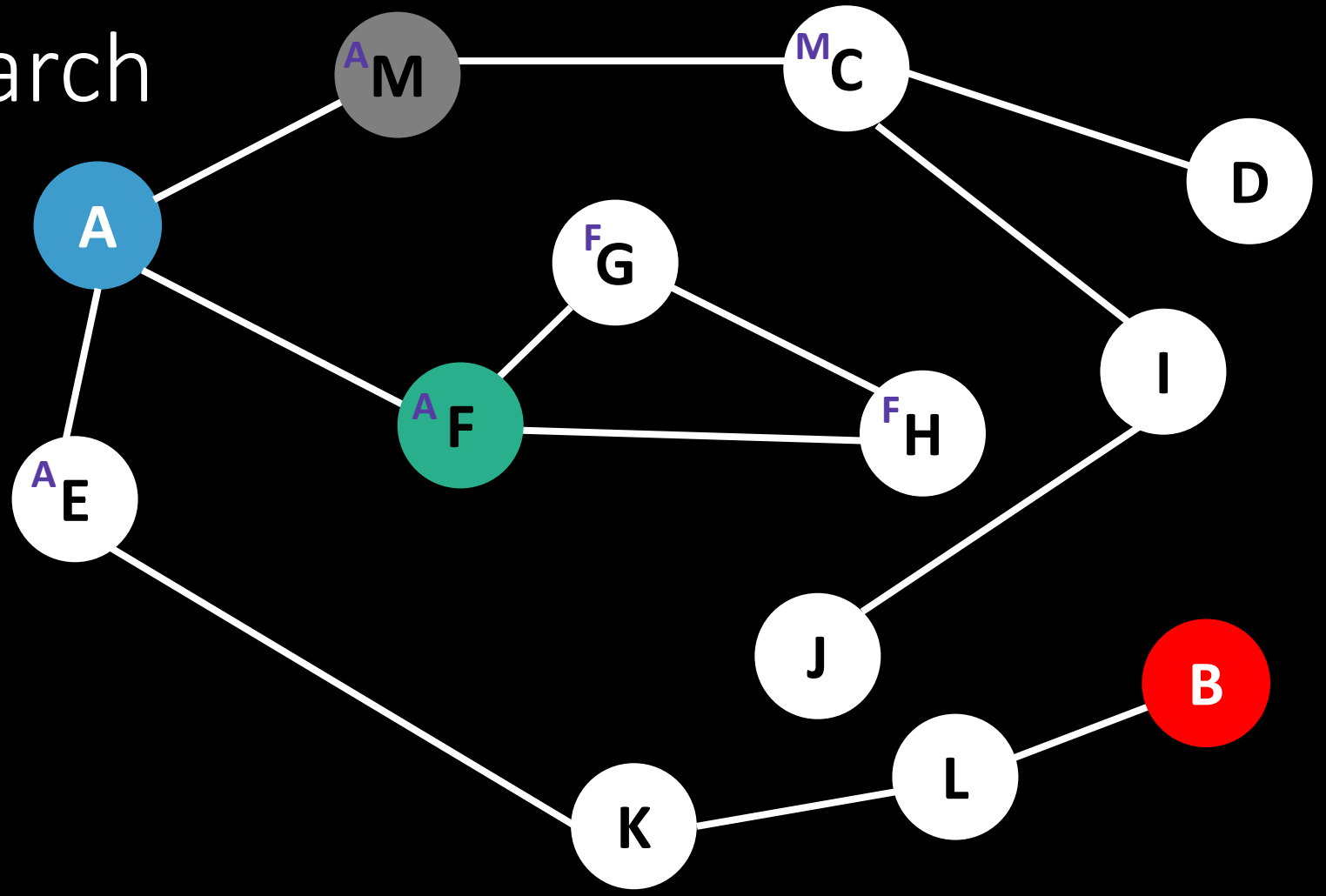
Search Frontier

A	M	F	E	C	G	H	

Breadth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (enqueue)
3. **Dequeue** a node from the queue
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left in the queue



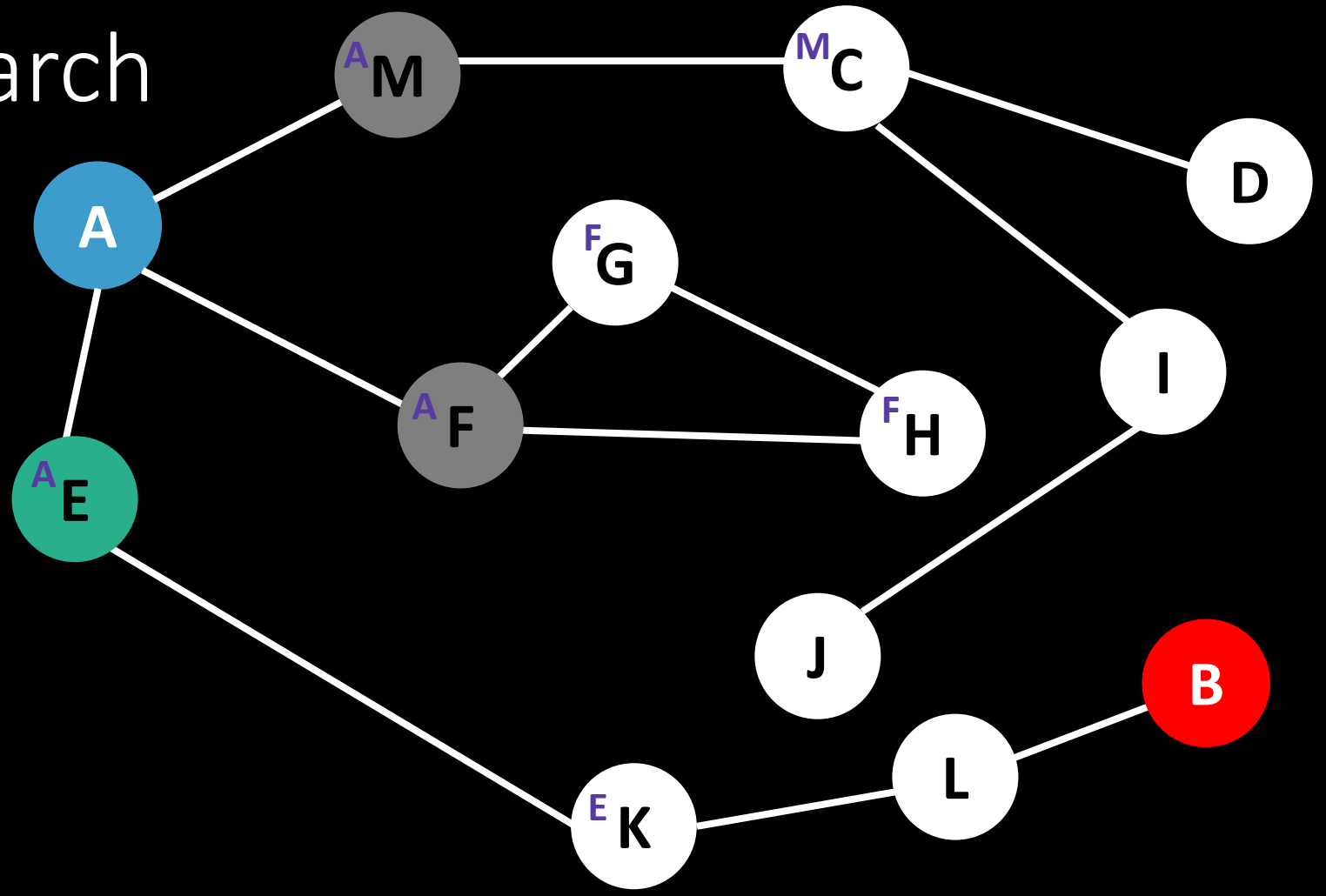
Search Frontier

A	M	F	E	C	G	H	

Breadth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (enqueue)
3. **Dequeue** a node from the queue
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left in the queue



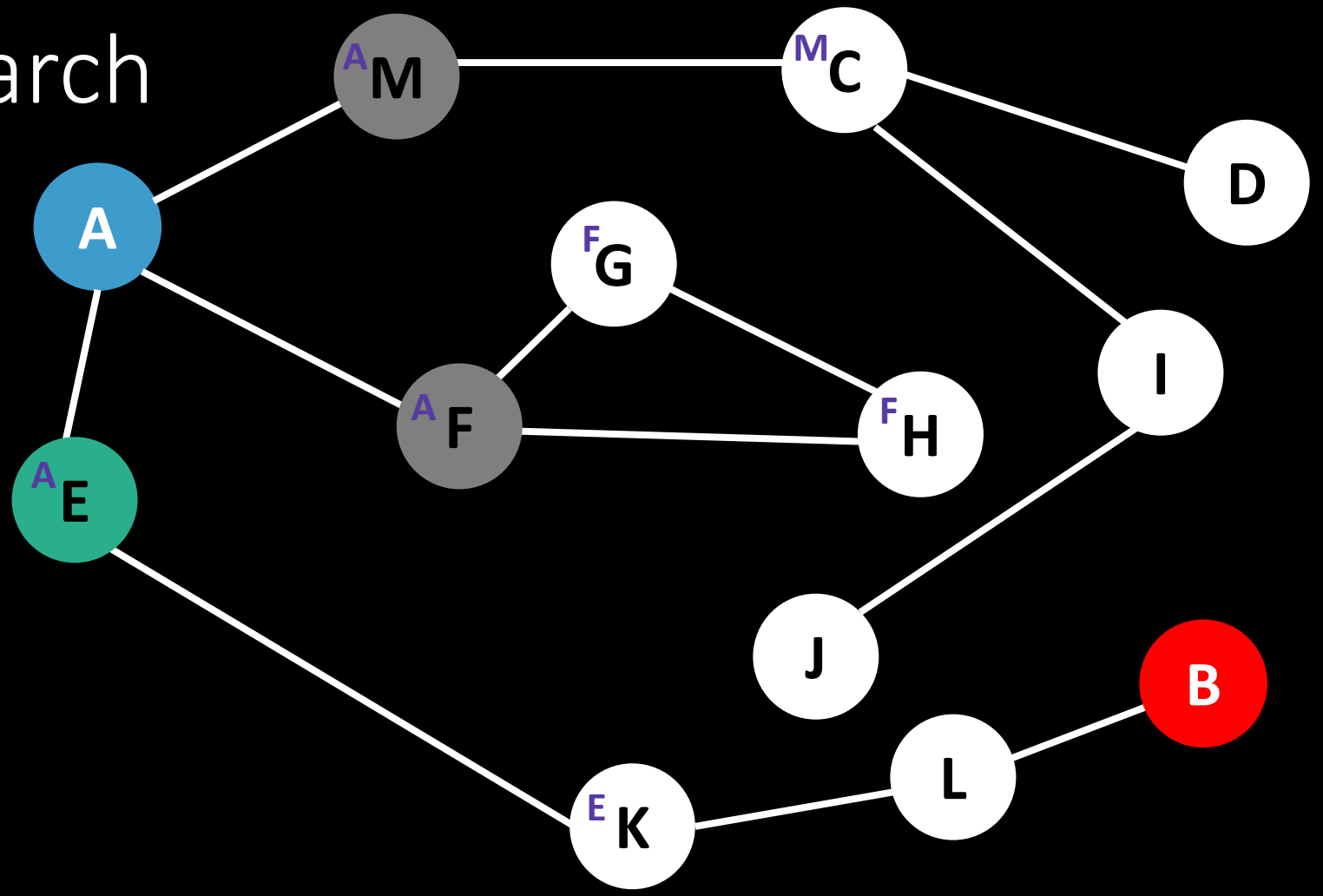
Search Frontier

~~A~~	~~M~~	~~F~~	~~E~~	C	G	H	K

Breadth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (enqueue)
3. **Dequeue** a node from the queue
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left in the queue



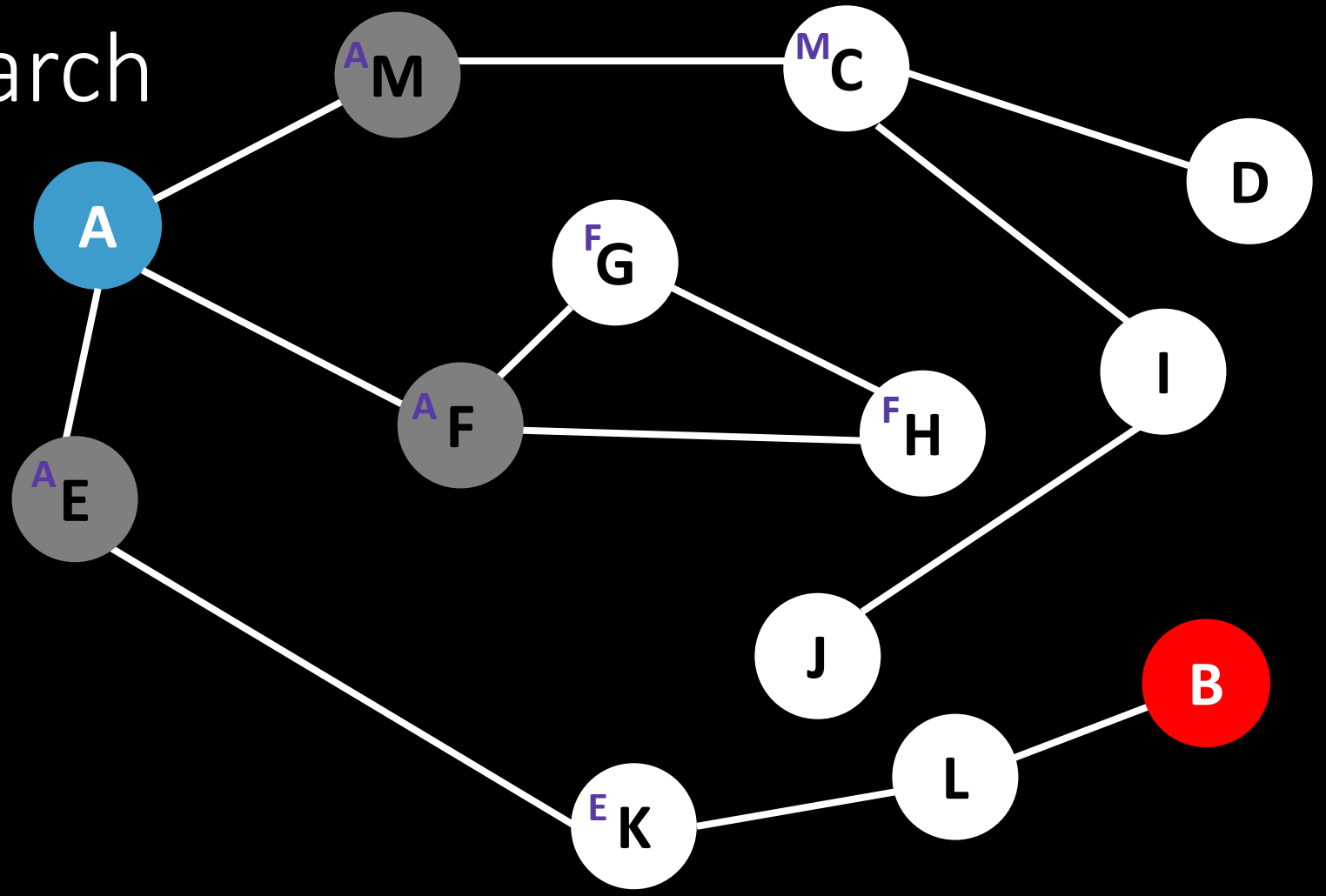
Search Frontier

C	G	H	K				

Breadth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (enqueue)
3. **Dequeue** a node from the queue
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left in the queue



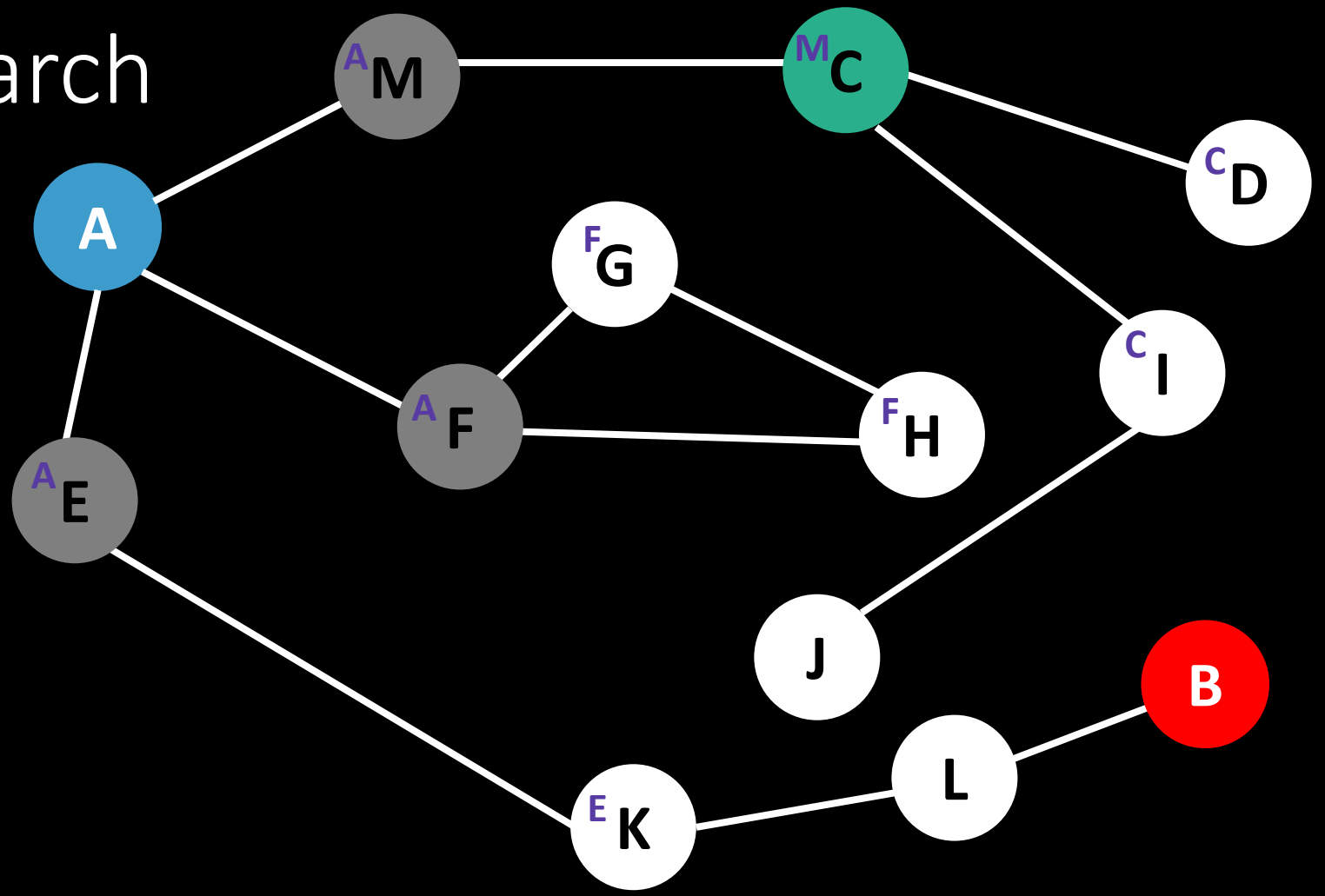
Search Frontier

C	G	H	K				

Breadth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (enqueue)
3. **Dequeue** a node from the queue
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left in the queue



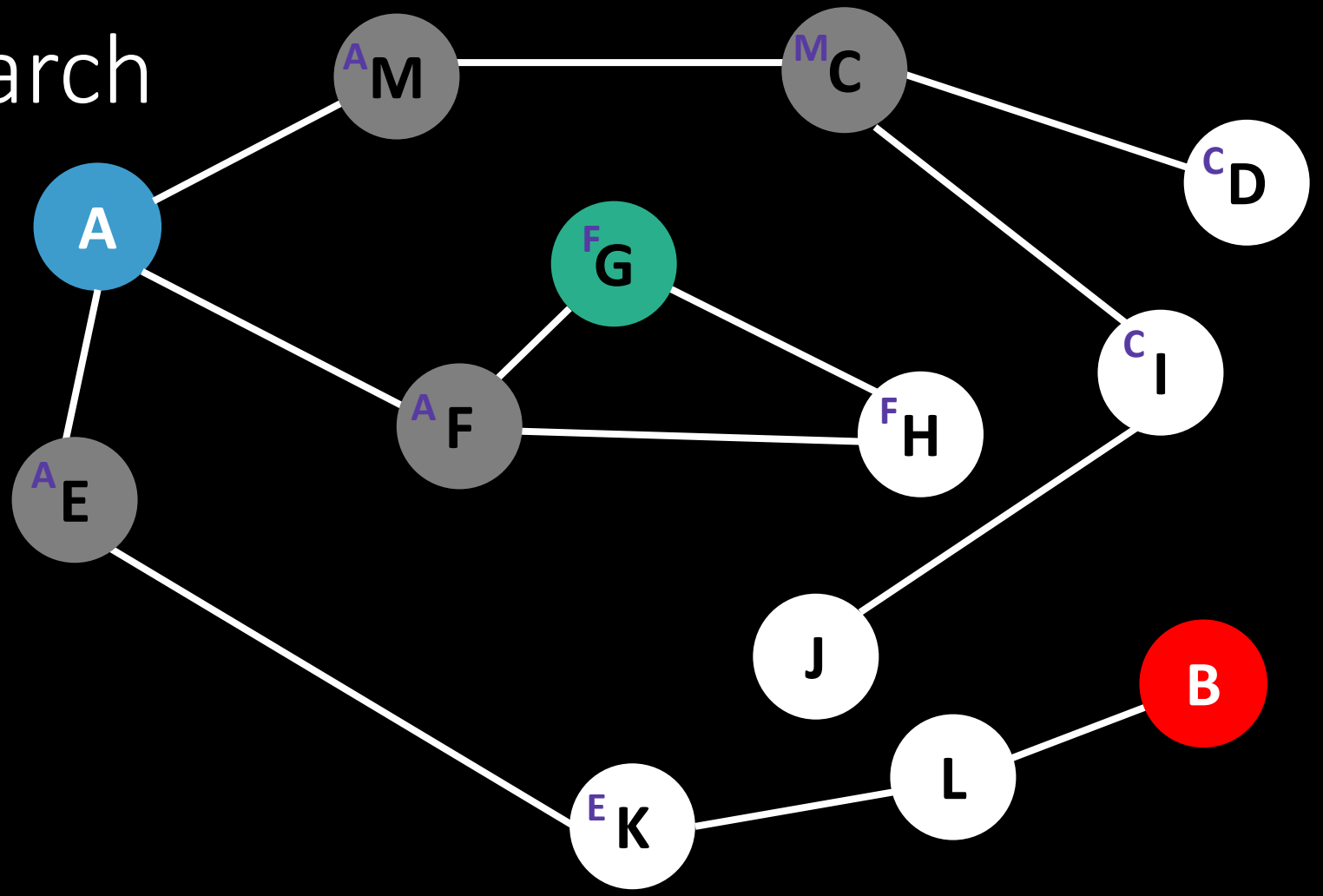
Search Frontier

C	G	H	K	D			

Breadth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (enqueue)
3. **Dequeue** a node from the queue
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left in the queue



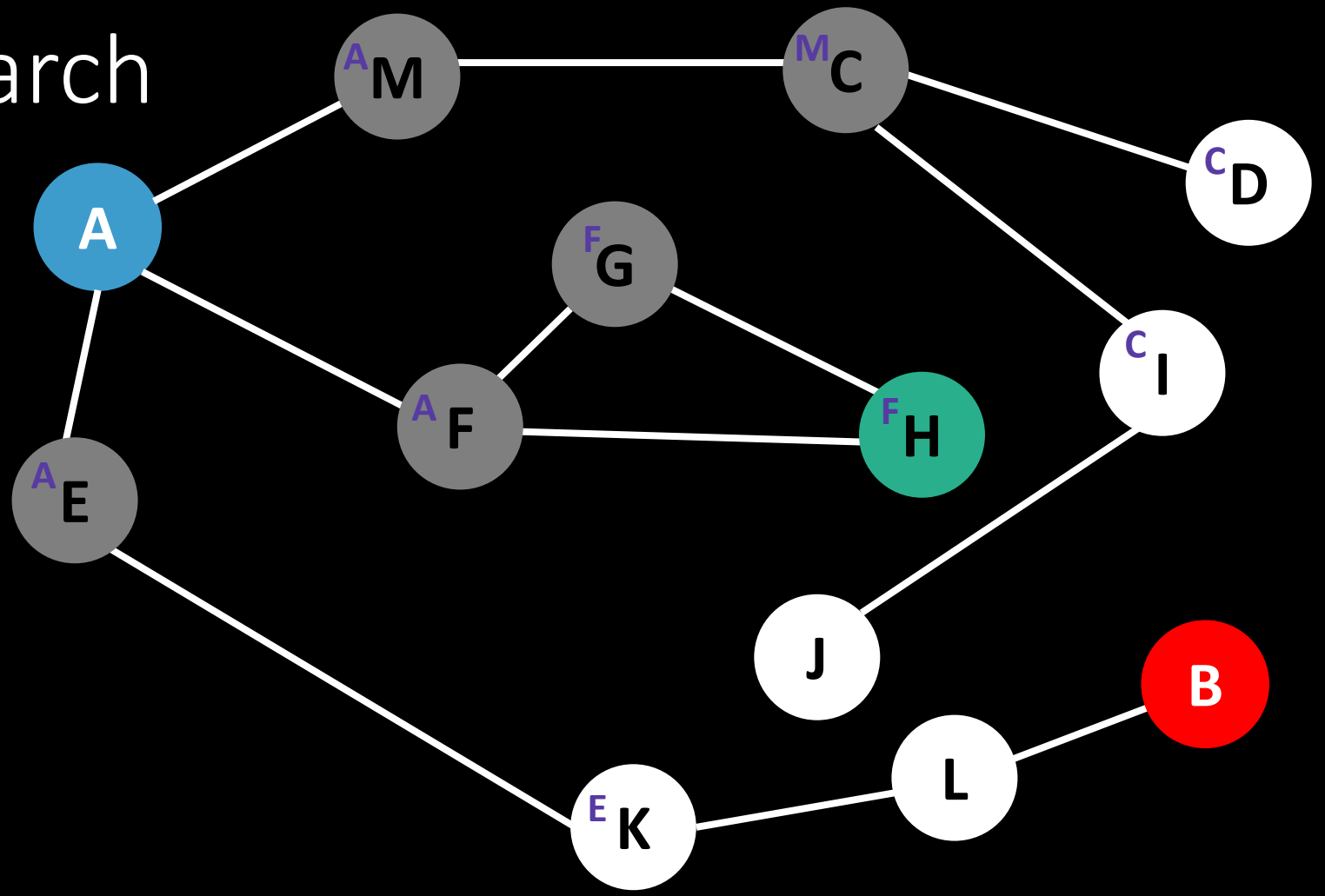
Search Frontier

C	G	H	K	D	I	H	

Breadth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (enqueue)
3. **Dequeue** a node from the queue
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left in the queue



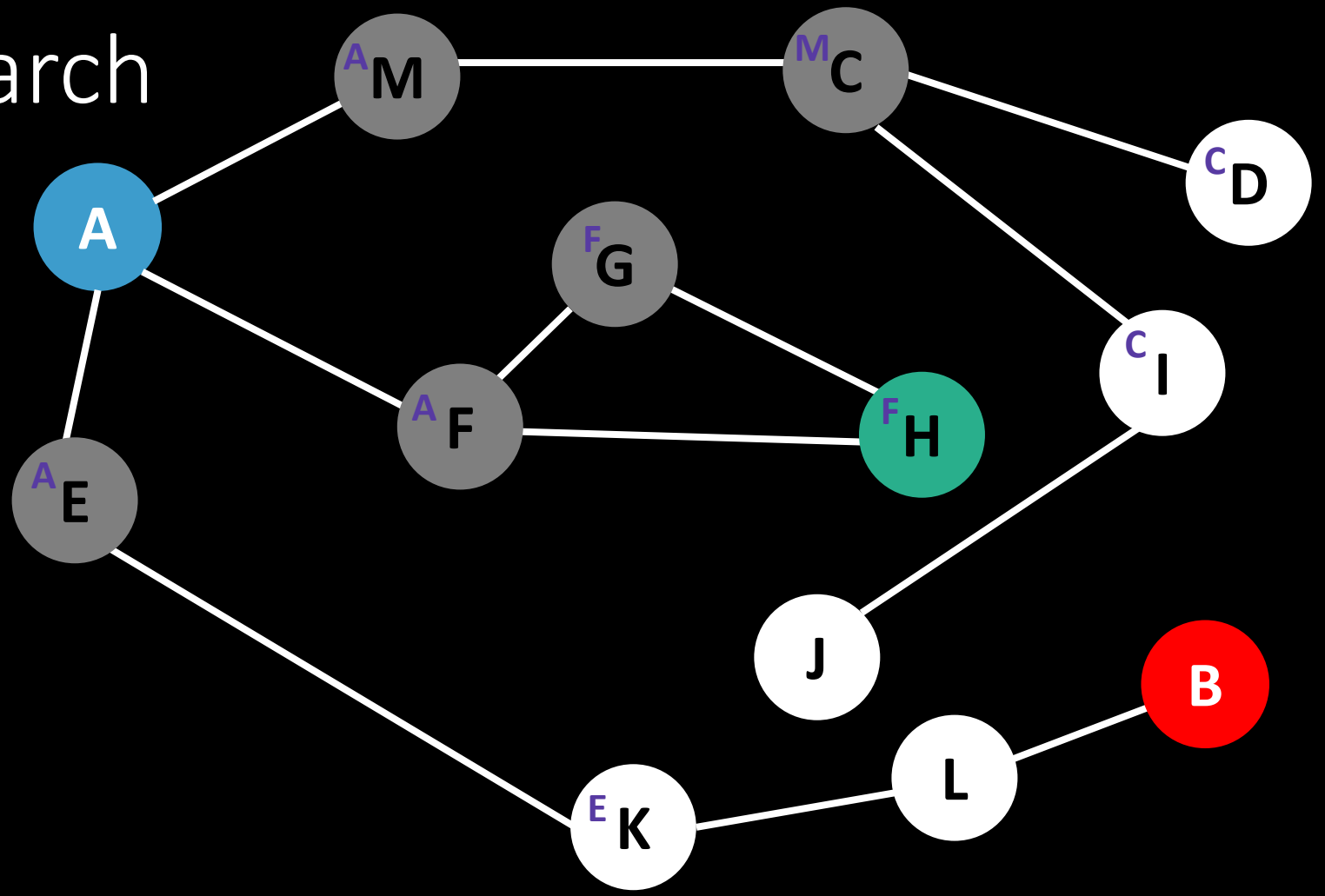
Search Frontier

C	G	M	K	D	I	H	

Breadth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (enqueue)
3. **Dequeue** a node from the queue
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left in the queue



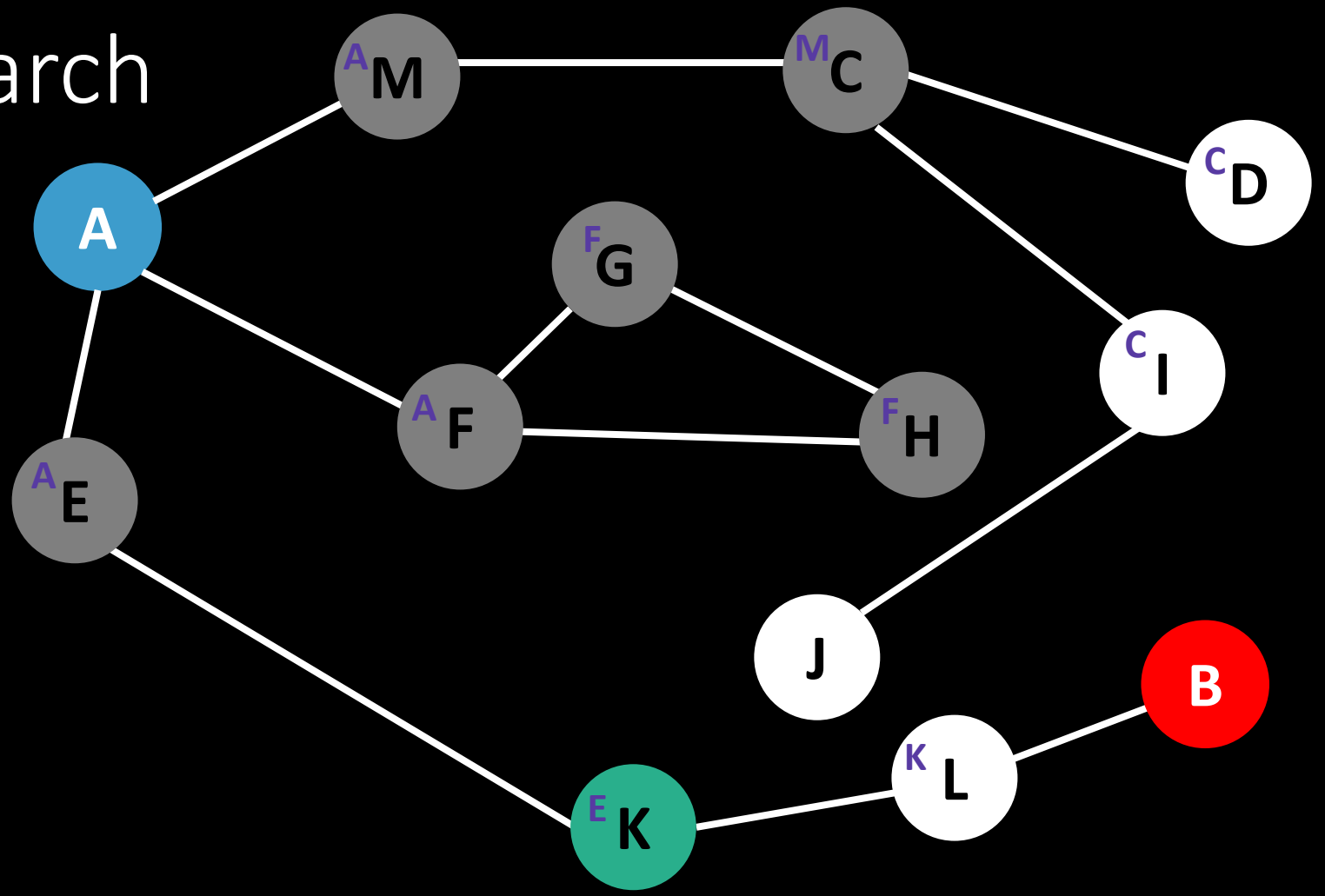
Search Frontier

C	G	M	K	D	I	H	

Breadth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (enqueue)
3. **Dequeue** a node from the queue
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left in the queue



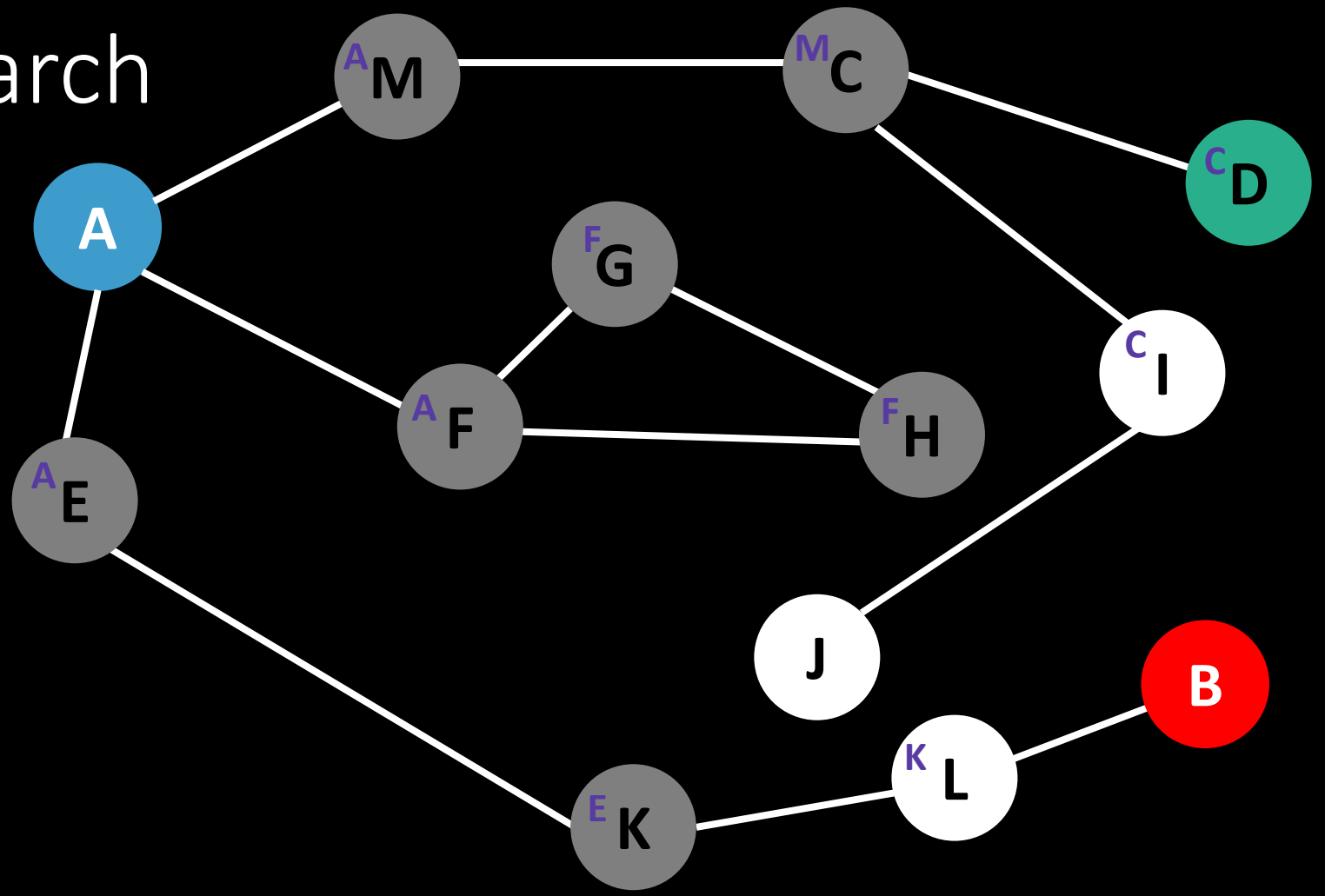
Search Frontier

~~C~~	~~G~~	~~H~~	~~K~~	D	I	H	L

Breadth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (enqueue)
3. **Dequeue** a node from the queue
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left in the queue



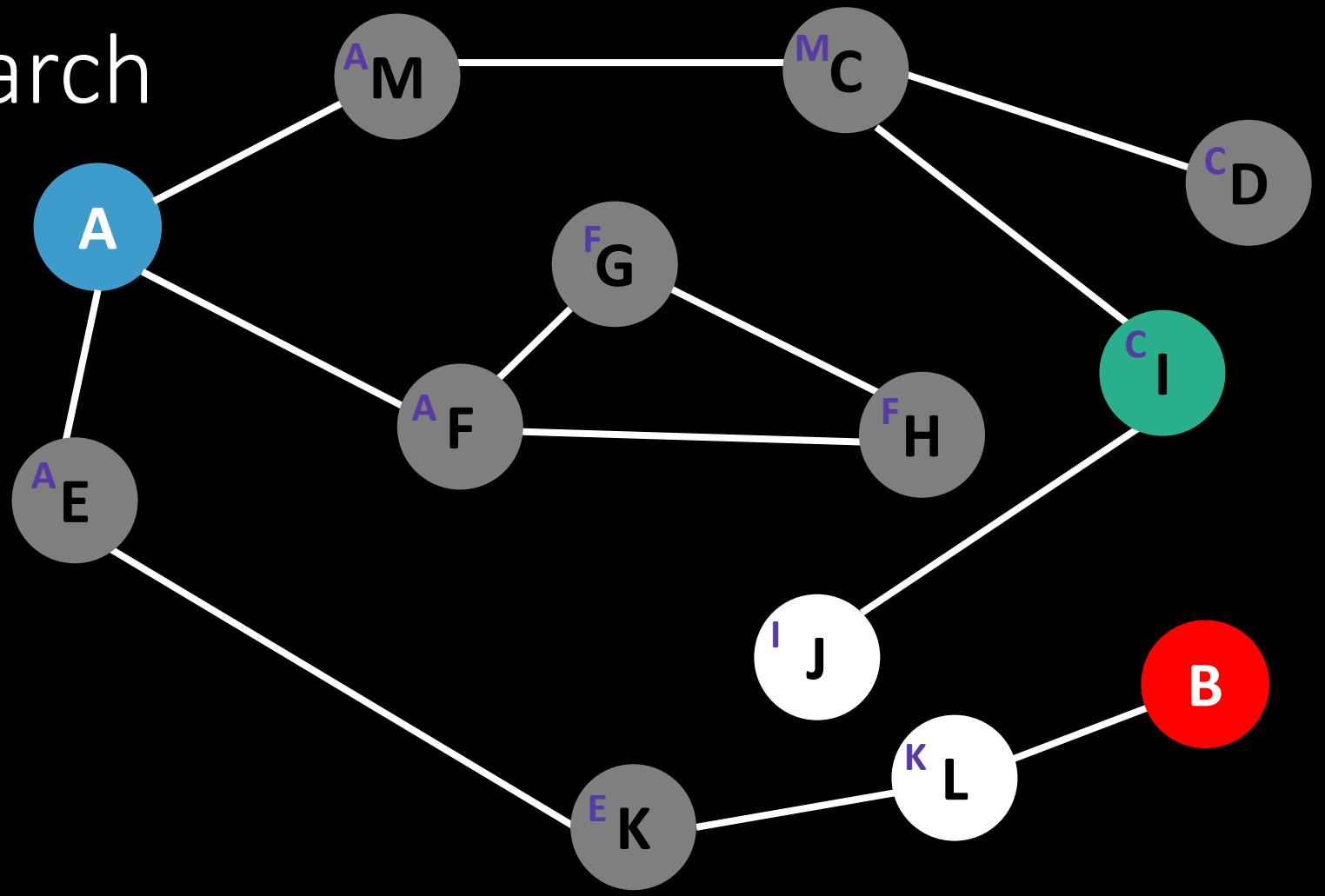
Search Frontier

D	I	H	L				

Breadth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (enqueue)
3. **Dequeue** a node from the queue
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left in the queue



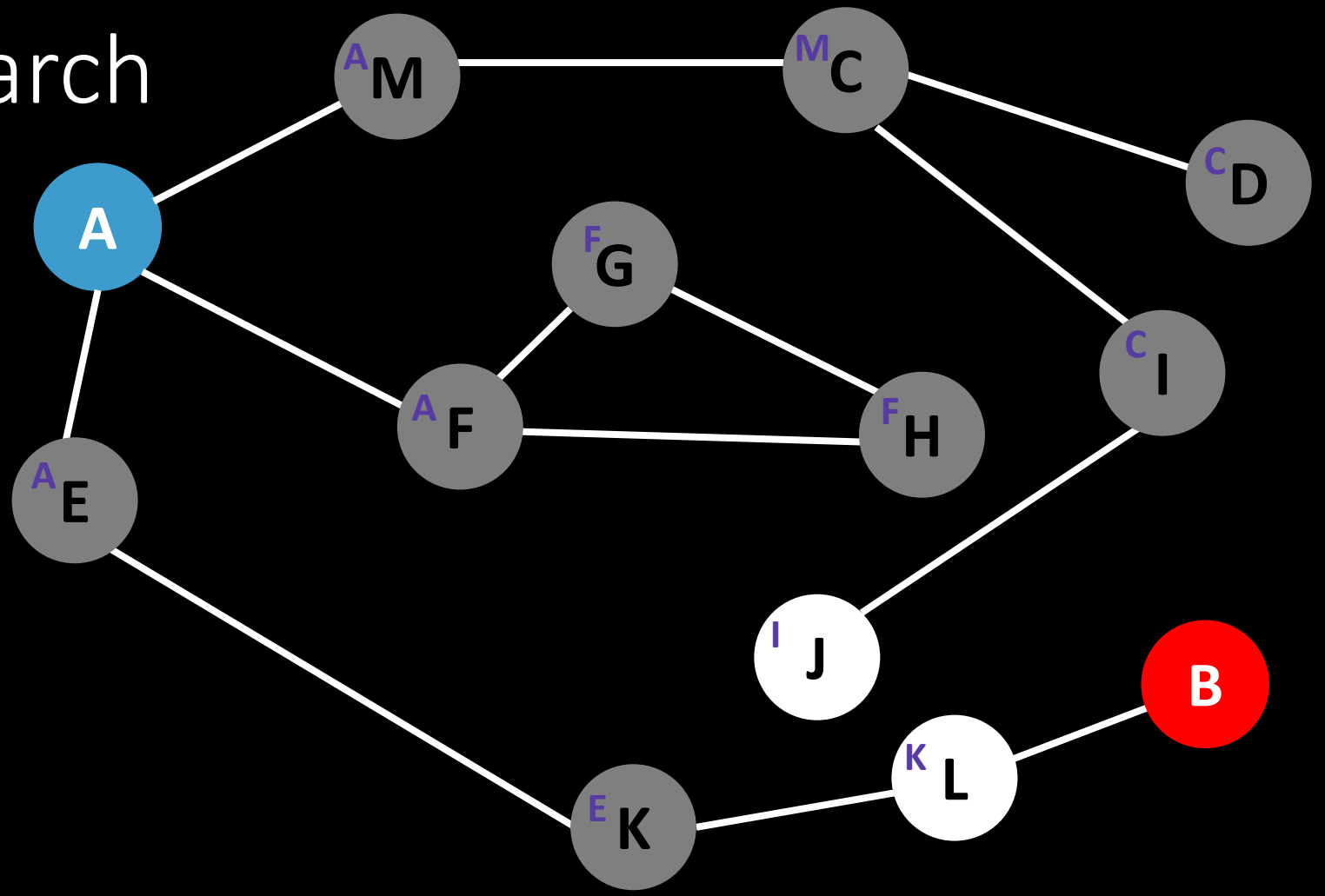
Search Frontier

~~D~~	~~I~~	H	L	J			

Breadth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (enqueue)
3. **Dequeue** a node from the queue
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left in the queue



Already explored, do not re-expand

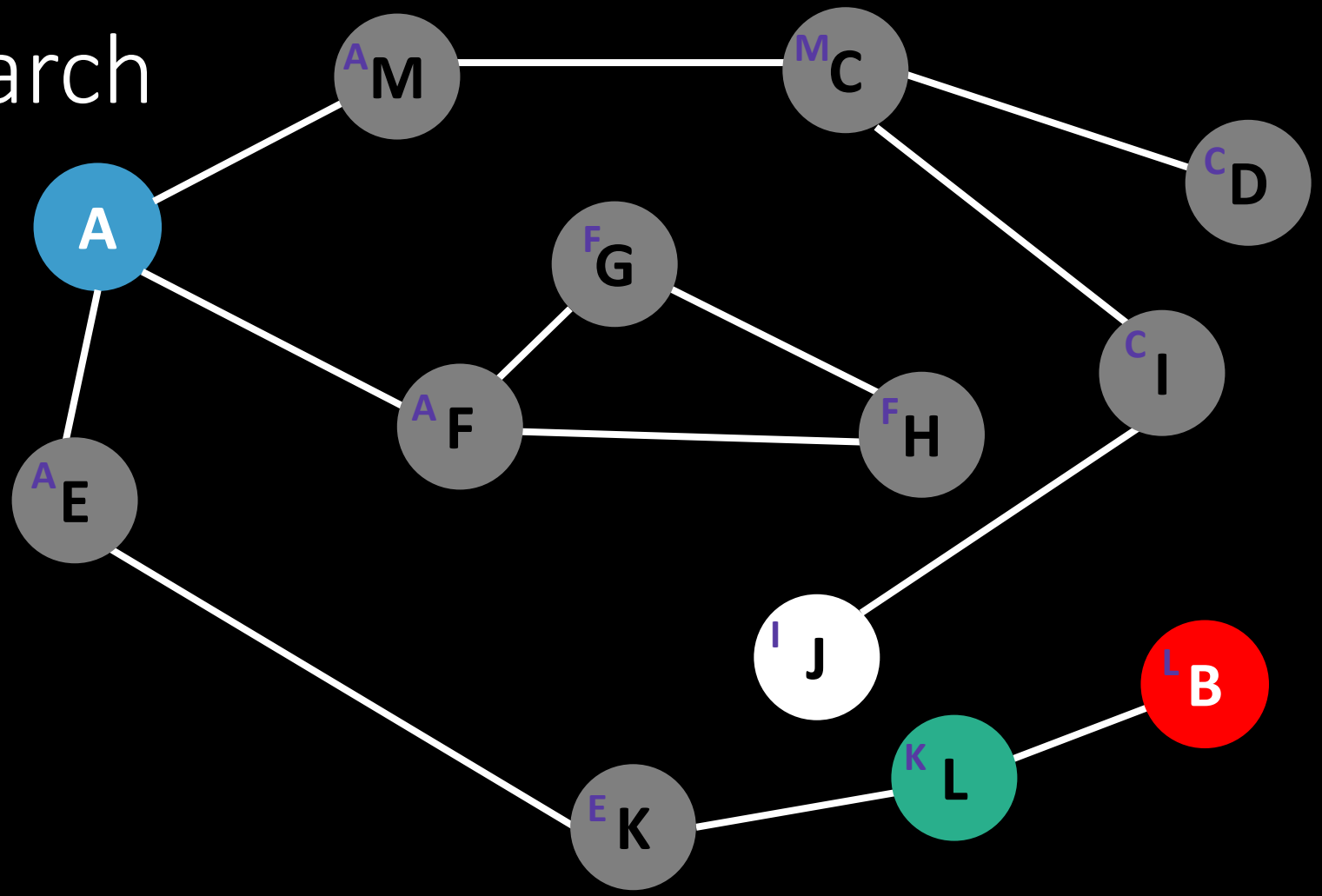
Search Frontier

D	I	H	L	J			

Breadth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (enqueue)
3. **Dequeue** a node from the queue
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left in the queue



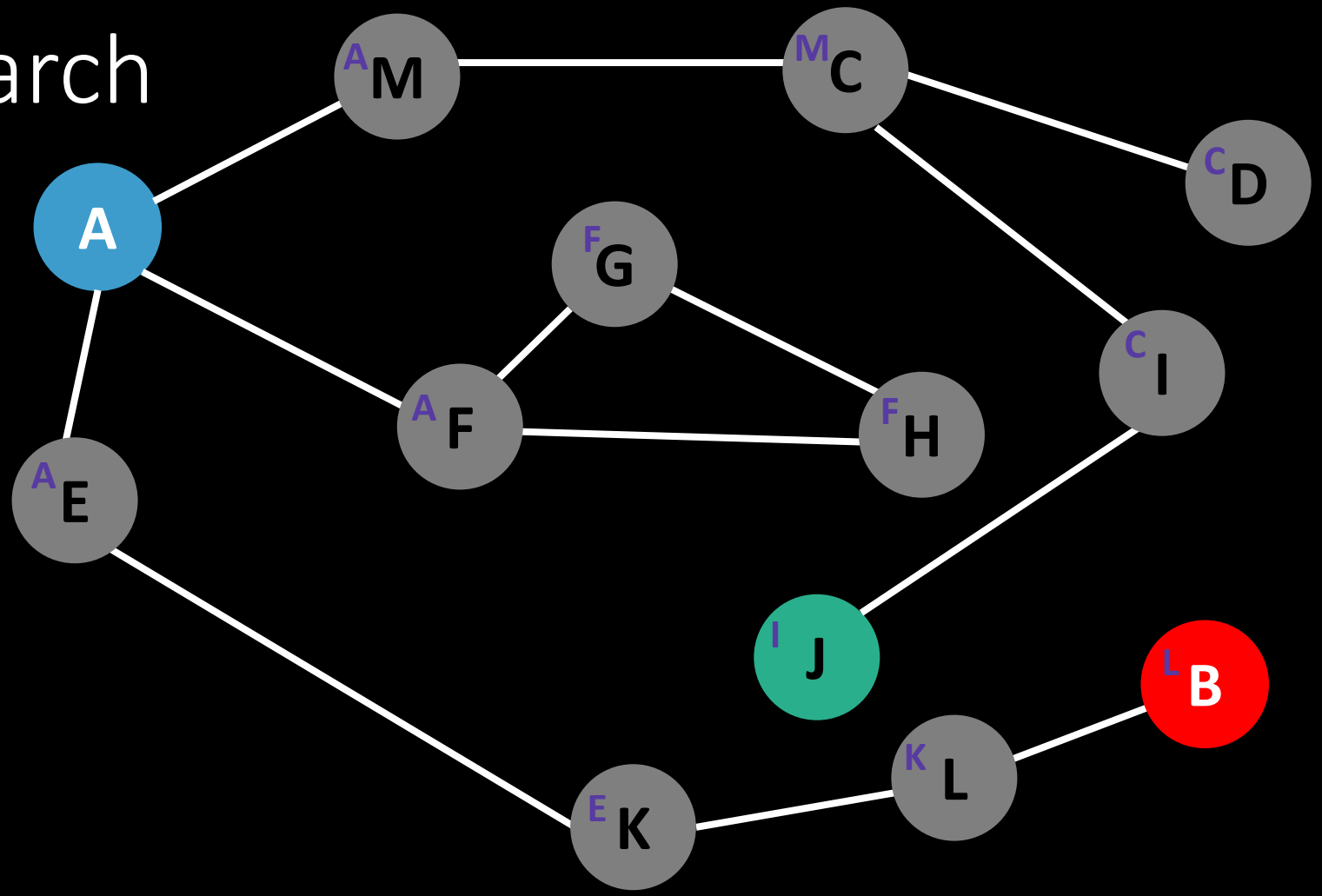
Search Frontier

D	I	H	L	J	B		

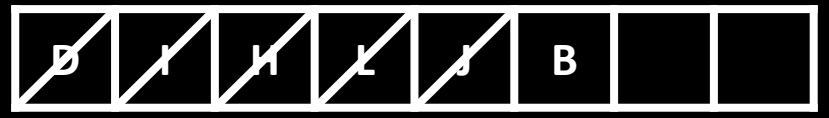
Breadth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (enqueue)
3. **Dequeue** a node from the queue
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left in the queue



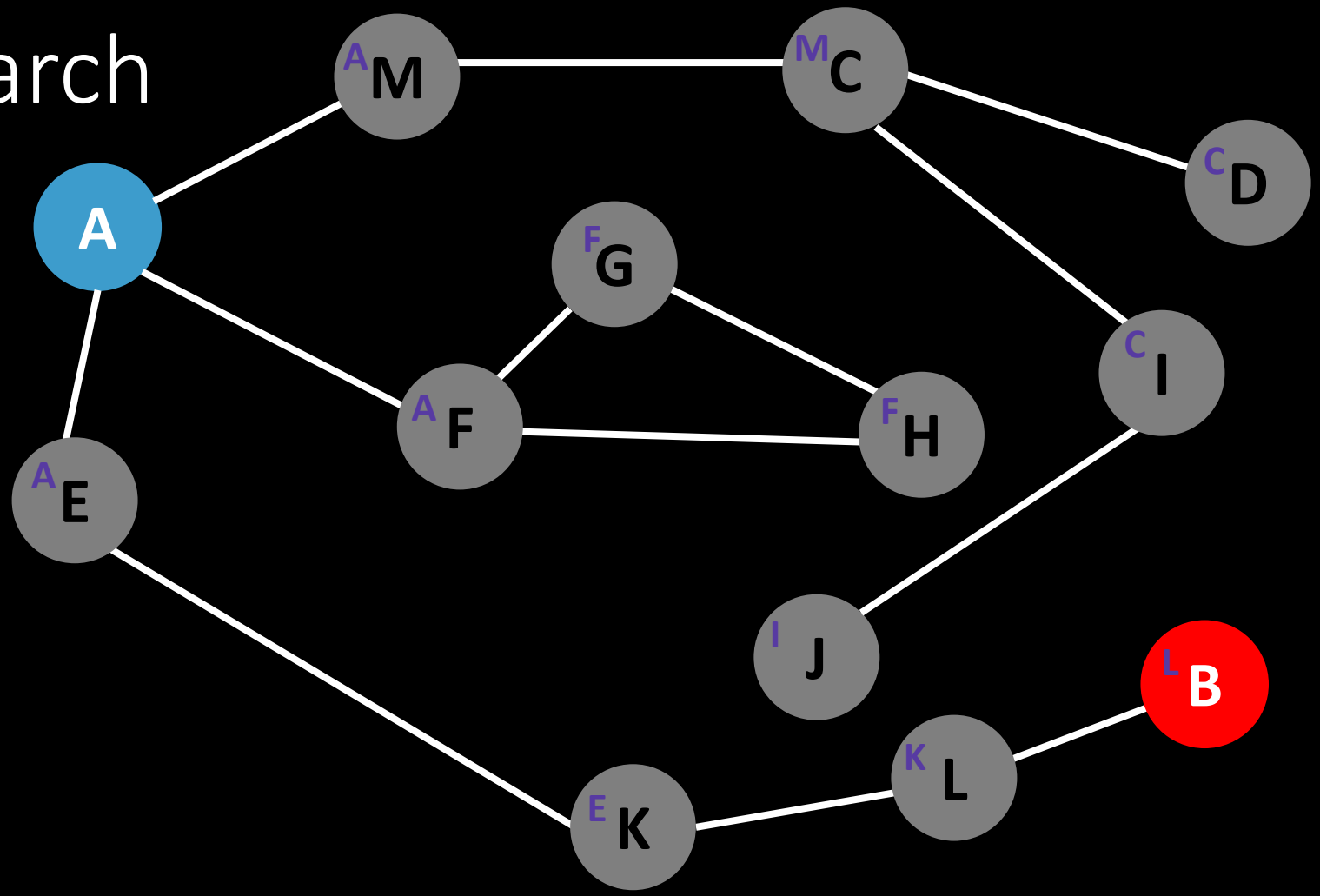
Search Frontier



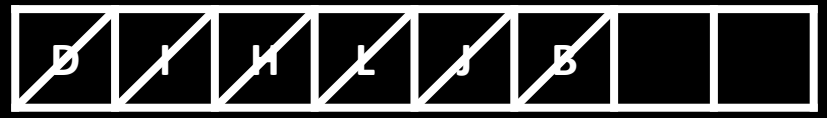
Breadth-First Search

Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (enqueue)
3. **Dequeue** a node from the queue
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there are no nodes left in the queue



Search Frontier

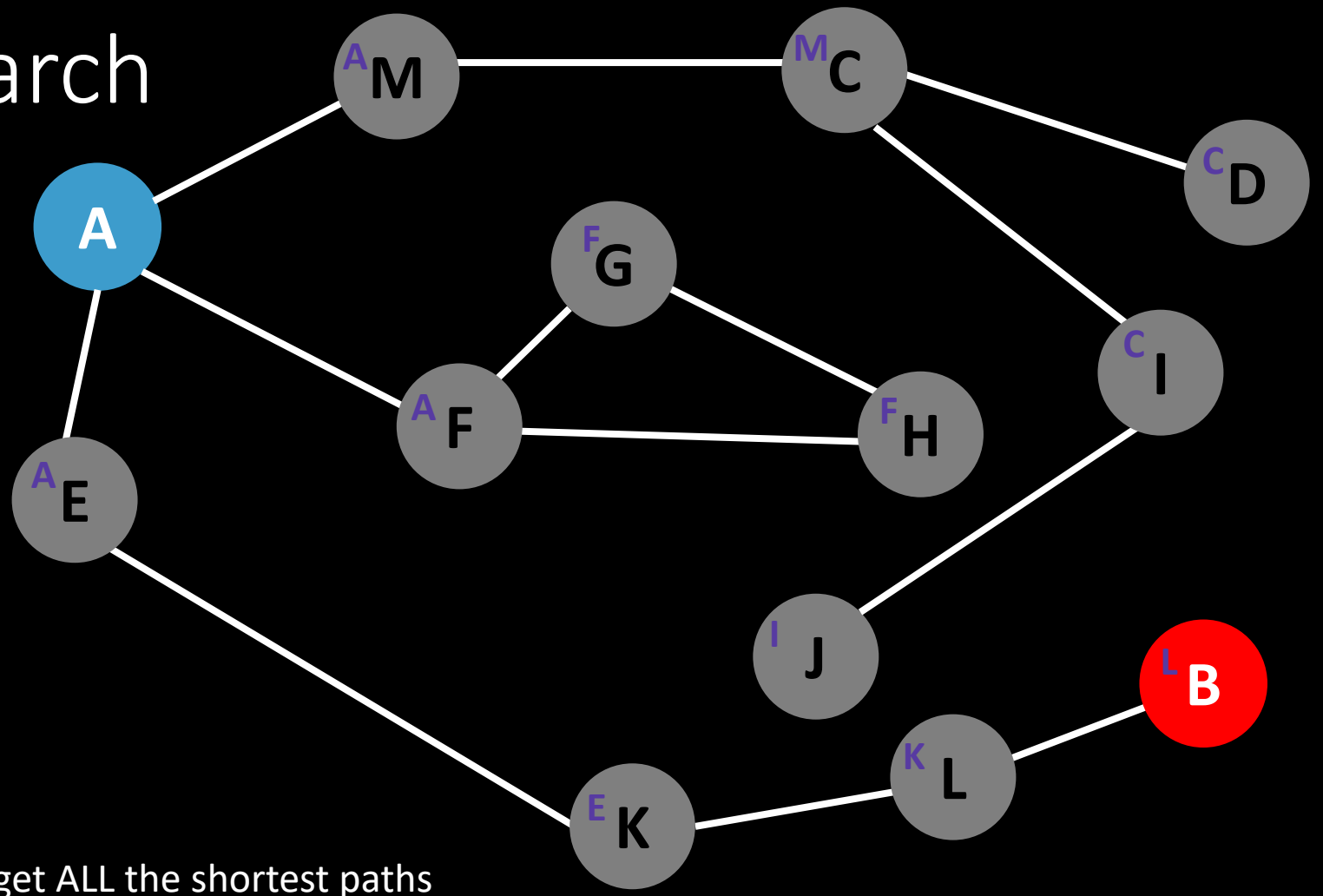


Breadth-First Search

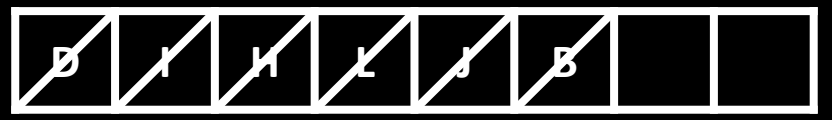
Algorithm

1. **Expand** starting node
2. Add all neighboring nodes to the **search frontier** (enqueue)
3. **Dequeue** a node from the queue
4. **Expand** the node
5. Repeat 2-4 until : the ending node is found OR there **are no nodes left in the queue**

} To get ALL the shortest paths from the starting node



Search Frontier

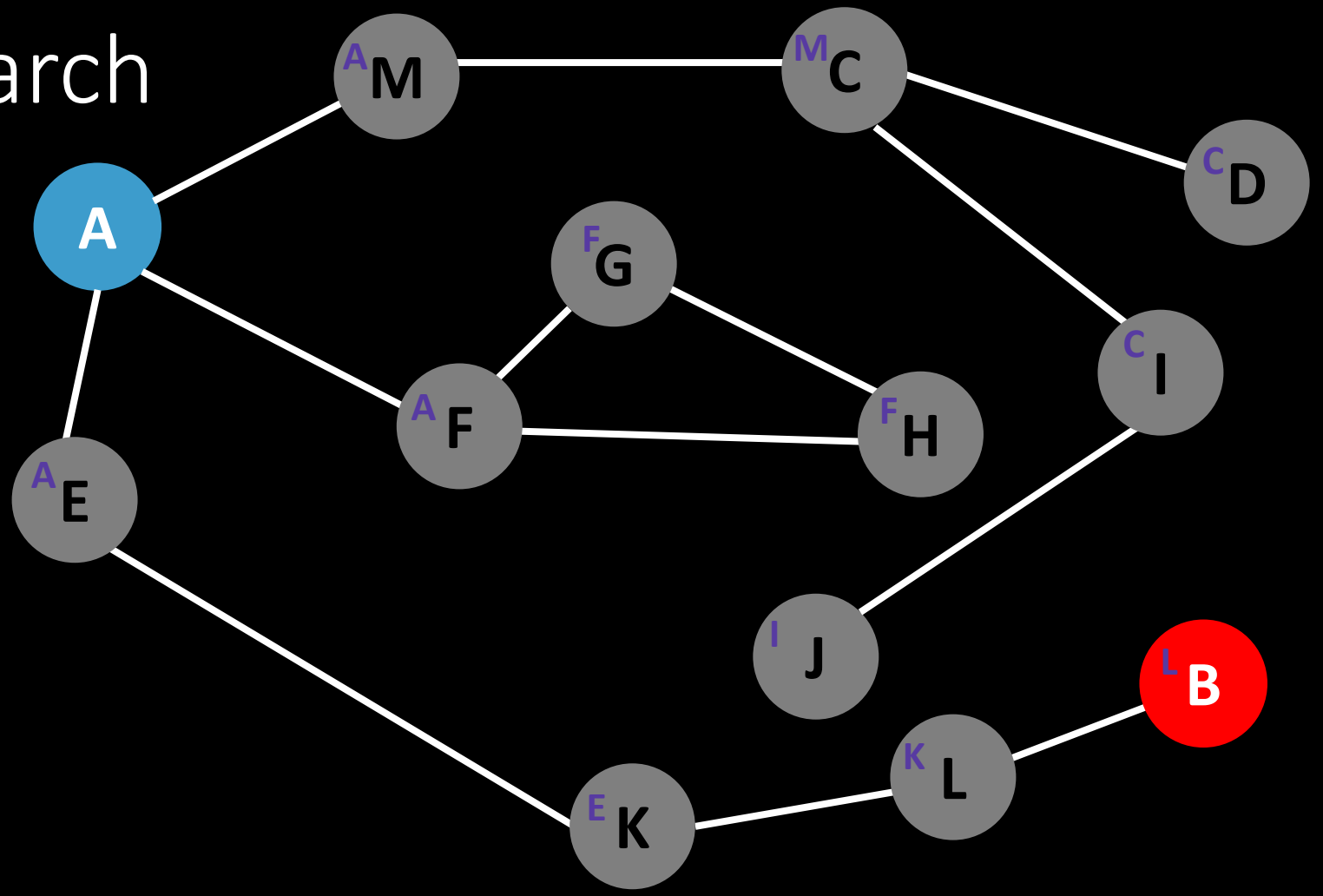


Breadth-First Search

- How to retrieve the shortest path between nodes A and B once the algorithm terminates?

Breadth-First Search

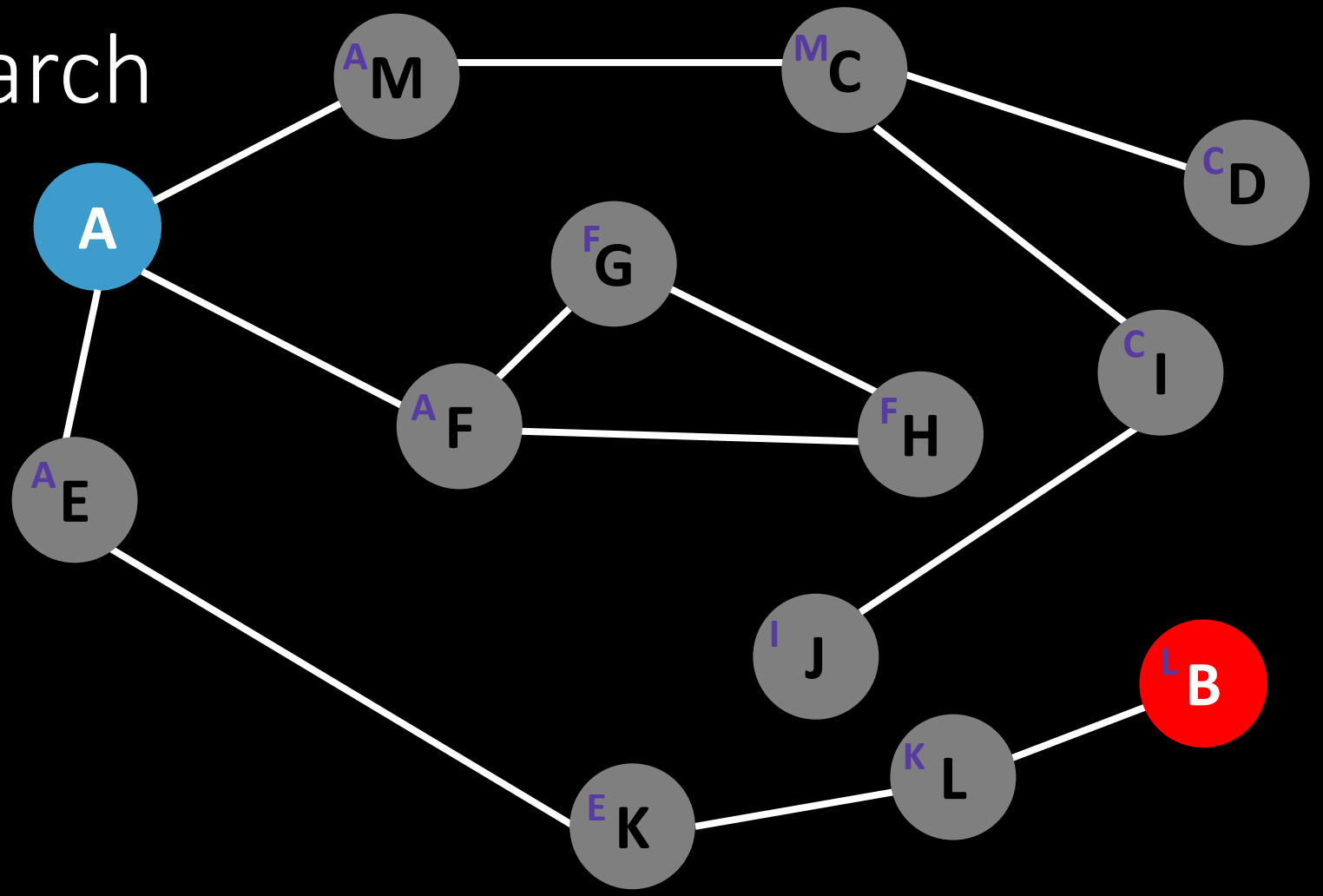
- How to retrieve the shortest path between nodes A and B once the algorithm terminates?



Breadth-First Search

Q: How to retrieve the shortest path between nodes A and B once the algorithm terminates?

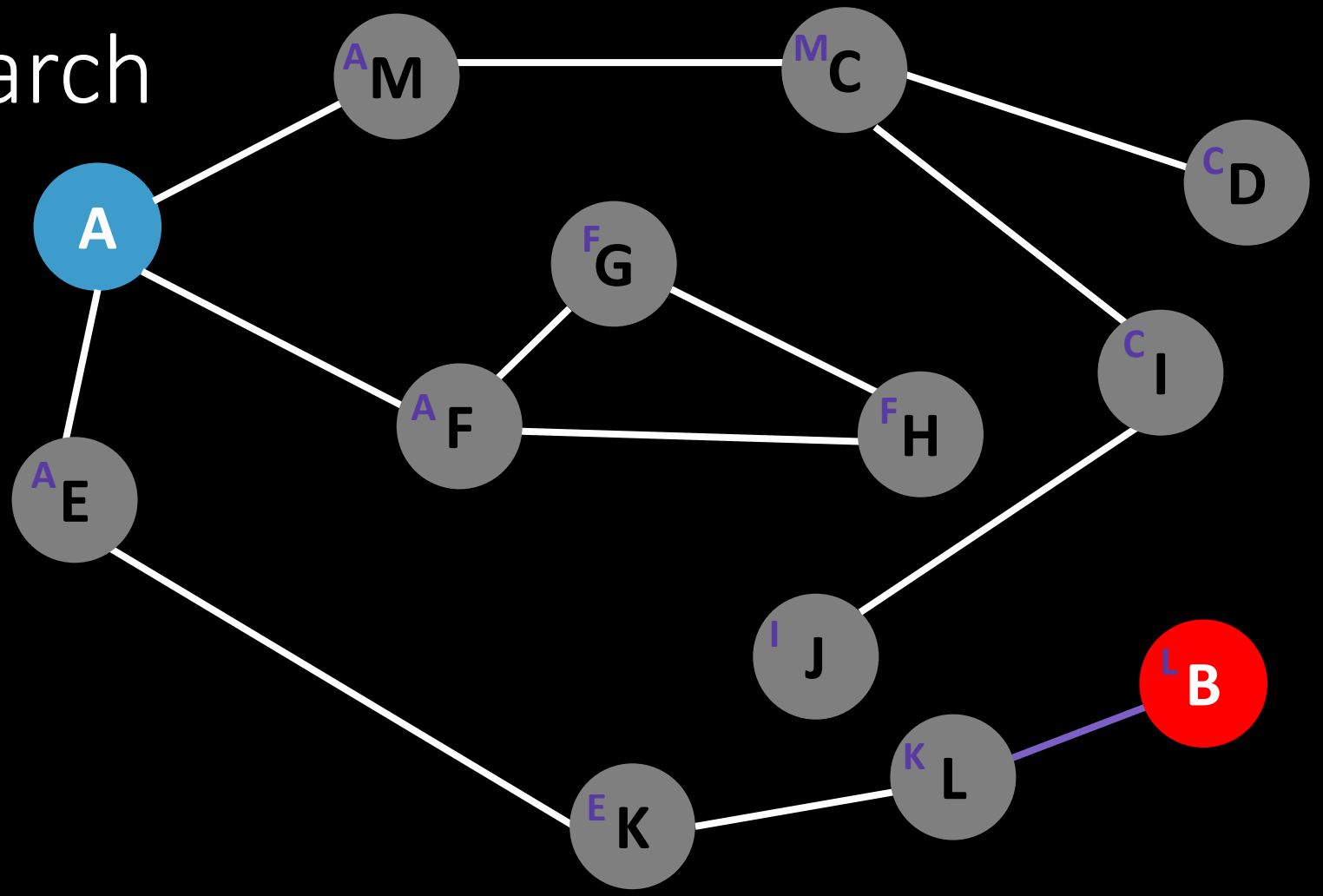
A: Trace back using the indicator of the previous node



Breadth-First Search

Q: How to retrieve the shortest path between nodes A and B once the algorithm terminates?

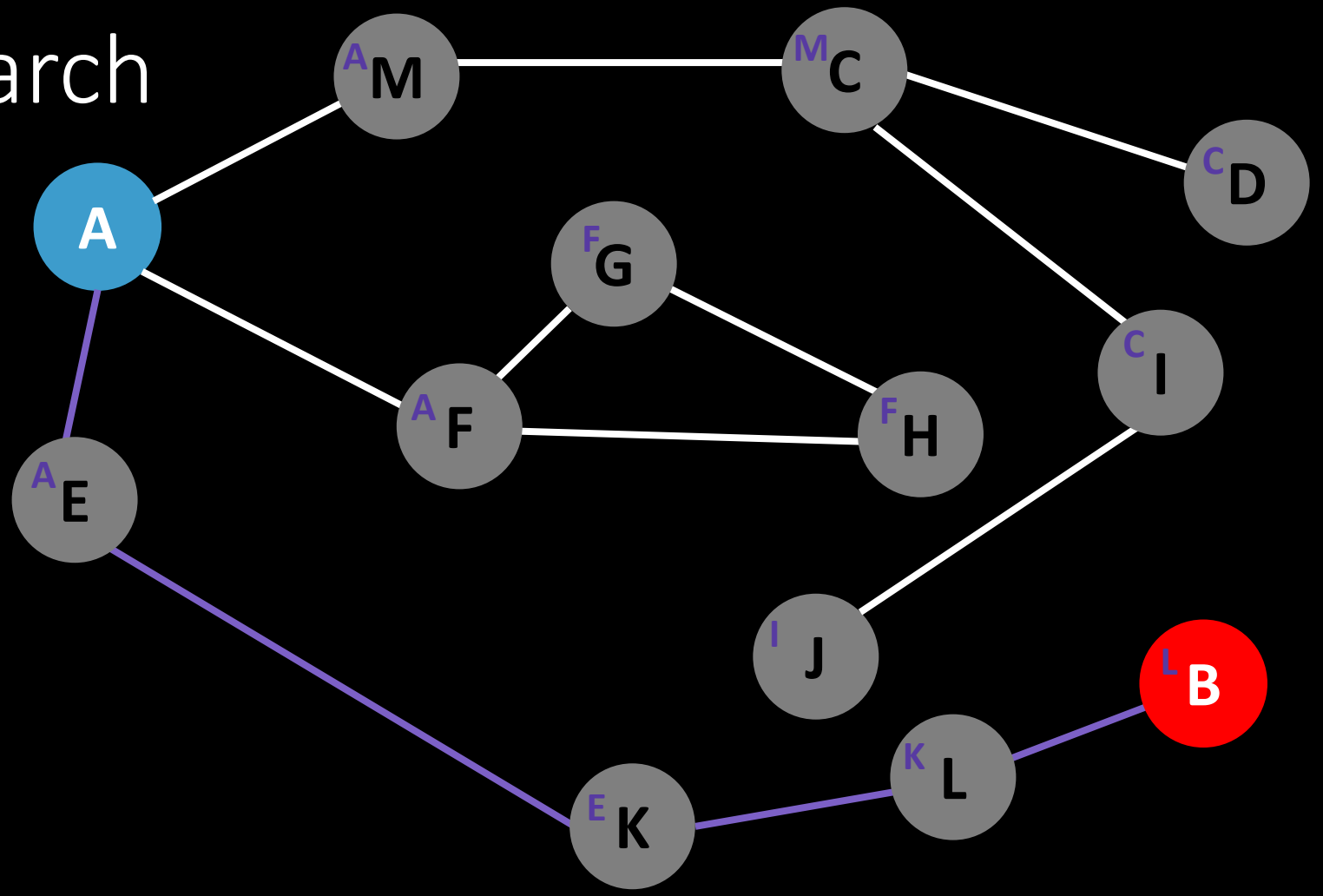
A: Trace back using the indicator of the previous node



Breadth-First Search

Q: How to retrieve the shortest path between nodes A and B once the algorithm terminates?

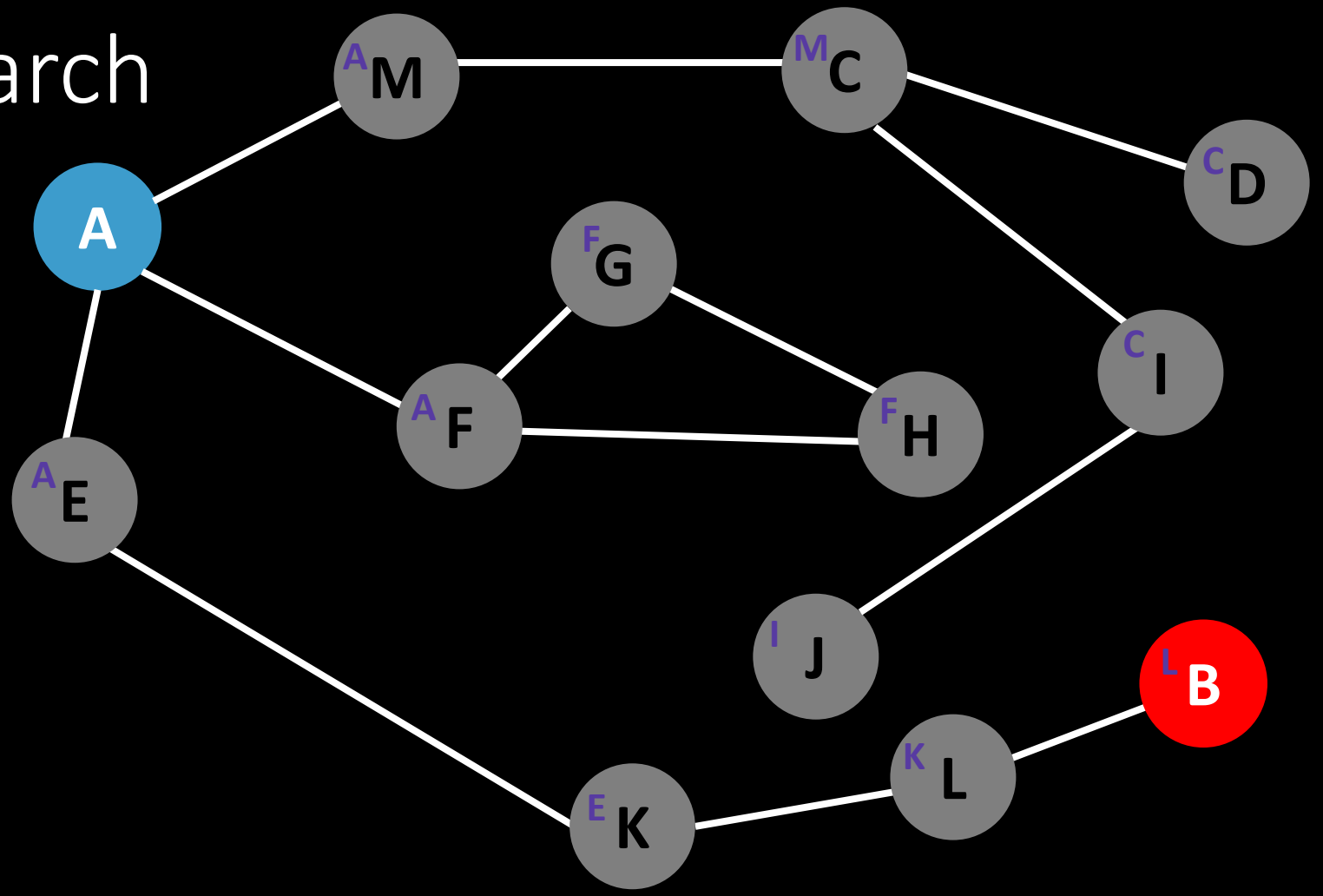
A: Trace back using the indicator of the previous node



Breadth-First Search

Q: How to retrieve the shortest path between nodes A and H once the algorithm terminates?

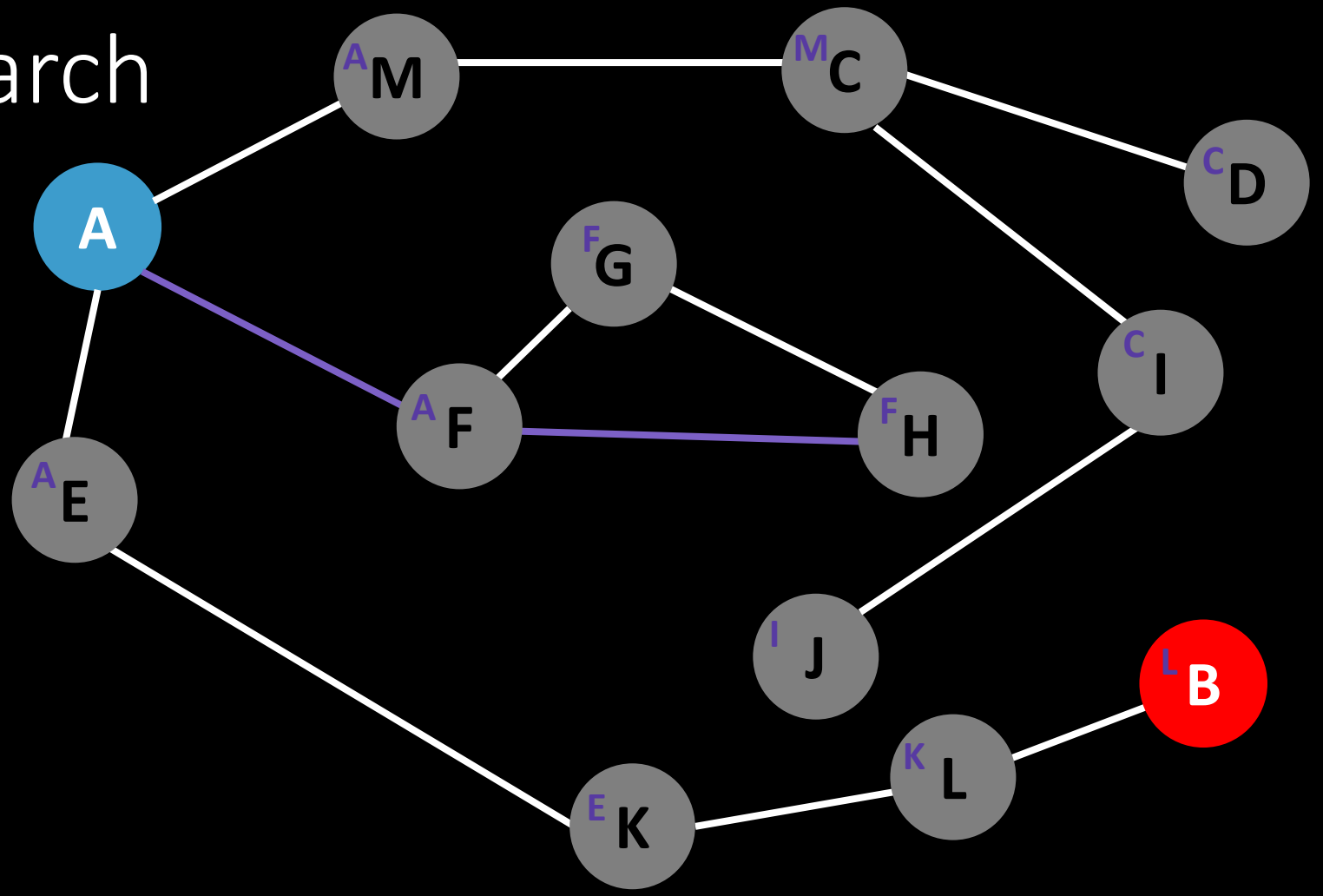
A: Trace back using the indicator of the previous node



Breadth-First Search

Q: How to retrieve the shortest path between nodes A and H once the algorithm terminates?

A: Trace back using the indicator of the previous node

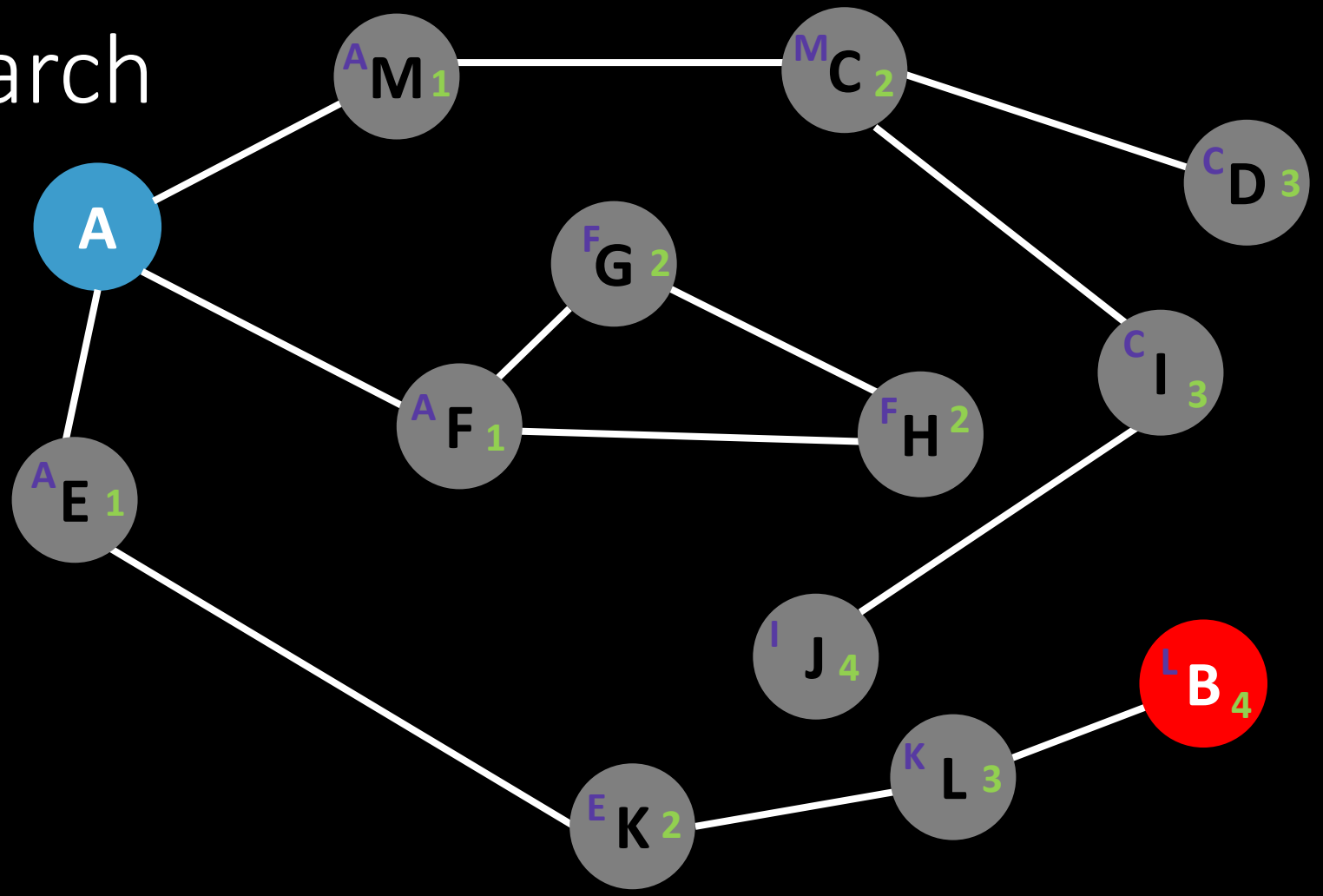


Breadth-First Search

- Sketch of proof that Breadth-First Search yields the shortest path

Breadth-First Search

- May also want to keep track of the node **level** uncovered by the BFS expansion



Depth-First Search PseudoCode

```
stack = []
stack.push(starting_node)

while (stack not empty) and (target not found):
        node = stack.pop()
        if node is visited:
                skip
        if node = target:
                target = found
        for all unvisited neighbors of node:
                neighbor.prev = node
                stack.push(neighbor)
        node.visited = True
```

Retrieving the path, if one is found

```
node = target
path = [node]
while node is not starting_node:
        path.push(node.prev)
        node = node.prev
```

Breadth-First Search PseudoCode

```
queue = []
queue.enqueue(starting_node)

while (queue not empty) and (target not found):
        node = queue.dequeue()
        if node is visited:
                skip
        if node = target:
                target = found
        for all unvisited neighbors of node:
                neighbor.prev = node
                queue.enqueue(neighbor)
        node.visited = True
```

Retrieving the path, if one is found

```
node = target
path = [node]
while node is not starting_node:
        path.push(node.prev)
        node = node.prev
```

Breadth-First Search PseudoCode

```
queue = []
queue.enqueue(starting_node)
level = 0
while (queue not empty) and (target not found):
        node = queue.dequeue()
        node.level = level++
        if node is visited:
                skip
        if node = target:
                target = found
        for all unvisited neighbors of node:
                neighbor.prev = node
                queue.enqueue(neighbor)
        node.visited = True
```

Retrieving the path, if one is found

```
node = target
path = [node]
while node is not starting_node:
        path.push(node.prev)
        node = node.prev
```

Breadth-First Search

- Consider when the graph edges do not have unit weight
- Can we still use BFS to find the shortest path?

Breadth-First Search

- Consider when the graph edges do not have unit weight
- Can we still use BFS to find the shortest path?
- Next lecture: finding the shortest path in weighted graphs
- Next week: how to implement these algorithms

Visualization Resource

- https://visualgo.net/en/dfsbfs
- https://workshape.github.io/visual-graph-algorithms/