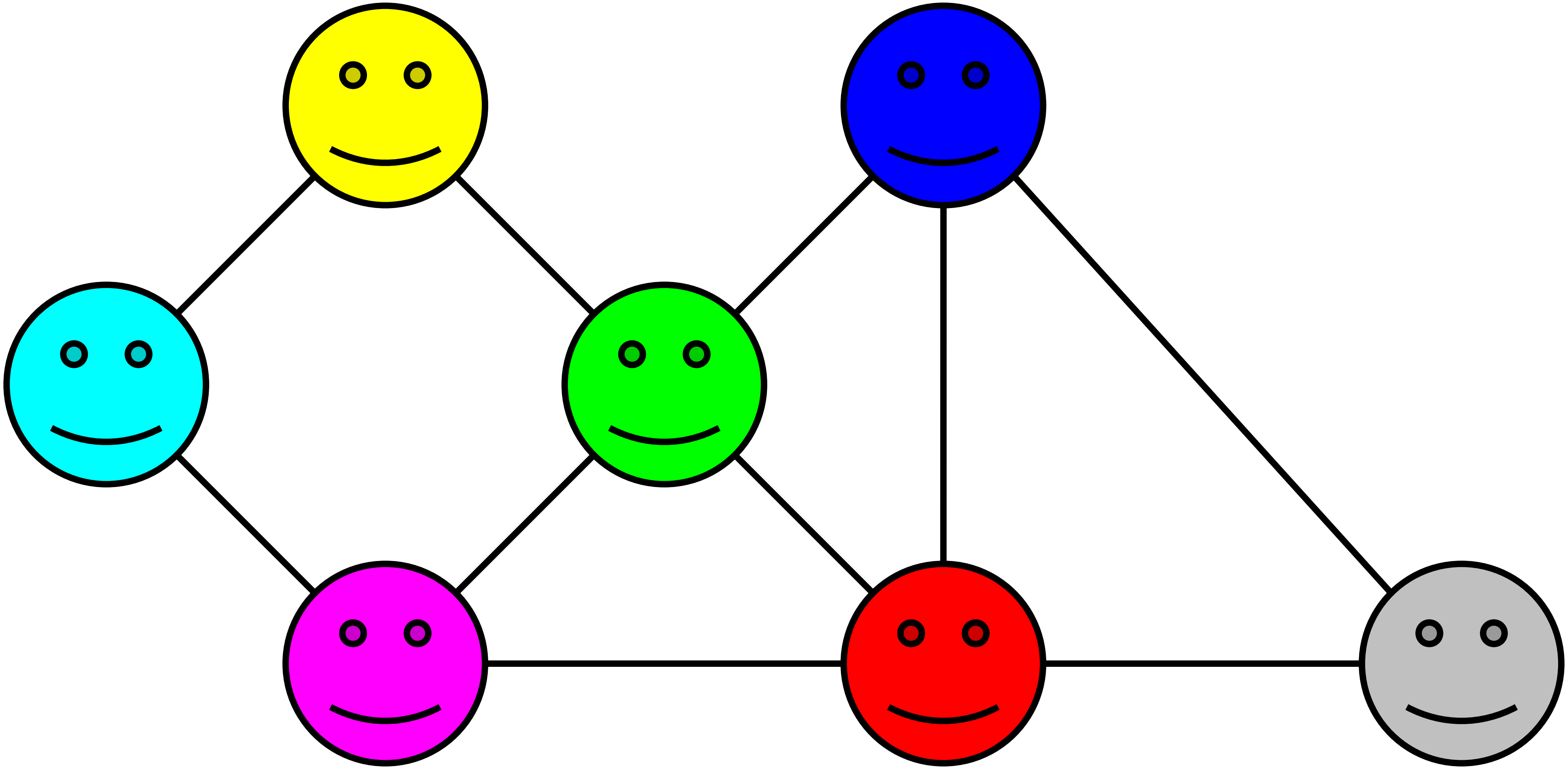
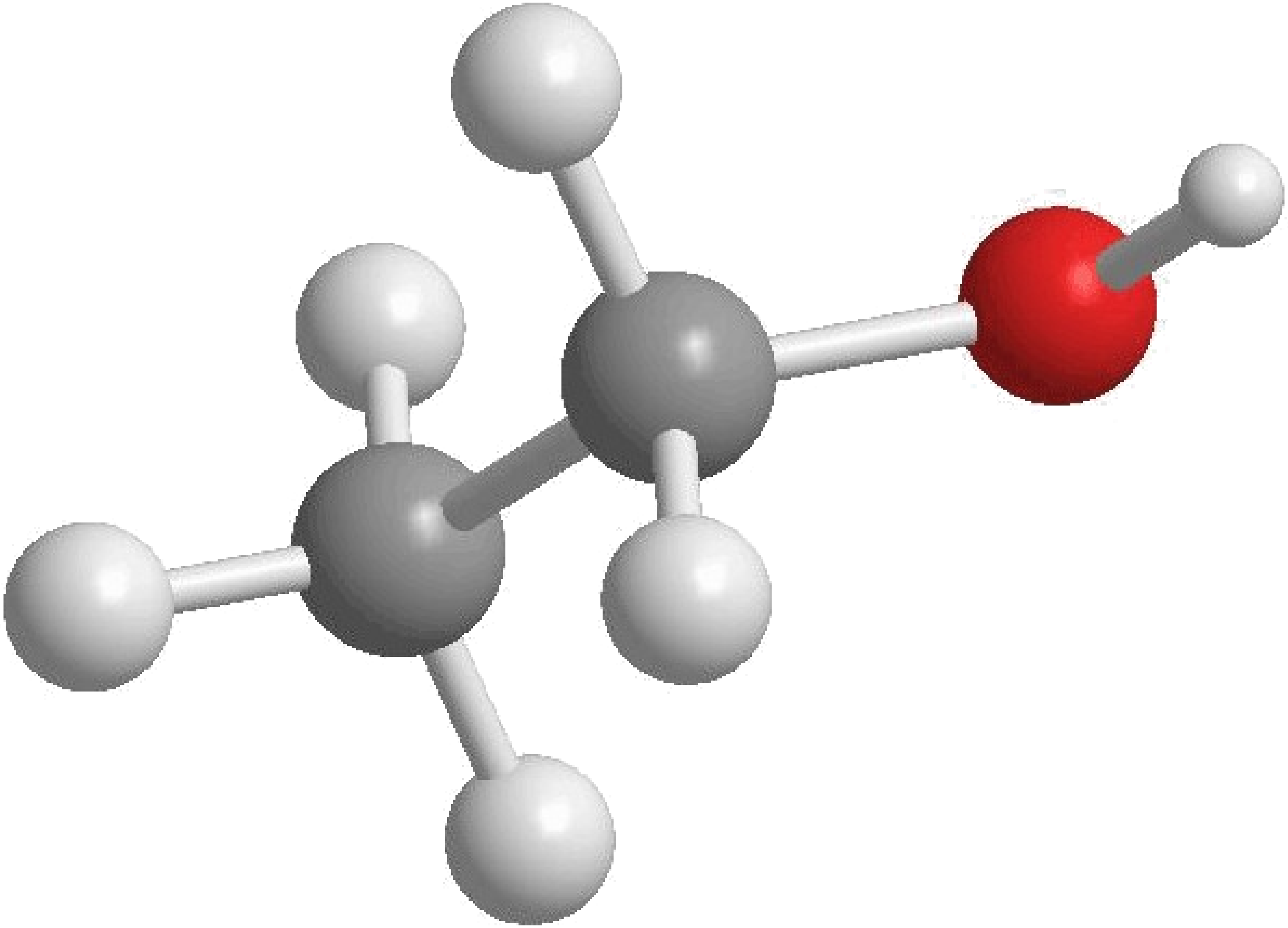
Graphs

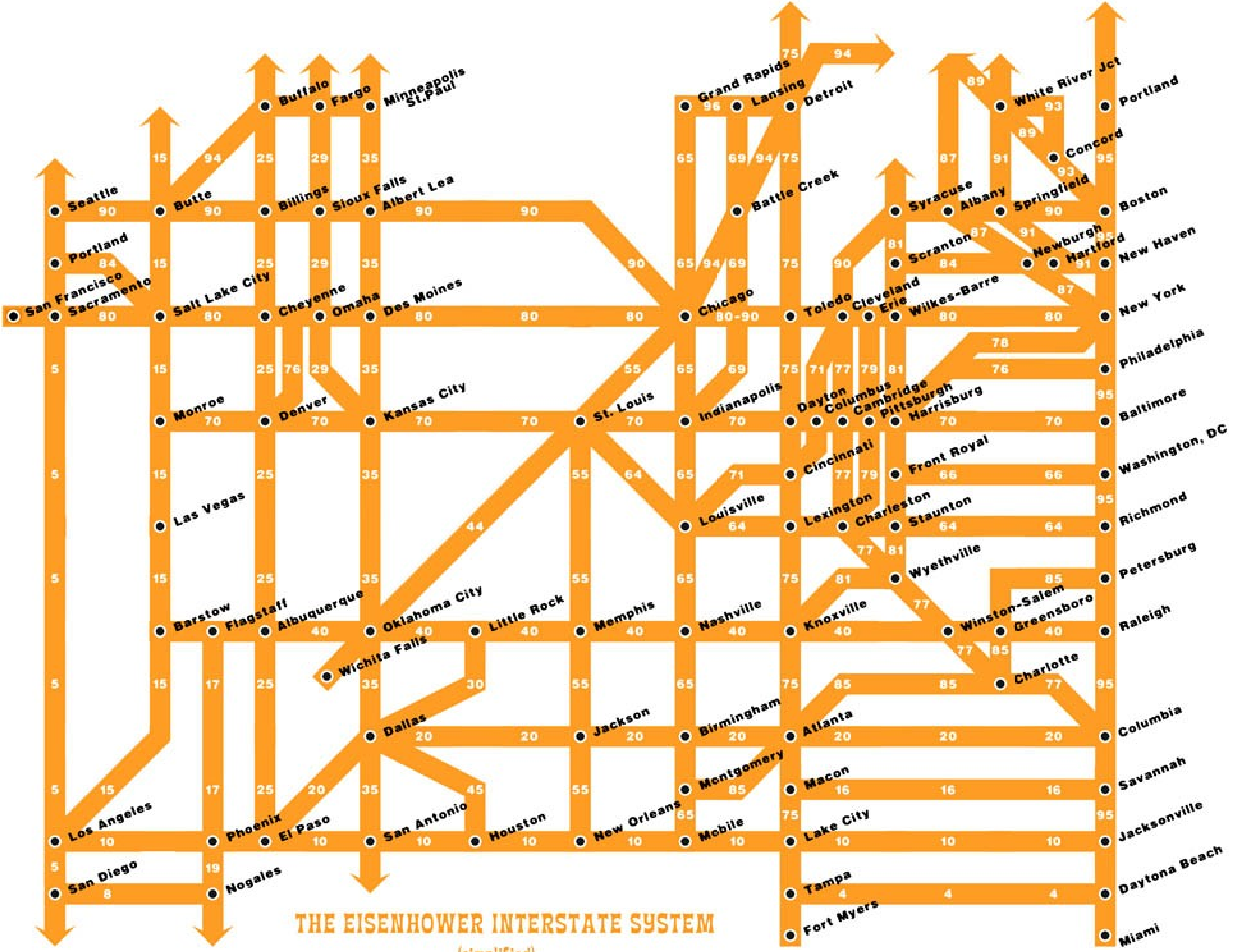
A Social Network



Chemical Bonds



http://4.bp.blogspot.com/-xCtBJ8lKHqA/Tjm0BONWBRI/AAAAAAAAAK4/-mHrbAUOHHg/s1600/Ethanol2.gif

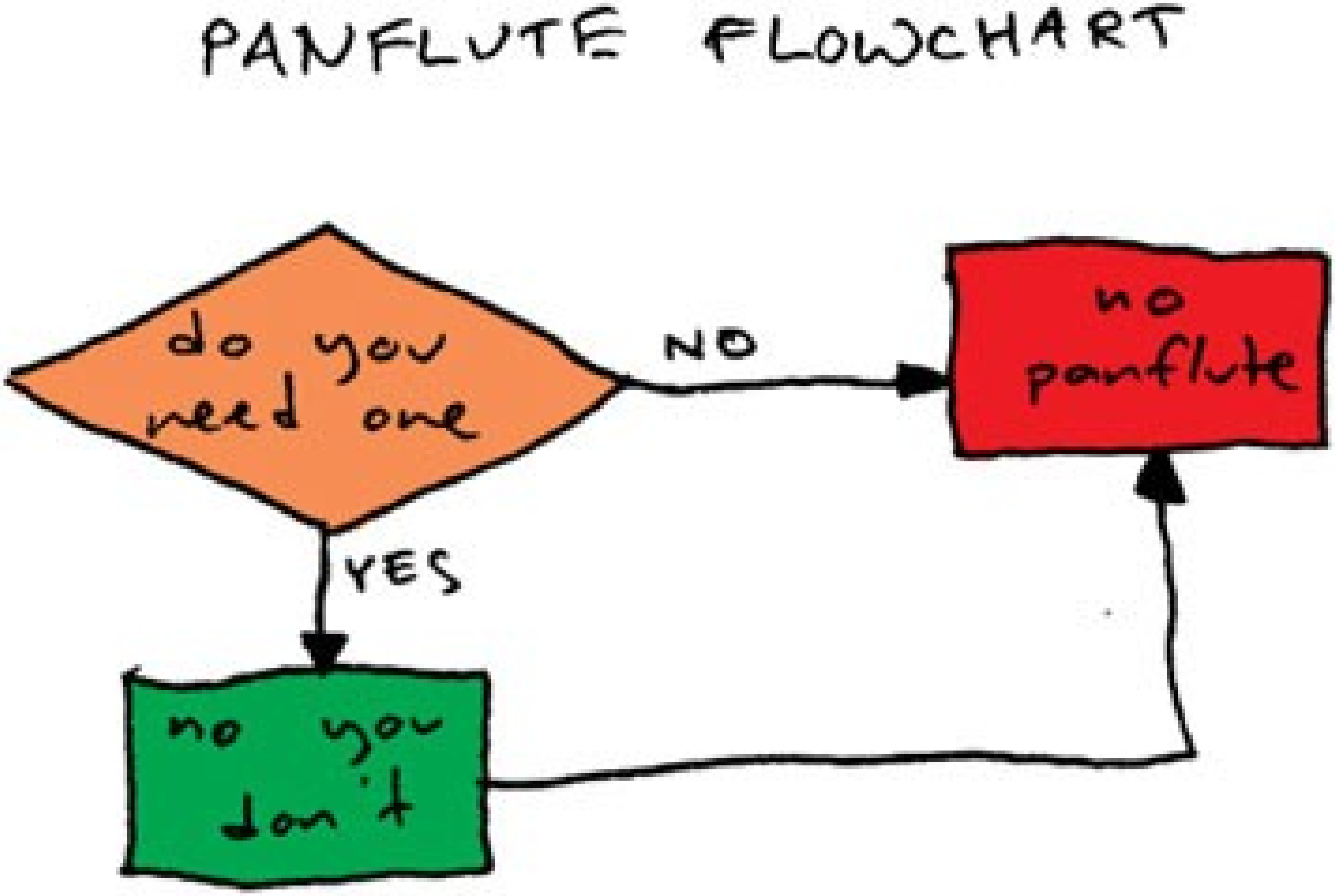


THE EISENHOWER INTERSTATE SYSTEM

(simplified)

CHRIS YATES 2007

http://strangemaps.files.wordpress.com/2007/02/fullinterstatemap-web.jpg

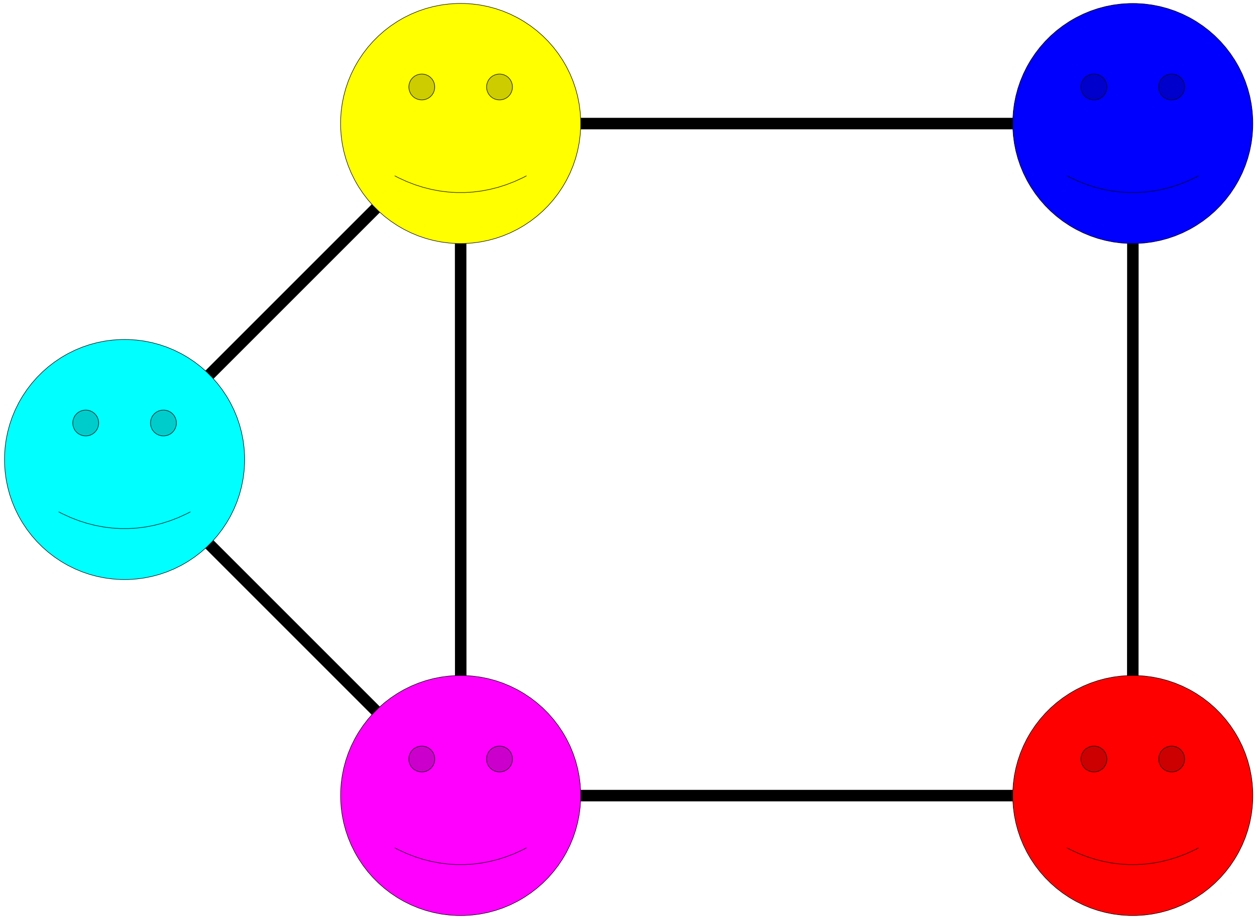


http://www.toothpastefordinner.com/

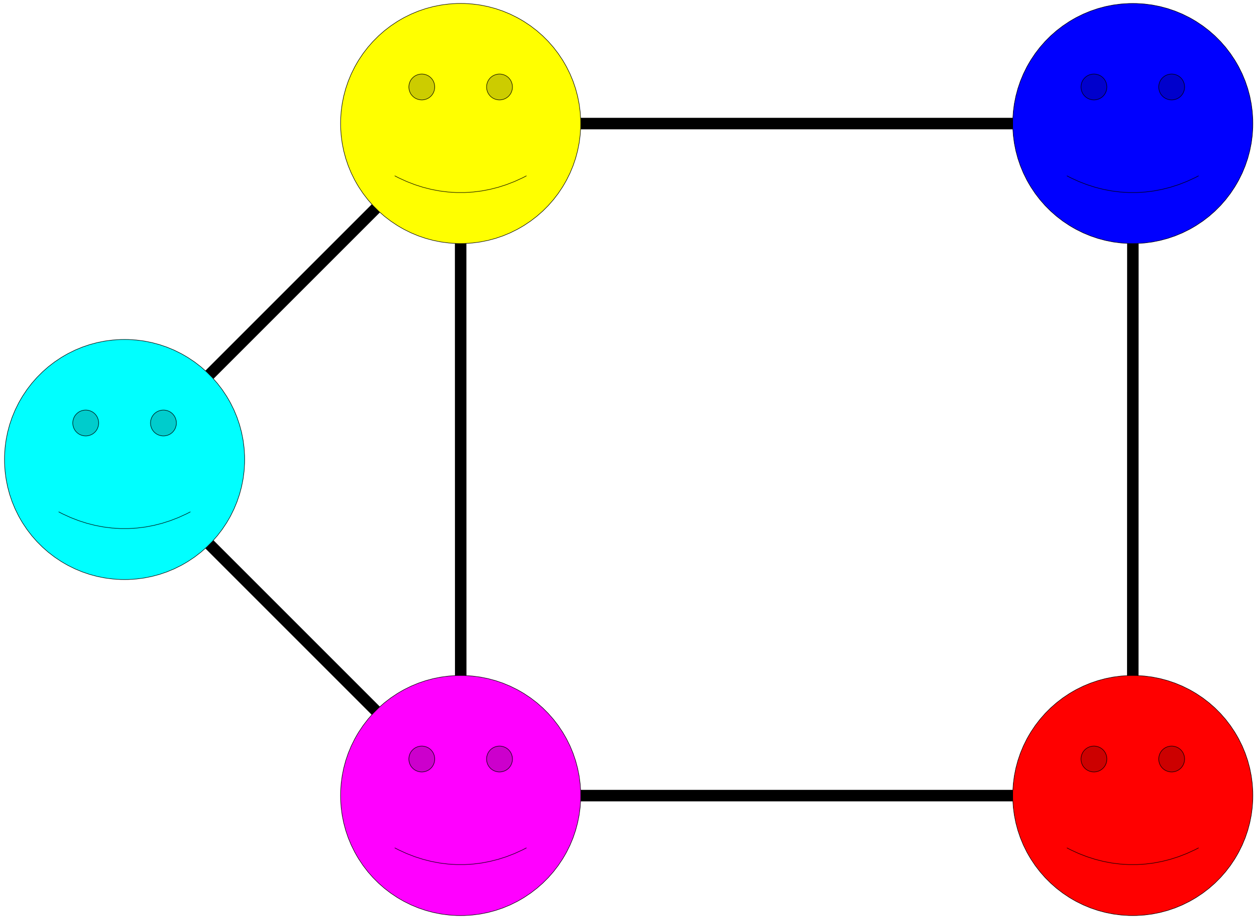
A ***graph*** is a mathematical structure for representing relationships.

A ***graph*** is a mathematical structure for representing relationships.

A ***graph*** is a mathematical structure for representing relationships.

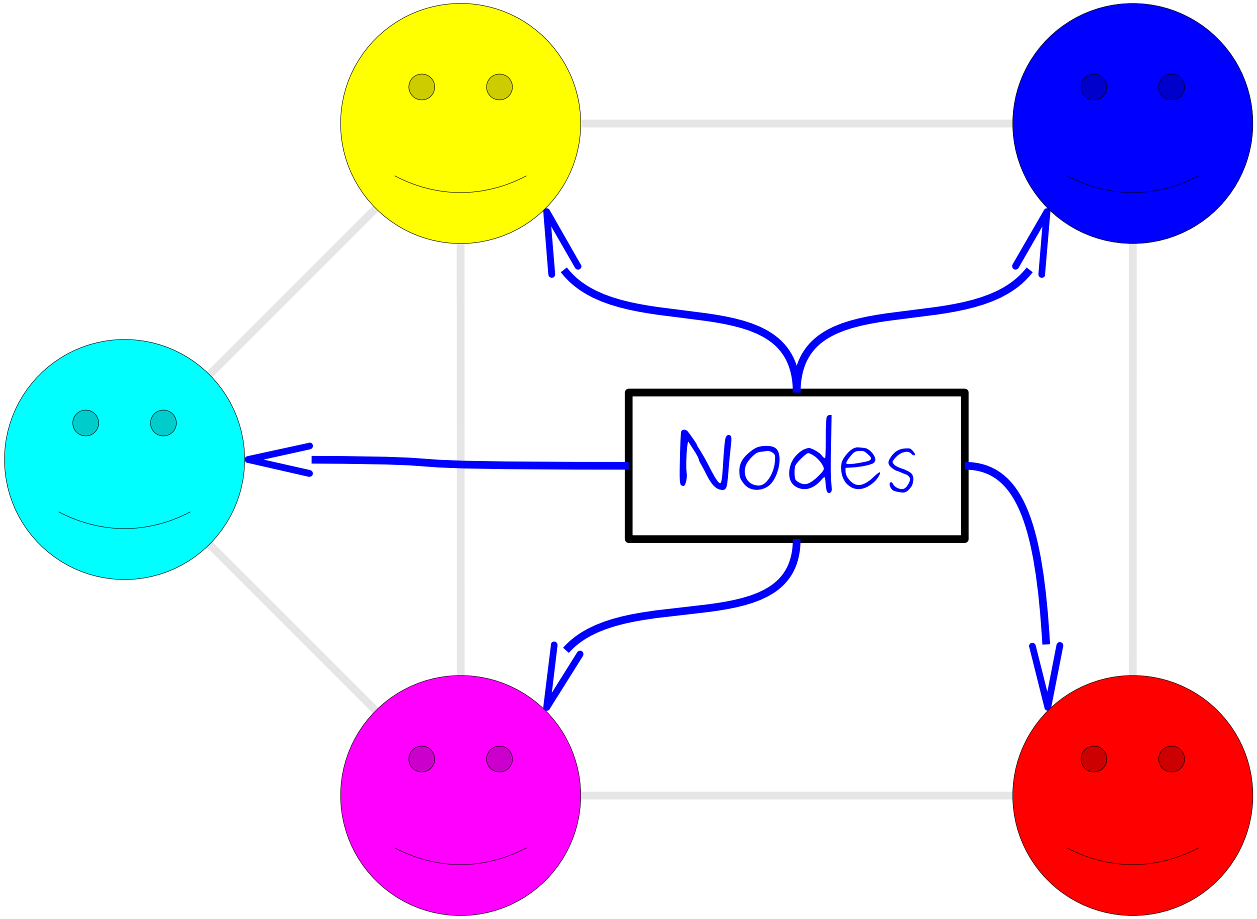


A ***graph*** is a mathematical structure for representing relationships.



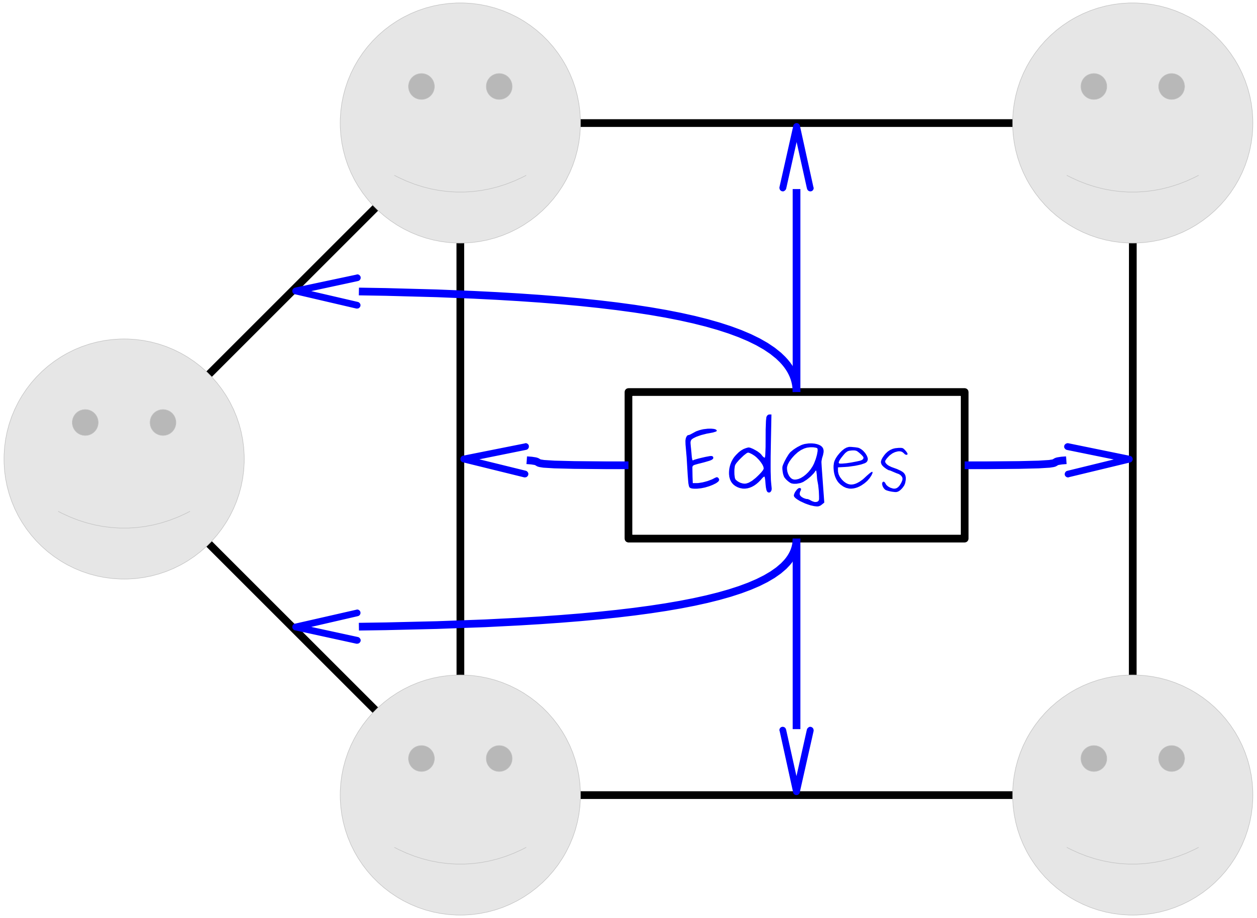
A graph consists of a set of ***nodes*** connected by ***edges***.

A ***graph*** is a mathematical structure for representing relationships.



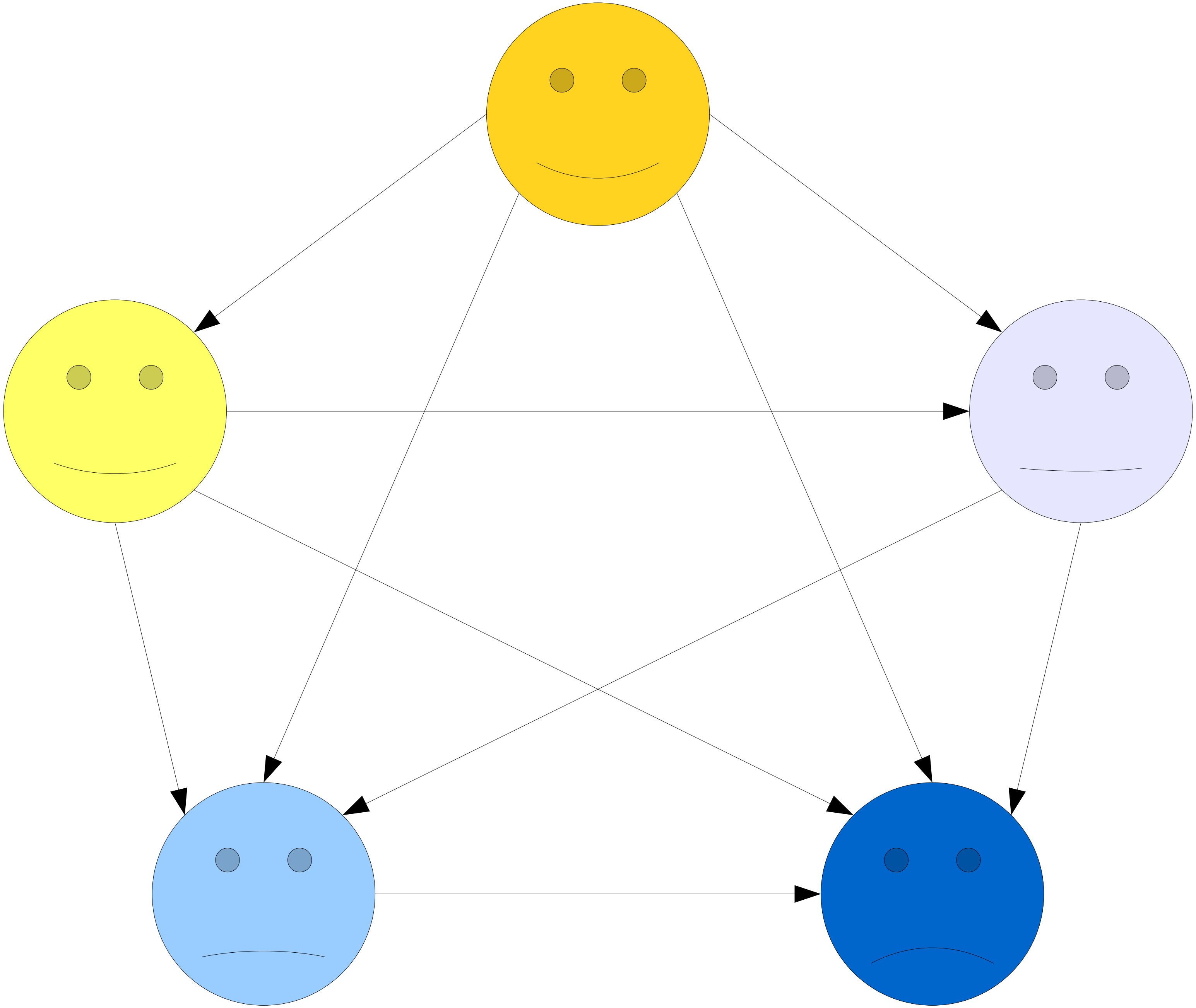
A graph consists of a set of ***nodes*** connected by ***edges***.

A ***graph*** is a mathematical structure for representing relationships.

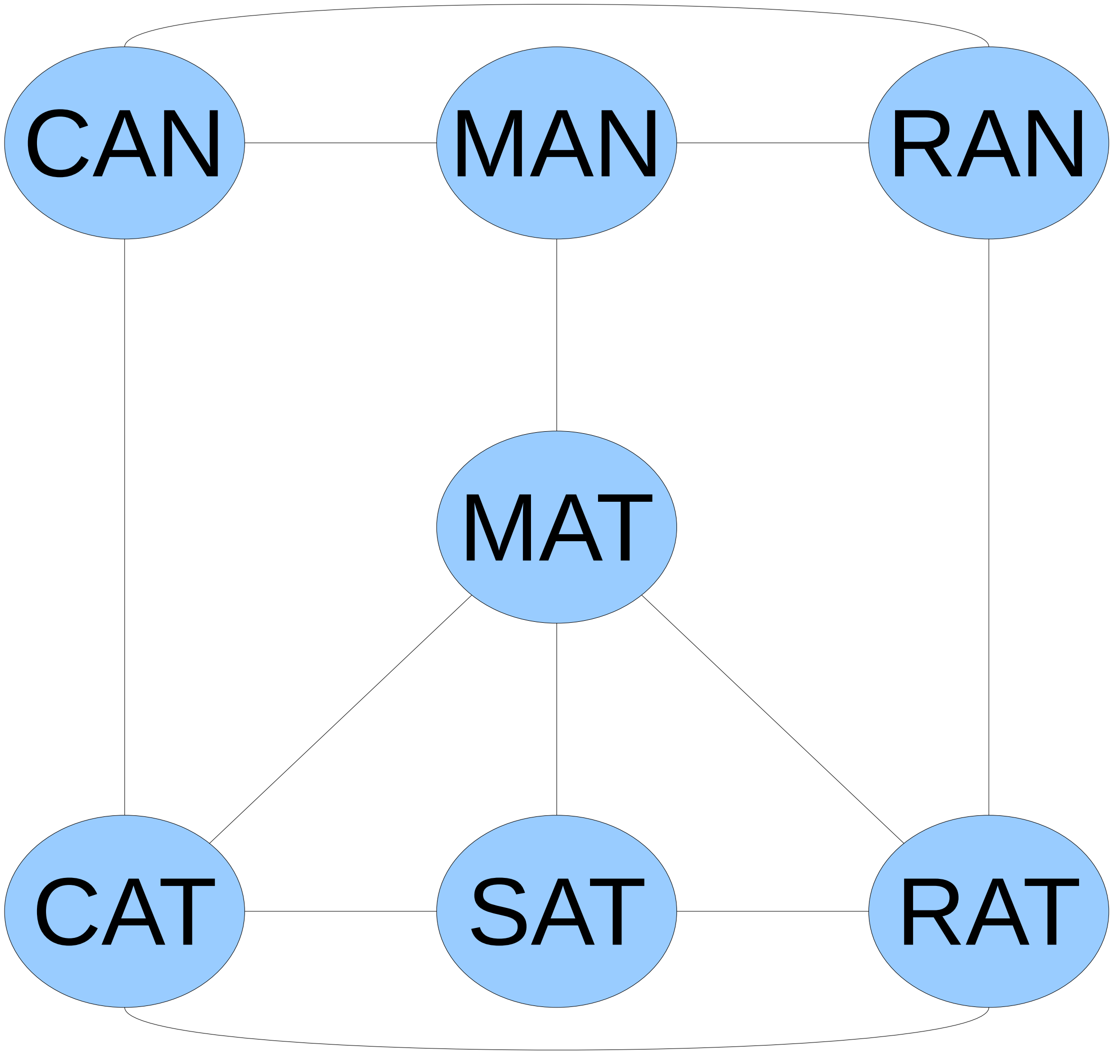


A graph consists of a set of ***nodes*** connected by ***edges***.

Some graphs are ***directed***.



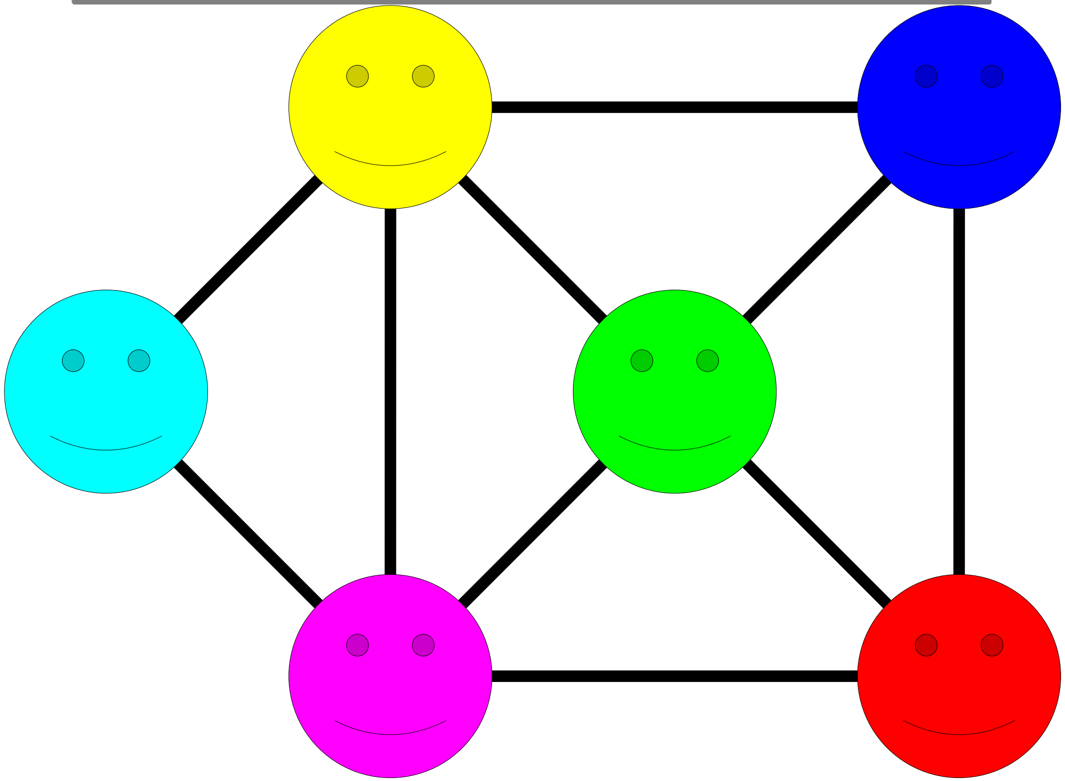
Some graphs are ***undirected***.



How can we represent graphs in C++?

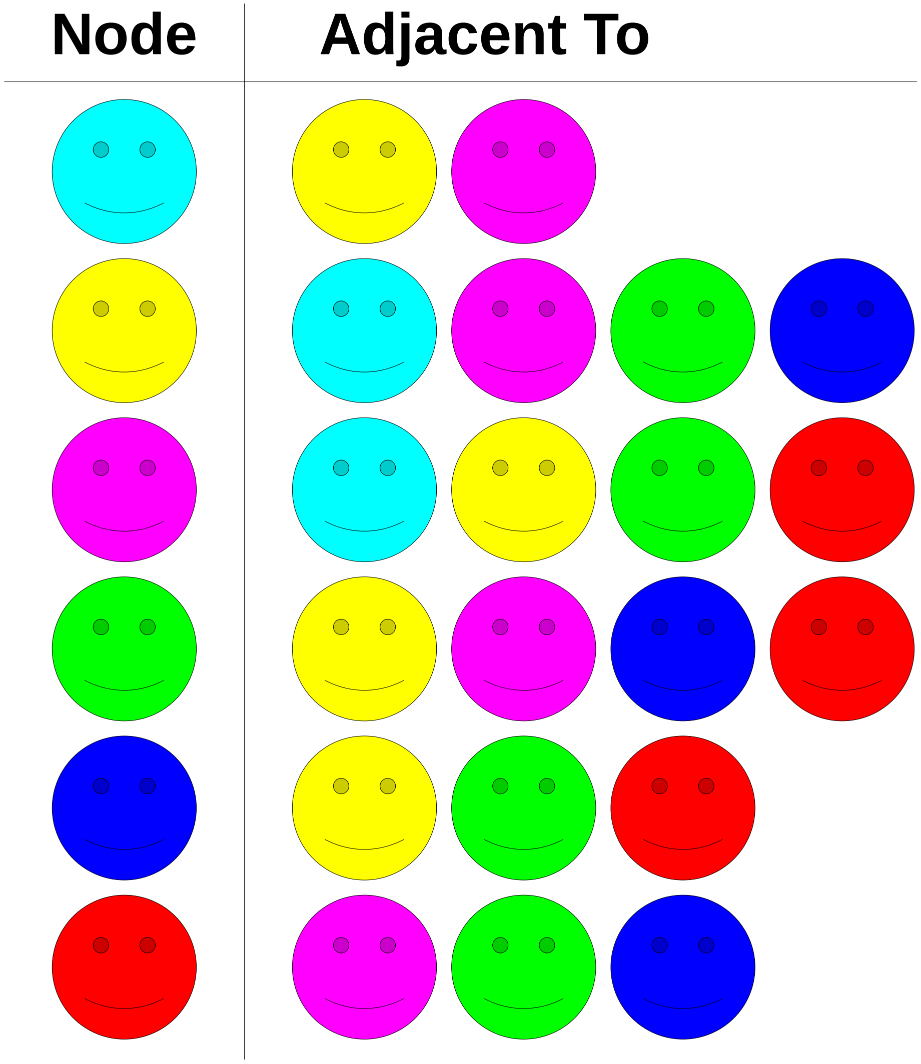
Representing Graphs

We can represent a graph as a map from nodes to the list of nodes each node is connected to.



Map<***Node***, Vector<***Node***>>

Node Vector<***Node***>


Node
Adjacent To

Representing Graphs

- The approach we just saw is called an ***adjacency list*** in comes in a number of different forms:

```
Map<string, Vector<string>>
Map<string, Set<string>>
Vector<Vector<int>>
```

- The core idea is that we have some kind of mapping associating each node with its outgoing edges.

Representing Graphs

The approach we just saw is called an ***adjacency list*** in comes in a number of different forms:

`Map<string, Vector<string>>`

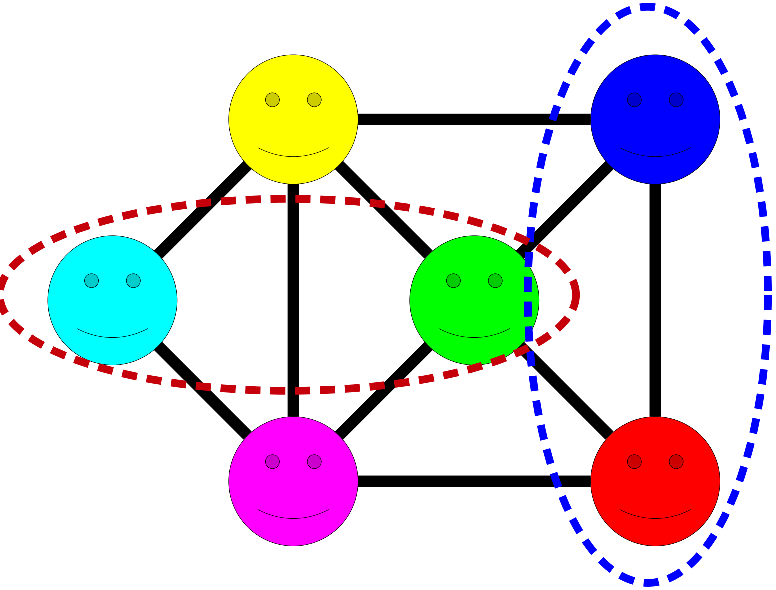
`Map<string, Set<string>>`

`Vector<Vec`

The core idea is that nd of mapping associating each node with its outgoing edges.

Question to ponder:
where have you seen this
before?

Other Graph Representations

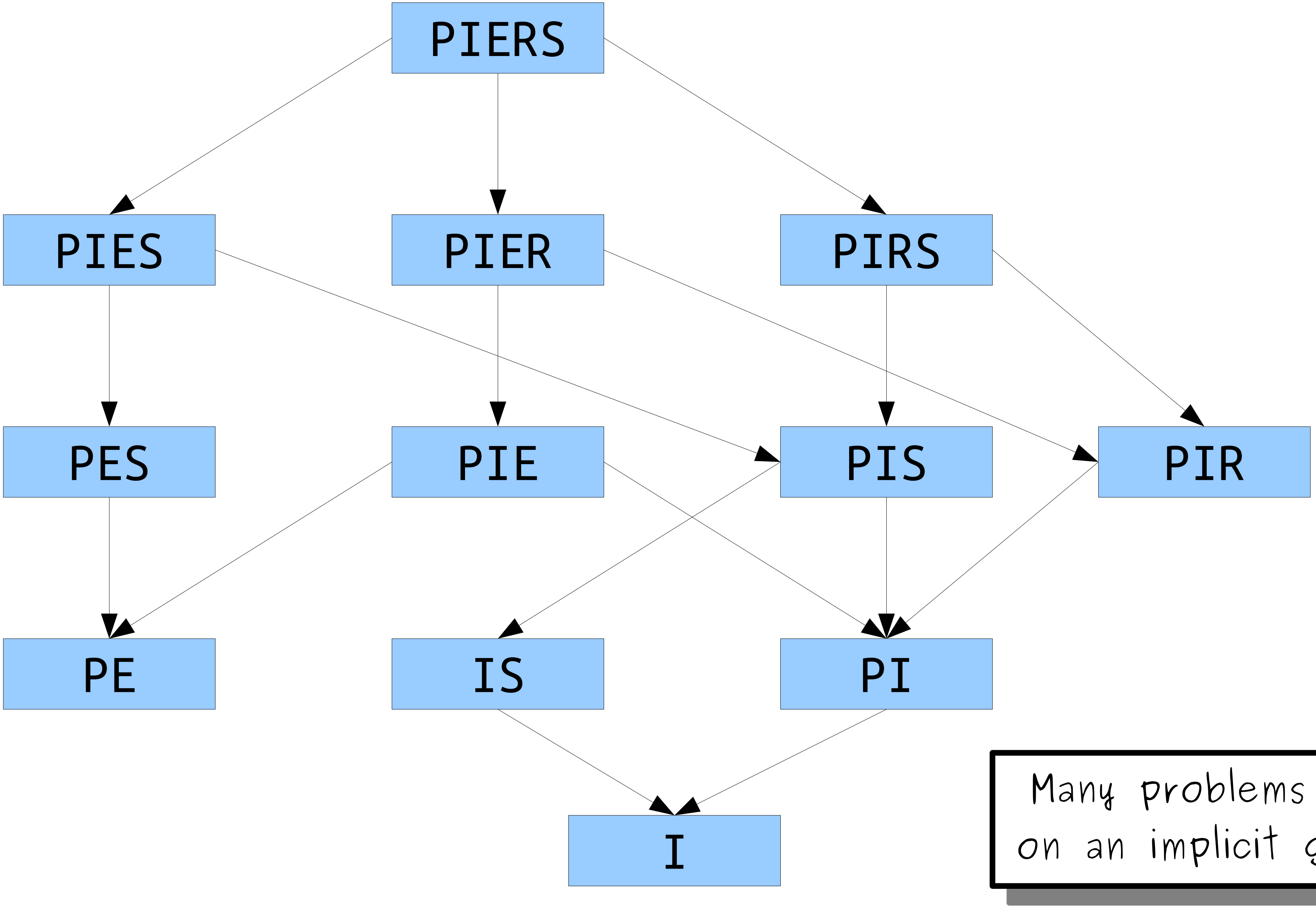


$$
\begin{array}{c|cccccc}
 & 🩵 & 🟡 & 🟣 & 🟢 & 🔵 & 🔴 \\
\hline
🩵 & 0 & 1 & 1 & 0 & 0 & 0 \\
🟡 & 1 & 0 & 1 & 1 & 1 & 0 \\
🟣 & 1 & 1 & 0 & 1 & 0 & 1 \\
🟢 & 0 & 1 & 1 & 0 & 1 & 1 \\
🔵 & 0 & 1 & 0 & 1 & 0 & 1 \\
🔴 & 0 & 0 & 1 & 1 & 1 & 0
\end{array}
$$

This representation is called an *adjacency matrix*.

For those of you in Math 51: if A is an adjacency matrix for a graph G, what is the significance of the matrix A^2?

Other Representations

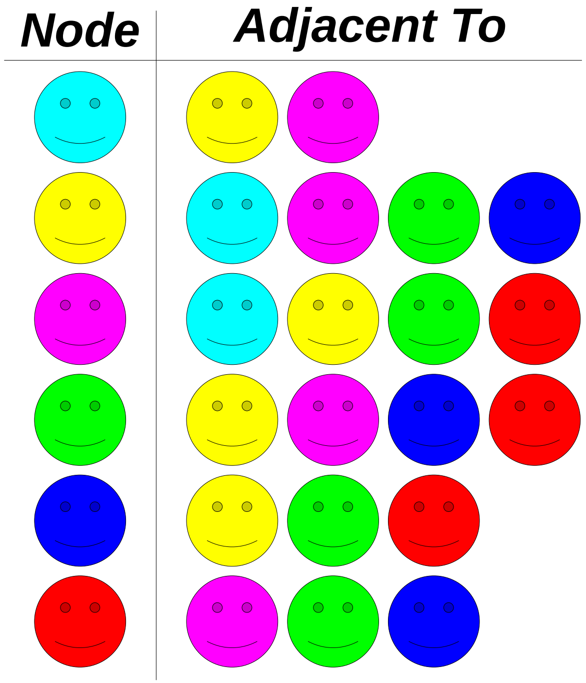


Many problems work on an implicit graph.

You’ll find graphs just
about everywhere you look.

They’re an ***extremely*** versatile and
powerful abstraction to be aware of.

Going forward, unless stated otherwise, assume we're using an ***adjacency list***.

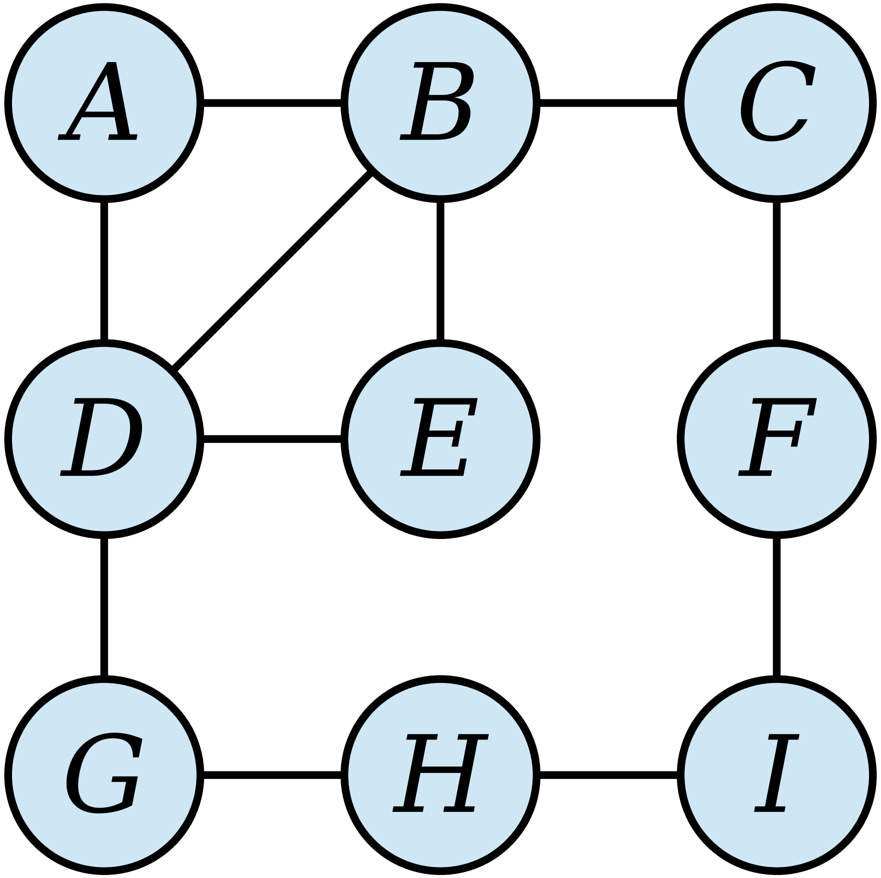


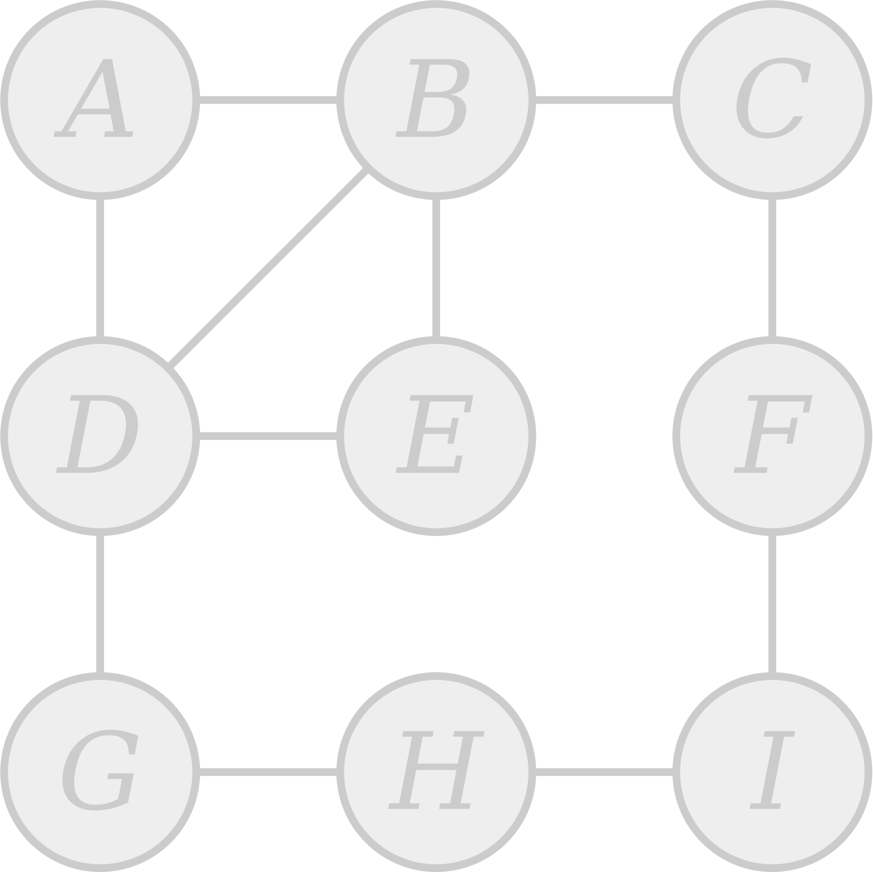
Traversing Graphs

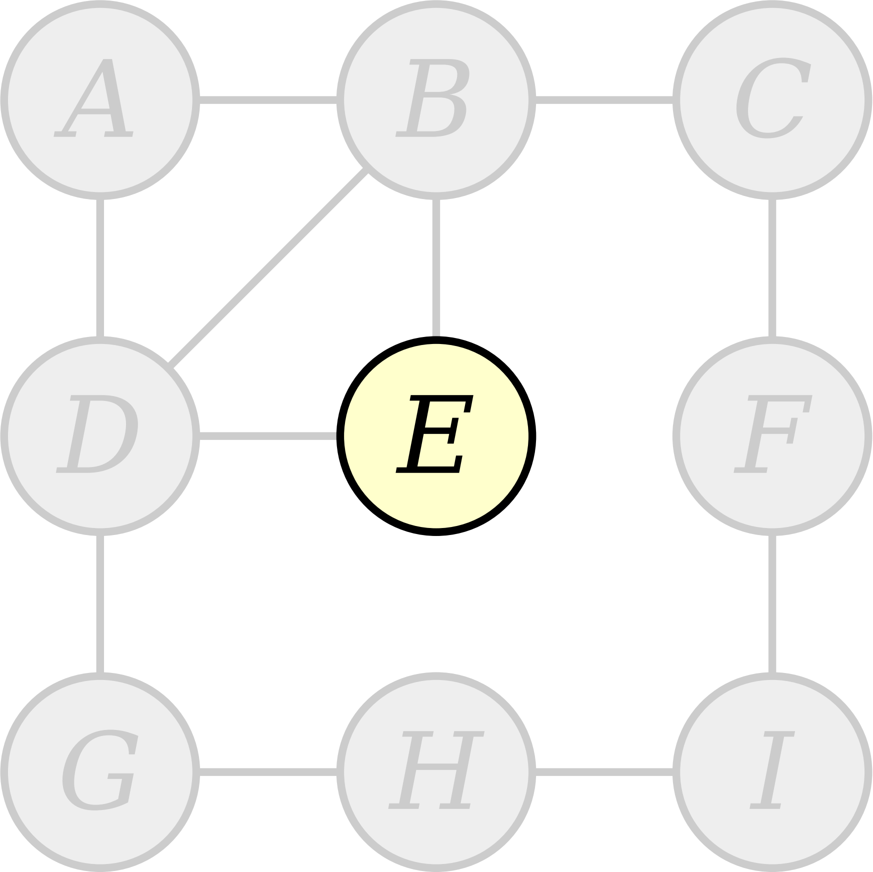
Iterating over a Graph

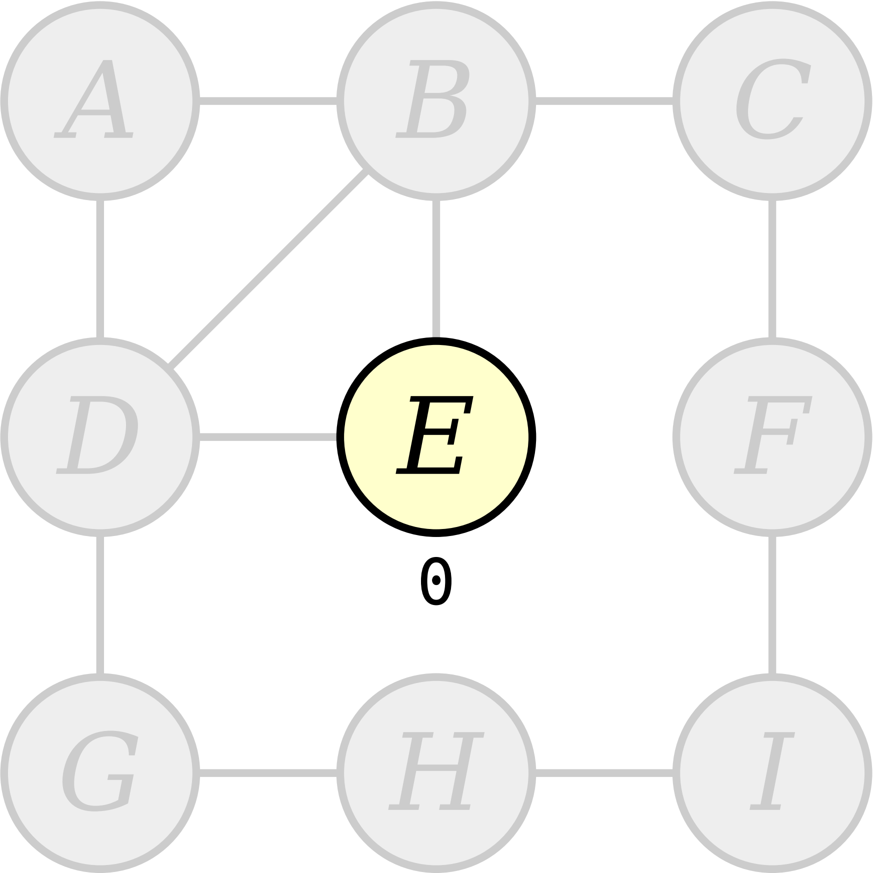
- In a singly-linked list, there’s pretty much one way to iterate over the list: start at the front and go forward!
- In a binary search tree, there are many traversal strategies:
  - An ***inorder traversal*** that produces all the elements in sorted order.
  - A ***postorder traversal*** is used to delete all the nodes in the BST.
- There are *many* ways to iterate over a graph, each of which have different properties.

One Search Strategy

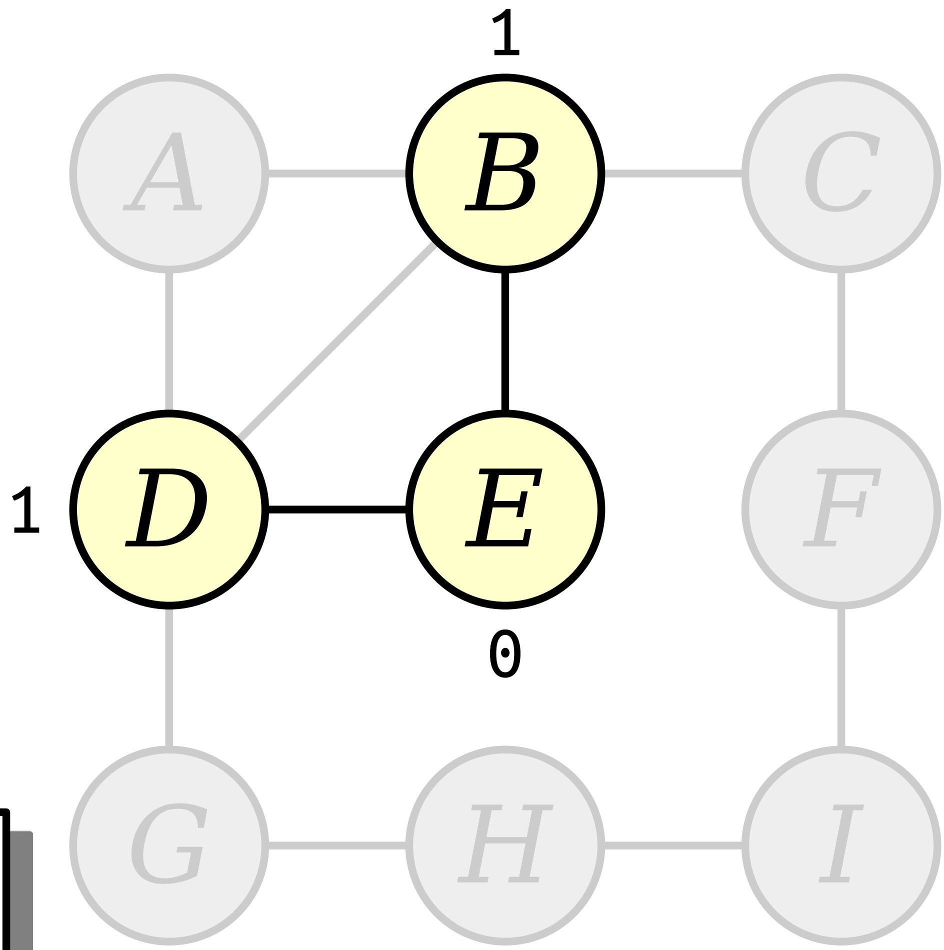

A
B
C
D
E
F
G
H
I


A
B
C
D
E
F
G
H
I

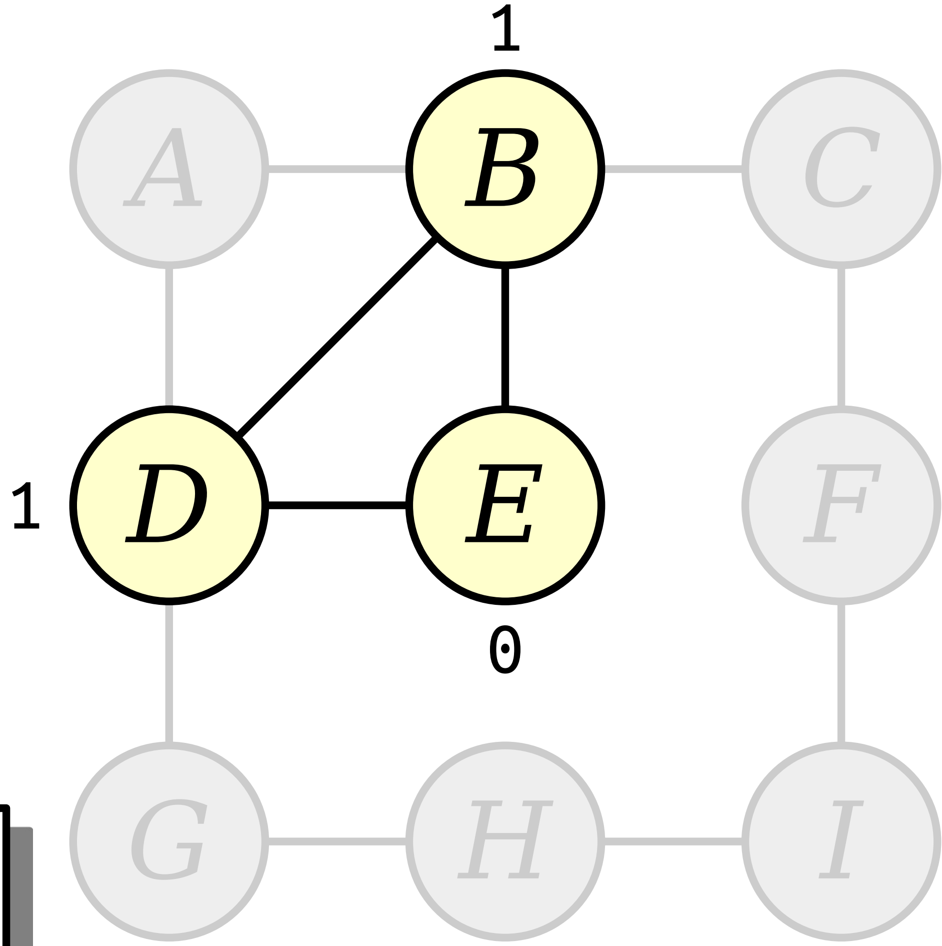

A
B
C
D
E
F
G
H
I



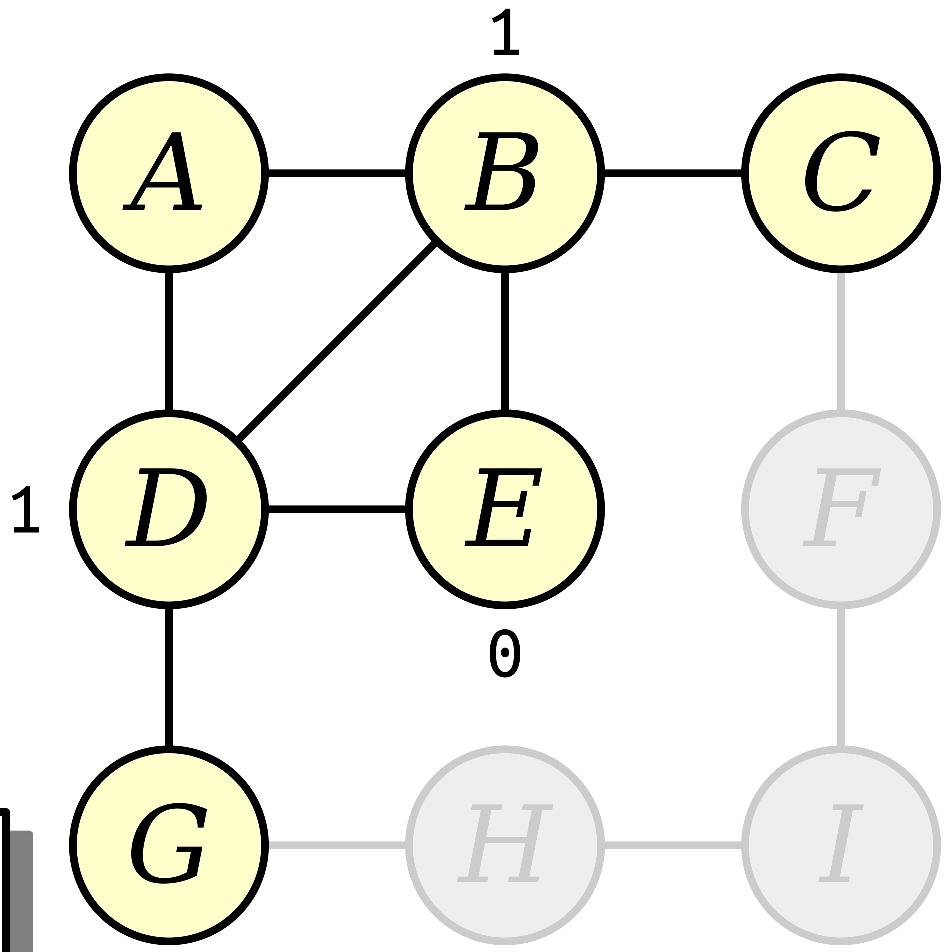
Core idea: Find everything one hop away from the start, then two hops away, then three hops away, etc.



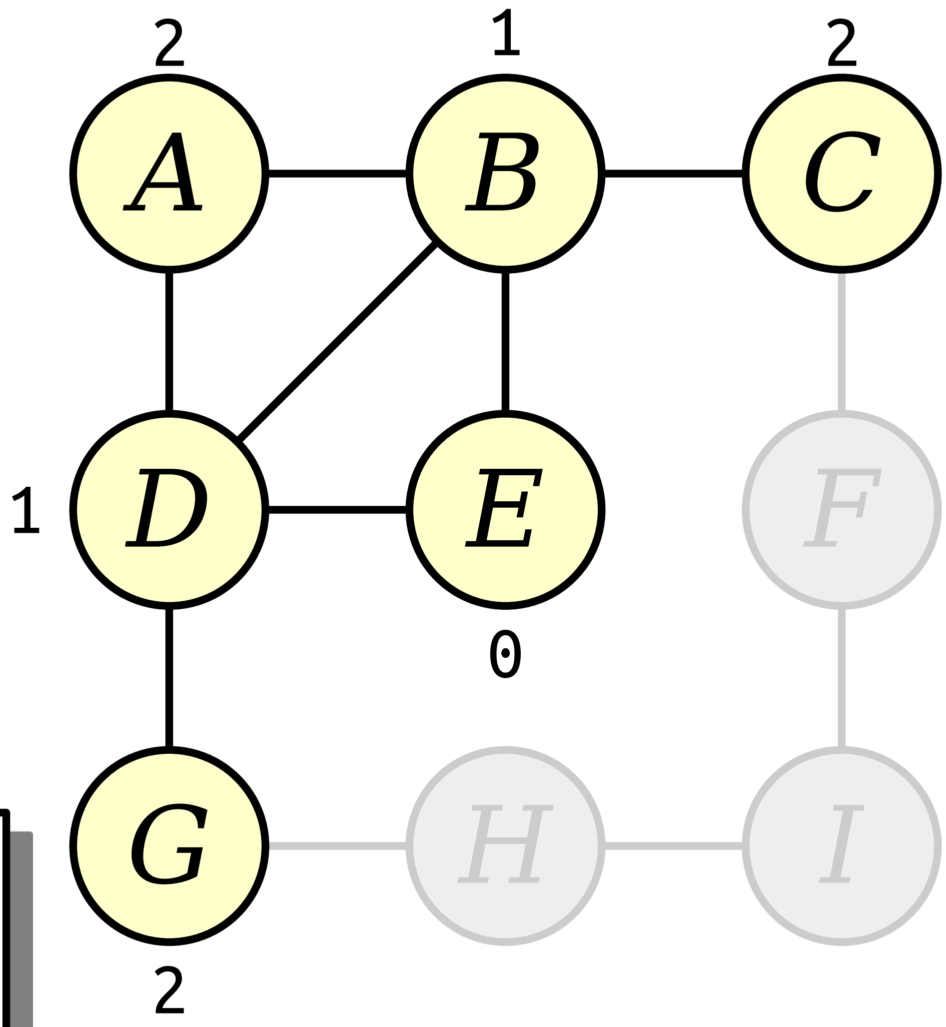
Core idea: Find everything one hop away from the start, then two hops away, then three hops away, etc.



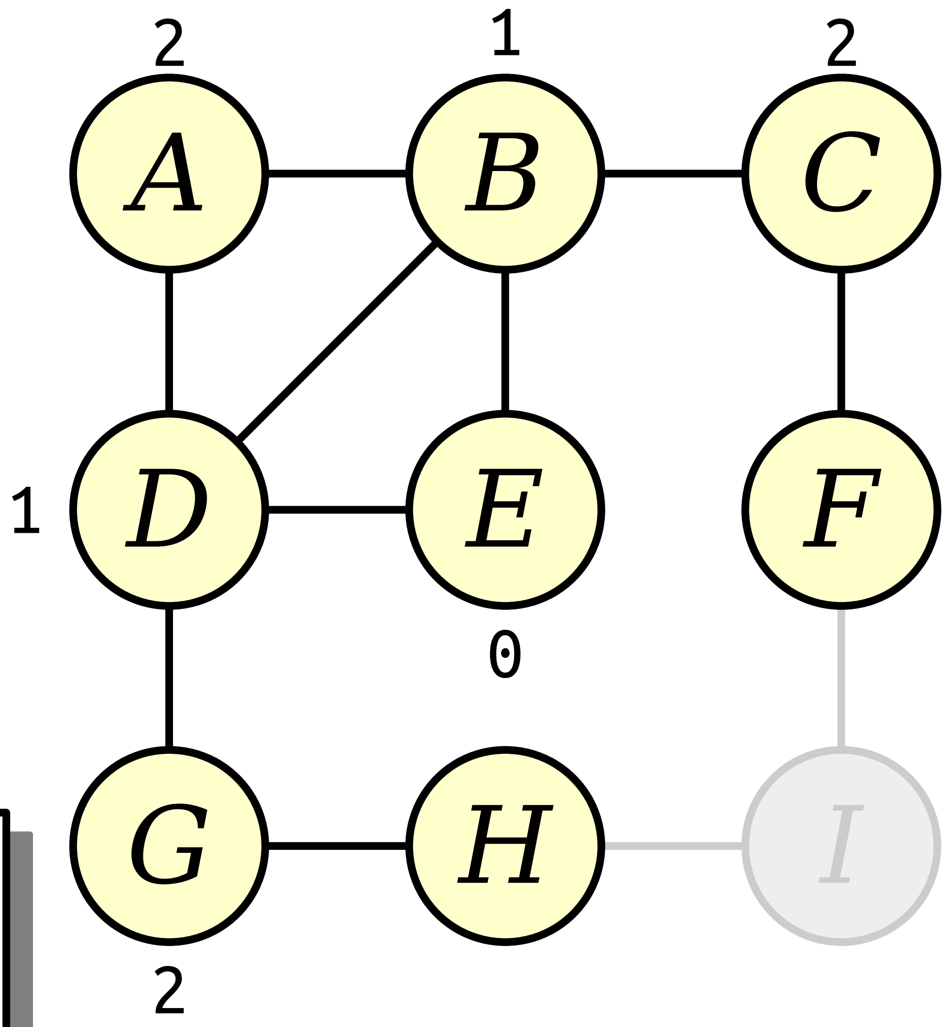
Core idea: Find everything one hop away from the start, then two hops away, then three hops away, etc.



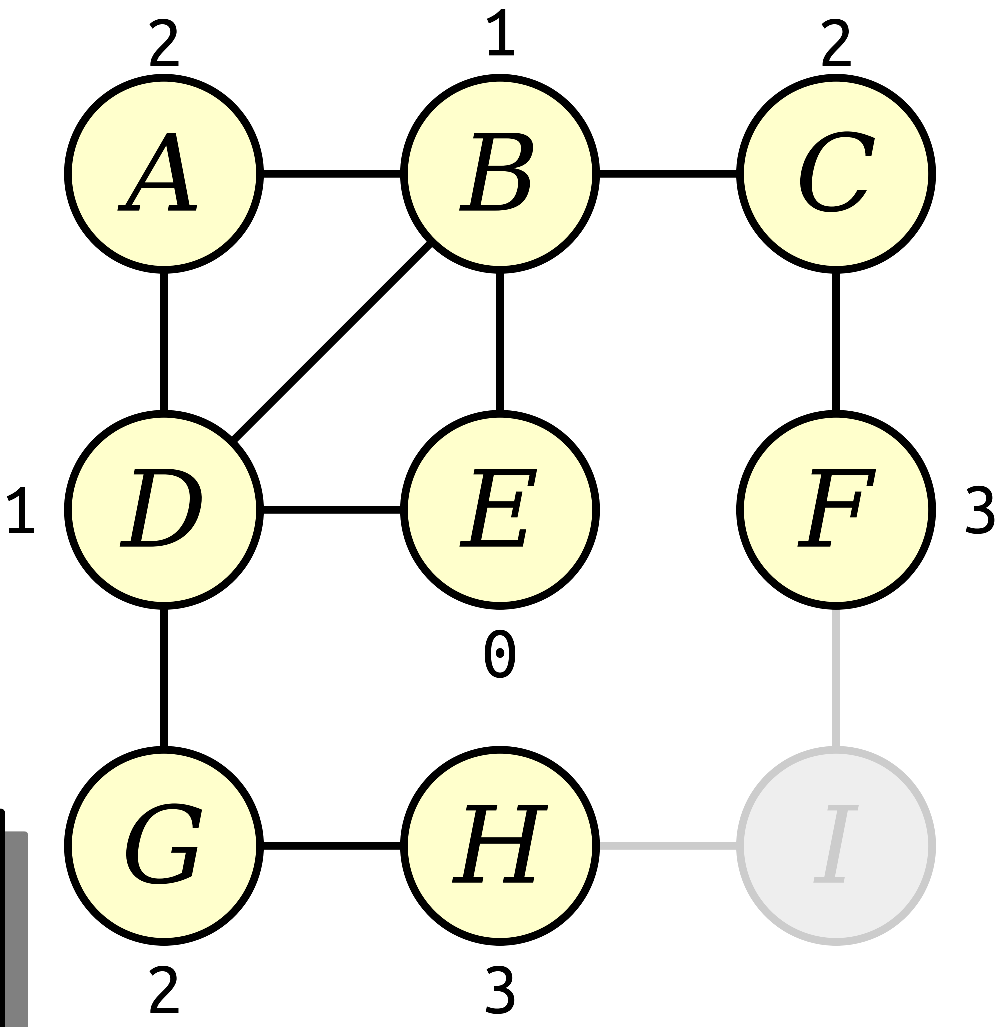
Core idea: Find everything one hop away from the start, then two hops away, then three hops away, etc.



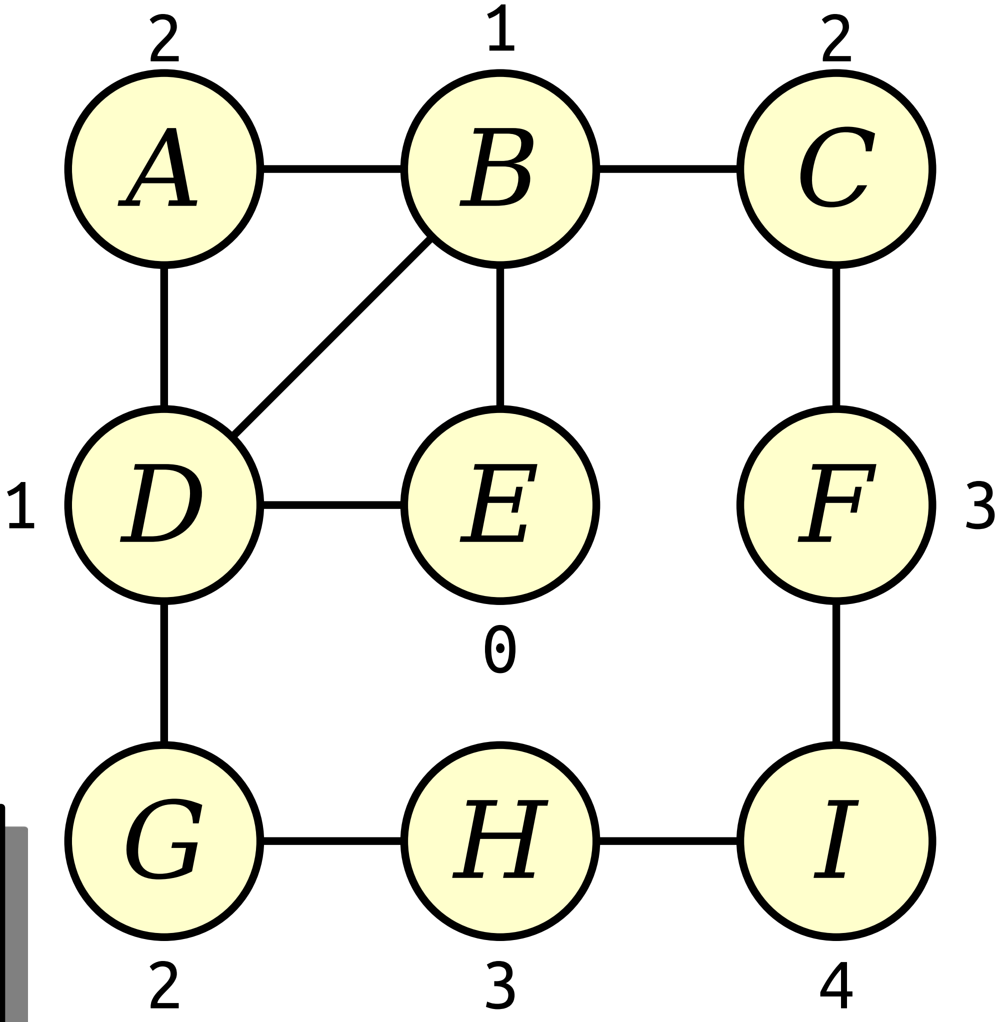
Core idea: Find everything one hop away from the start, then two hops away, then three hops away, etc.



Core idea: Find everything one hop away from the start, then two hops away, then three hops away, etc.

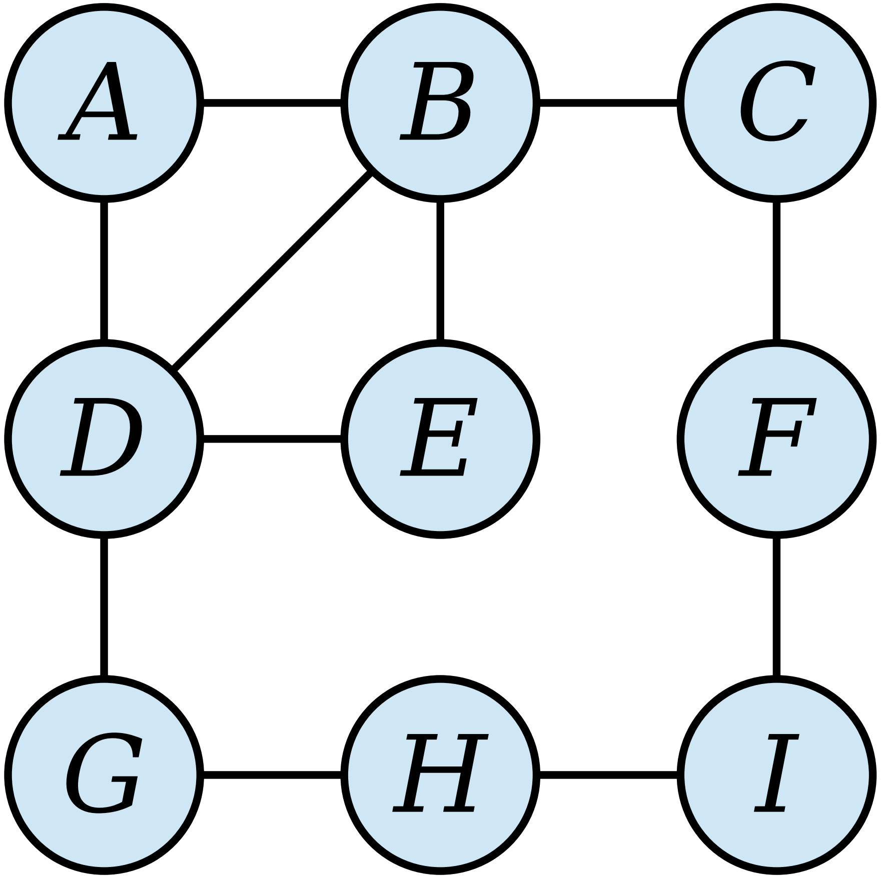


Core idea: Find everything one hop away from the start, then two hops away, then three hops away, etc.

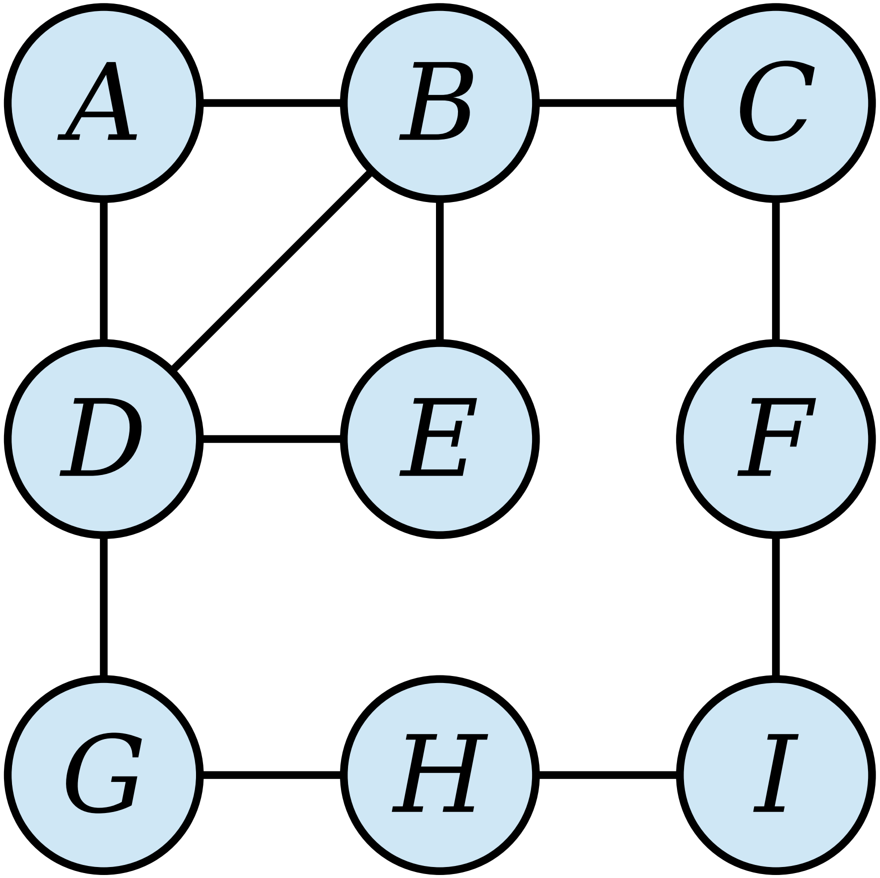


Core idea: Find everything one hop away from the start, then two hops away, then three hops away, etc.

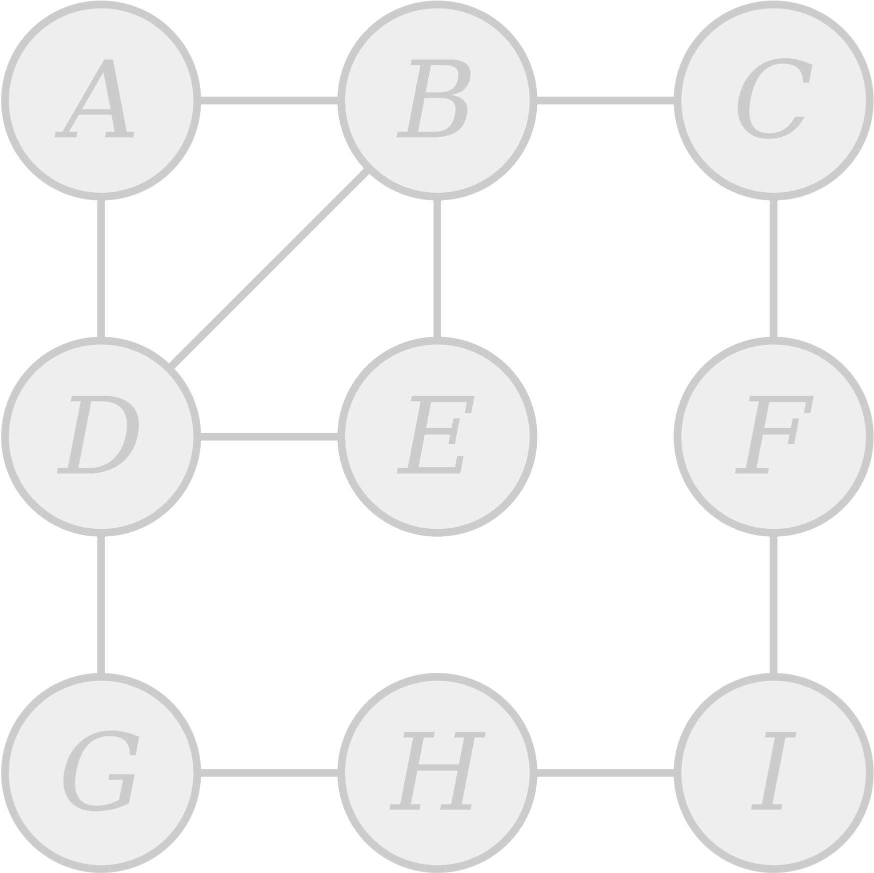
Implementing this Idea



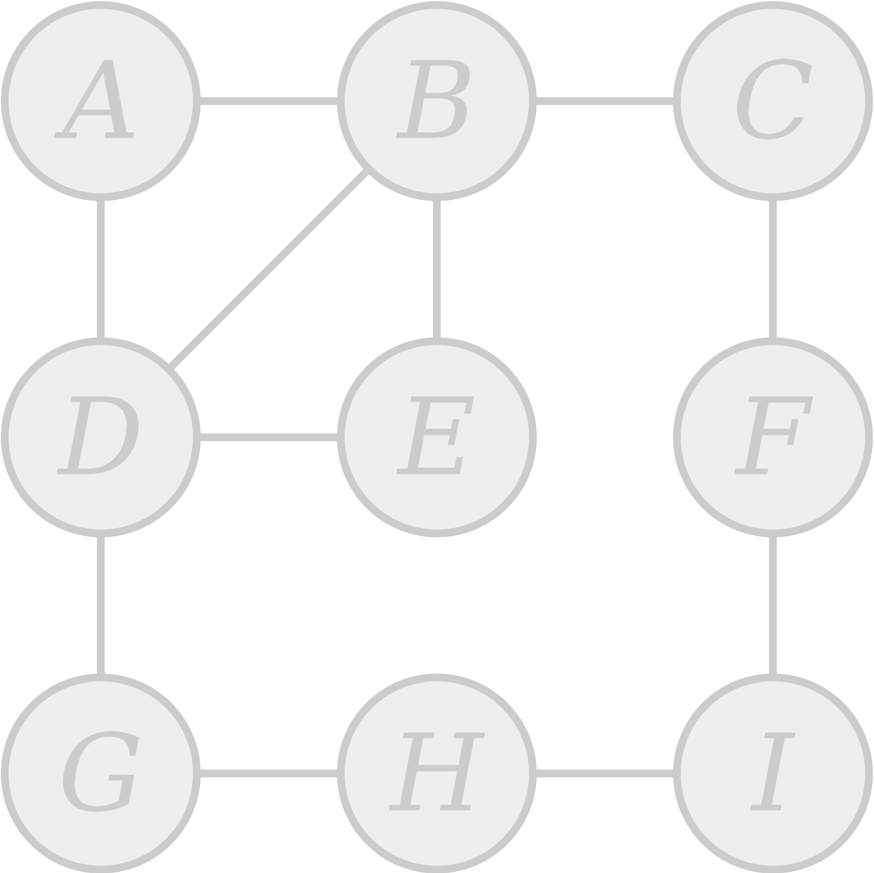
Visit nodes in ascending order of distance from the start node E.



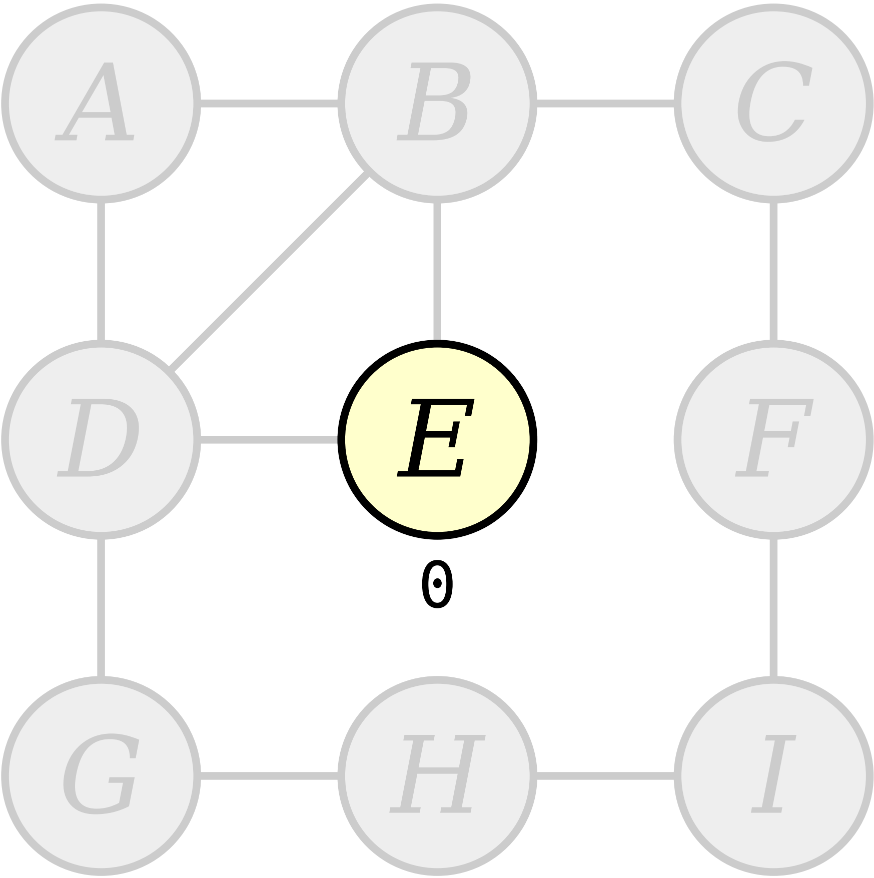
Load newly-discovered nodes into a queue.



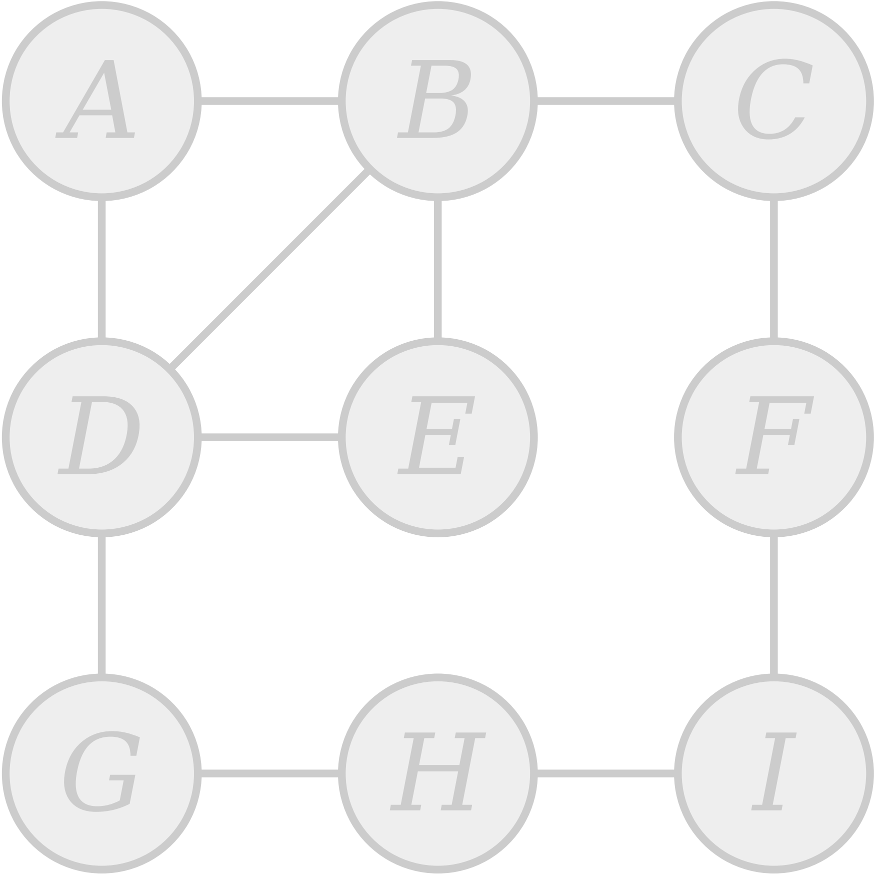
Visit nodes in ascending order of distance from the start node E.



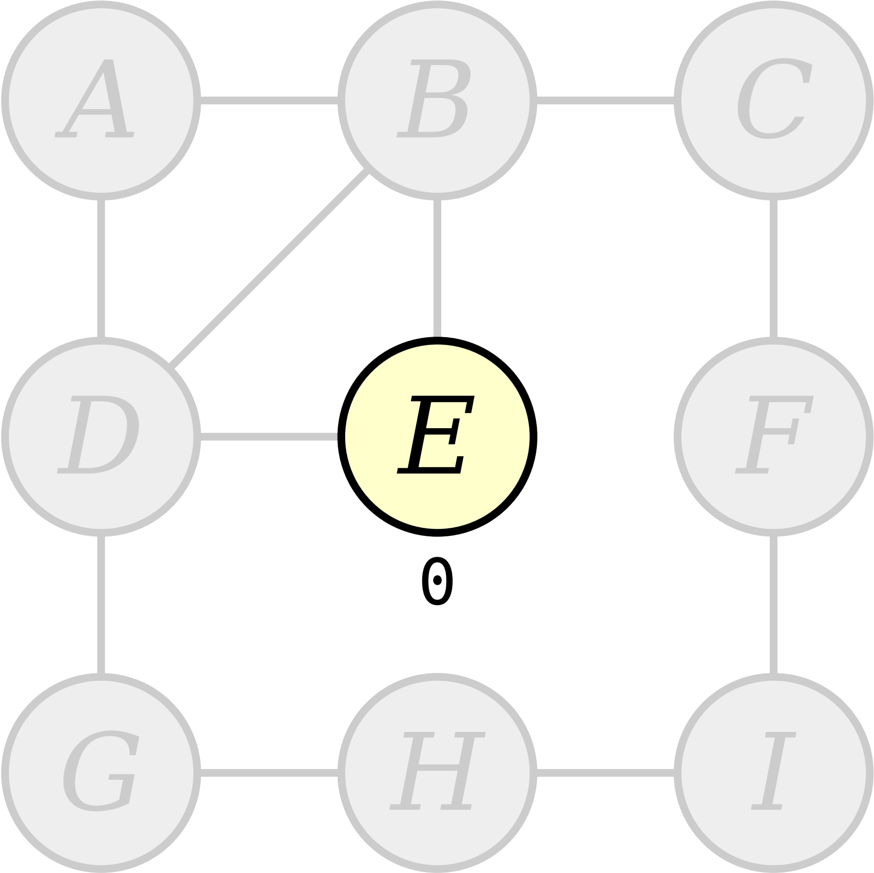
Load newly-discovered nodes into a queue.



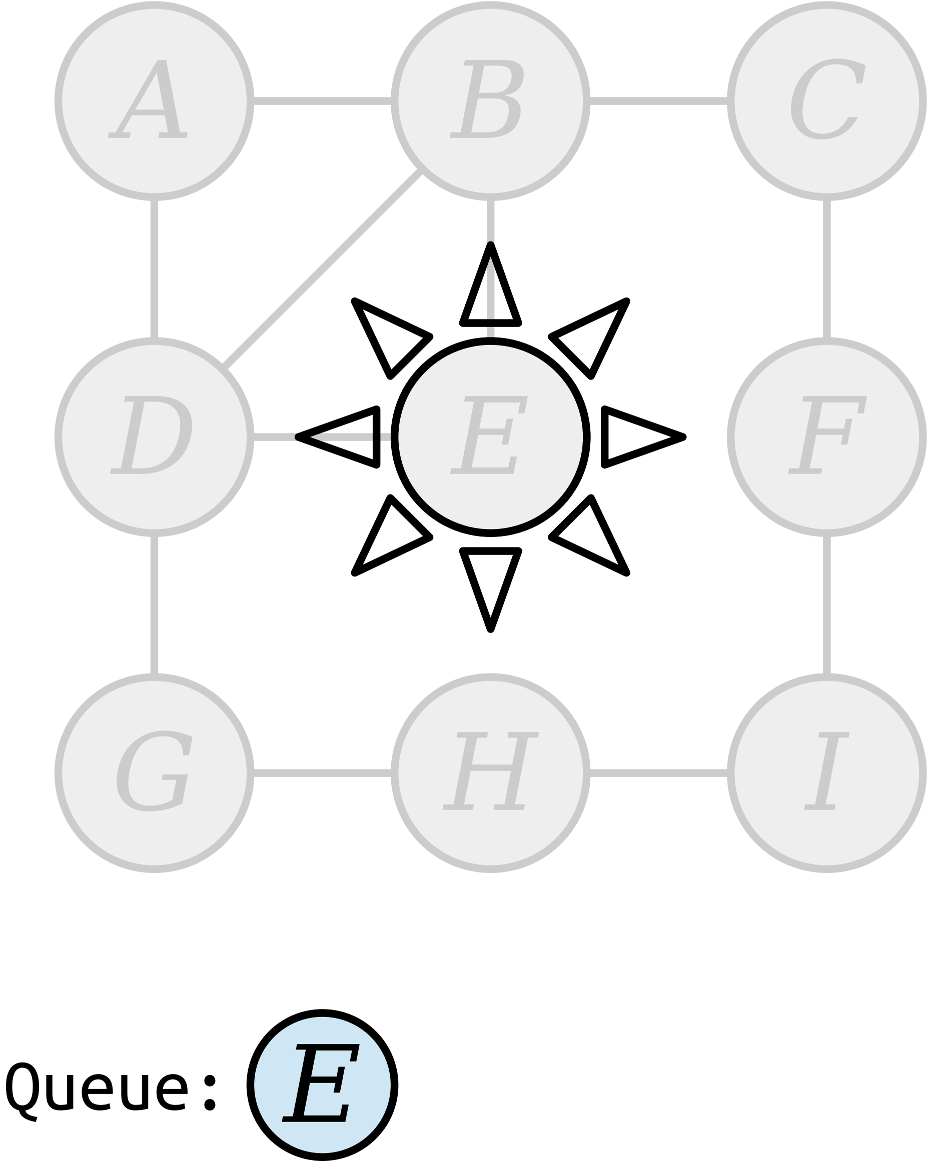
Visit nodes in ascending order of distance from the start node *E*.



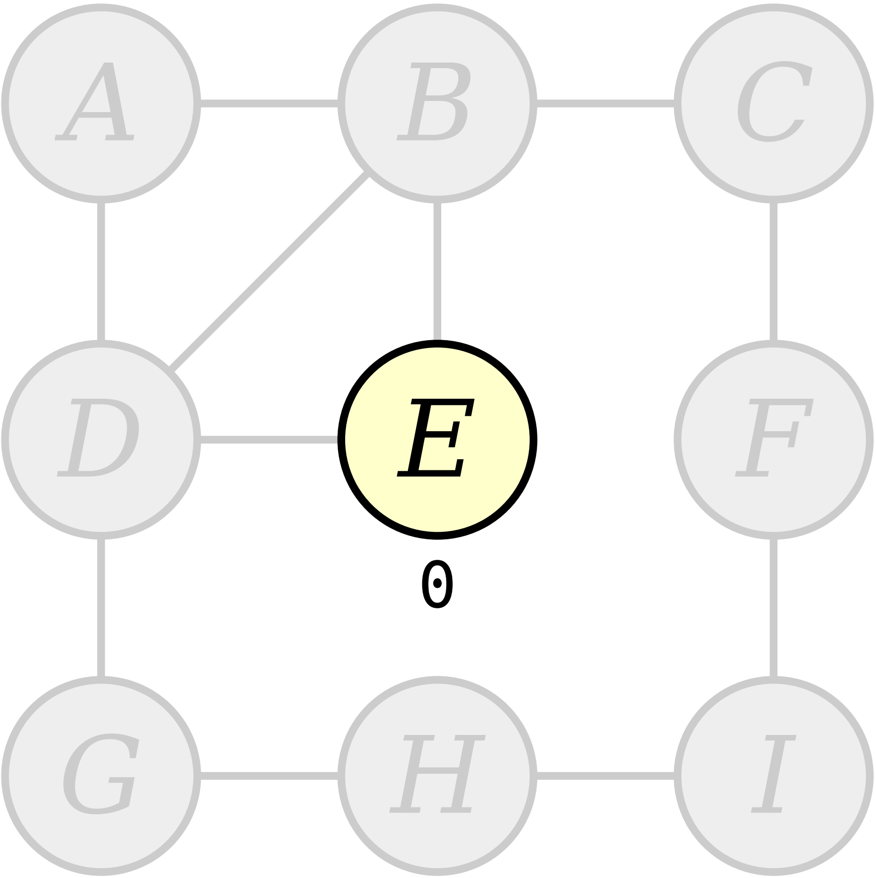
Load newly-discovered nodes into a queue.



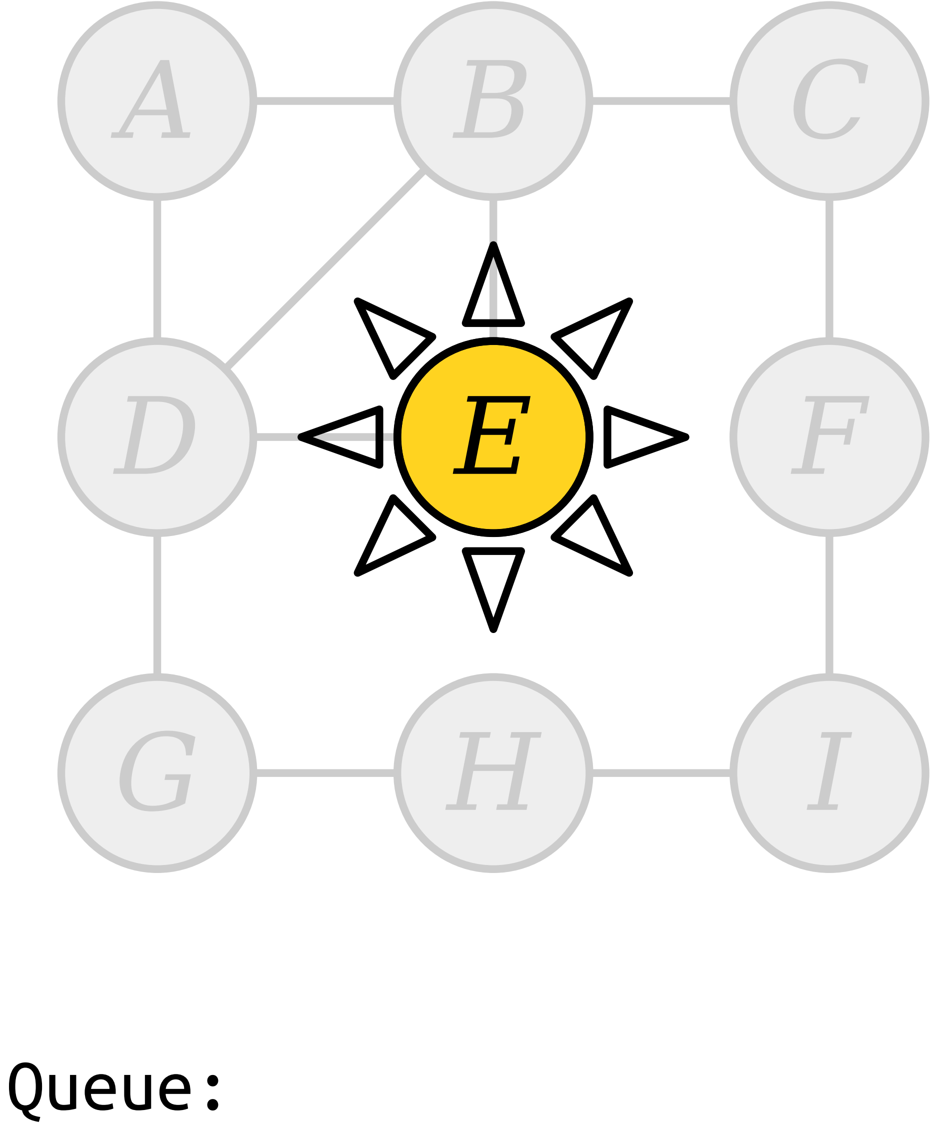
Visit nodes in ascending order of distance from the start node *E*.



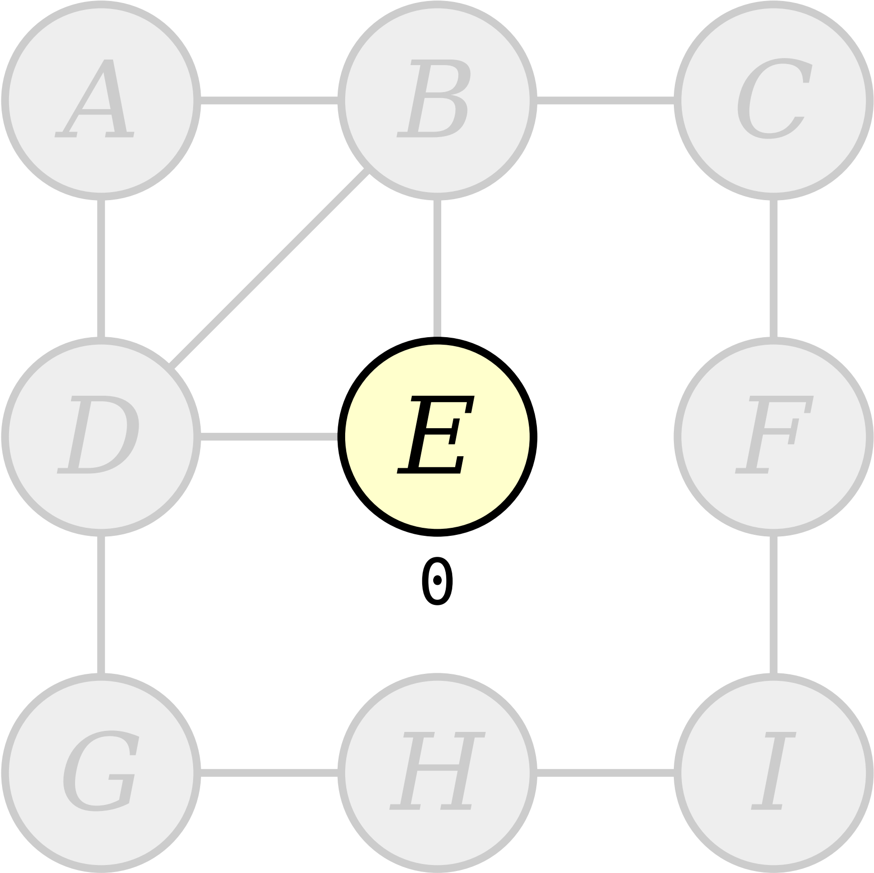
Load newly-discovered nodes into a queue.



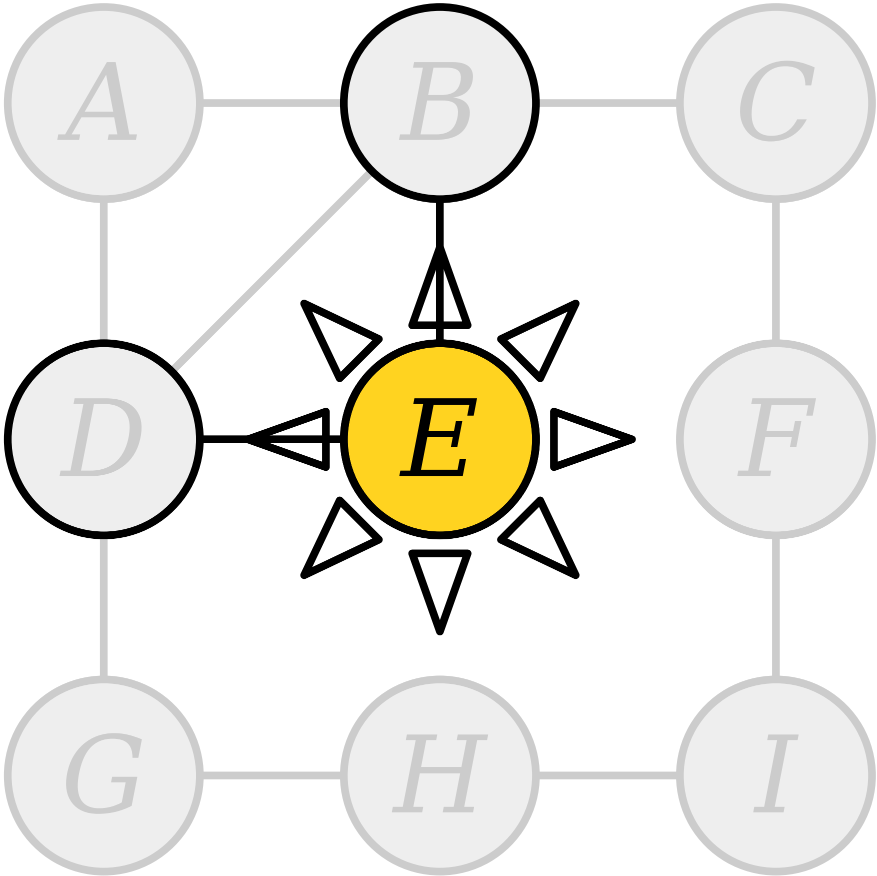
Visit nodes in ascending order of distance from the start node *E*.



Load newly-discovered nodes into a queue.

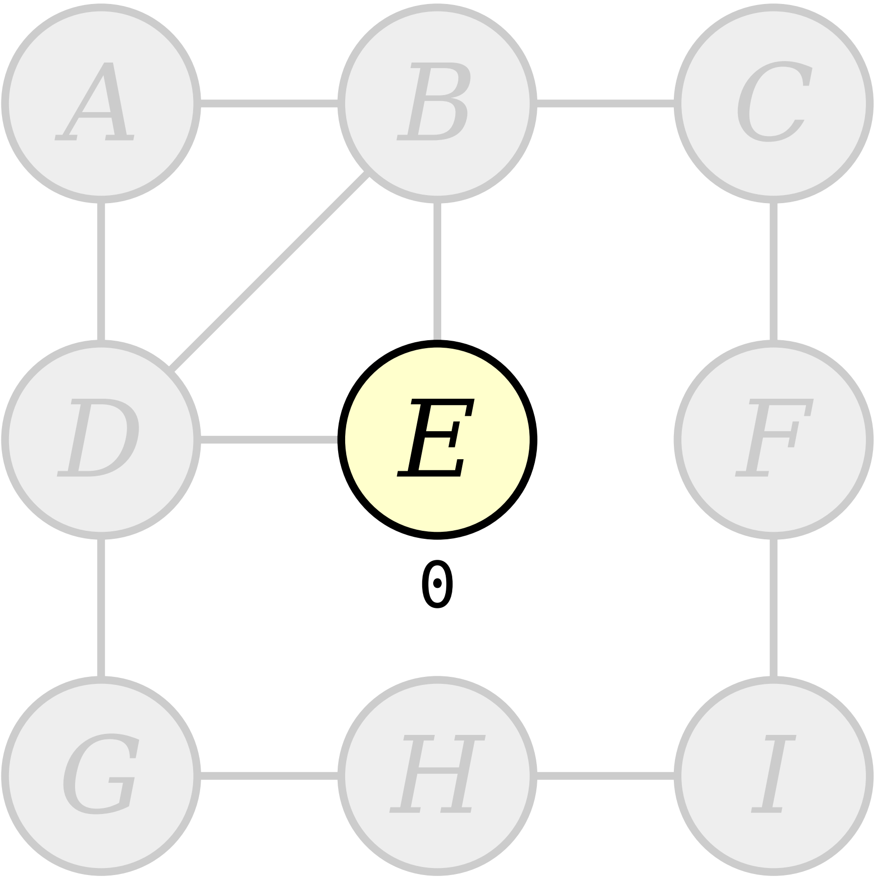


Visit nodes in ascending order of distance from the start node *E*.

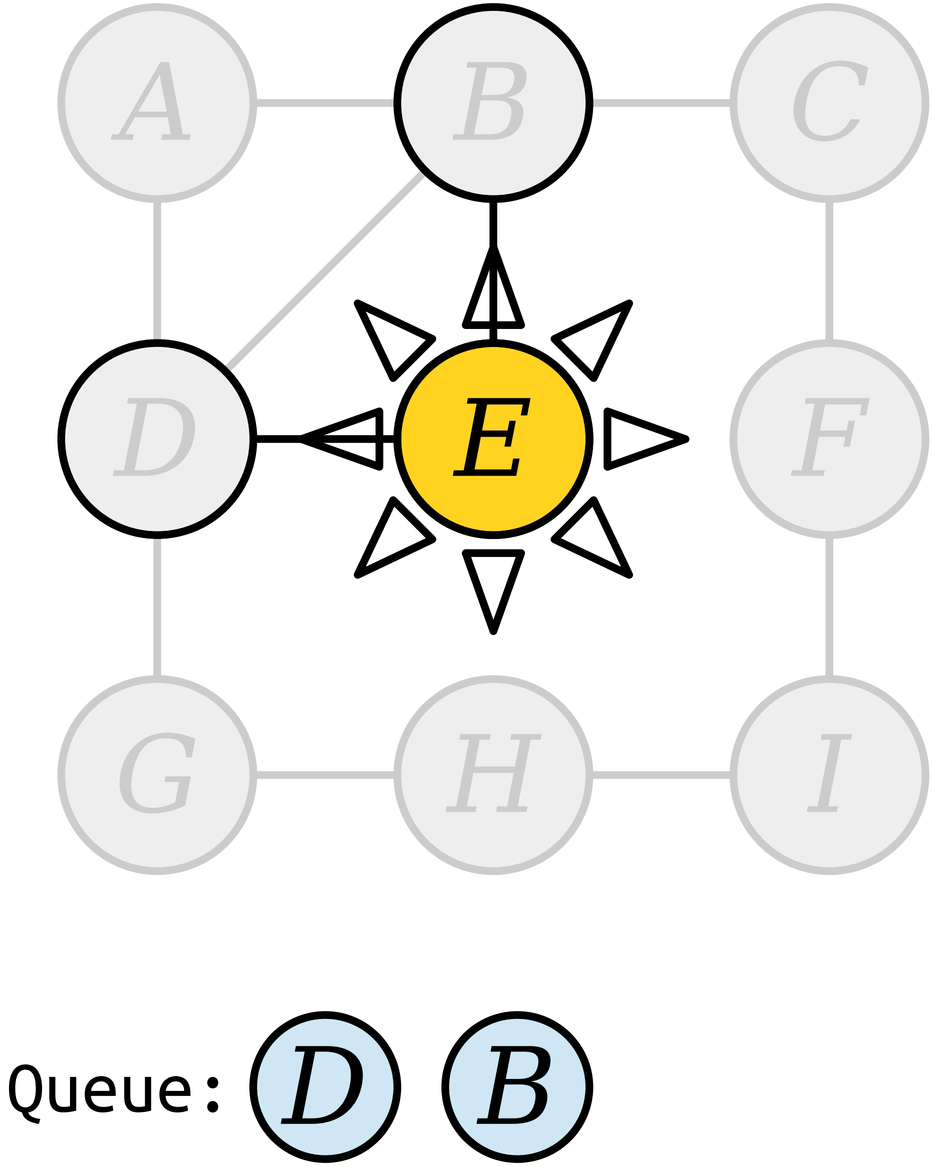


Queue:

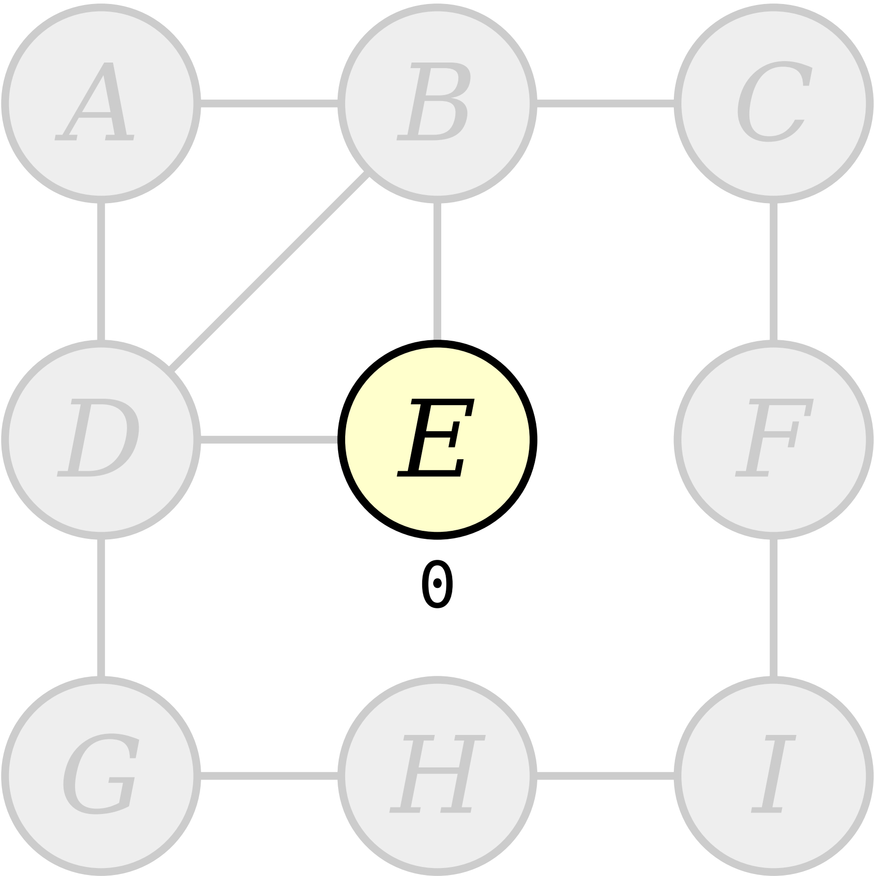
Load newly-discovered nodes into a queue.



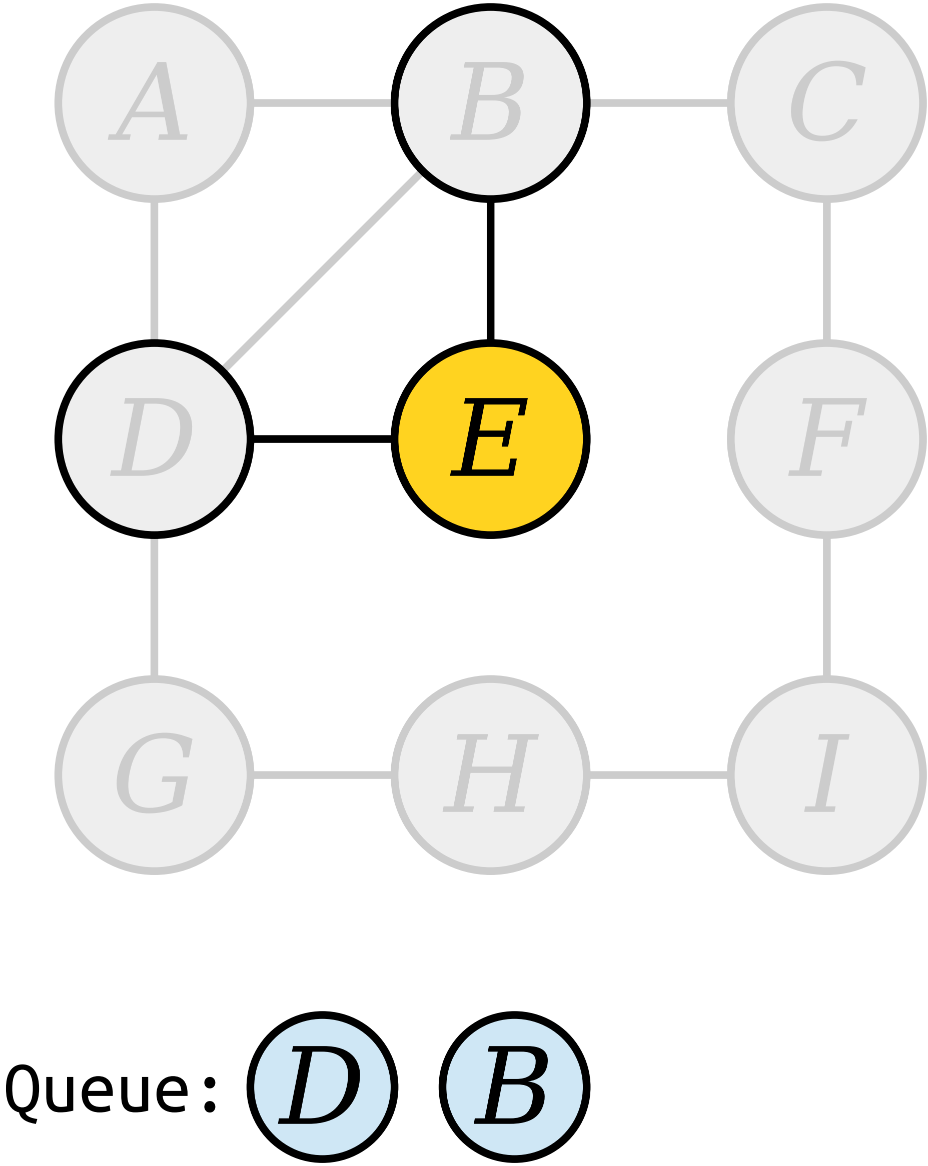
Visit nodes in ascending order of distance from the start node E.



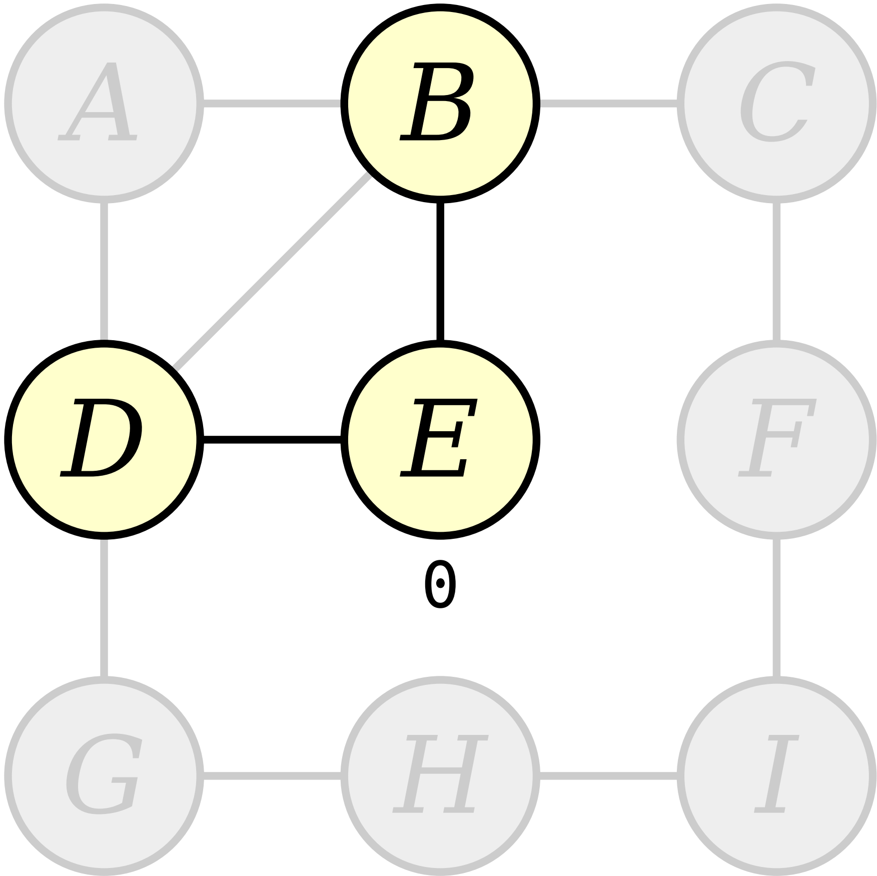
Load newly-discovered nodes into a queue.



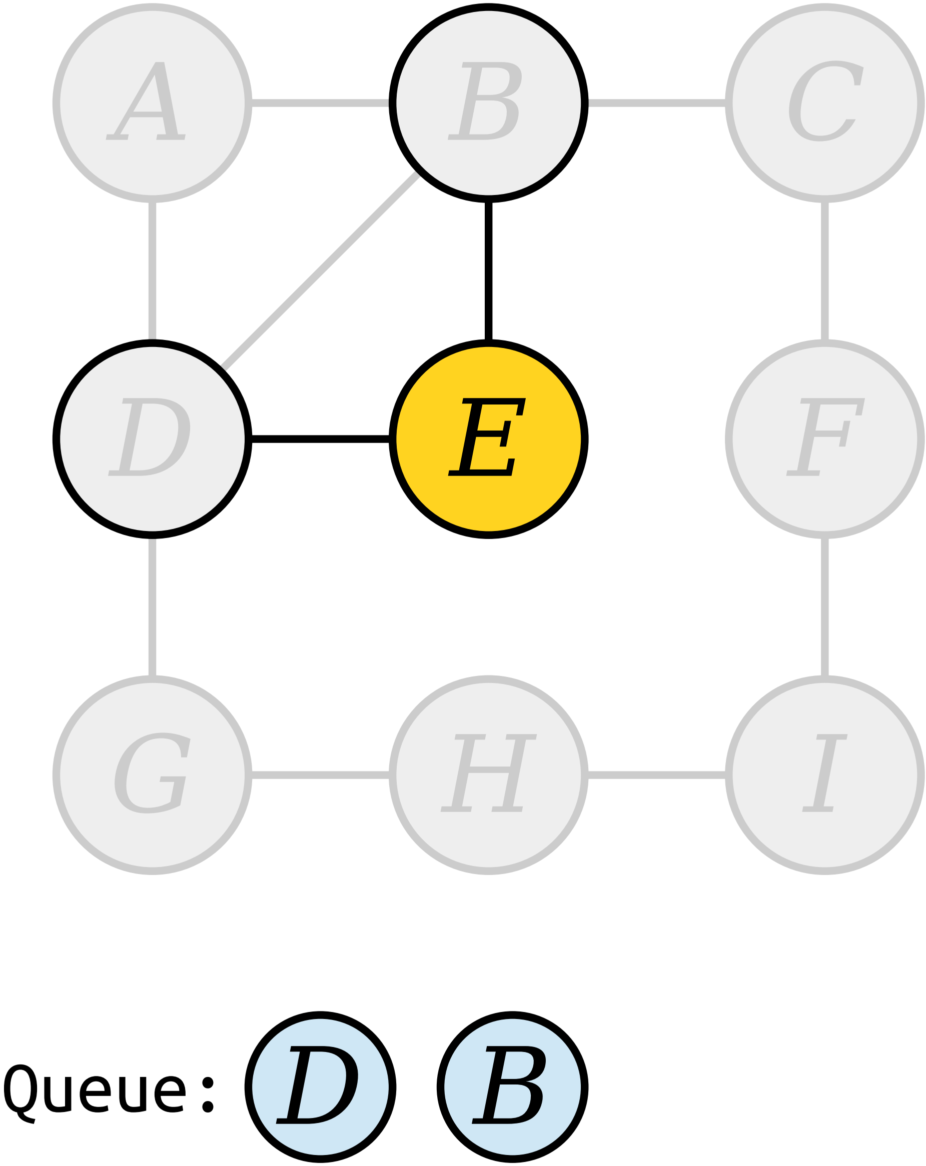
Visit nodes in ascending order of distance from the start node *E*.



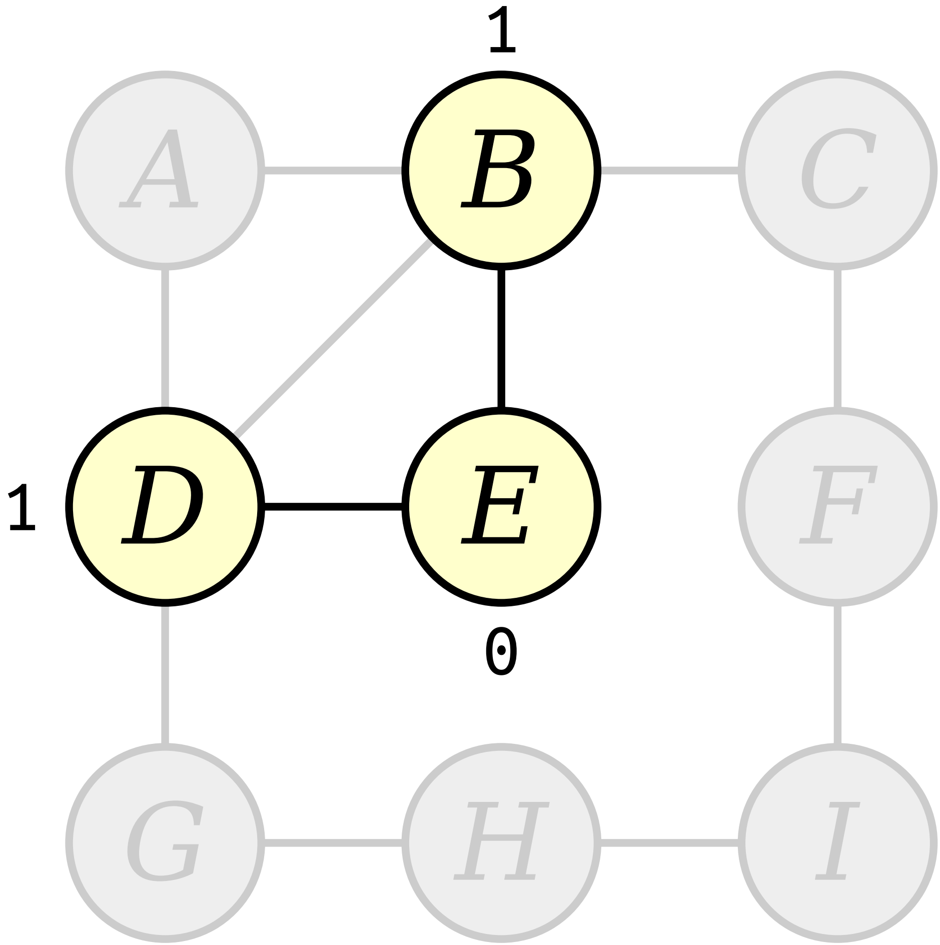
Load newly-discovered nodes into a queue.



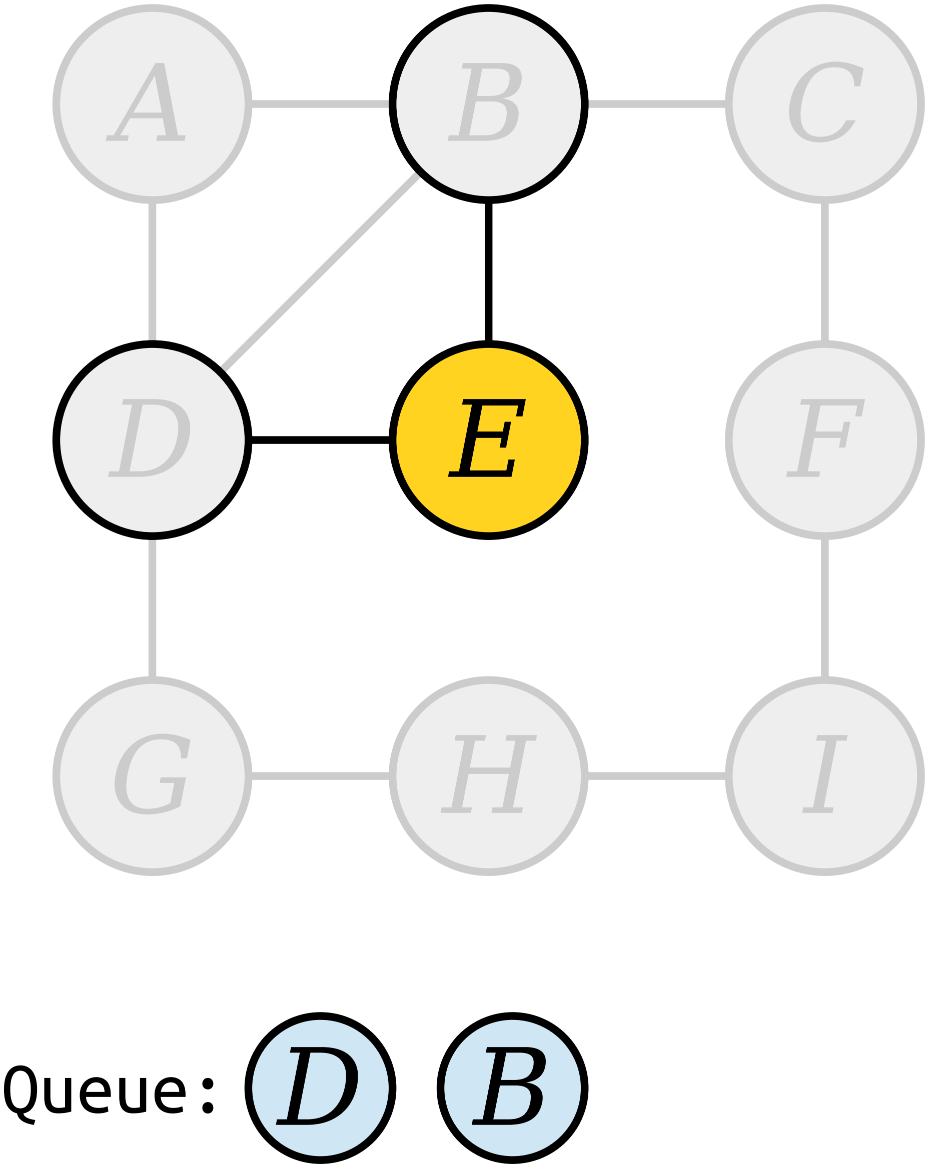
Visit nodes in ascending order of distance from the start node *E*.



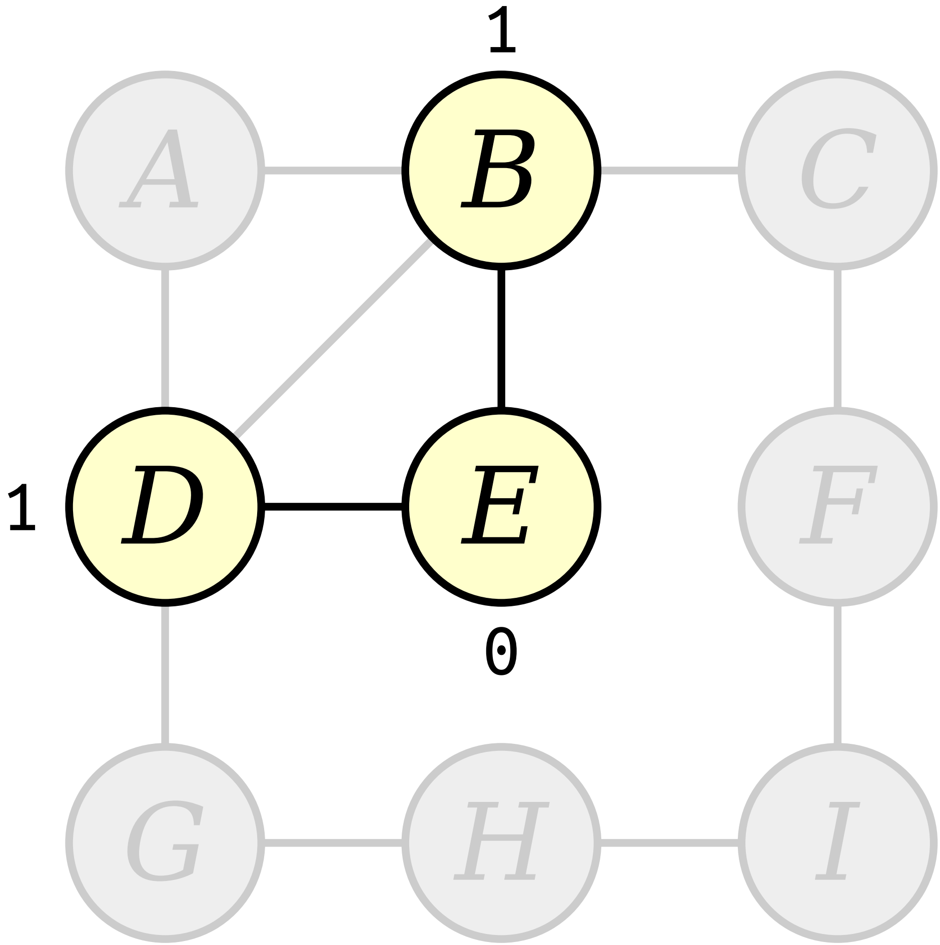
Load newly-discovered nodes into a queue.



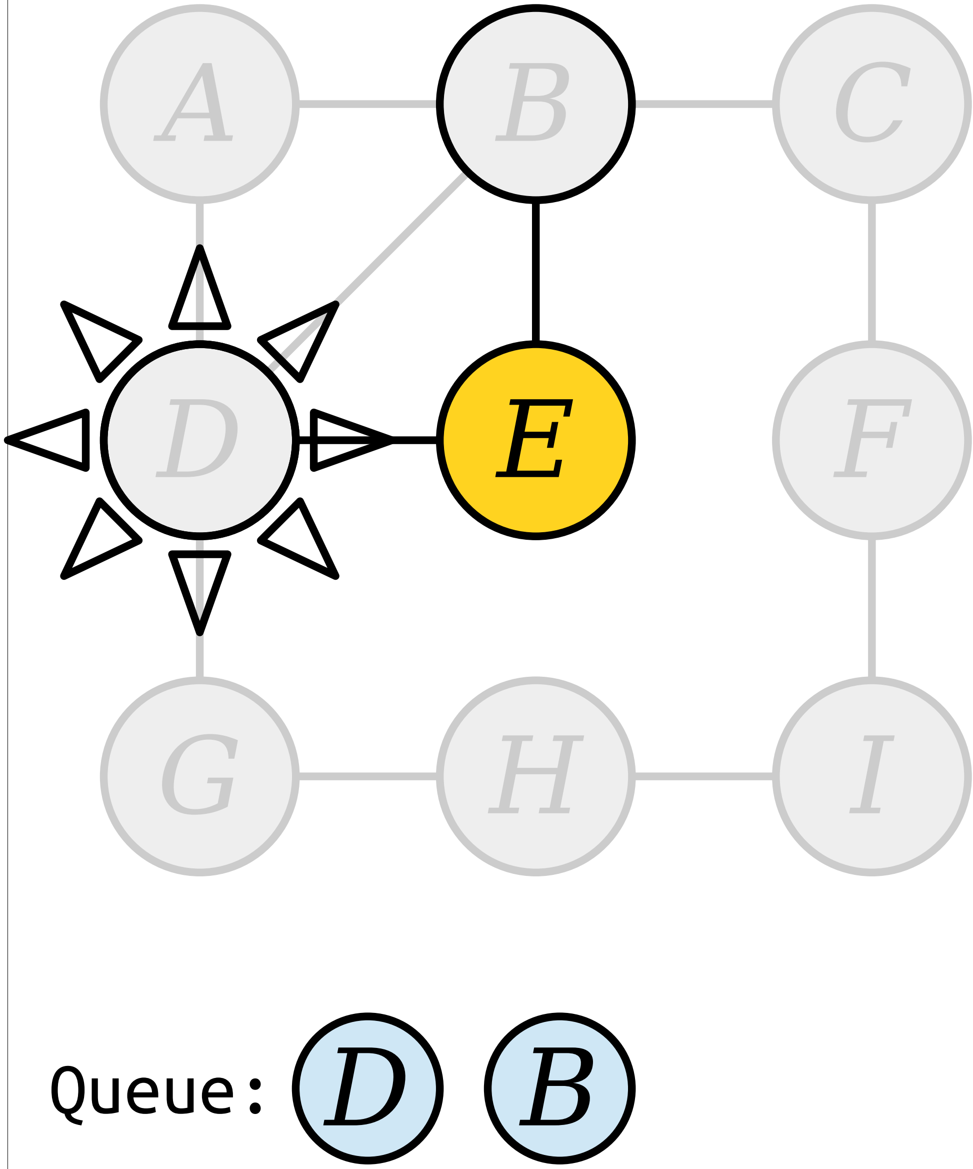
Visit nodes in ascending order of distance from the start node *E*.



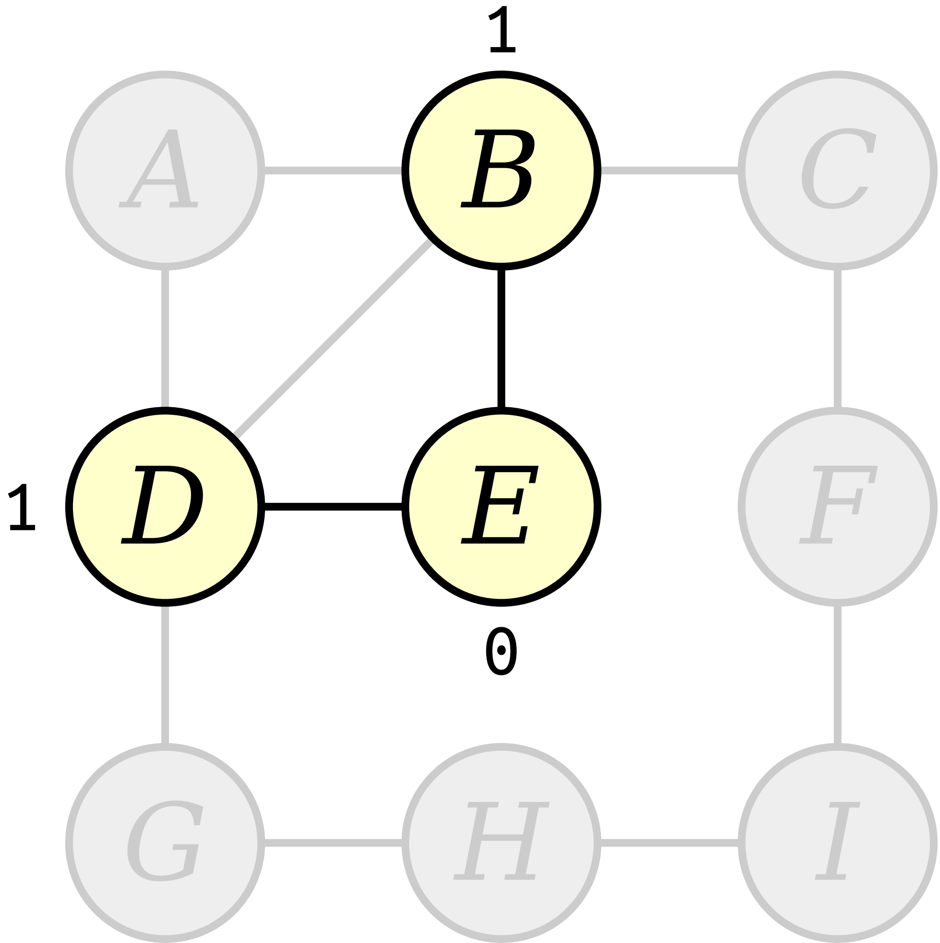
Load newly-discovered nodes into a queue.



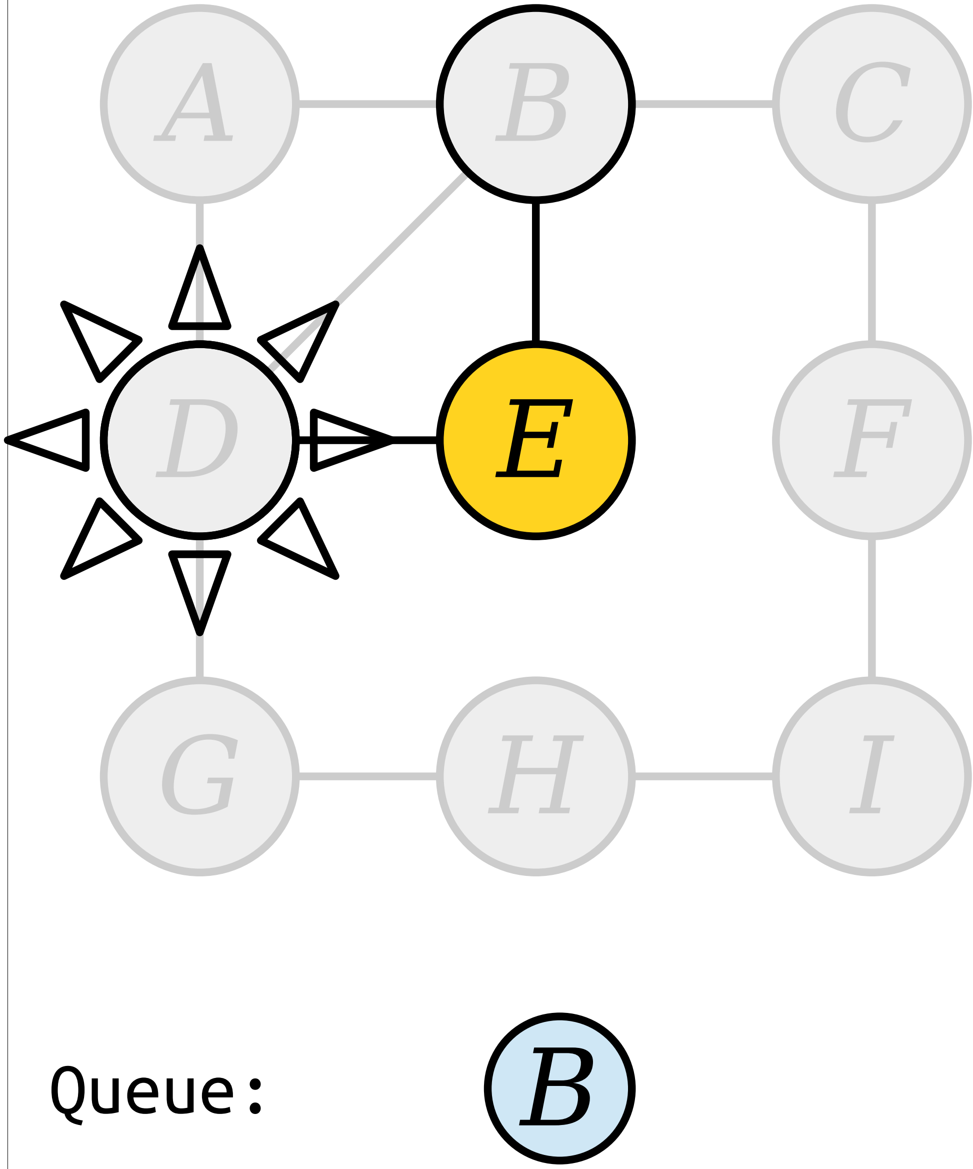
Visit nodes in ascending order of distance from the start node *E*.



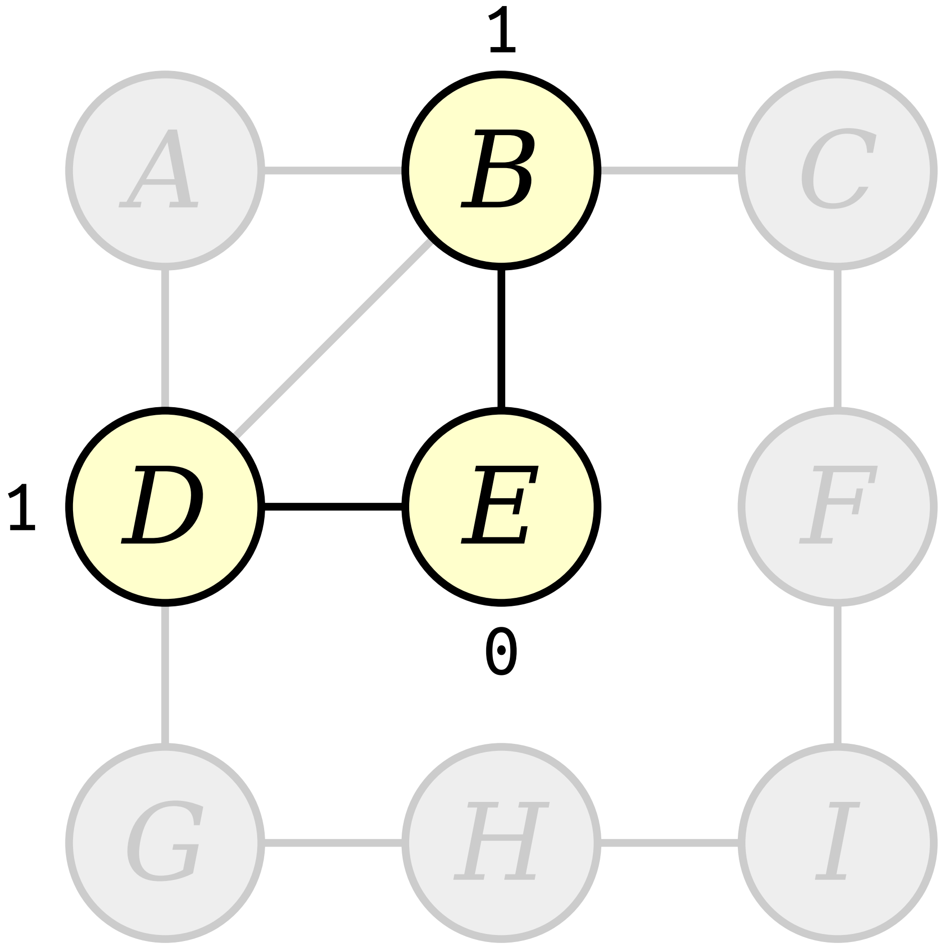
Load newly-discovered nodes into a queue.



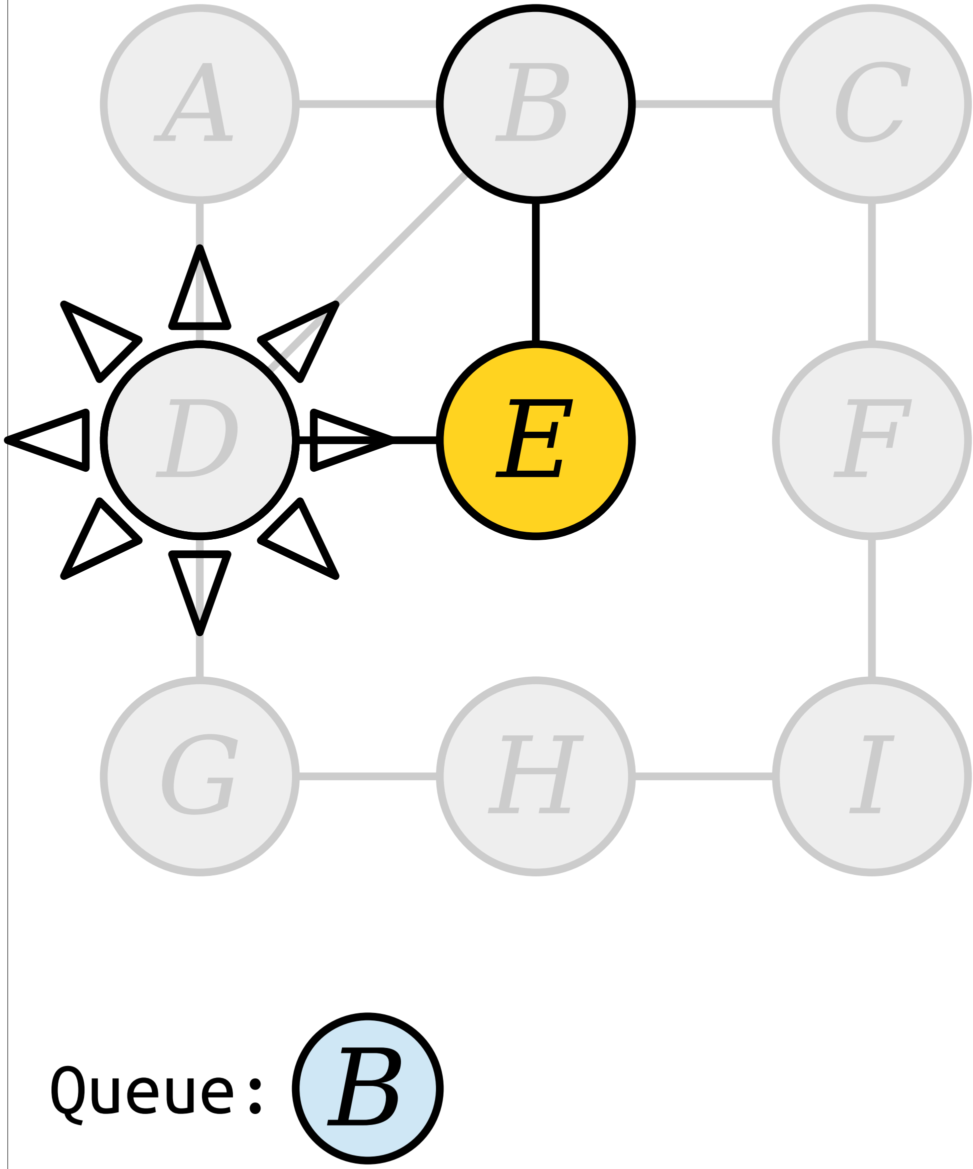
Visit nodes in ascending order of distance from the start node *E*.



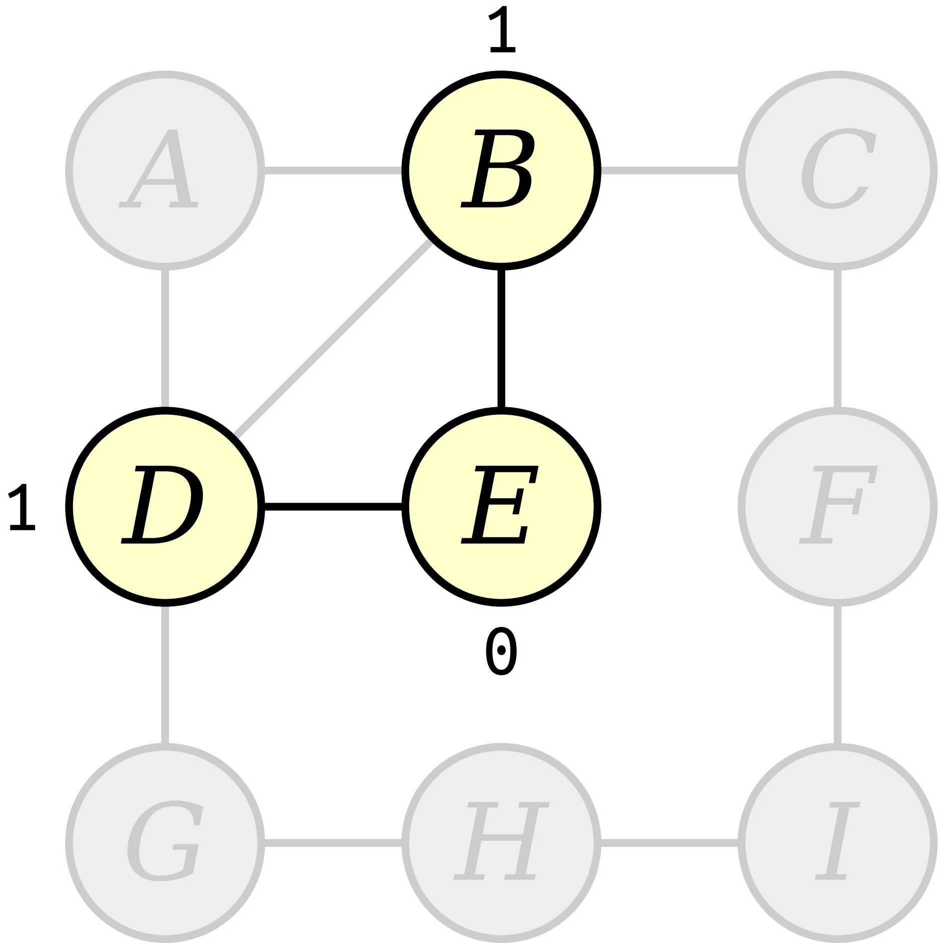
Load newly-discovered nodes into a queue.



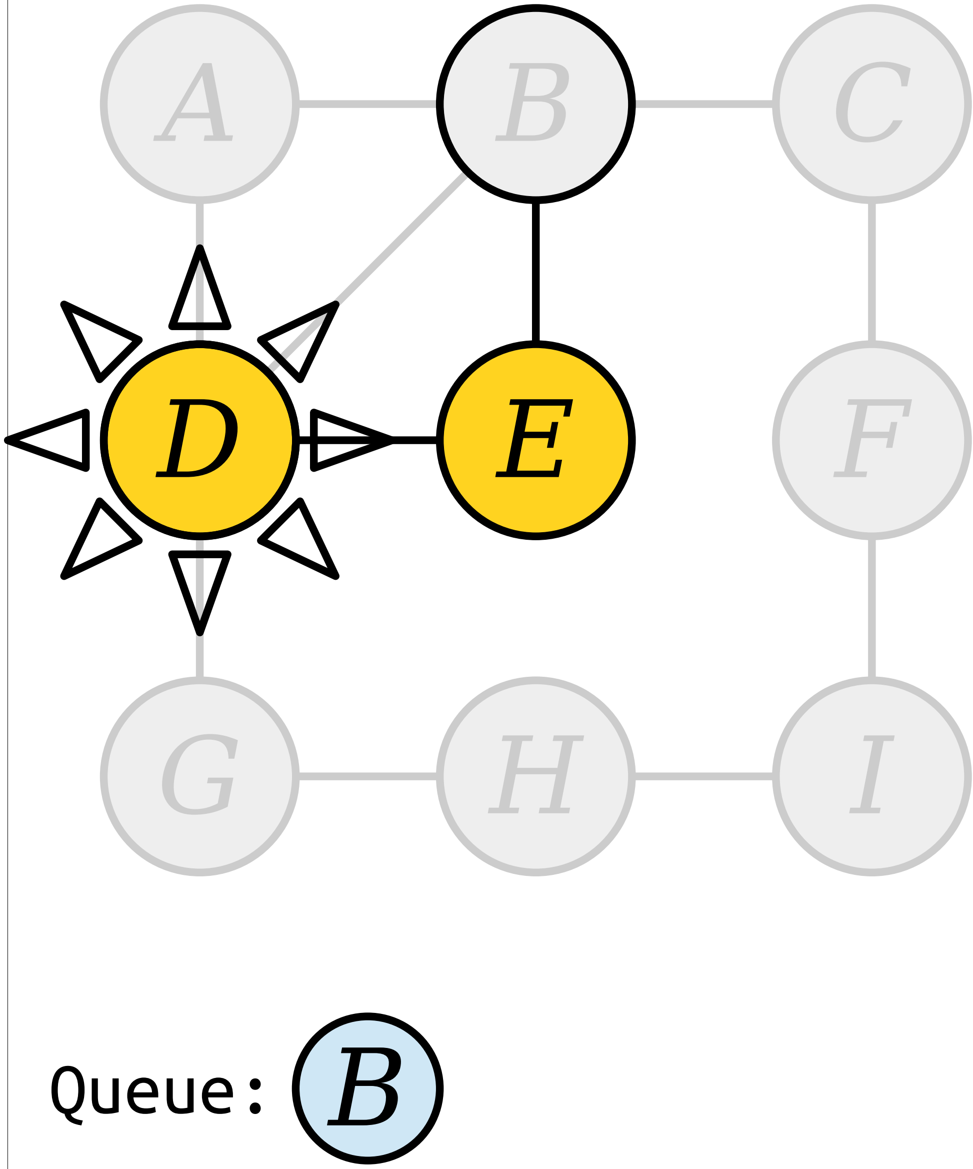
Visit nodes in ascending order of distance from the start node *E*.



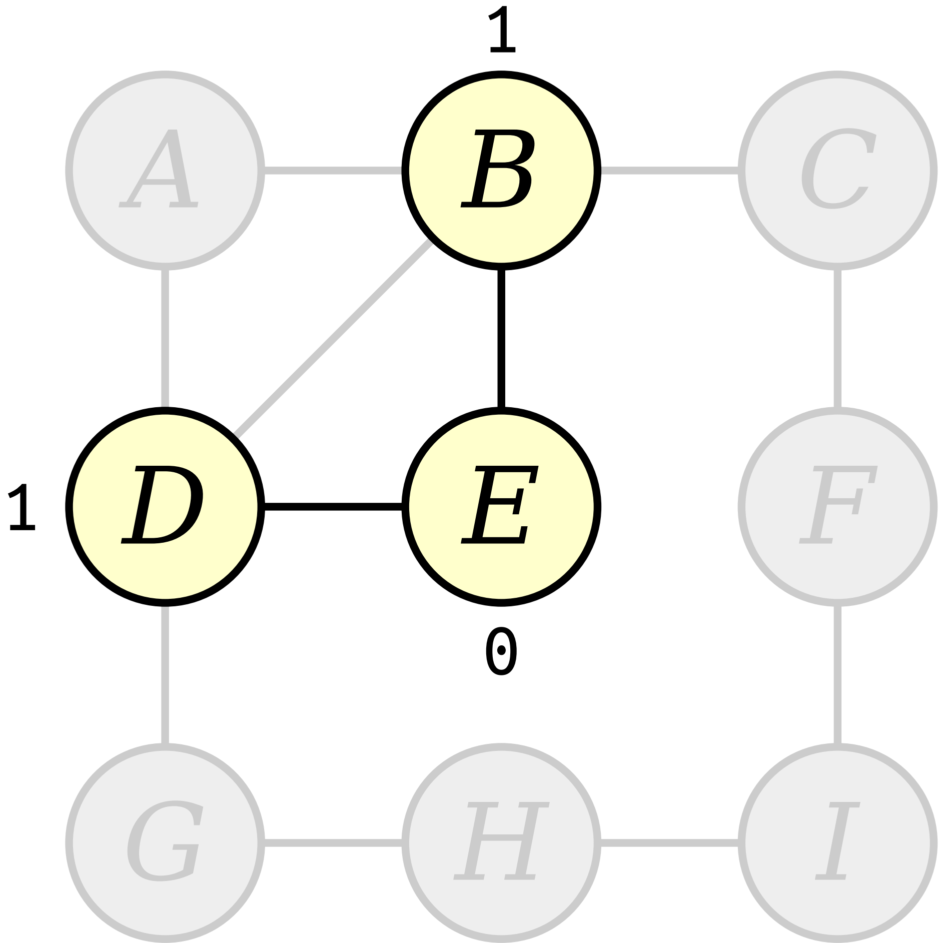
Load newly-discovered nodes into a queue.



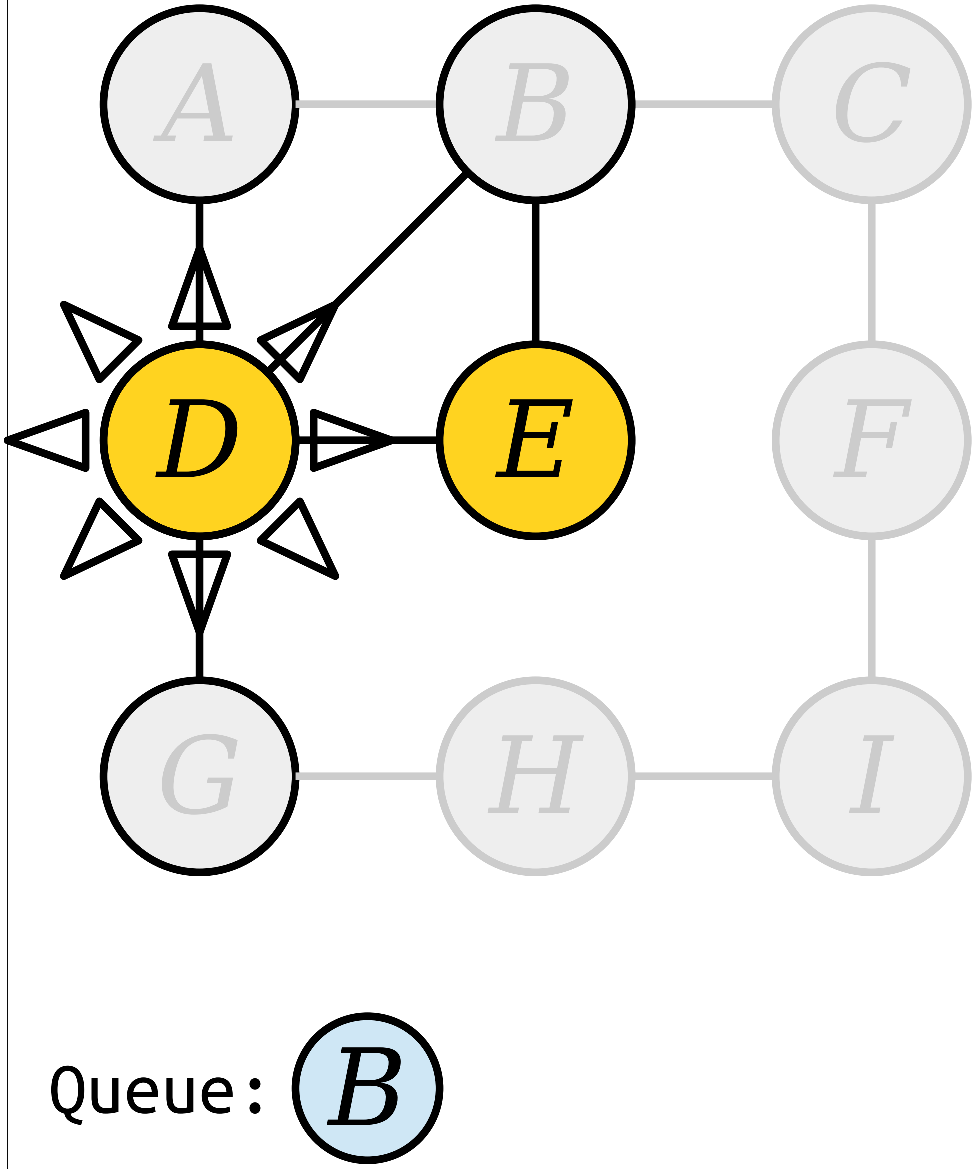
Visit nodes in ascending order of distance from the start node *E*.



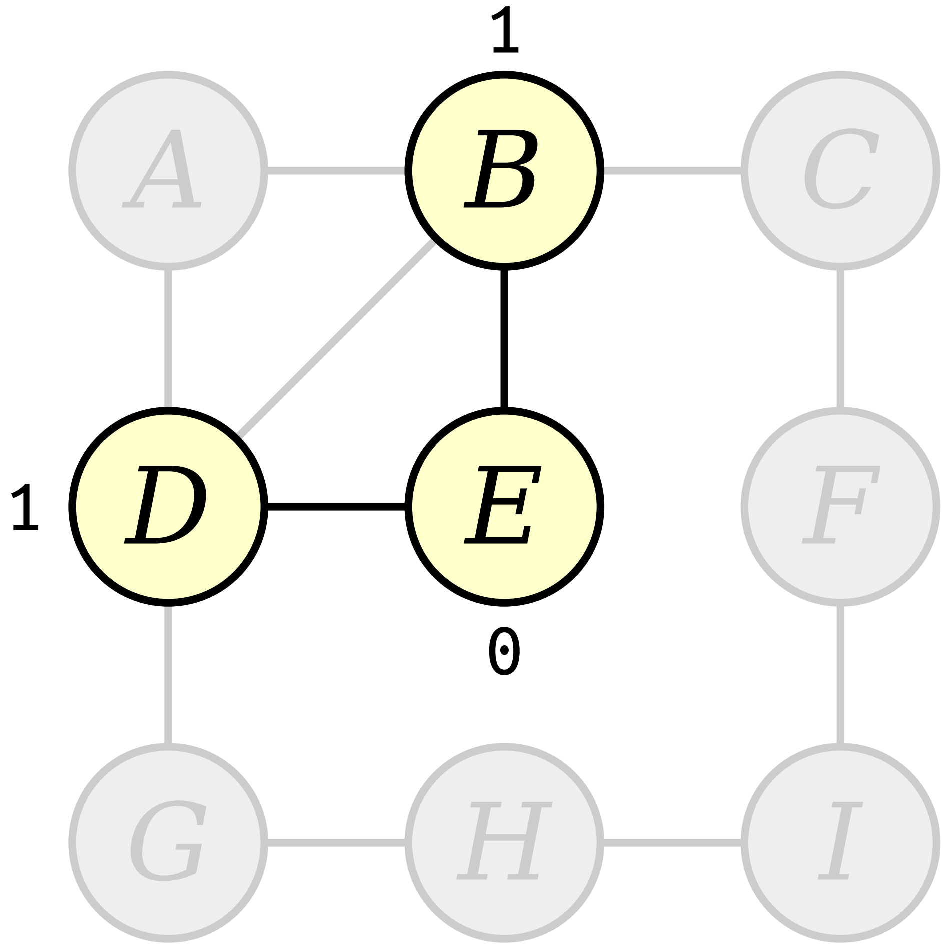
Load newly-discovered nodes into a queue.



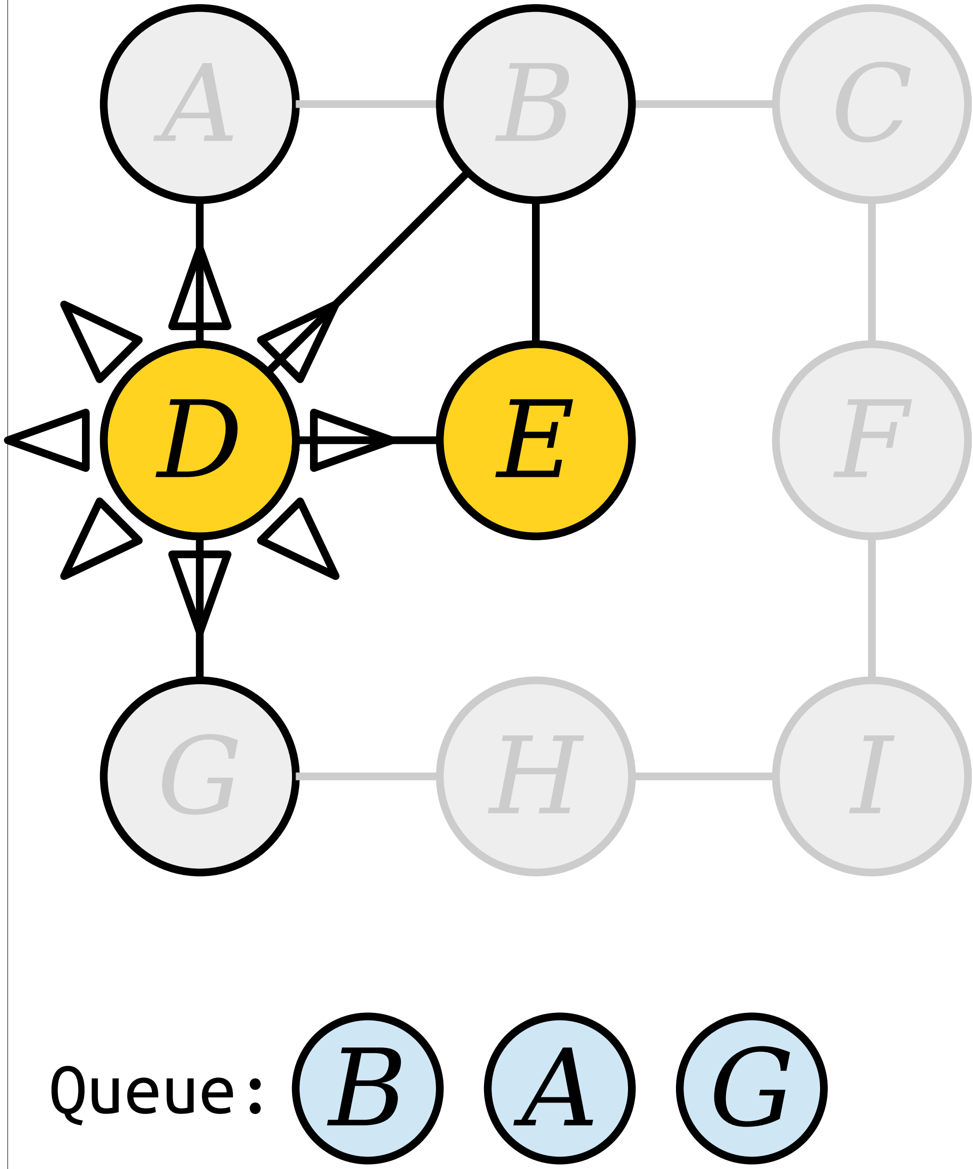
Visit nodes in ascending order of distance from the start node *E*.



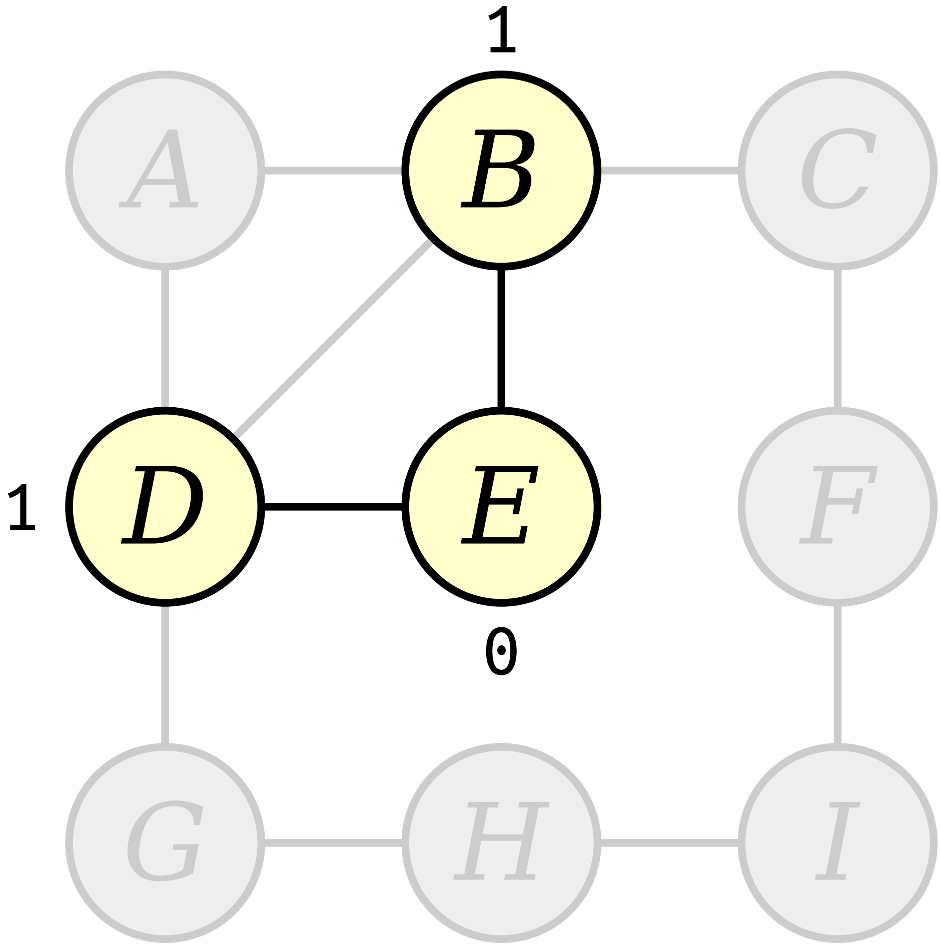
Load newly-discovered nodes into a queue.



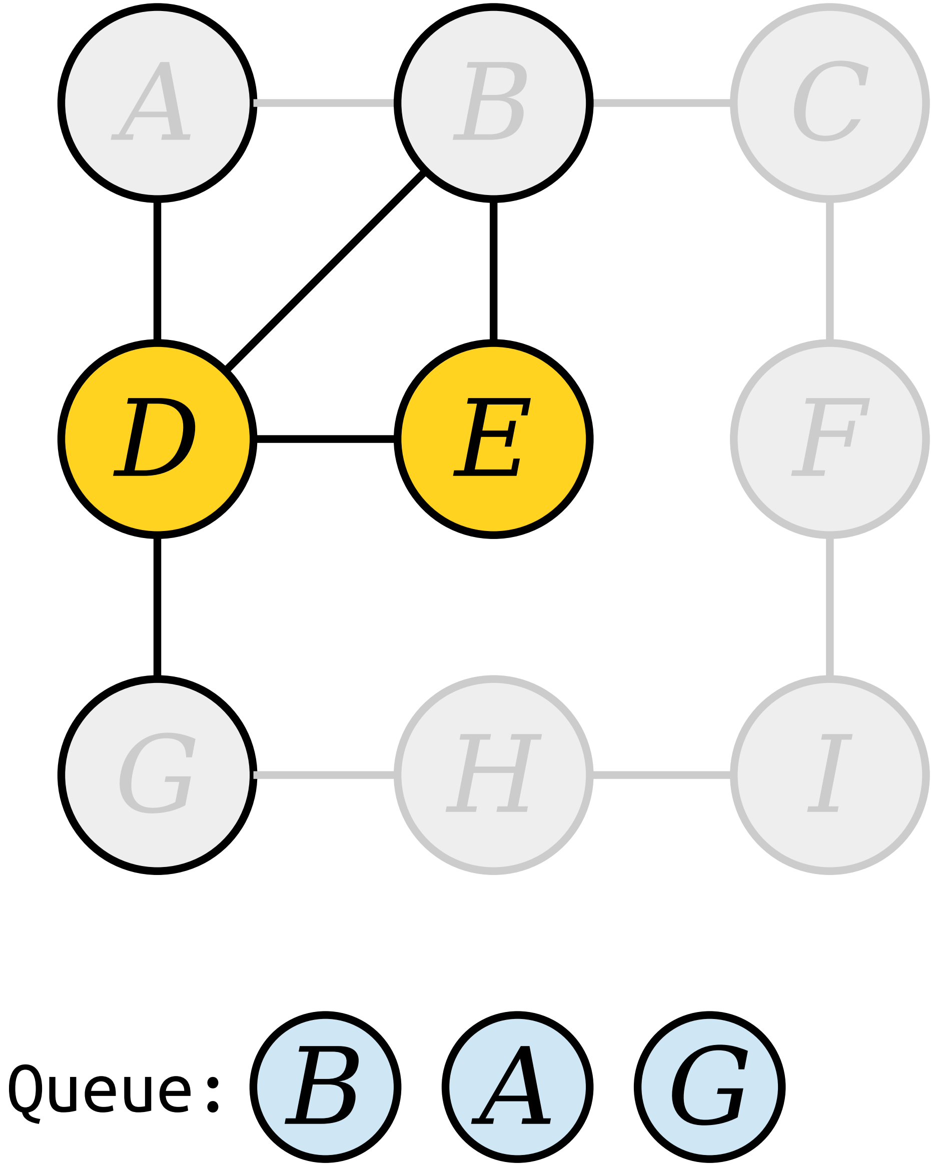
Visit nodes in ascending order of distance from the start node E.



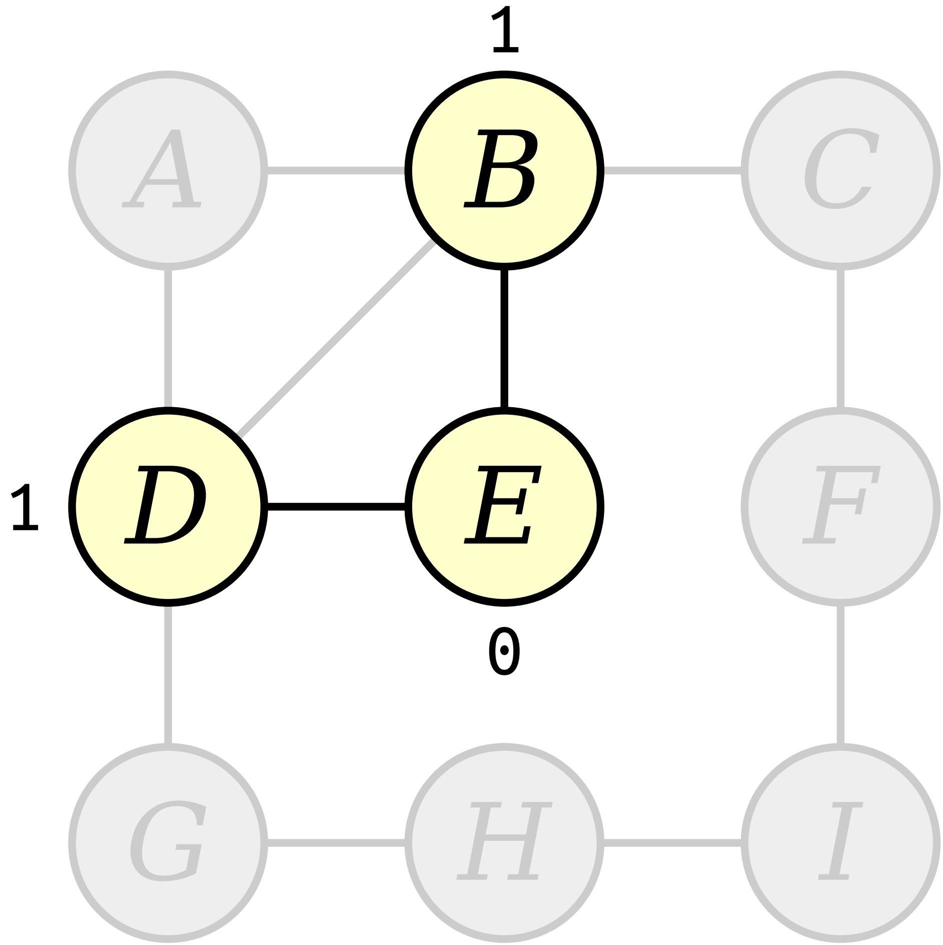
Load newly-discovered nodes into a queue.



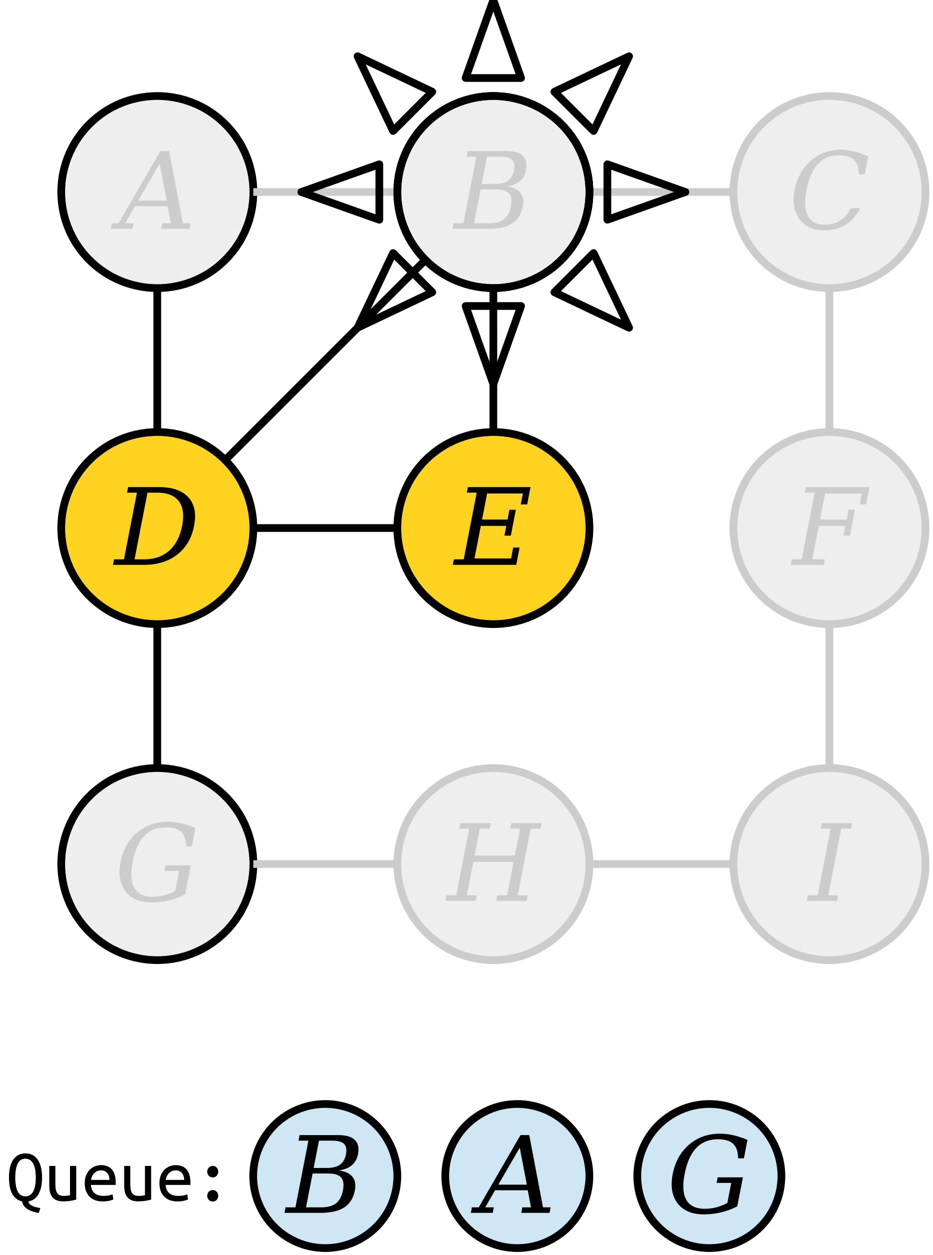
Visit nodes in ascending order of distance from the start node *E*.



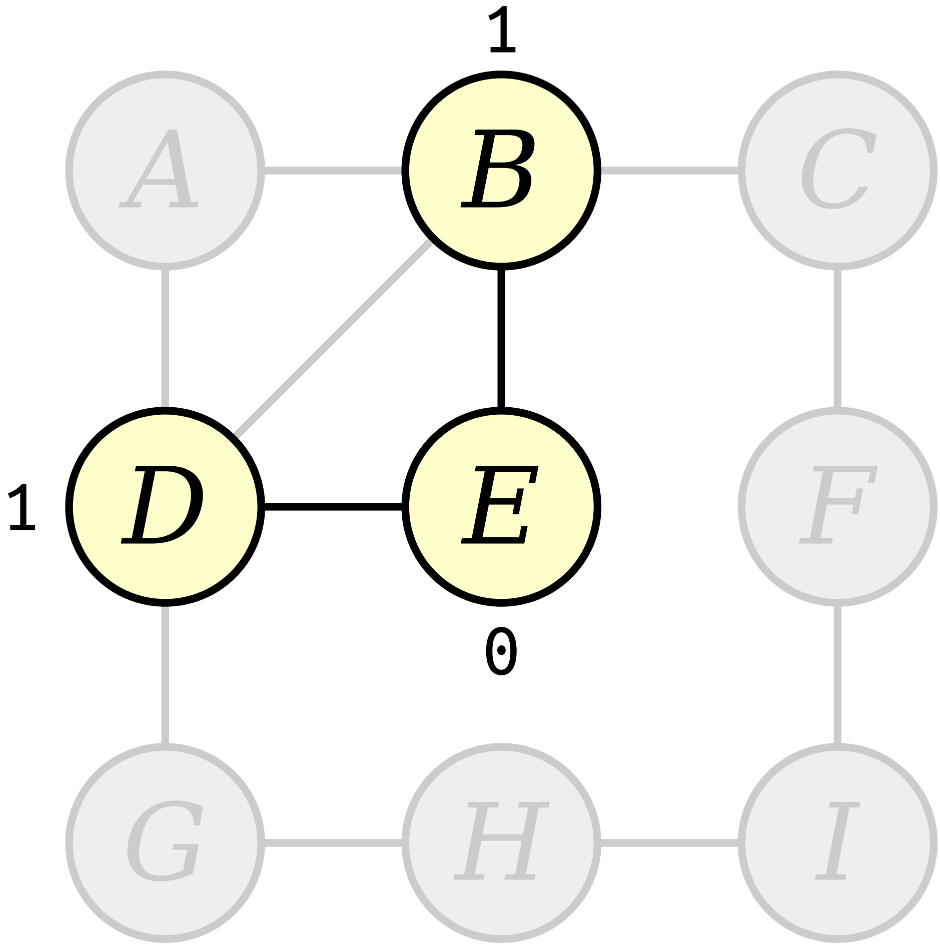
Load newly-discovered nodes into a queue.



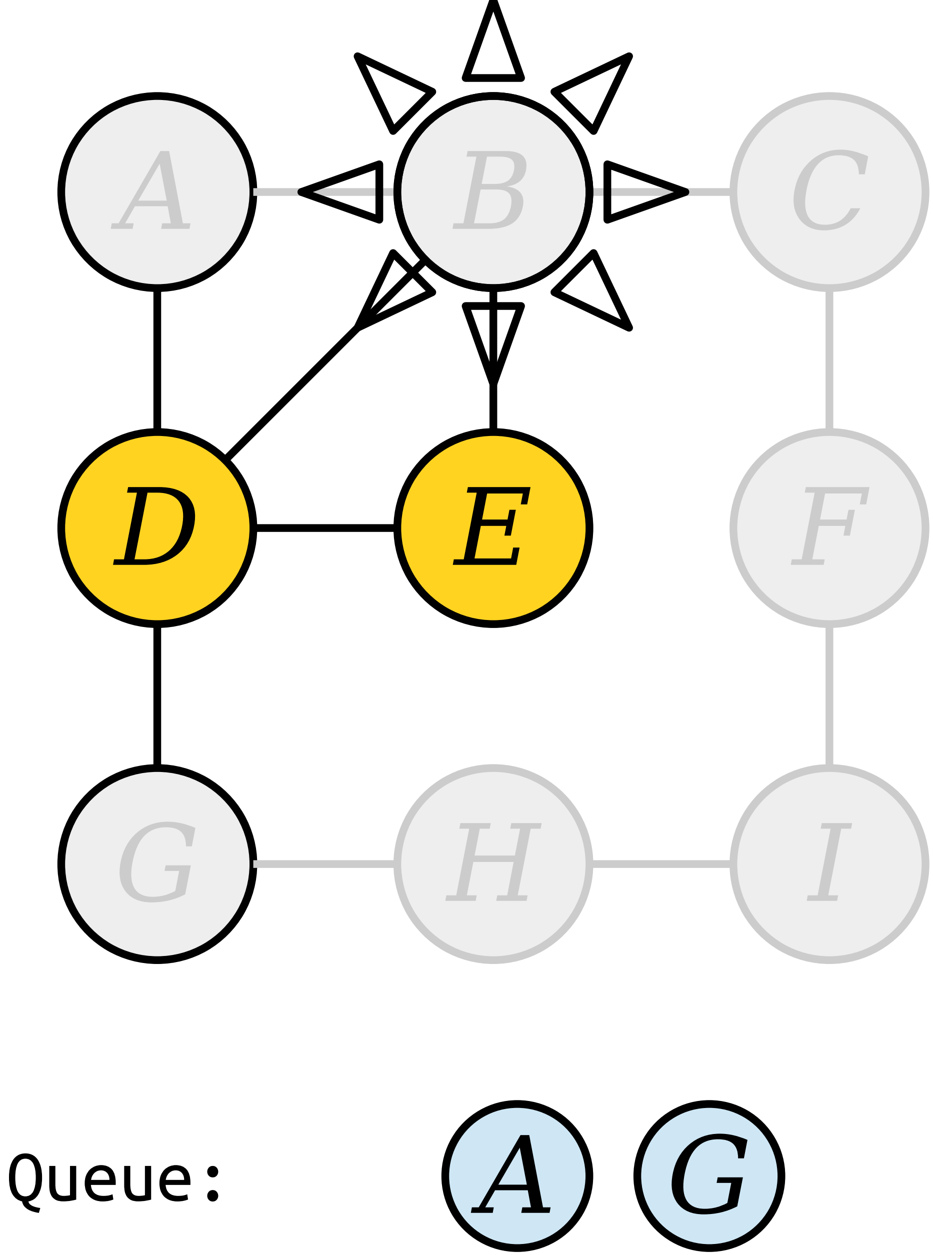
Visit nodes in ascending order of distance from the start node *E*.



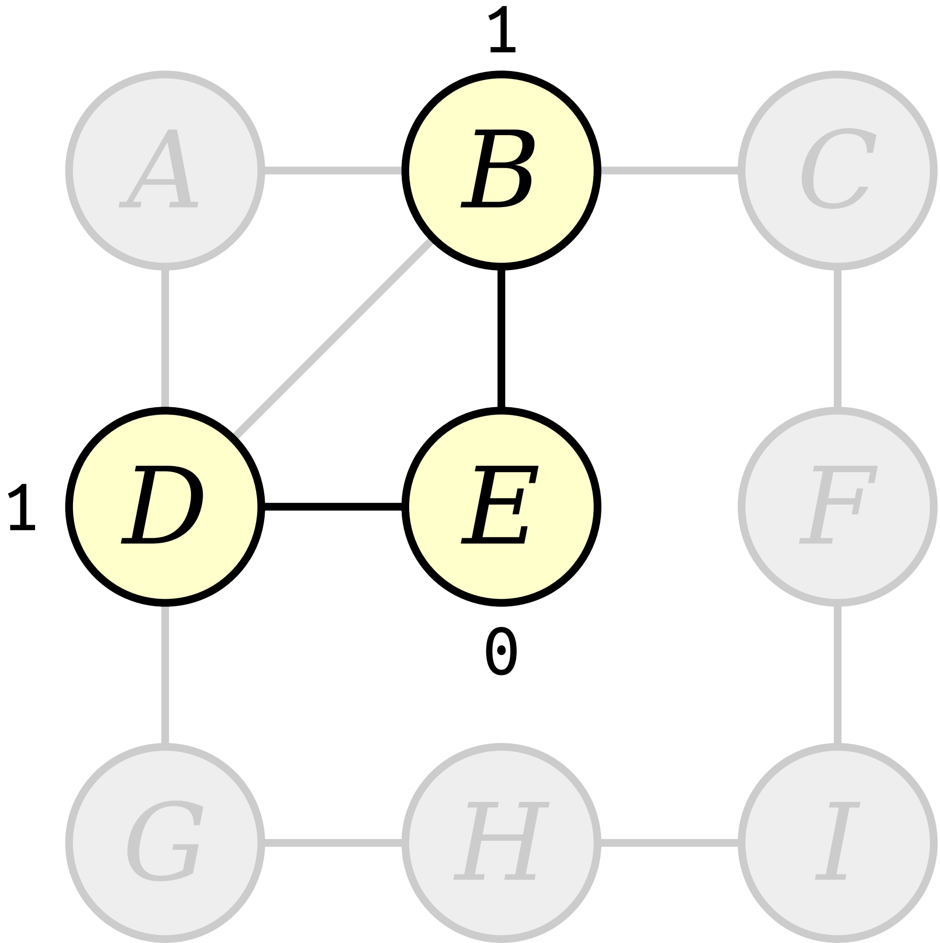
Load newly-discovered nodes into a queue.



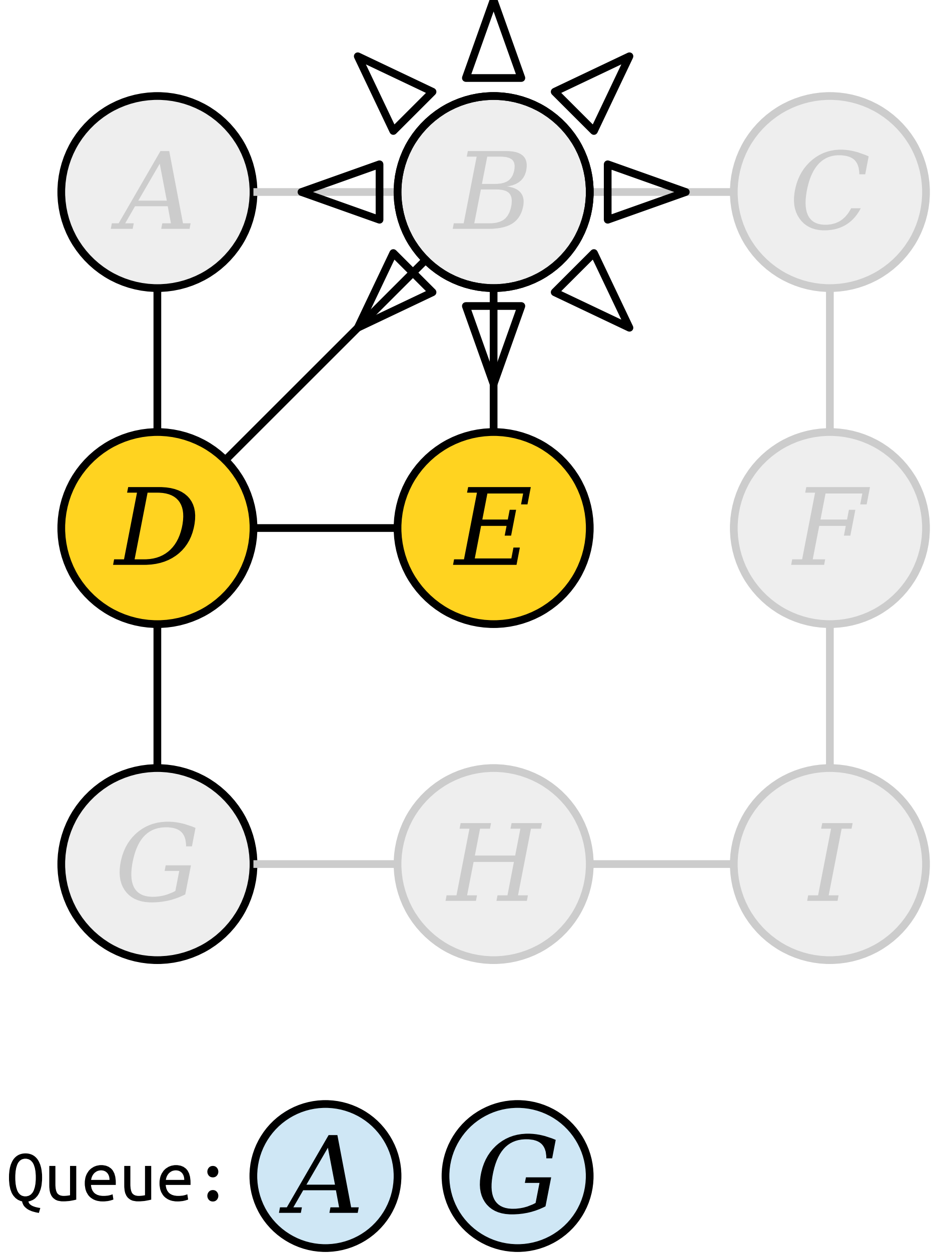
Visit nodes in ascending order of distance from the start node *E*.



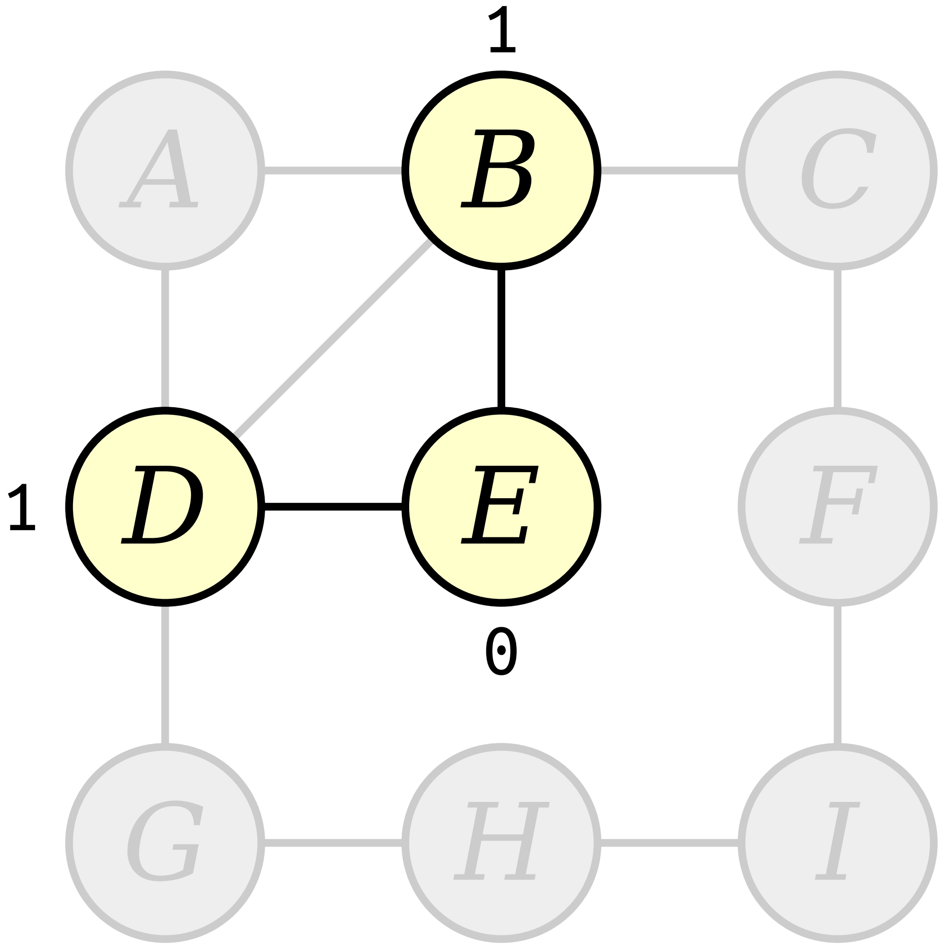
Load newly-discovered nodes into a queue.



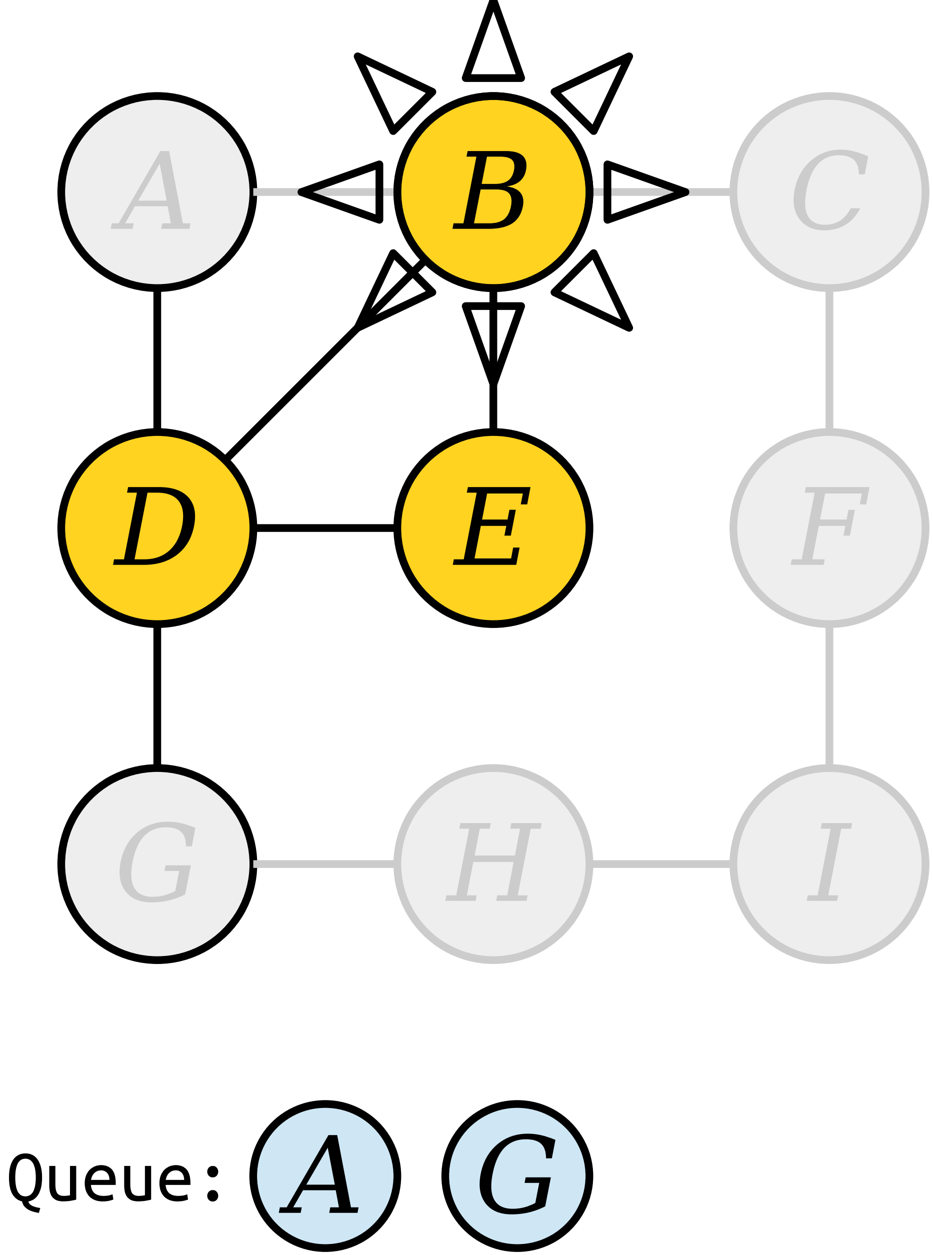
Visit nodes in ascending order of distance from the start node *E*.



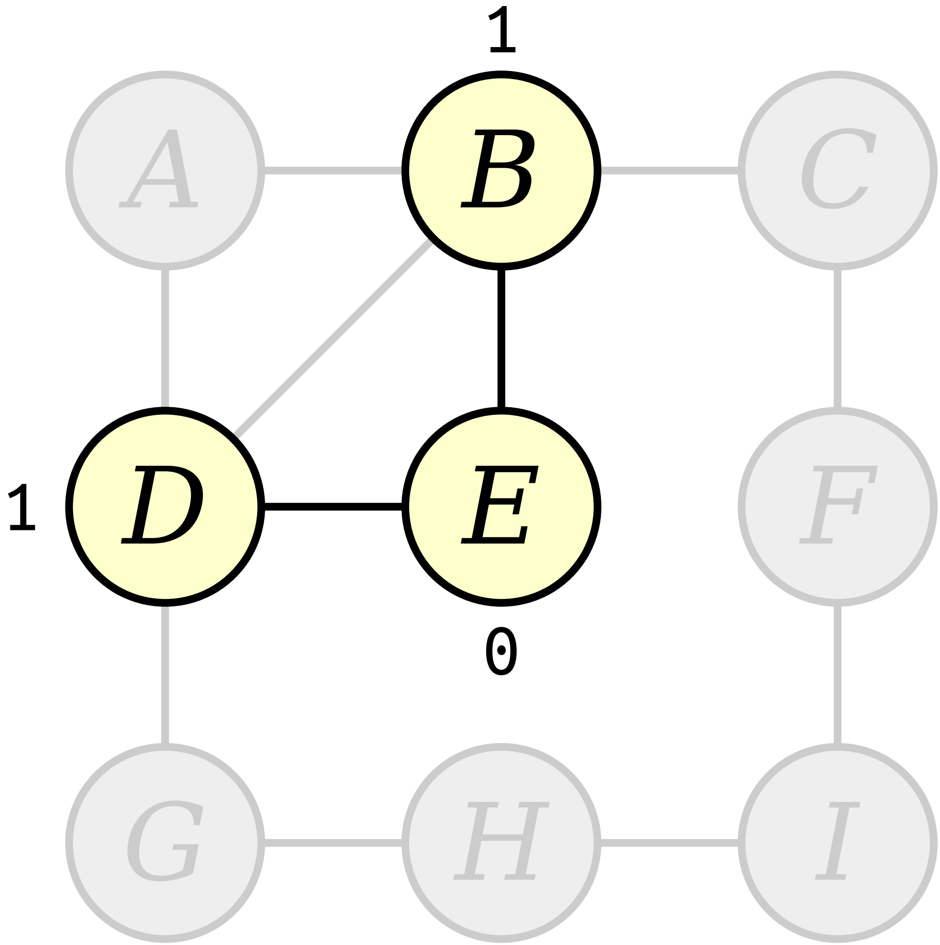
Load newly-discovered nodes into a queue.



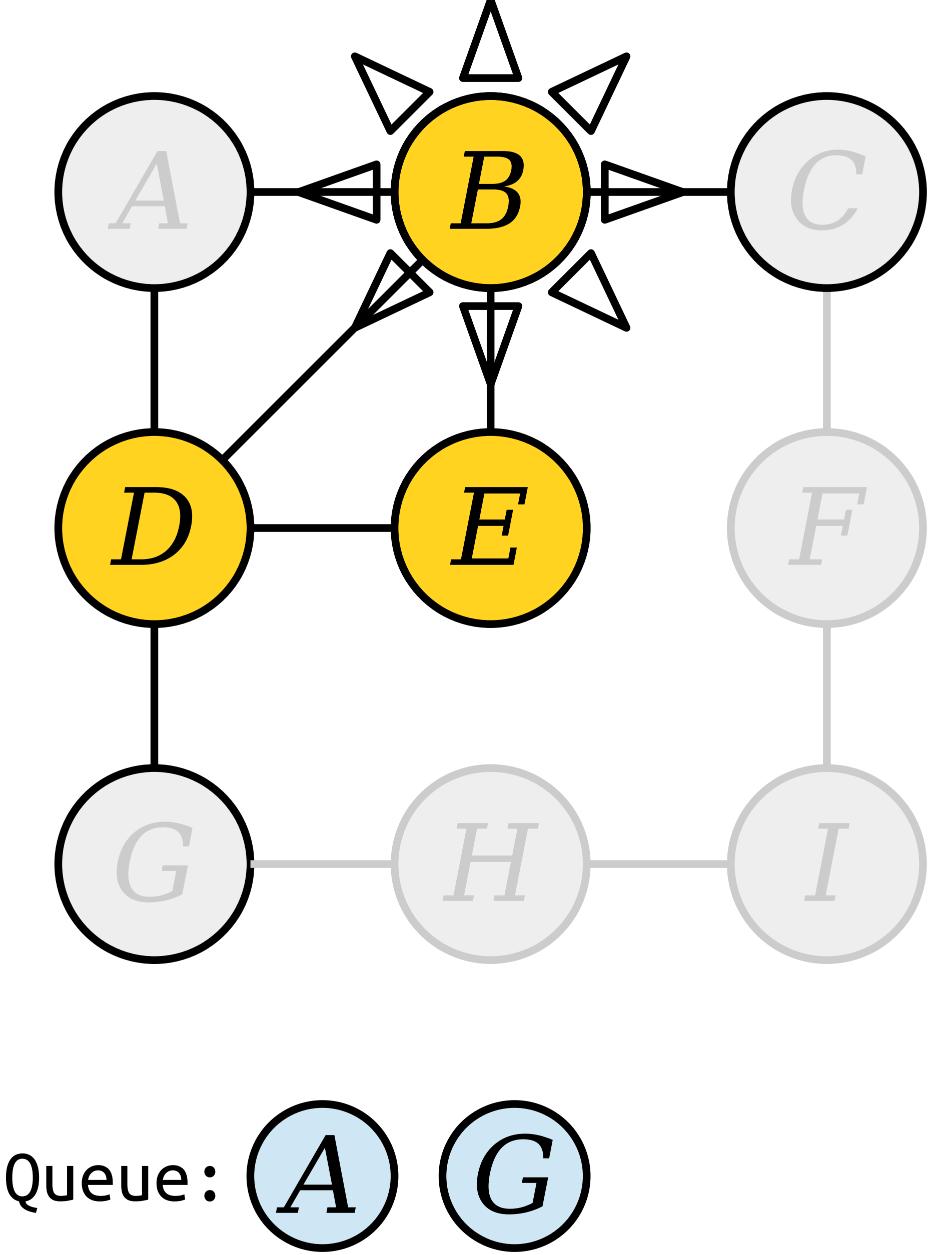
Visit nodes in ascending order of distance from the start node *E*.



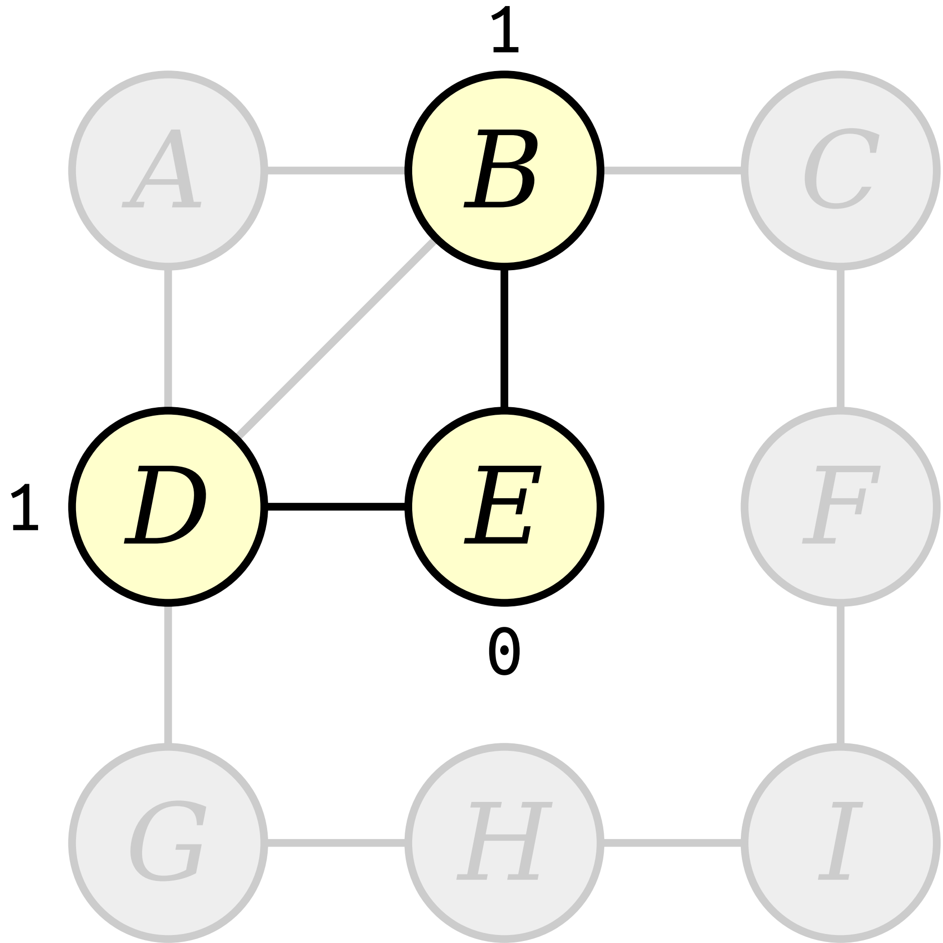
Load newly-discovered nodes into a queue.



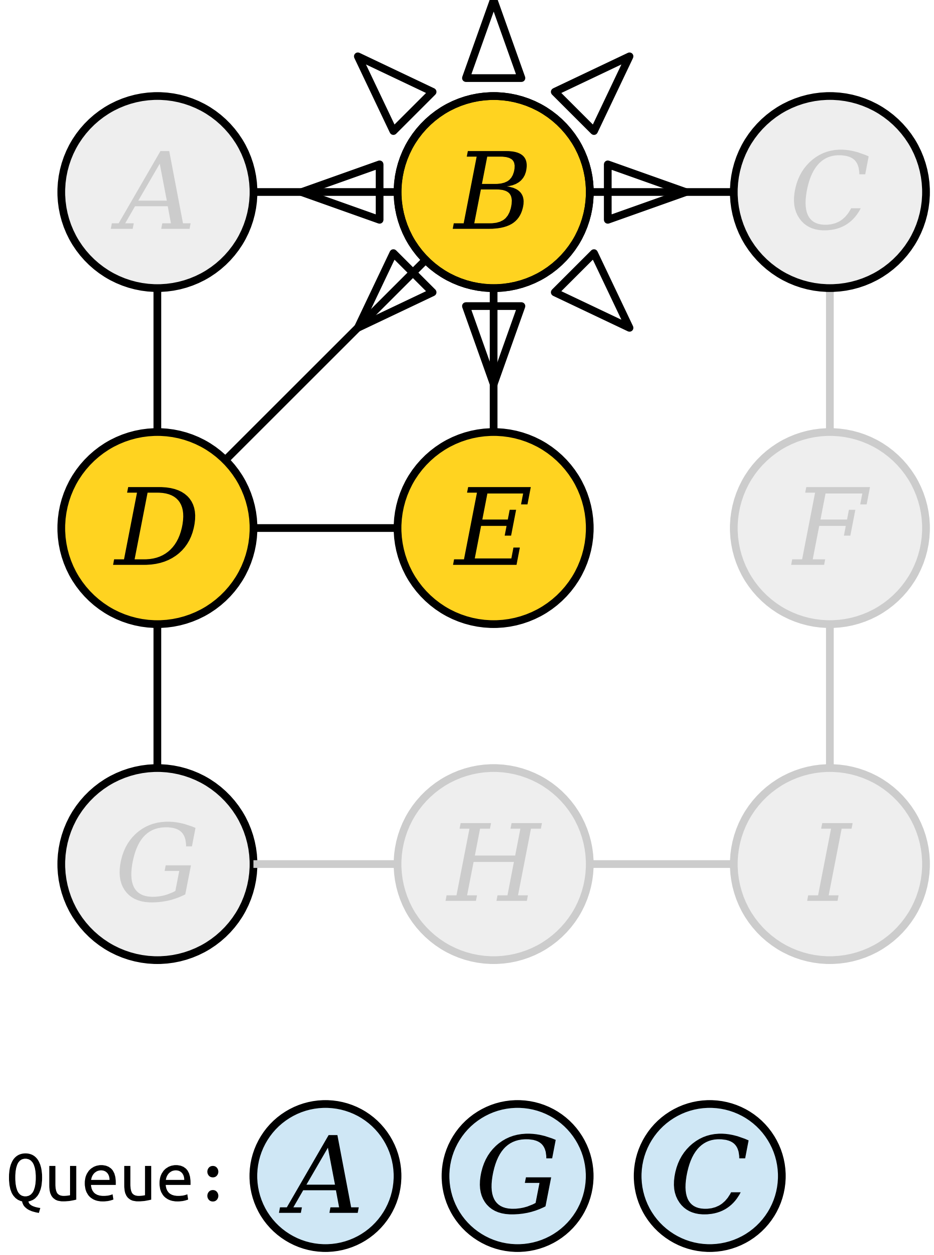
Visit nodes in ascending order of distance from the start node *E*.



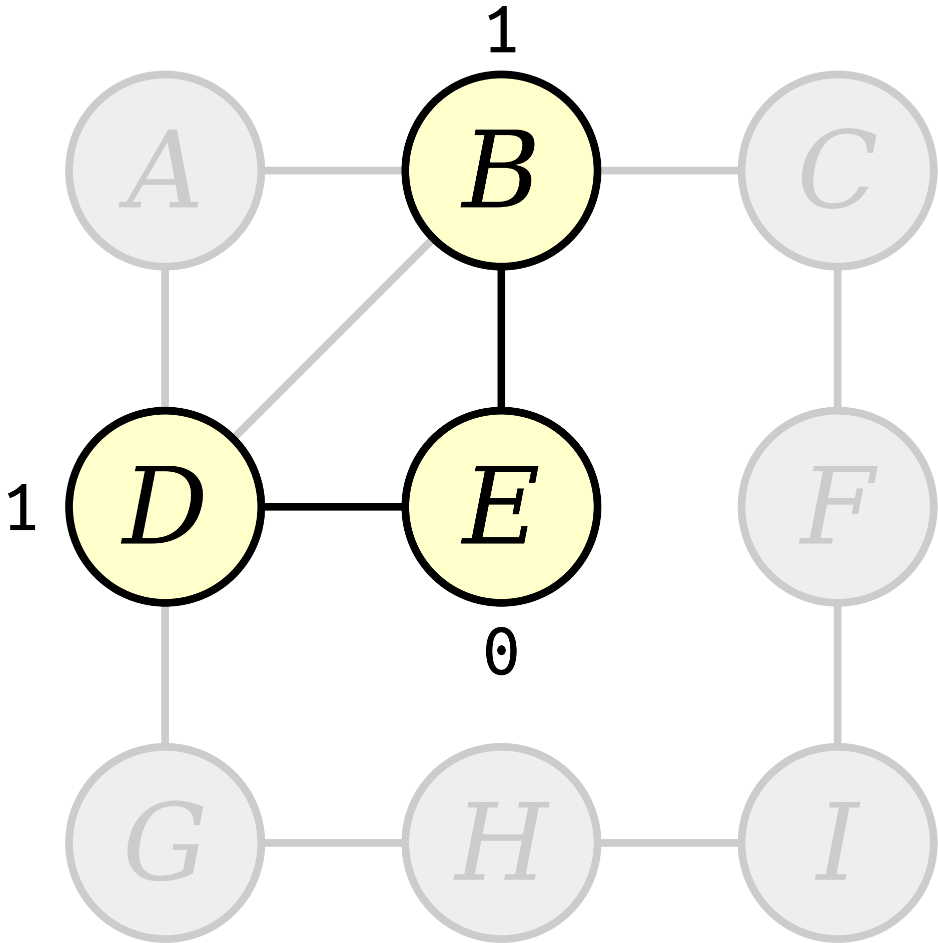
Load newly-discovered nodes into a queue.



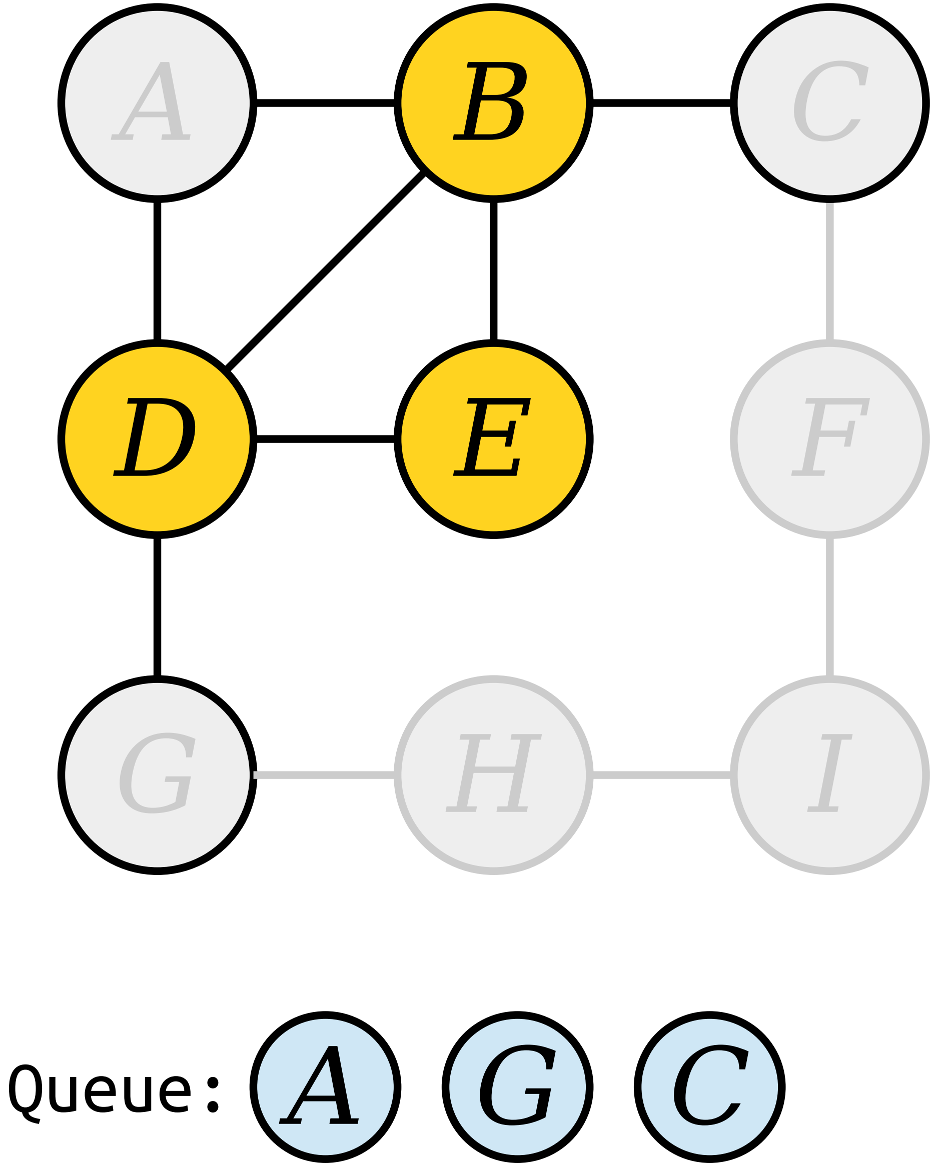
Visit nodes in ascending order of distance from the start node *E*.



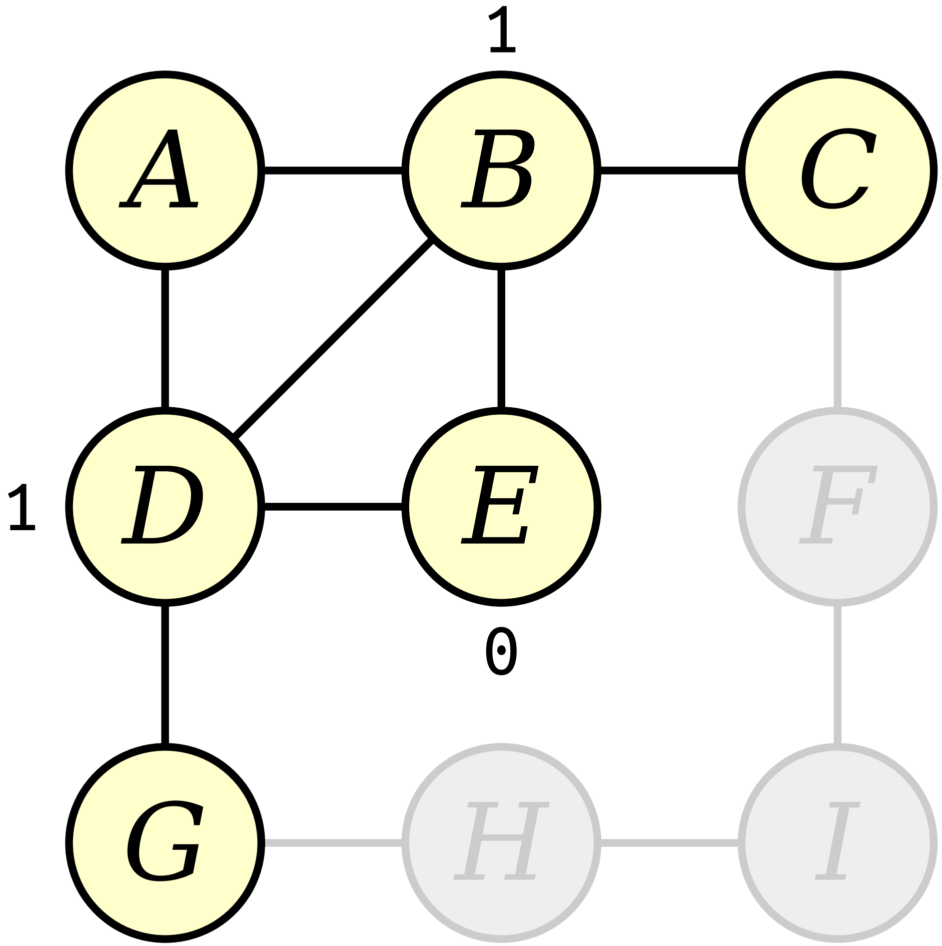
Load newly-discovered nodes into a queue.



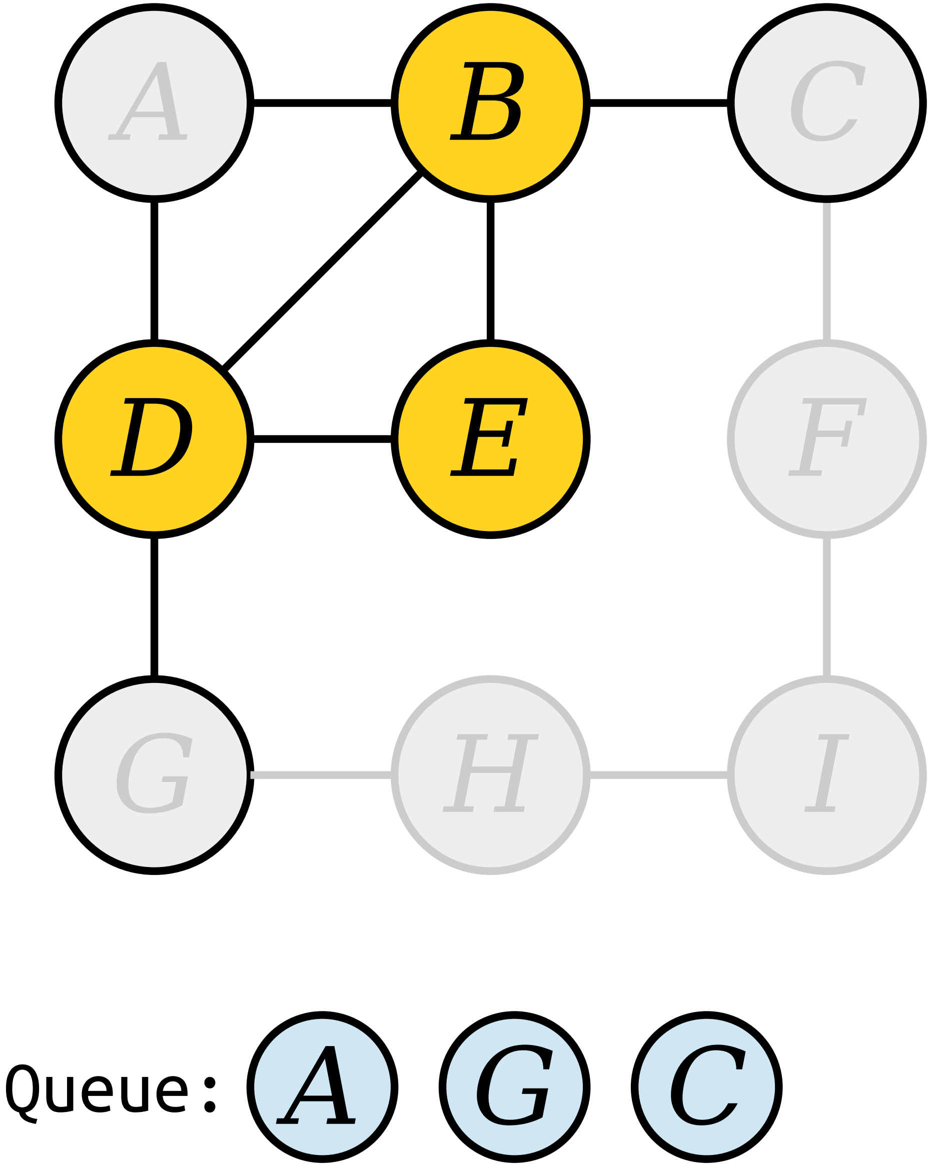
Visit nodes in ascending order of distance from the start node *E*.



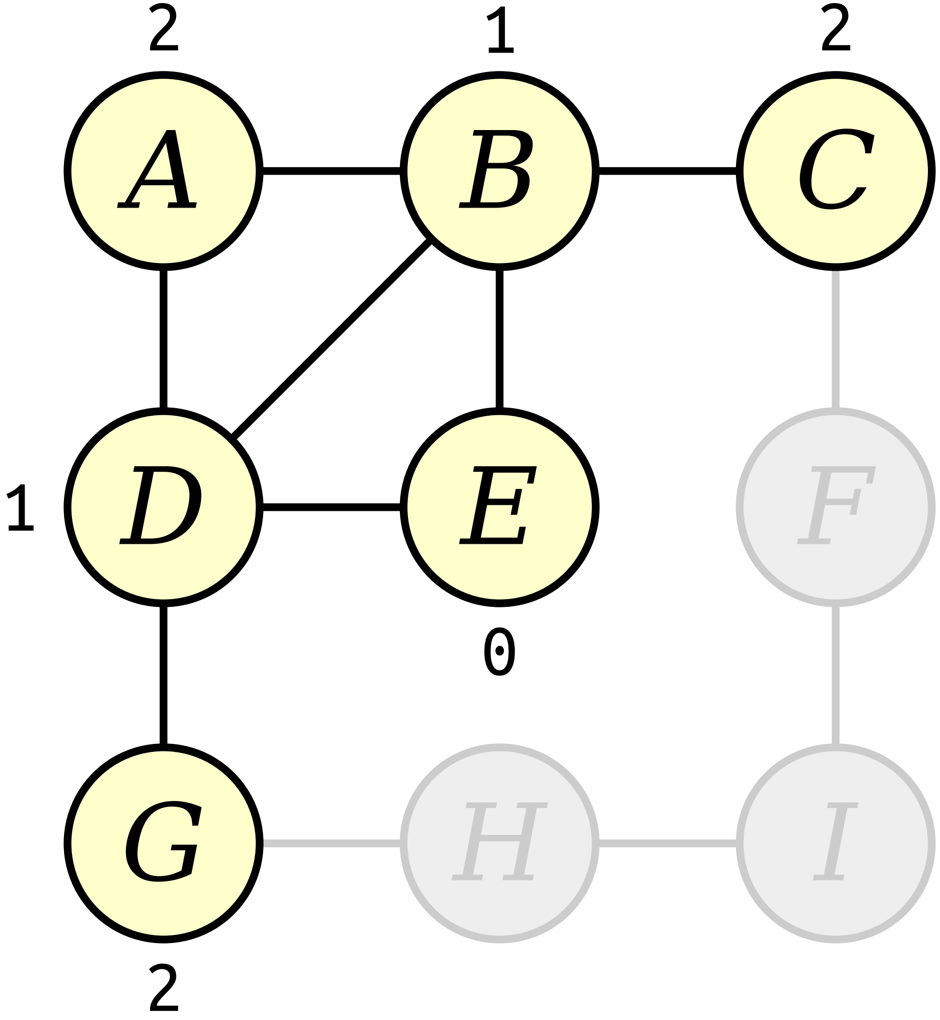
Load newly-discovered nodes into a queue.



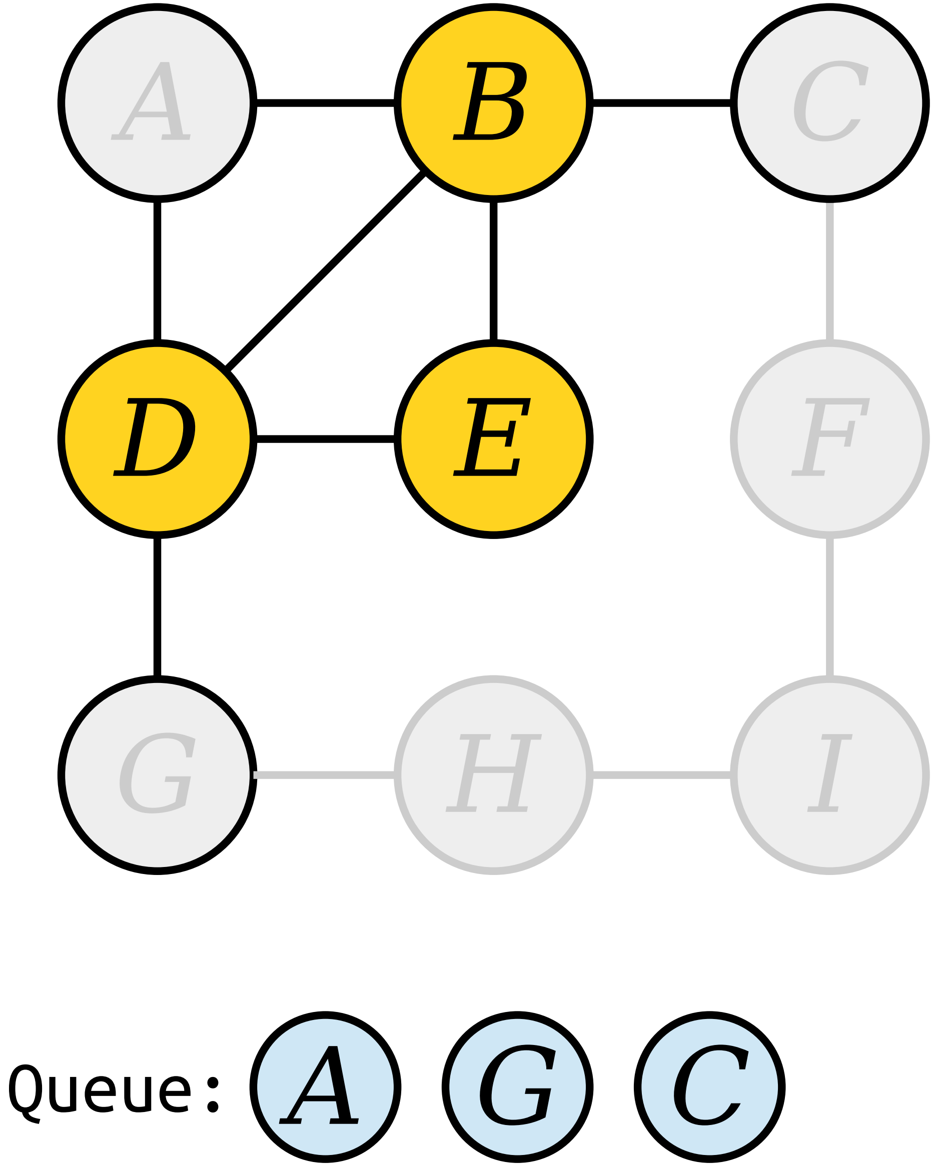
Visit nodes in ascending order of distance from the start node *E*.



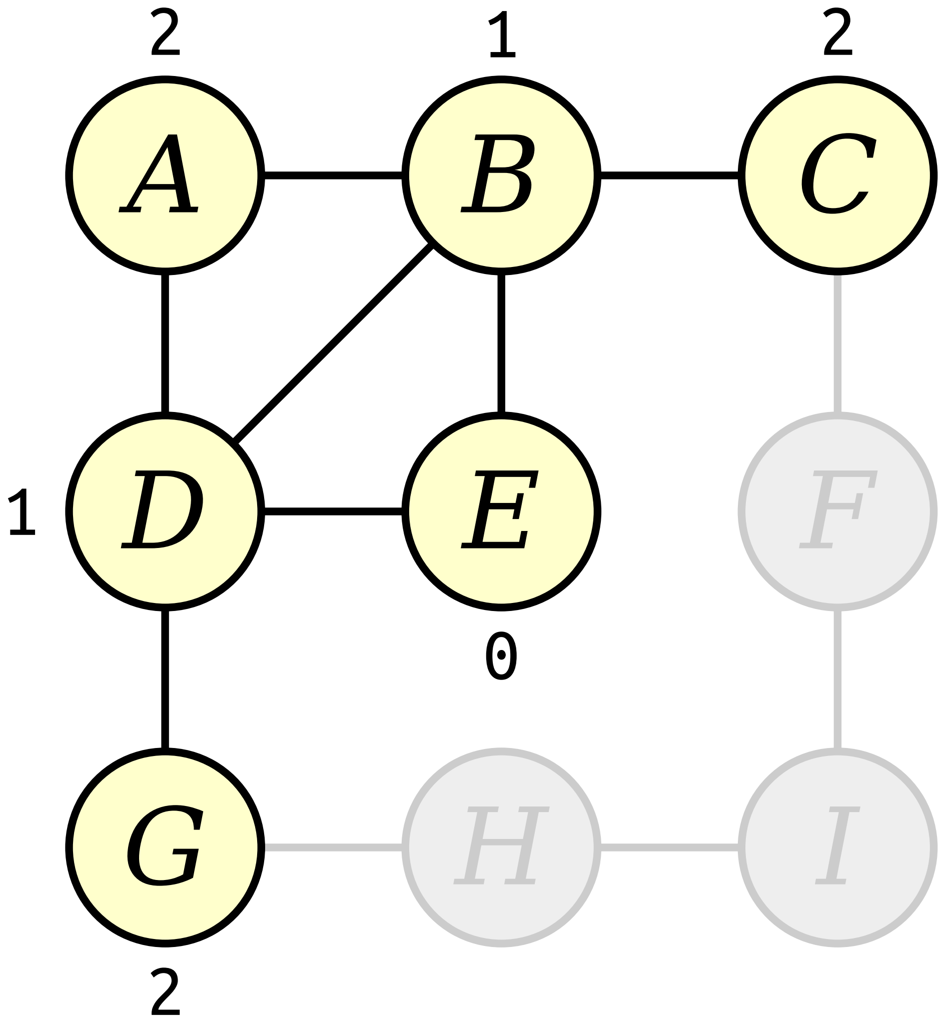
Load newly-discovered nodes into a queue.



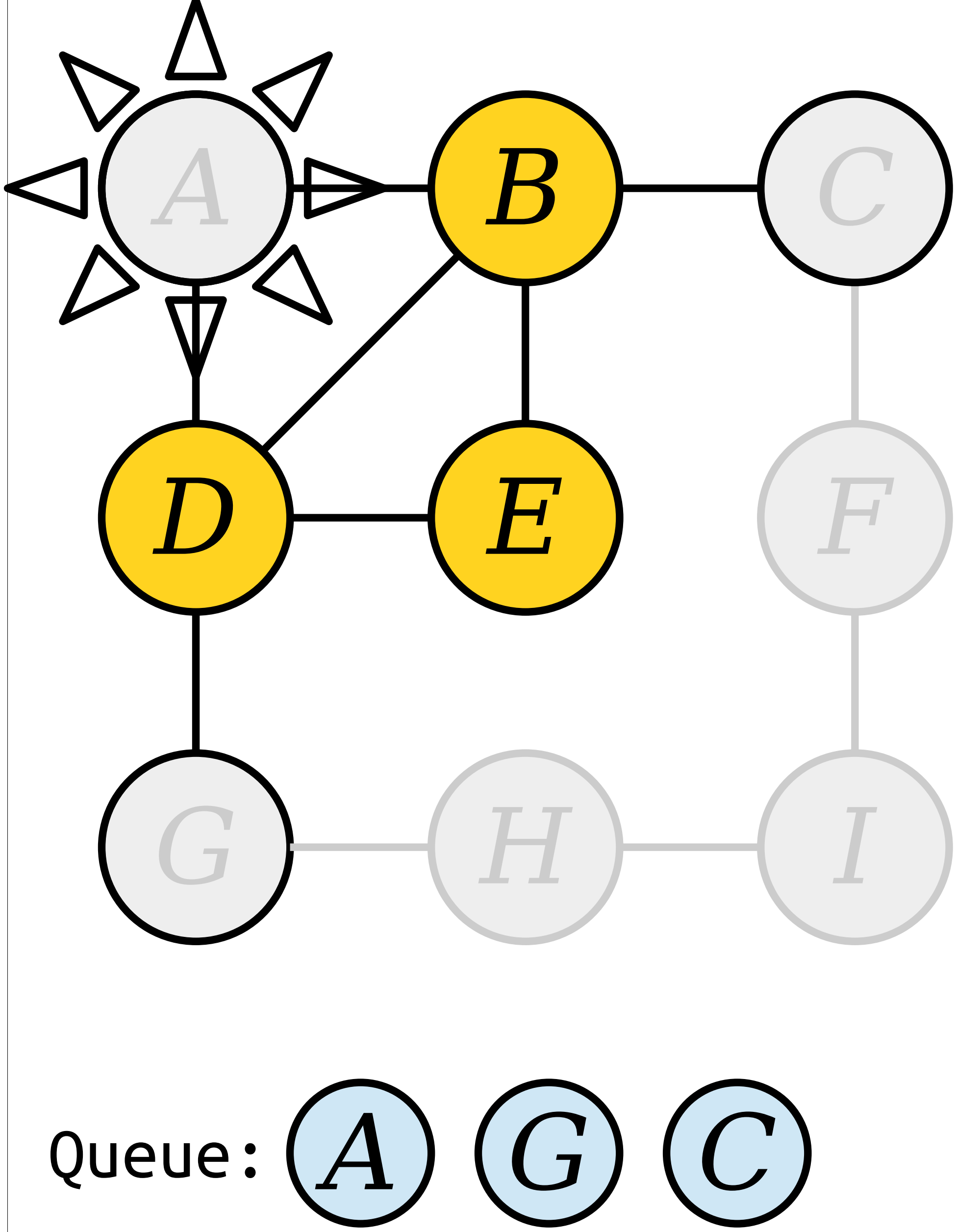
Visit nodes in ascending order of distance from the start node *E*.



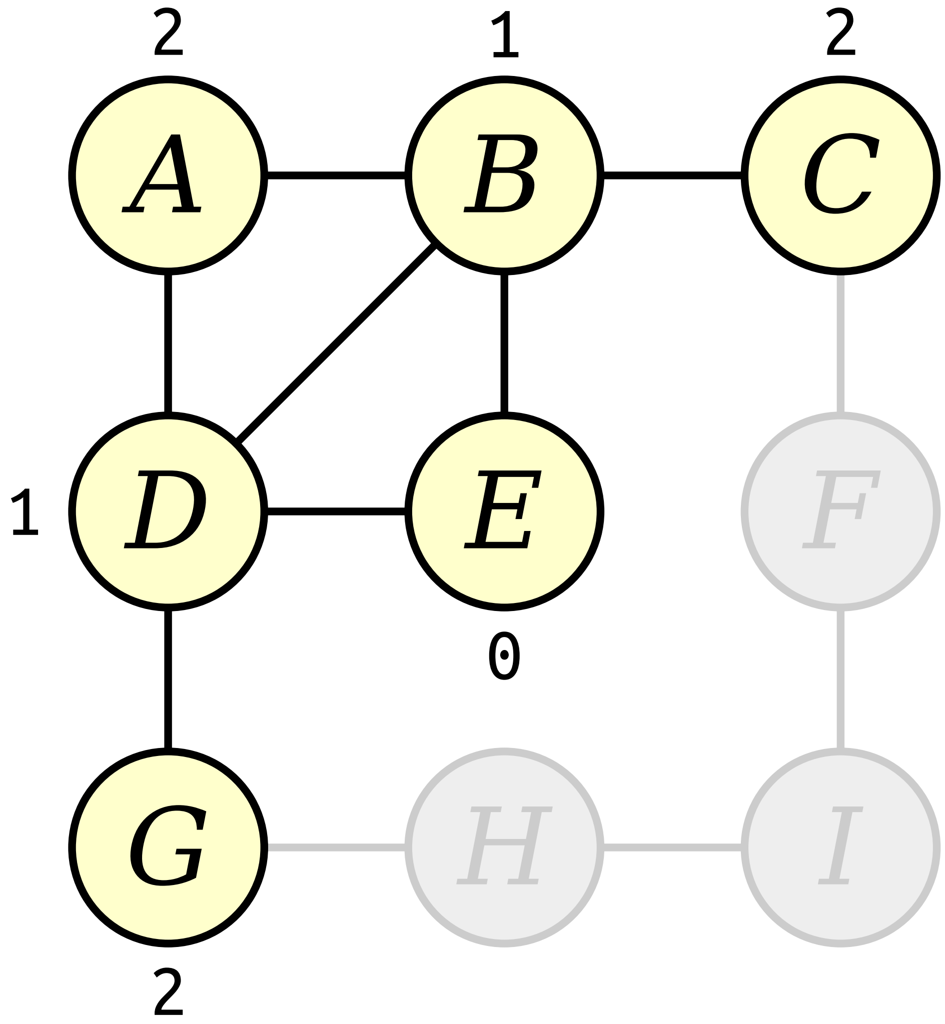
Load newly-discovered nodes into a queue.



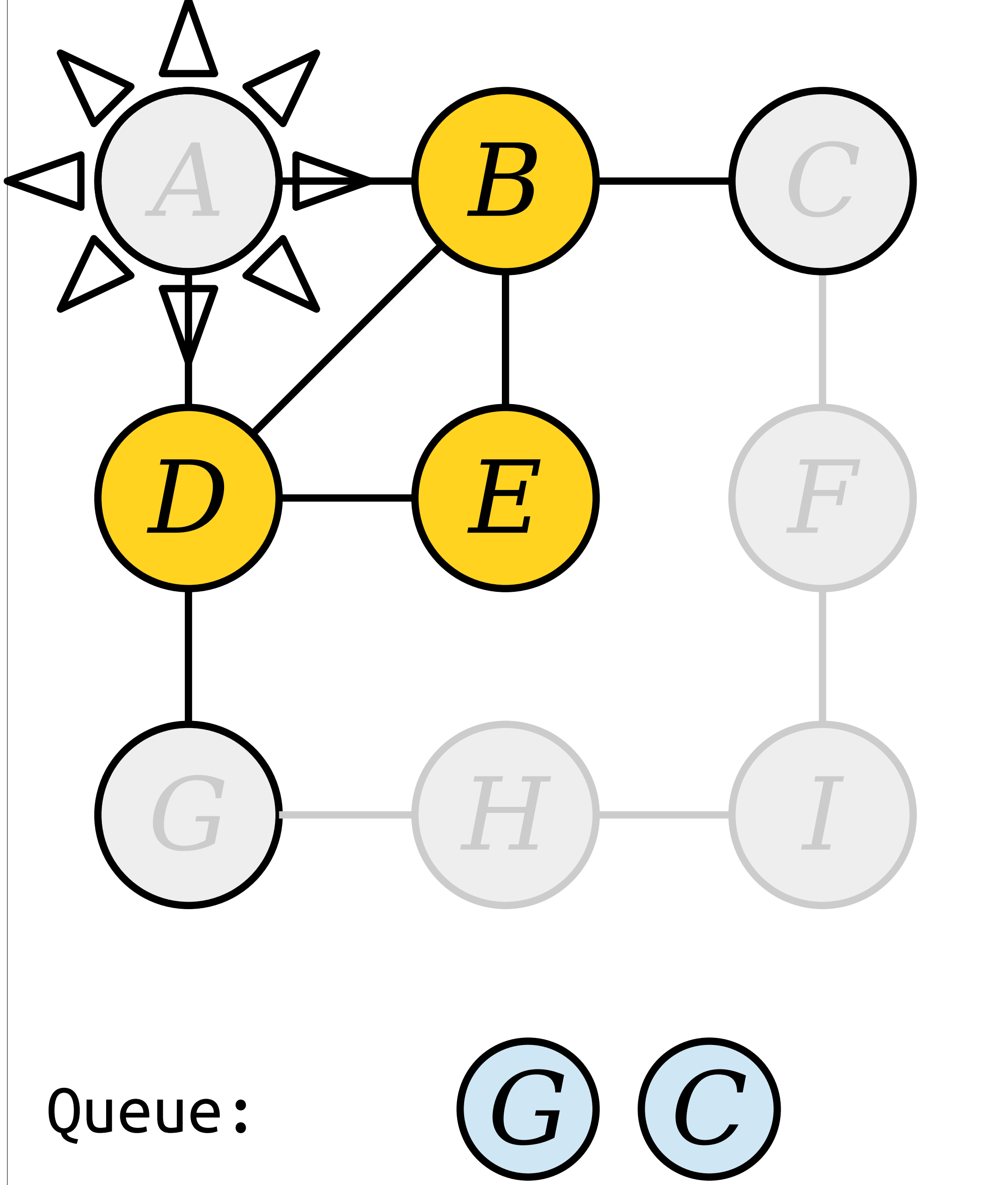
Visit nodes in ascending order of distance from the start node *E*.



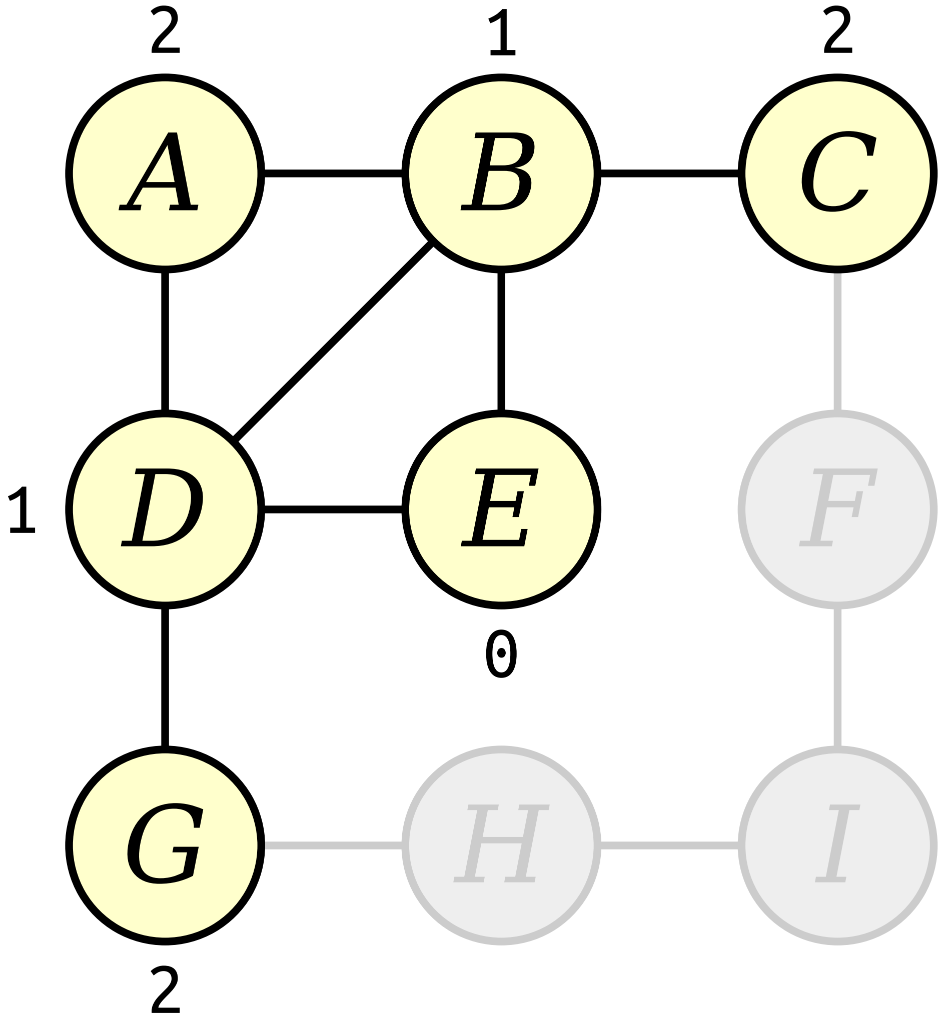
Load newly-discovered nodes into a queue.



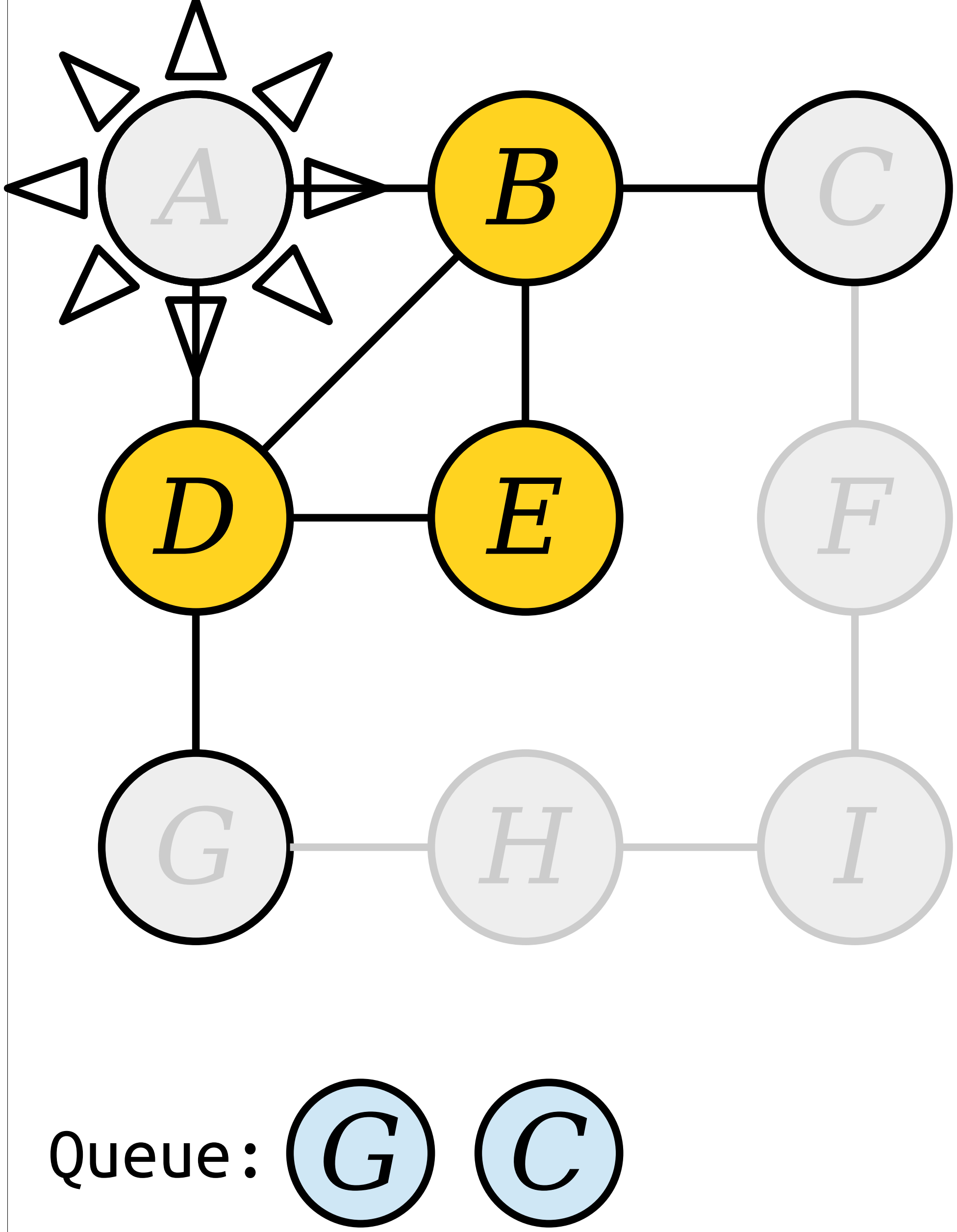
Visit nodes in ascending order of distance from the start node E.



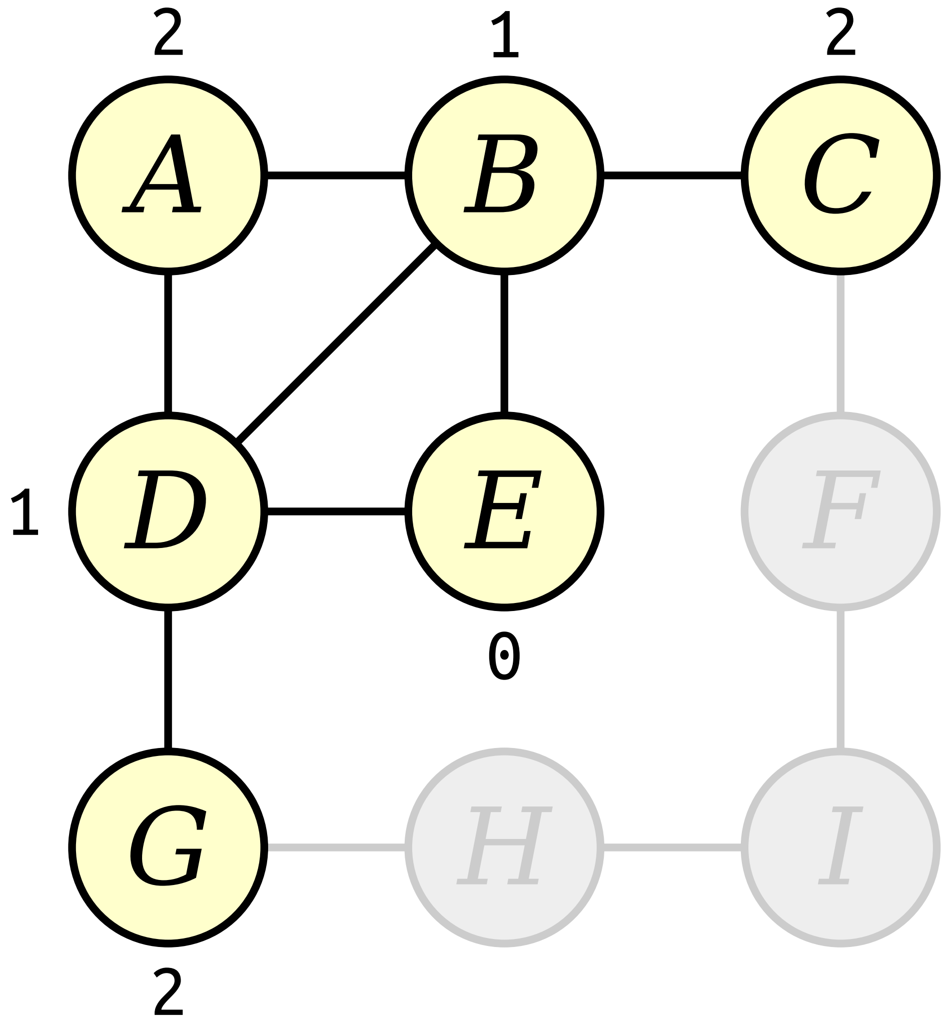
Load newly-discovered nodes into a queue.



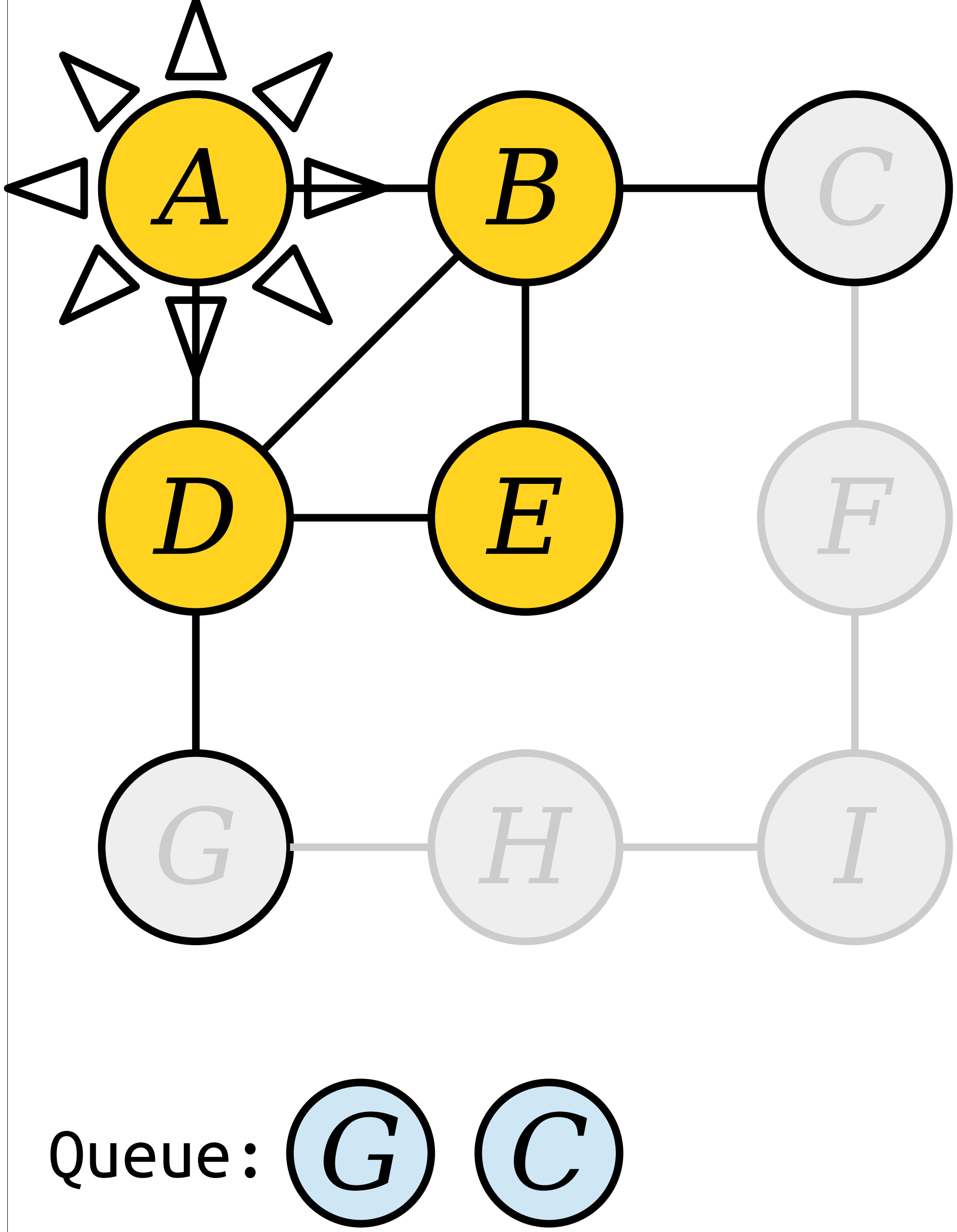
Visit nodes in ascending order of distance from the start node E.



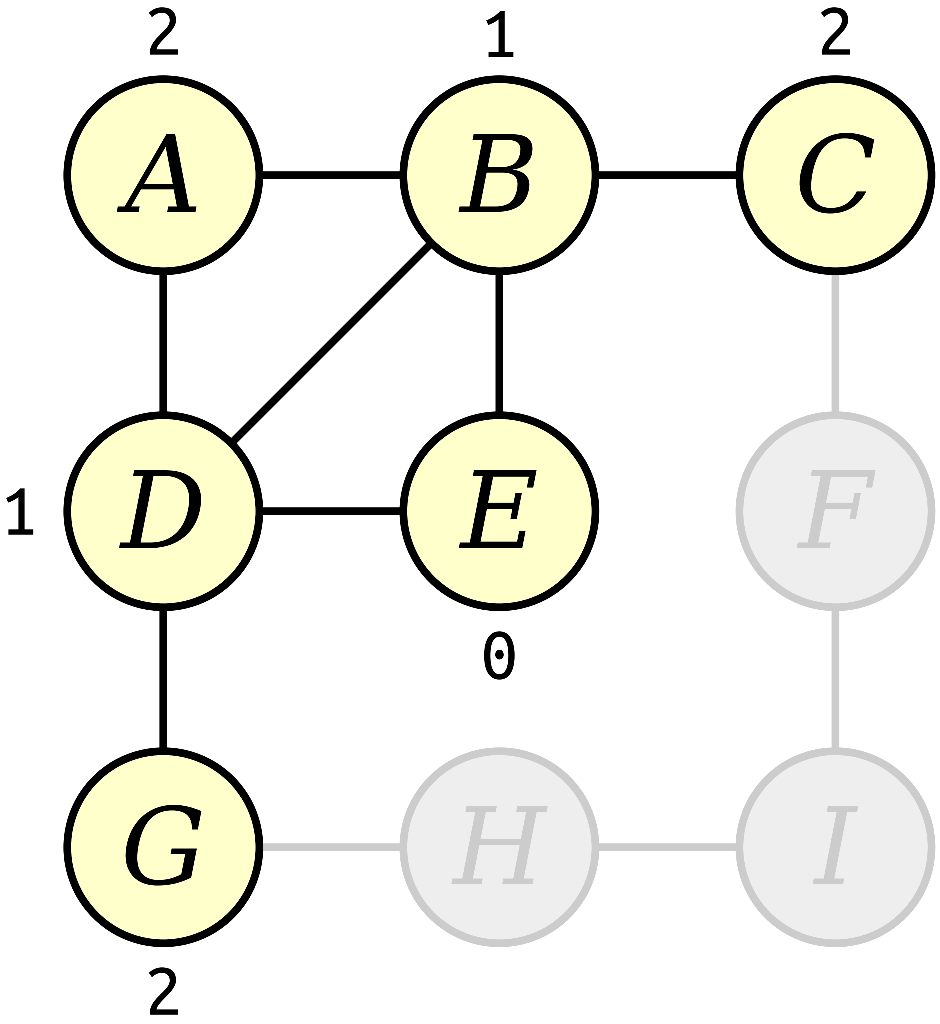
Load newly-discovered nodes into a queue.



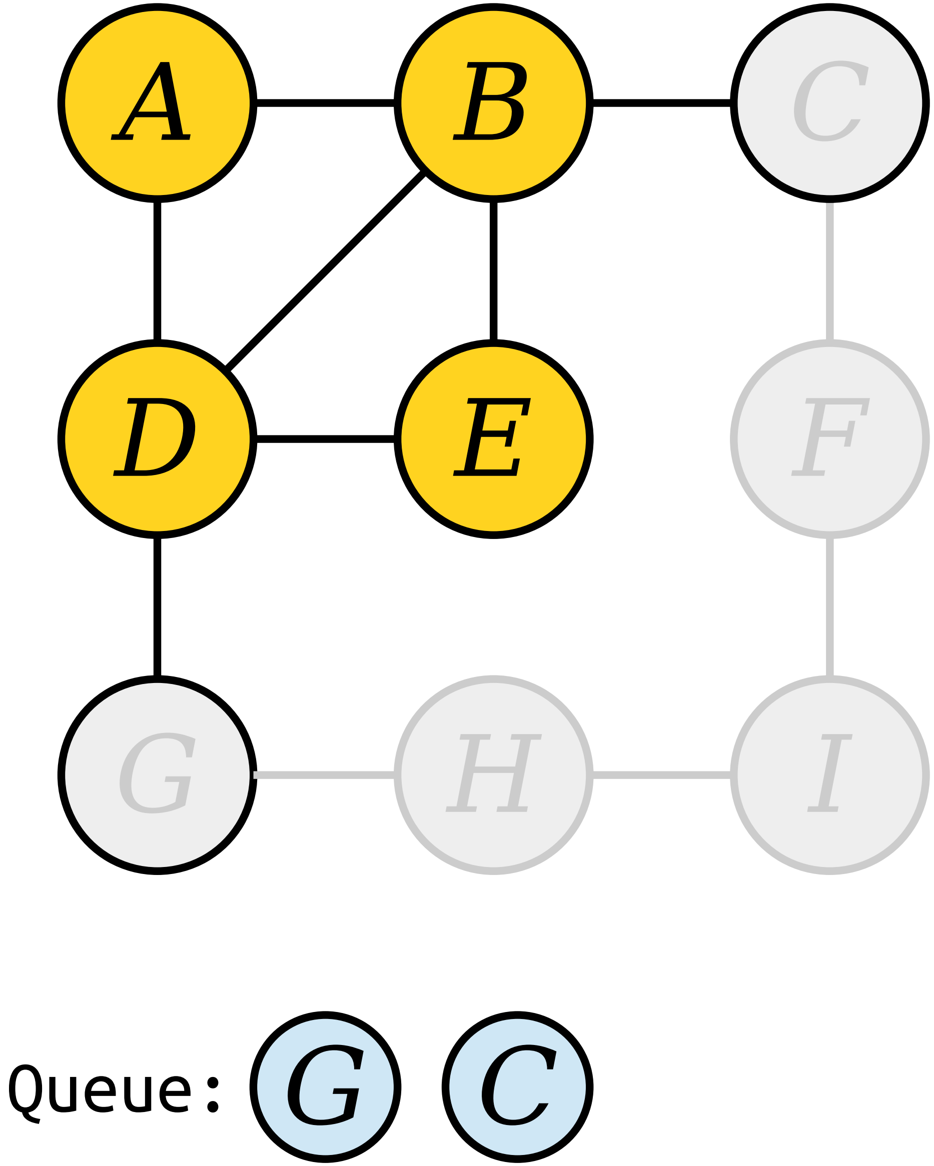
Visit nodes in ascending order of distance from the start node *E*.



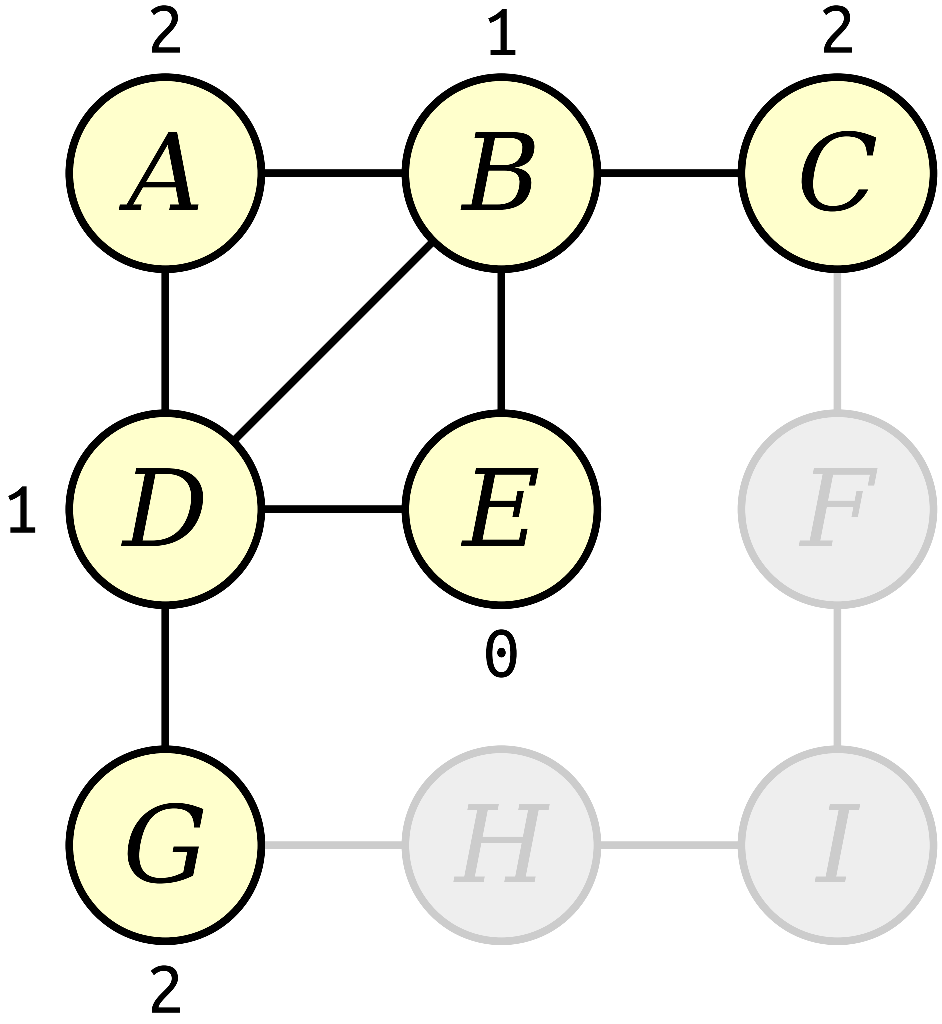
Load newly-discovered nodes into a queue.



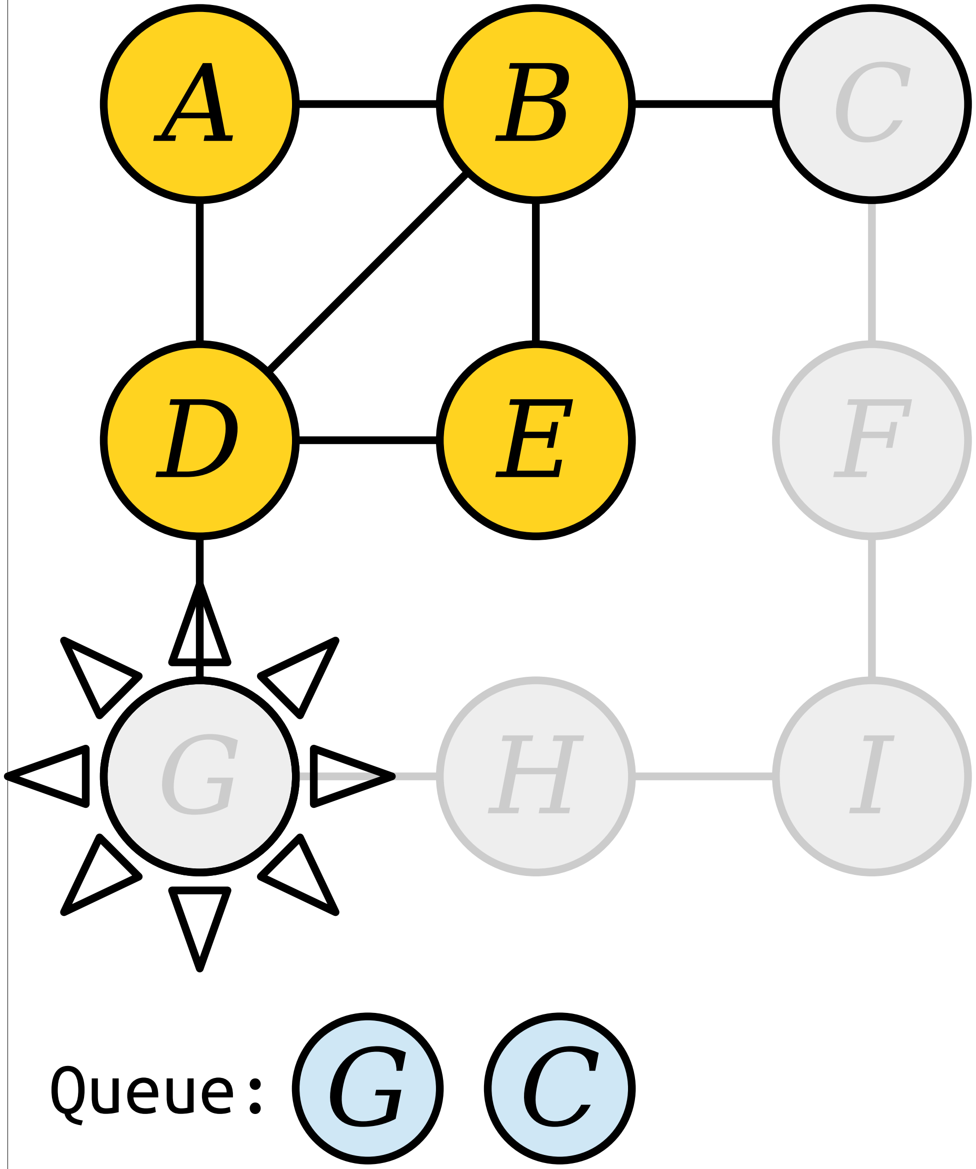
Visit nodes in ascending order of distance from the start node *E*.



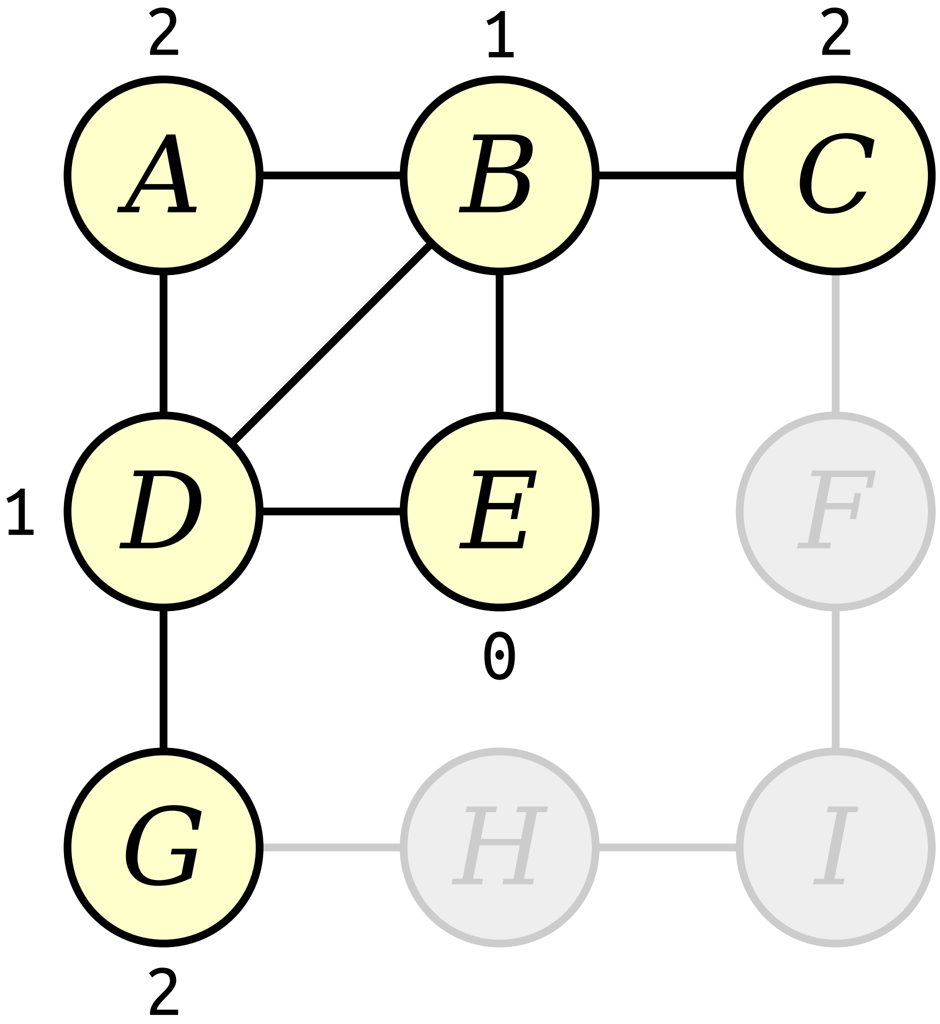
Load newly-discovered nodes into a queue.



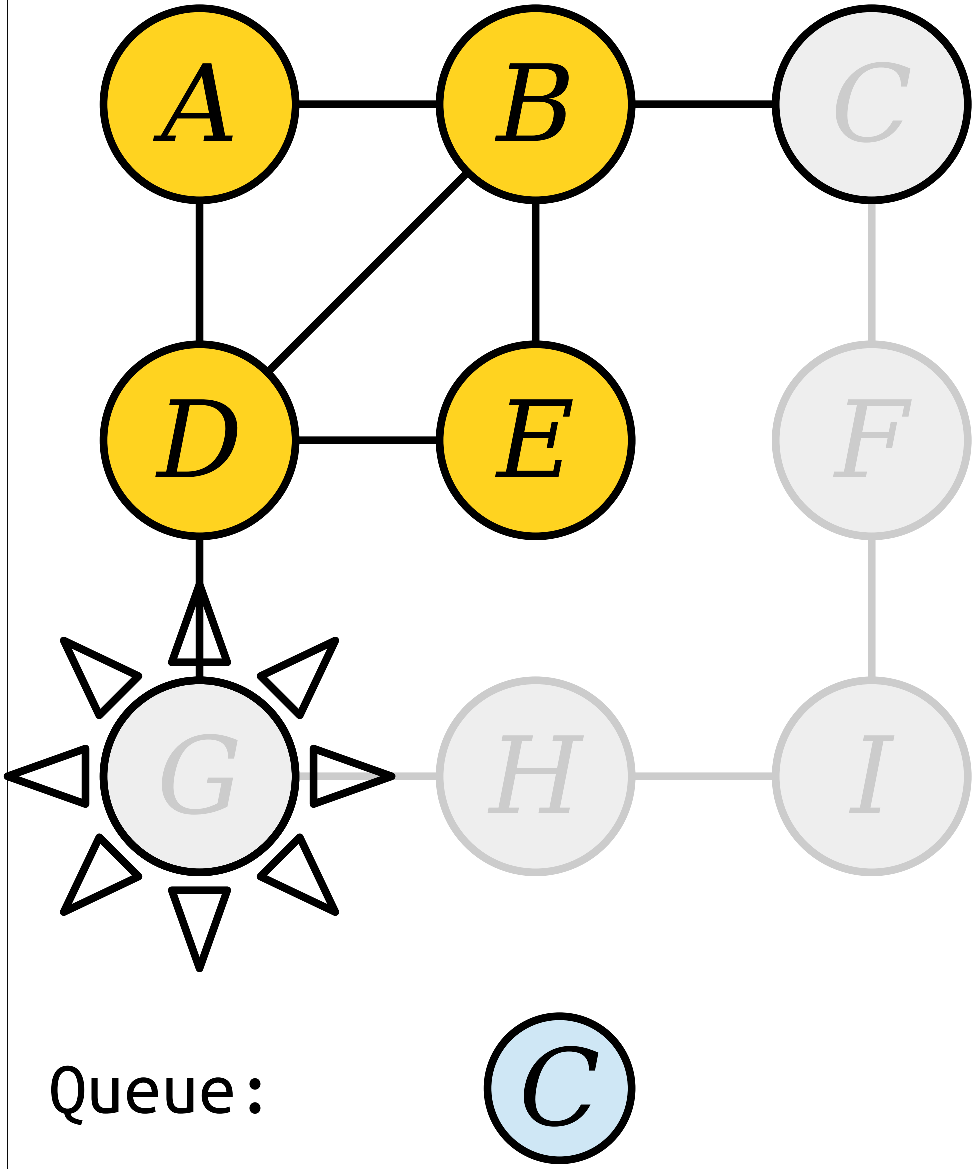
Visit nodes in ascending order of distance from the start node *E*.



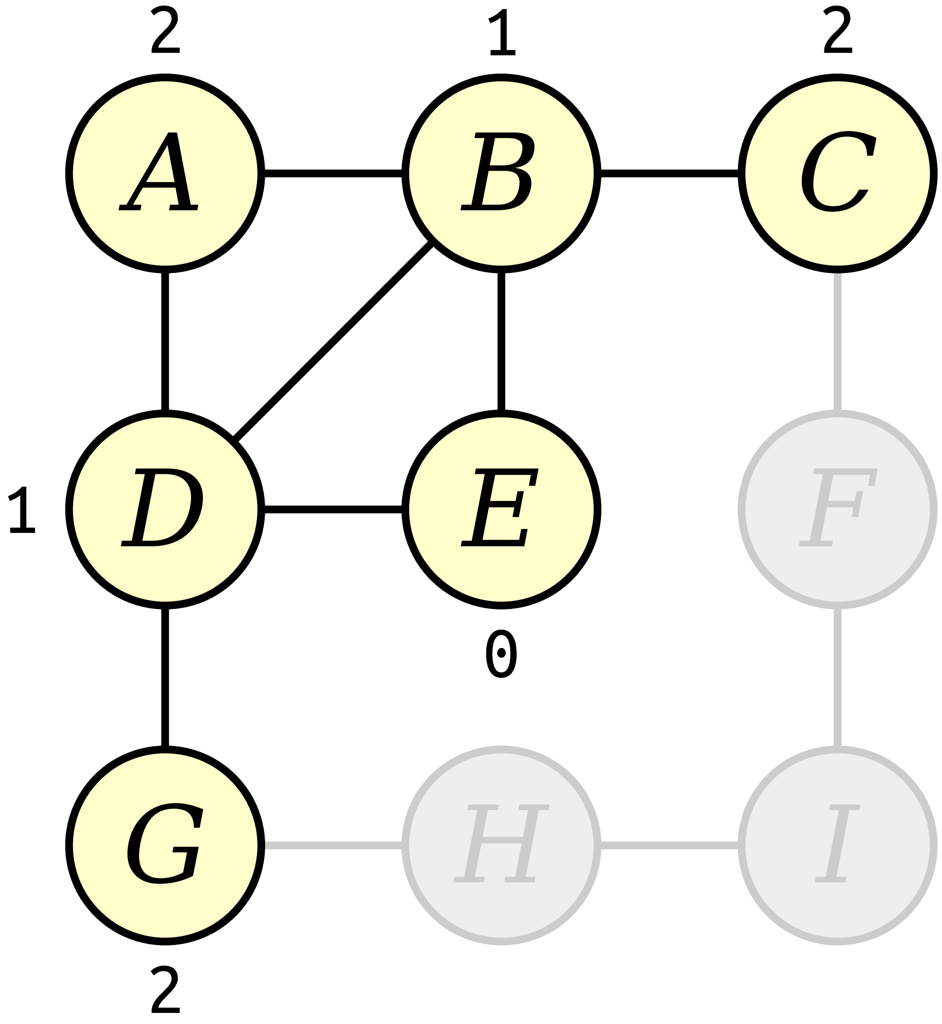
Load newly-discovered nodes into a queue.



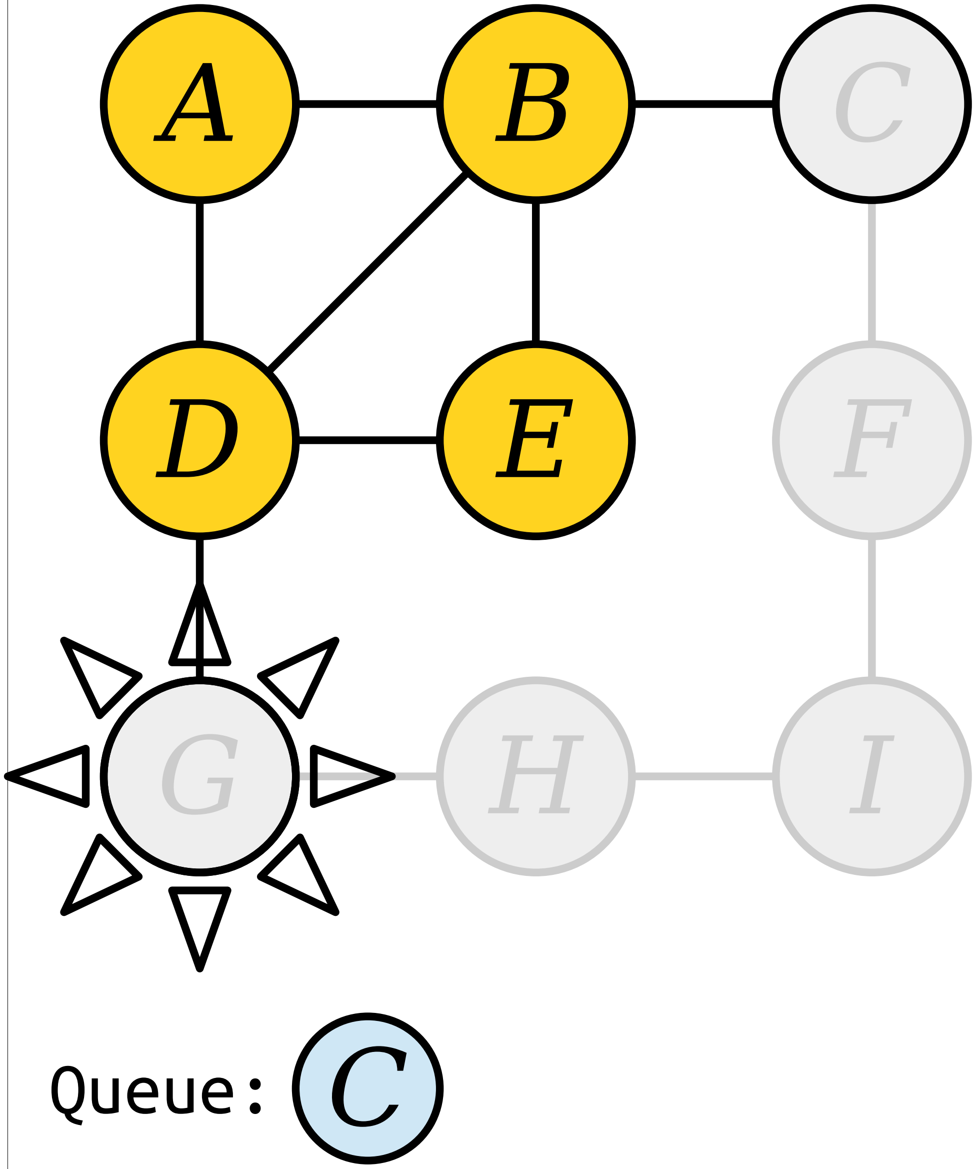
Visit nodes in ascending order of distance from the start node *E*.



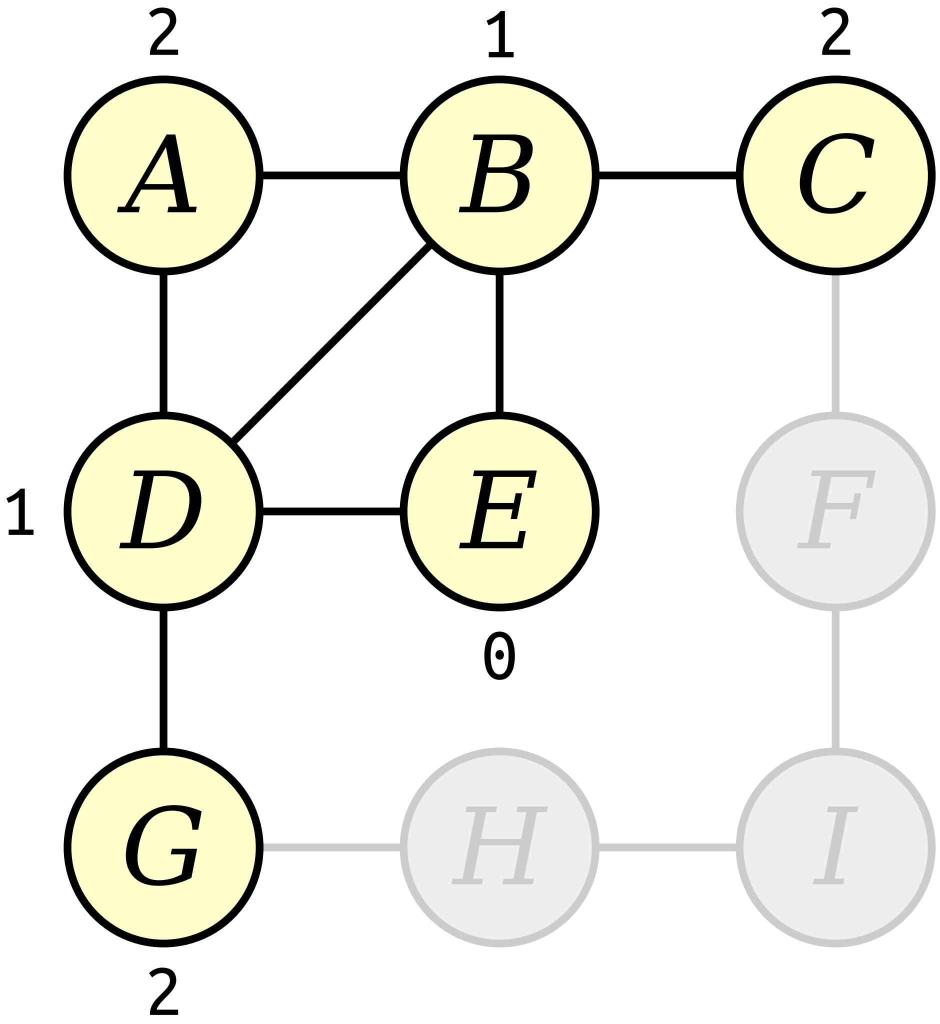
Load newly-discovered nodes into a queue.



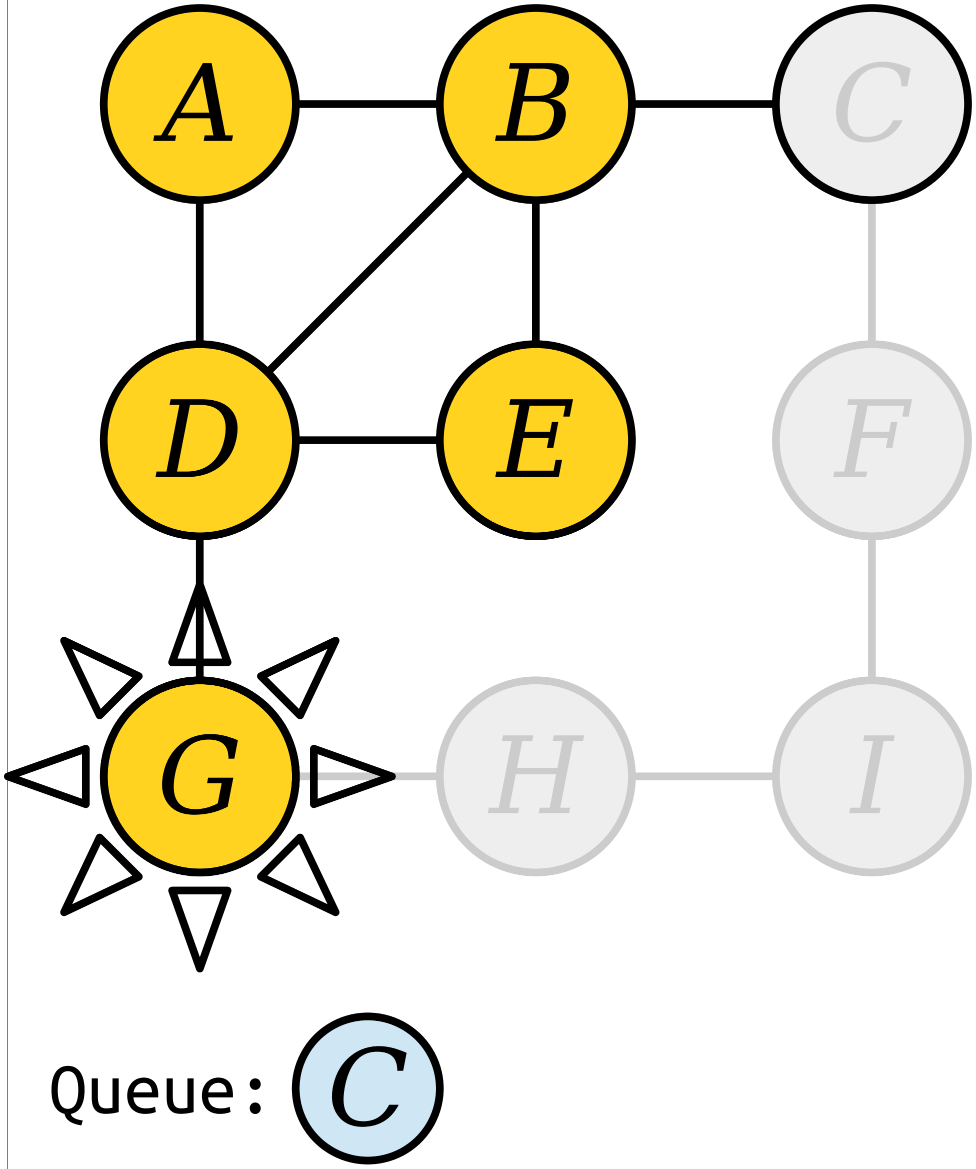
Visit nodes in ascending order of distance from the start node E.



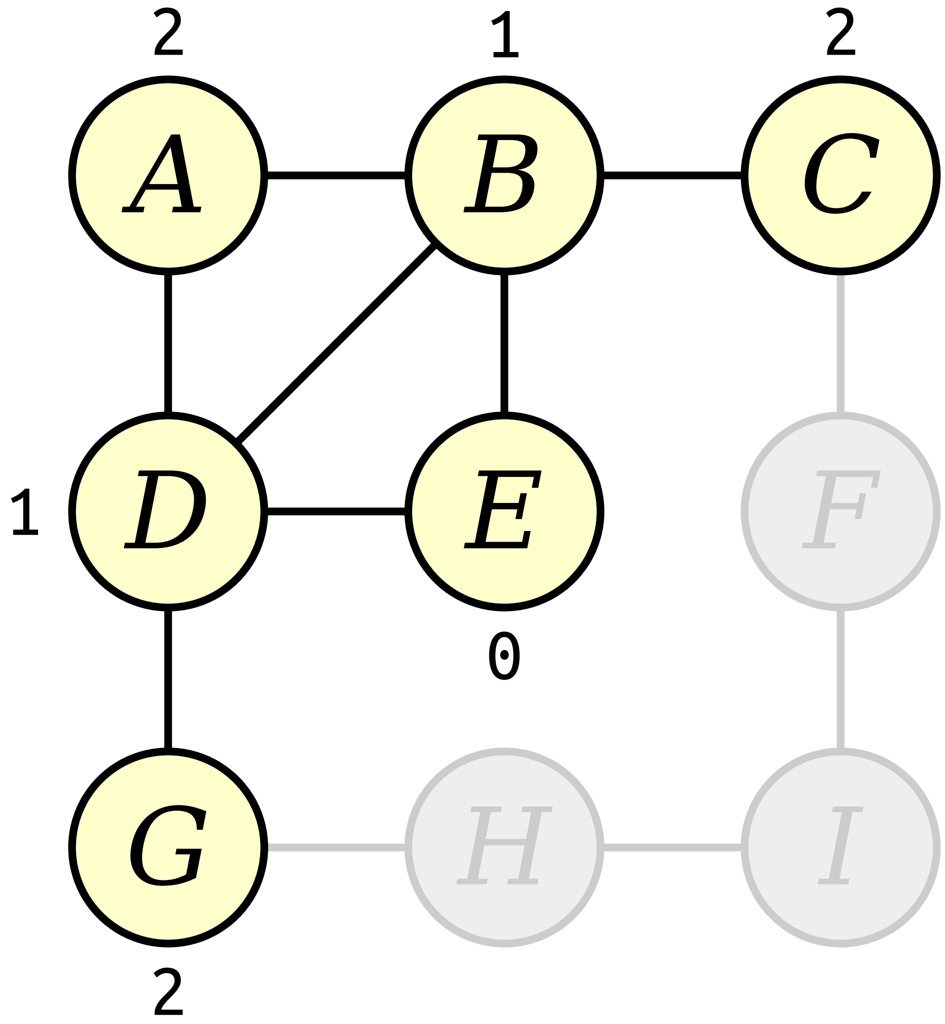
Load newly-discovered nodes into a queue.



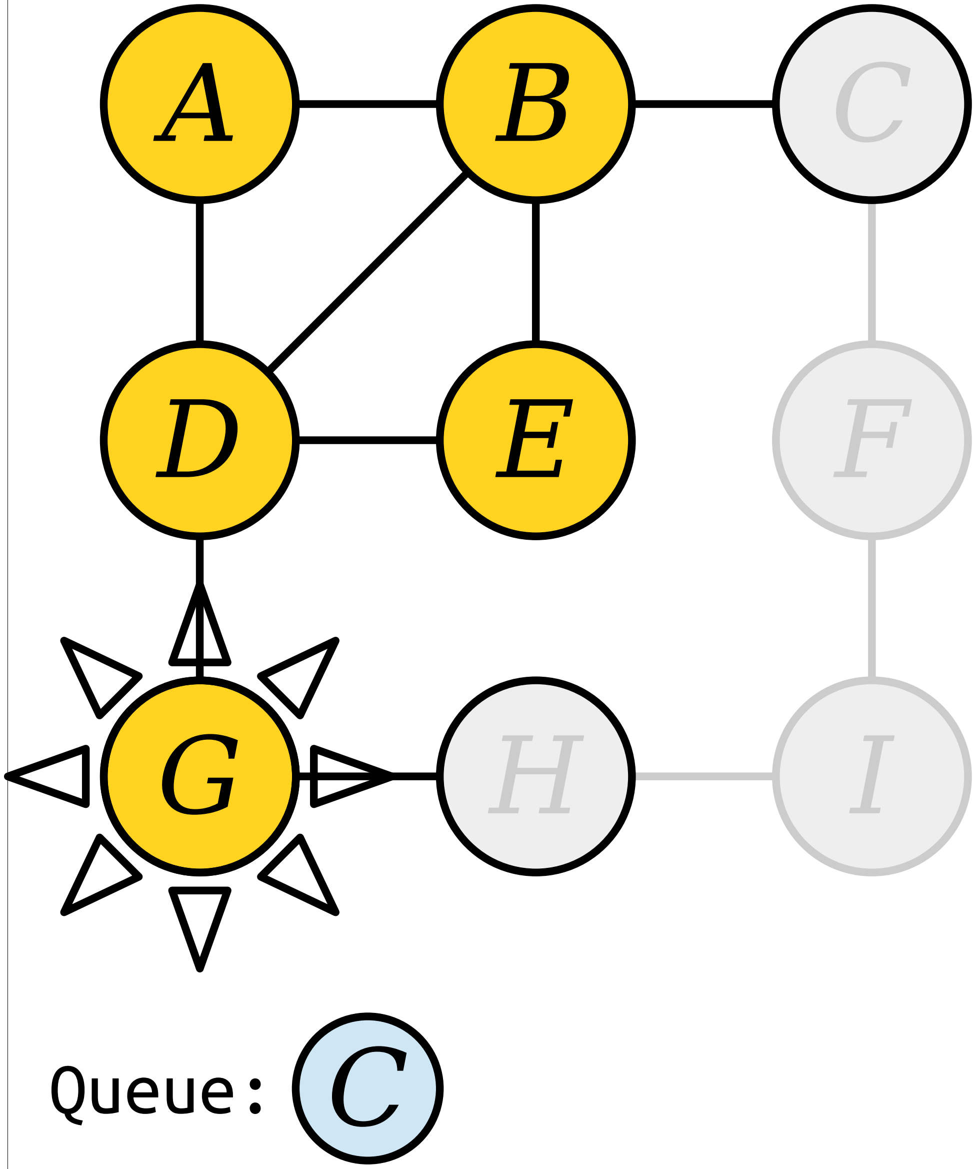
Visit nodes in ascending order of distance from the start node *E*.



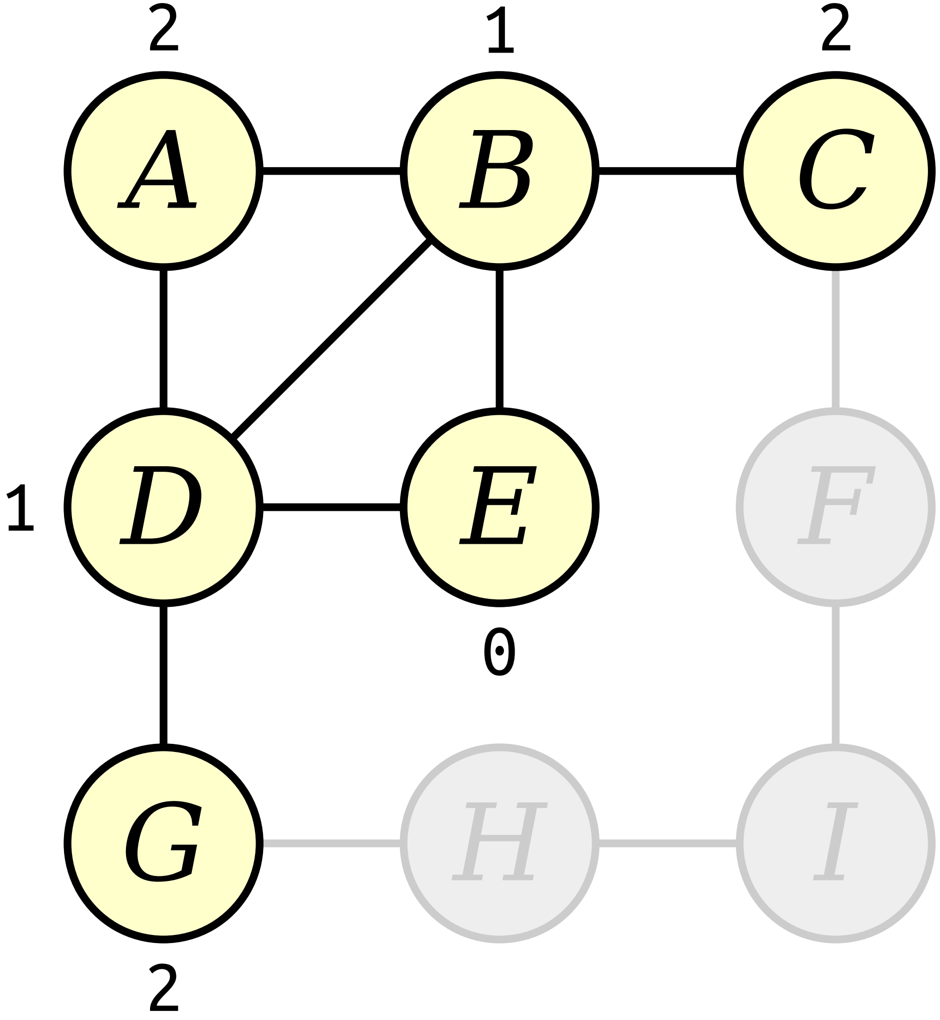
Load newly-discovered nodes into a queue.



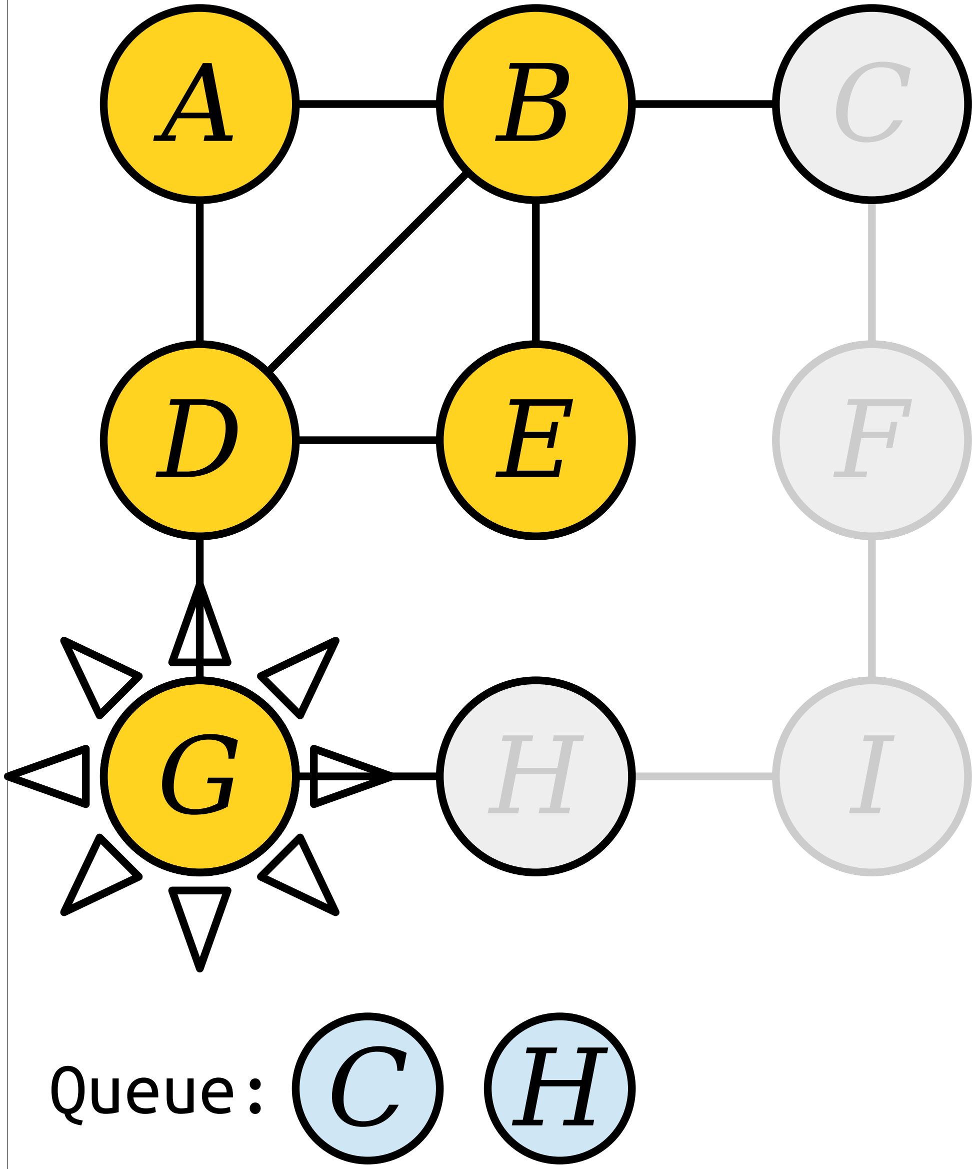
Visit nodes in ascending order of distance from the start node E.



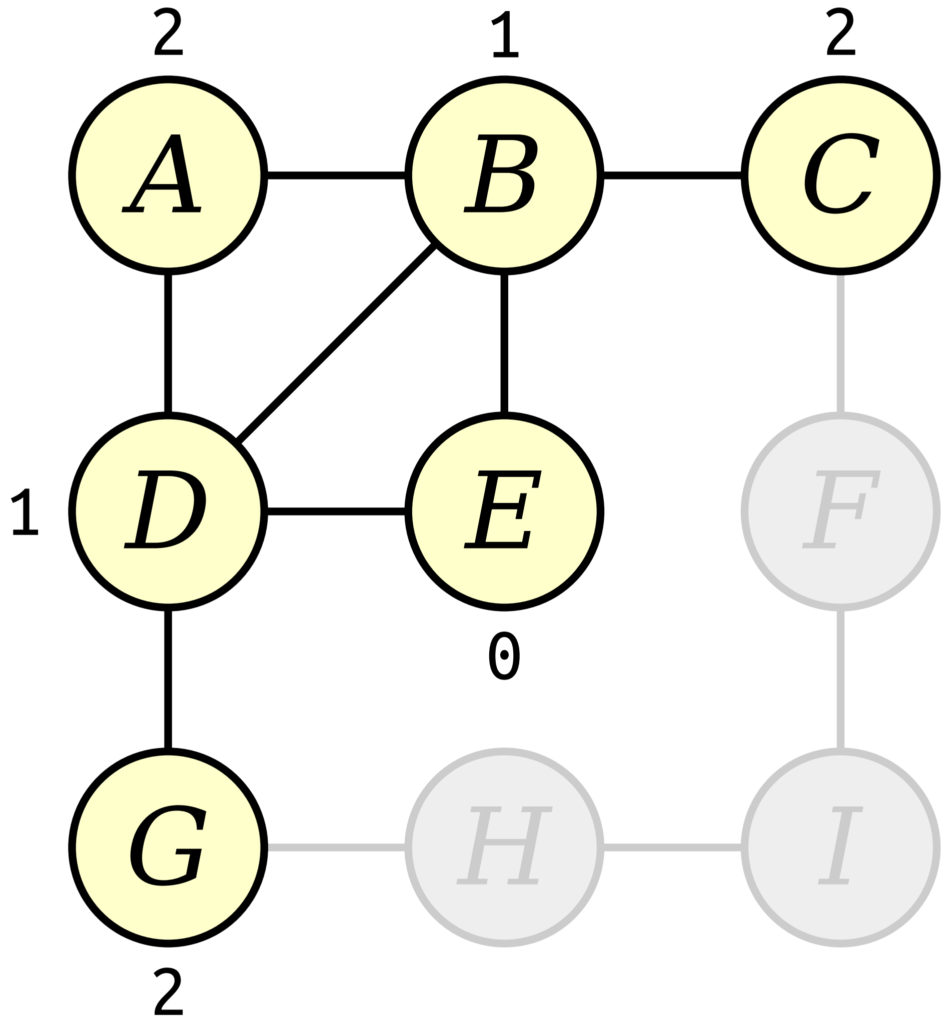
Load newly-discovered nodes into a queue.



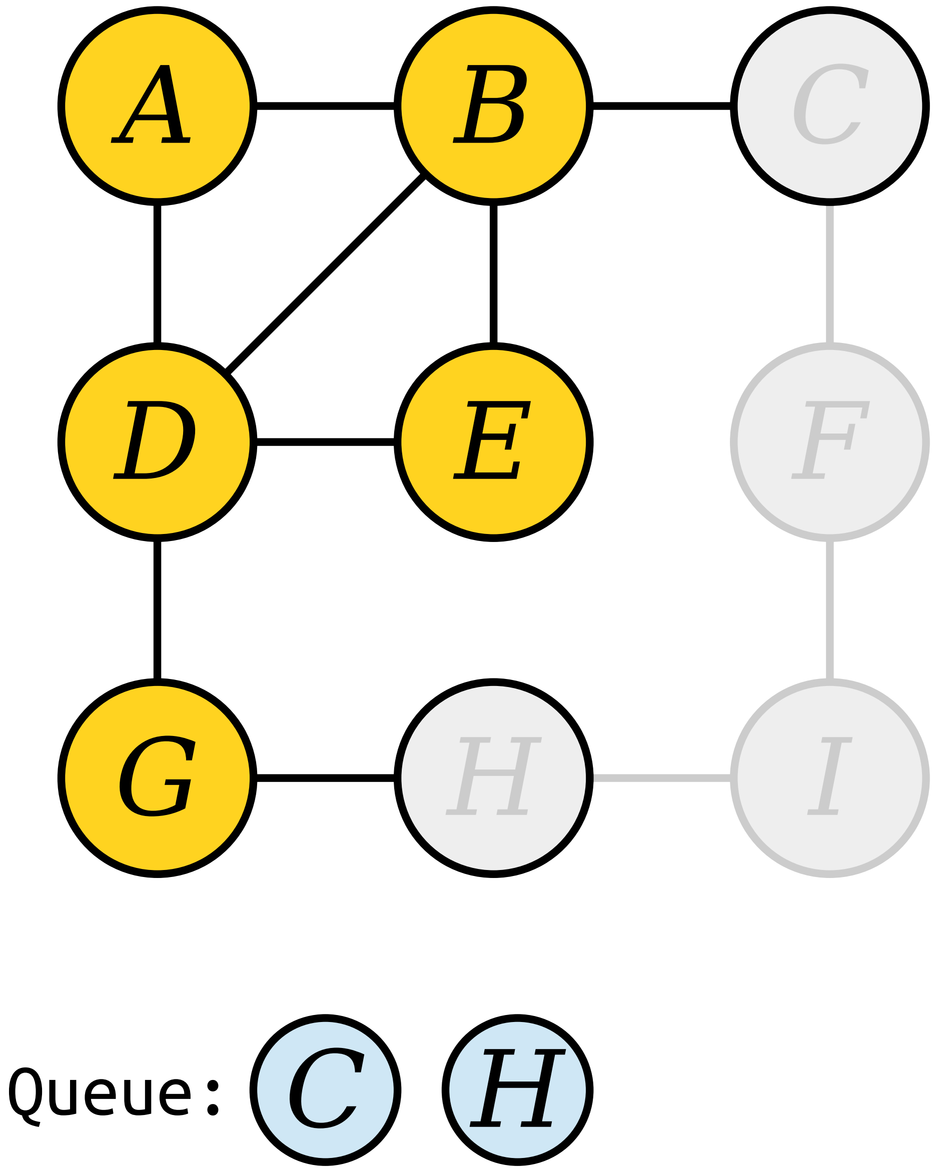
Visit nodes in ascending order of distance from the start node *E*.



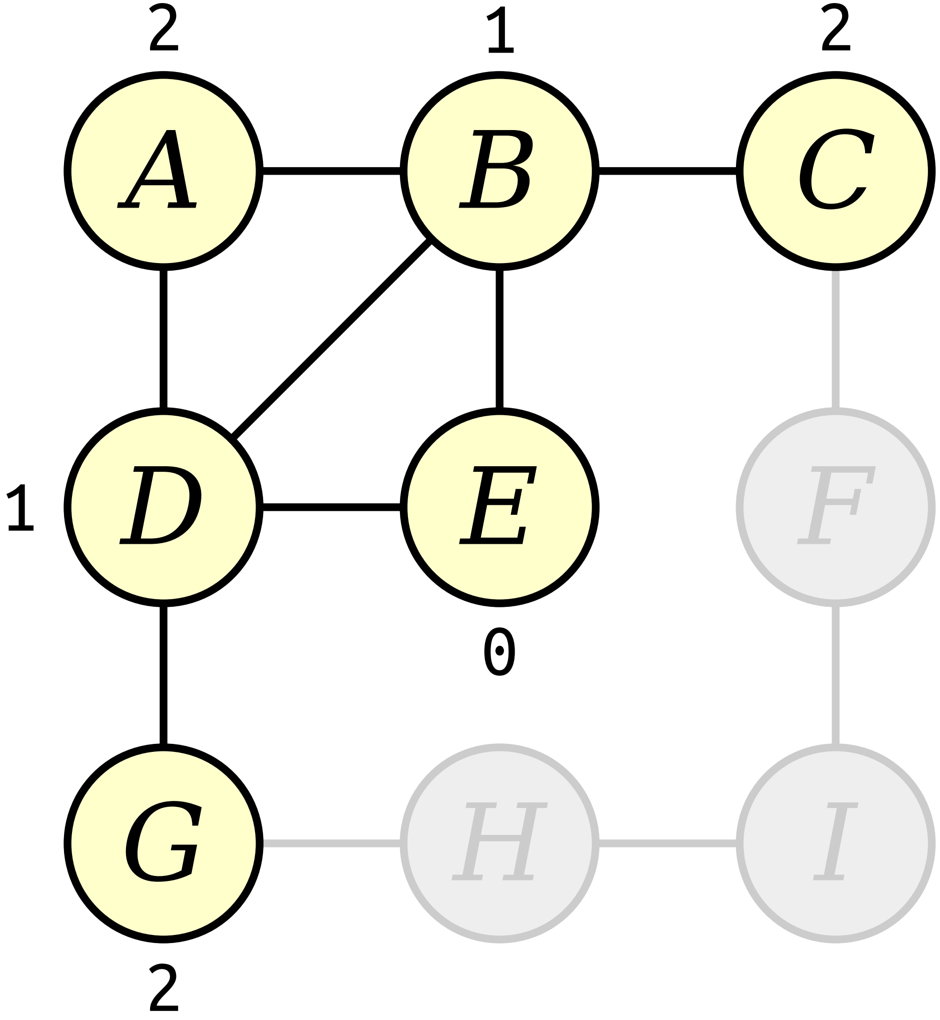
Load newly-discovered nodes into a queue.



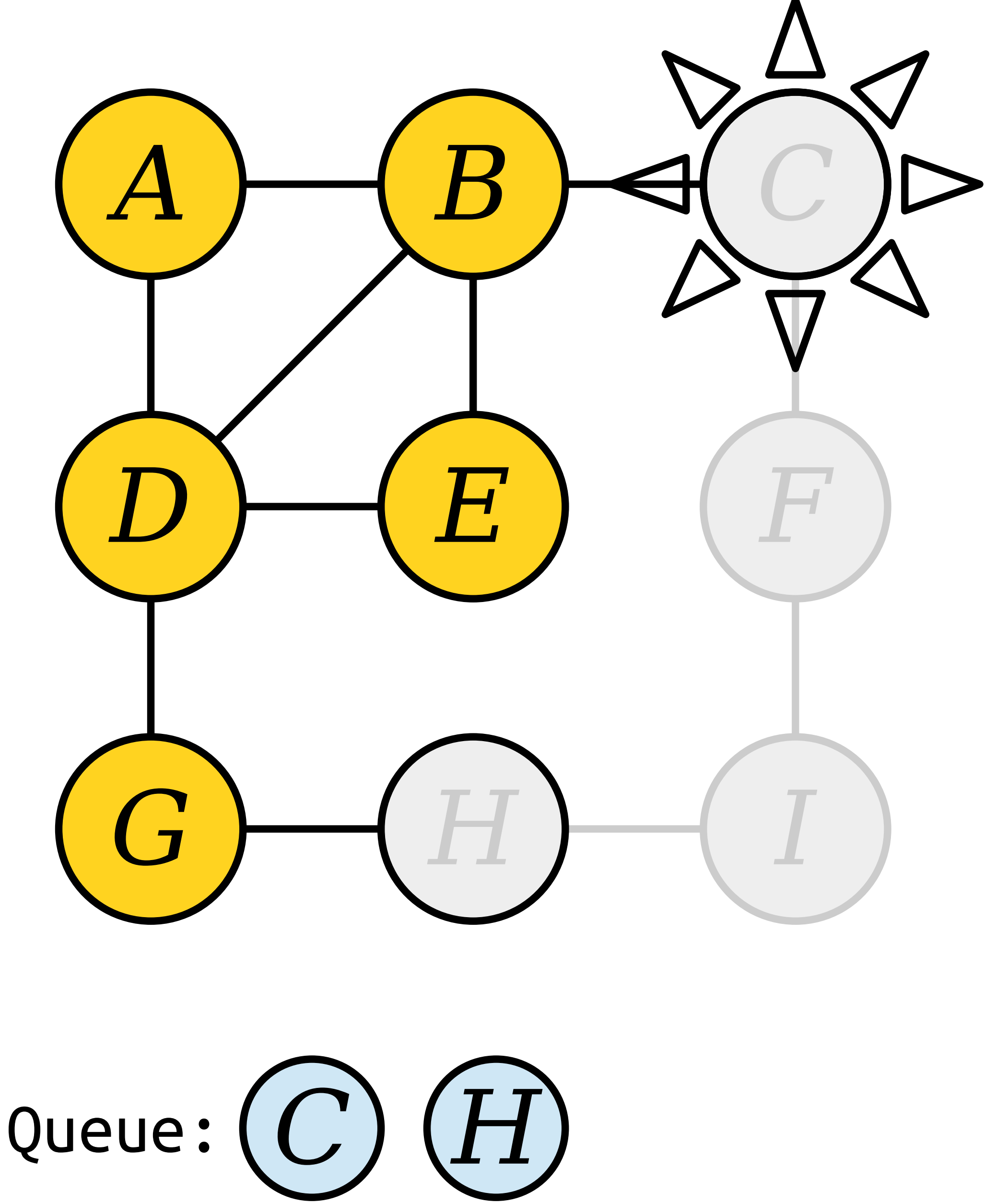
Visit nodes in ascending order of distance from the start node *E*.



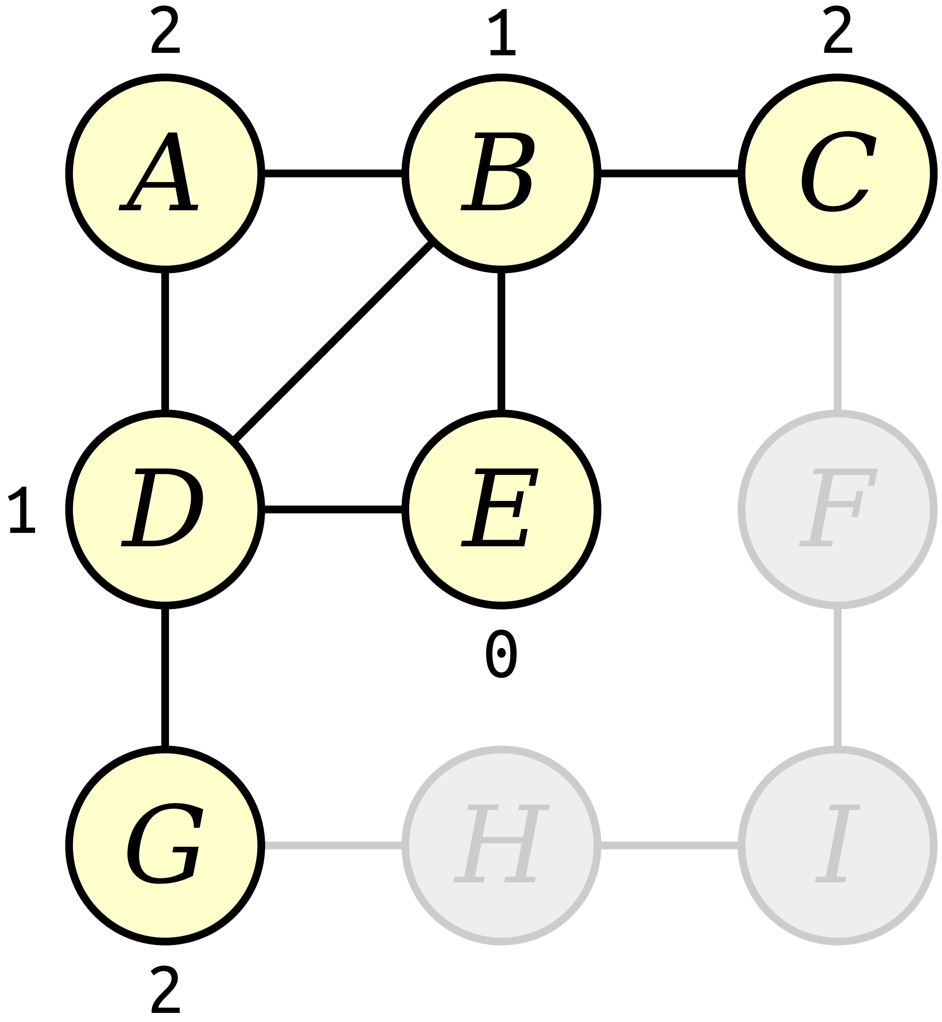
Load newly-discovered nodes into a queue.



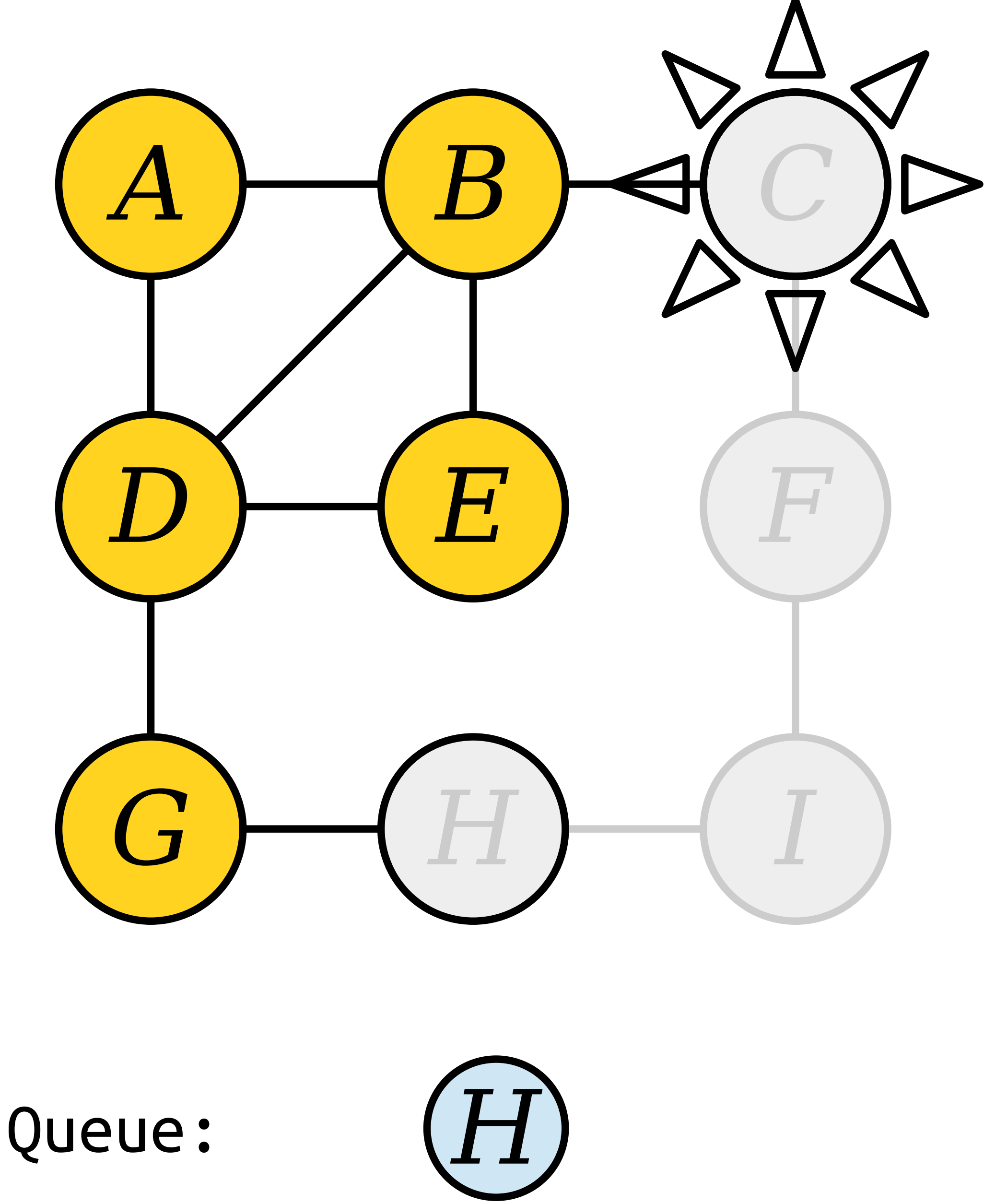
Visit nodes in ascending order of distance from the start node *E*.



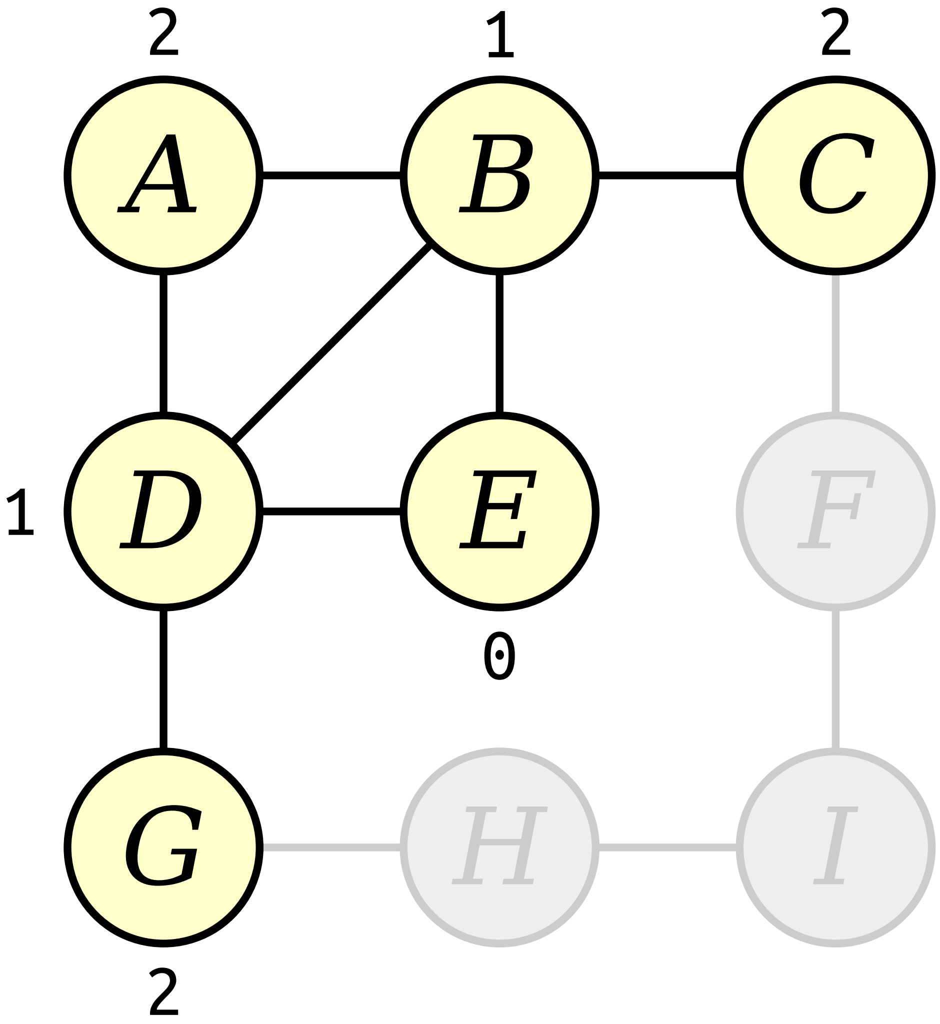
Load newly-discovered nodes into a queue.



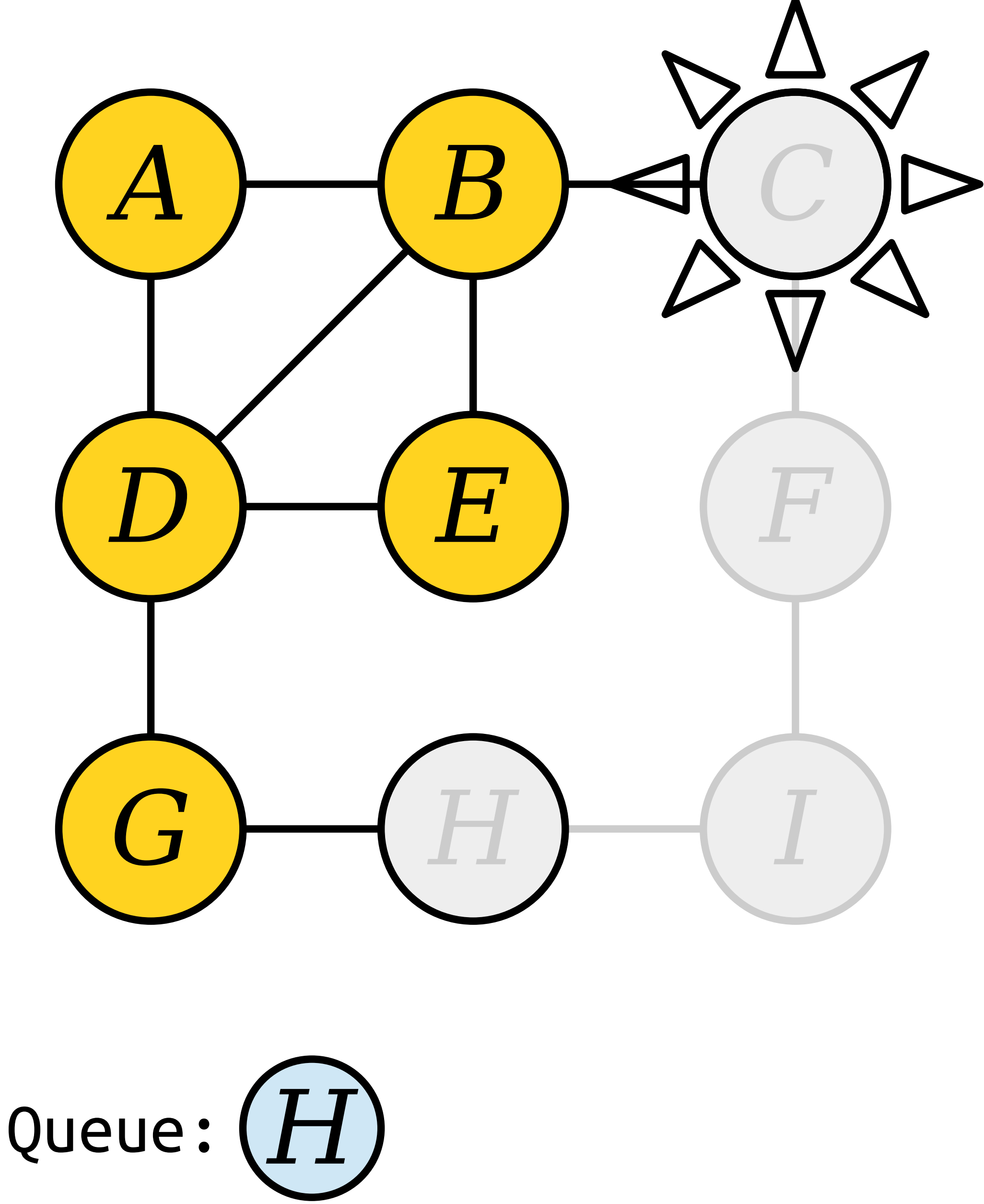
Visit nodes in ascending order of distance from the start node *E*.



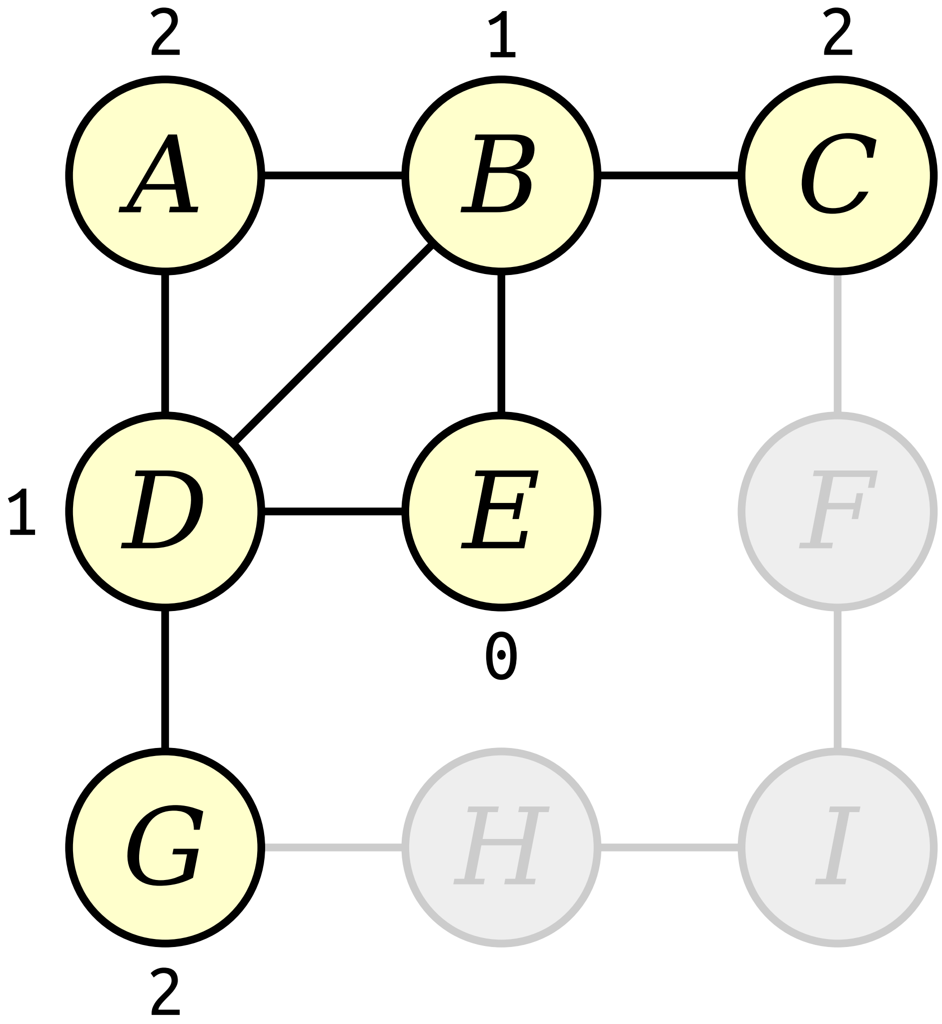
Load newly-discovered nodes into a queue.



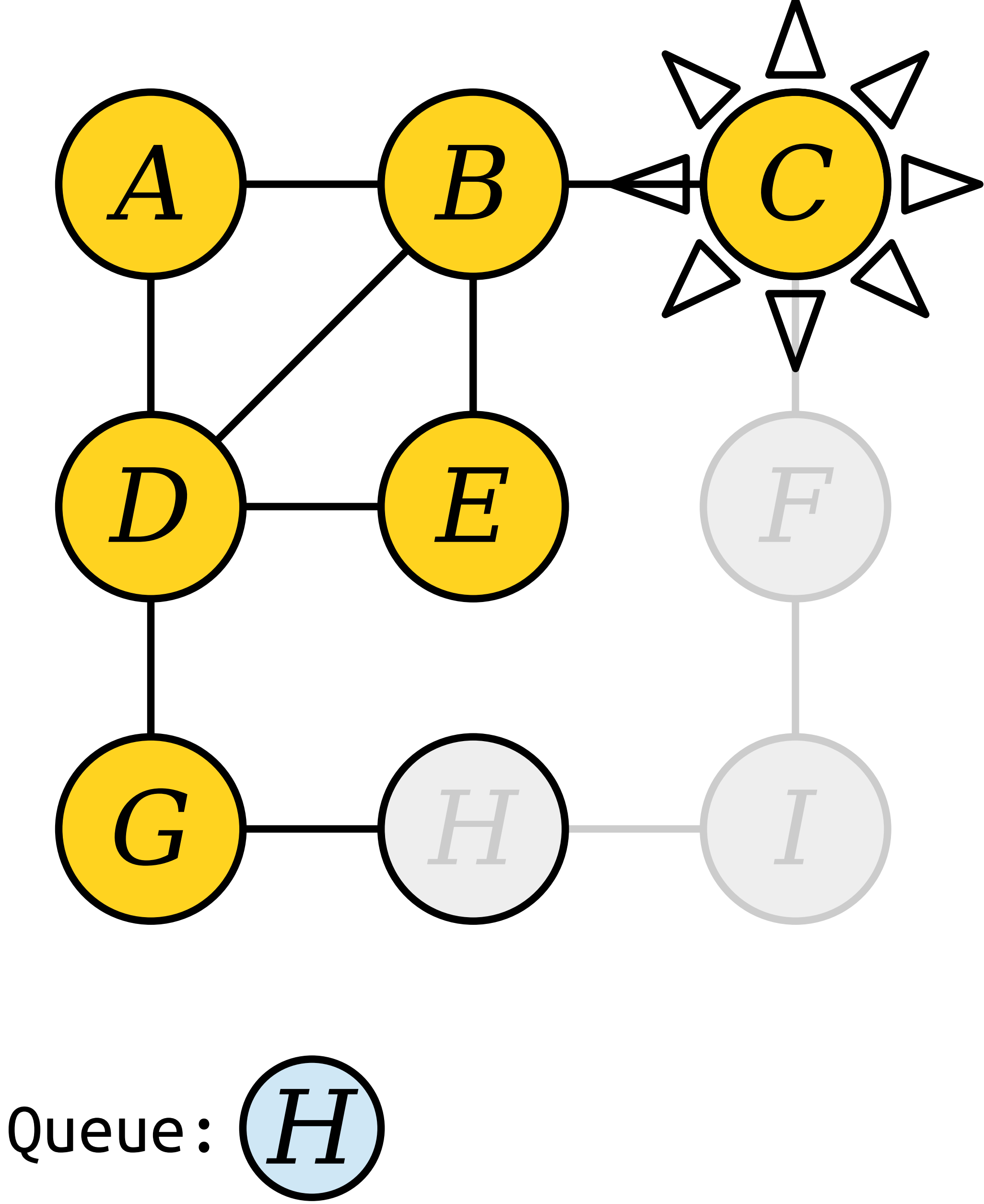
Visit nodes in ascending order of distance from the start node *E*.



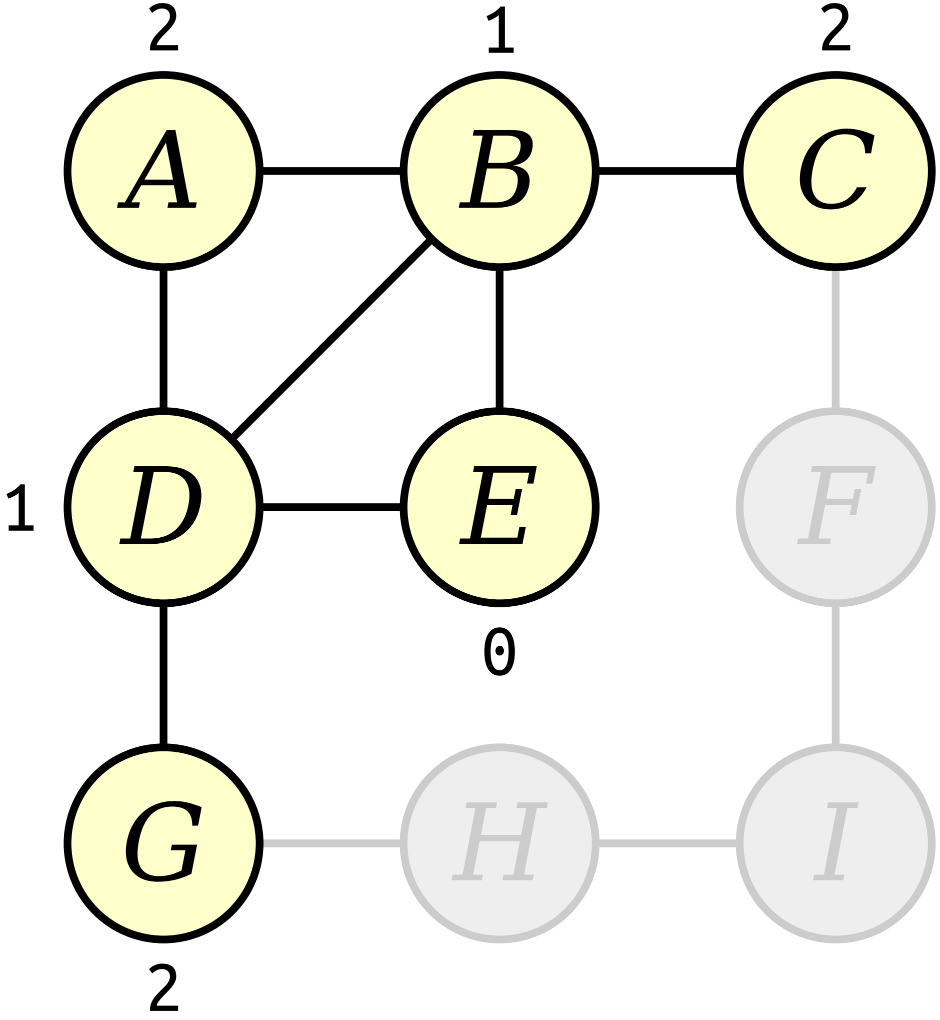
Load newly-discovered nodes into a queue.



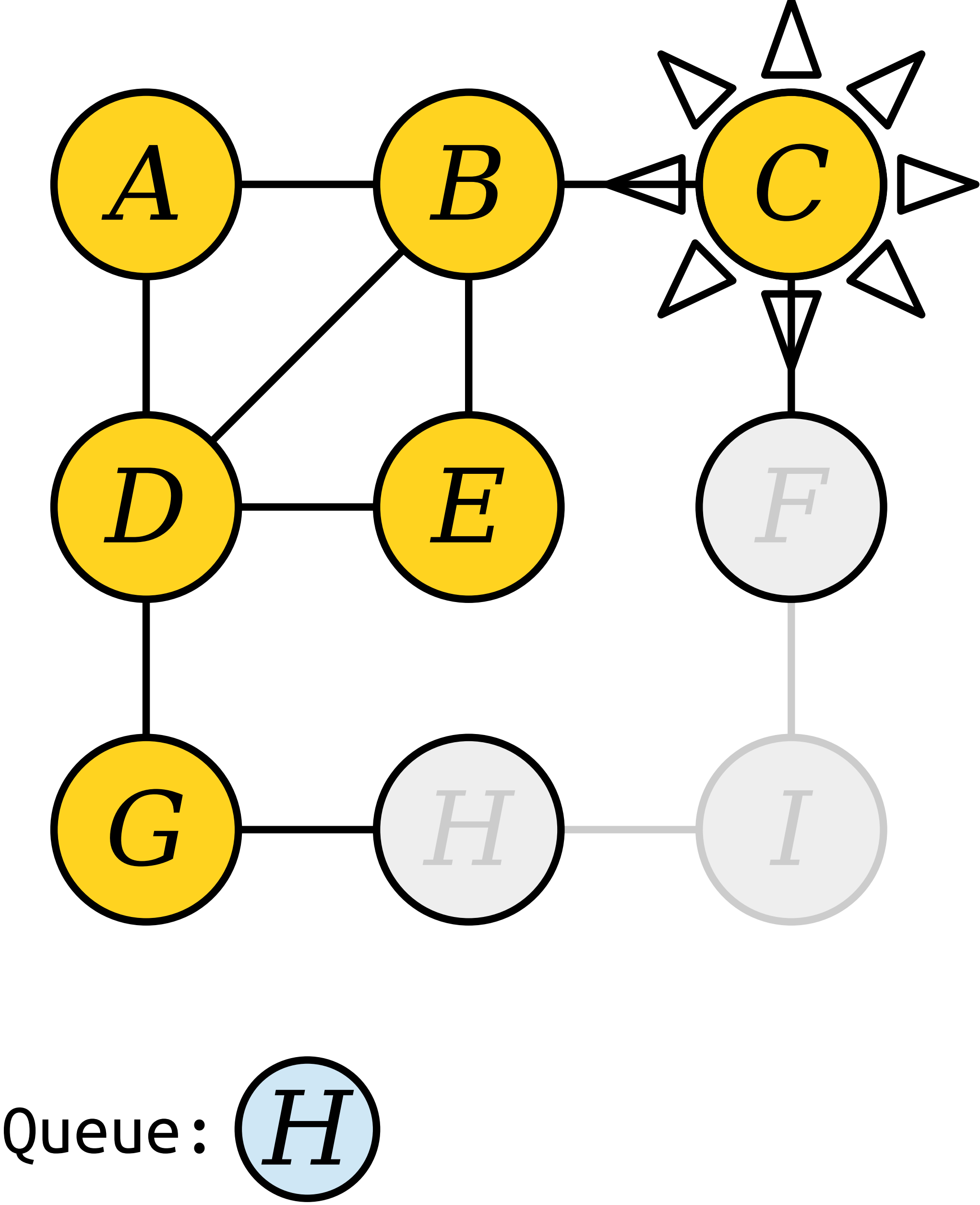
Visit nodes in ascending order of distance from the start node E.



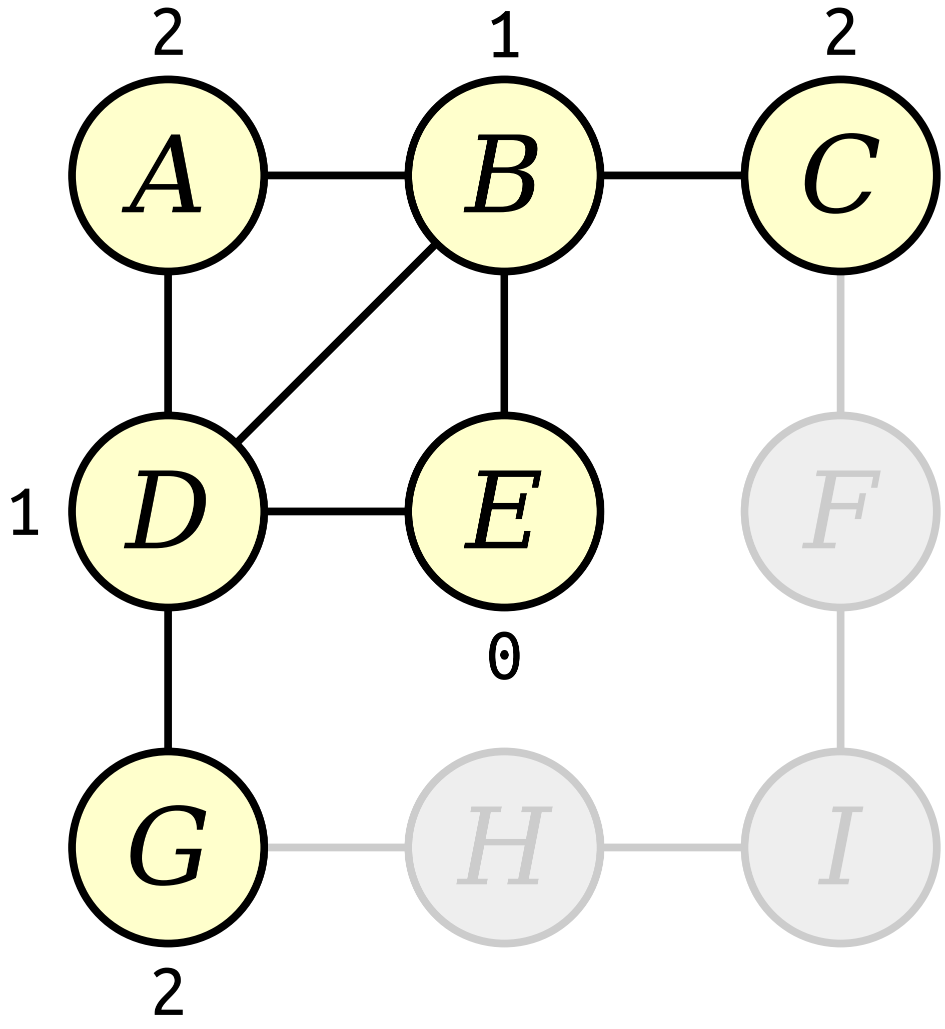
Load newly-discovered nodes into a queue.



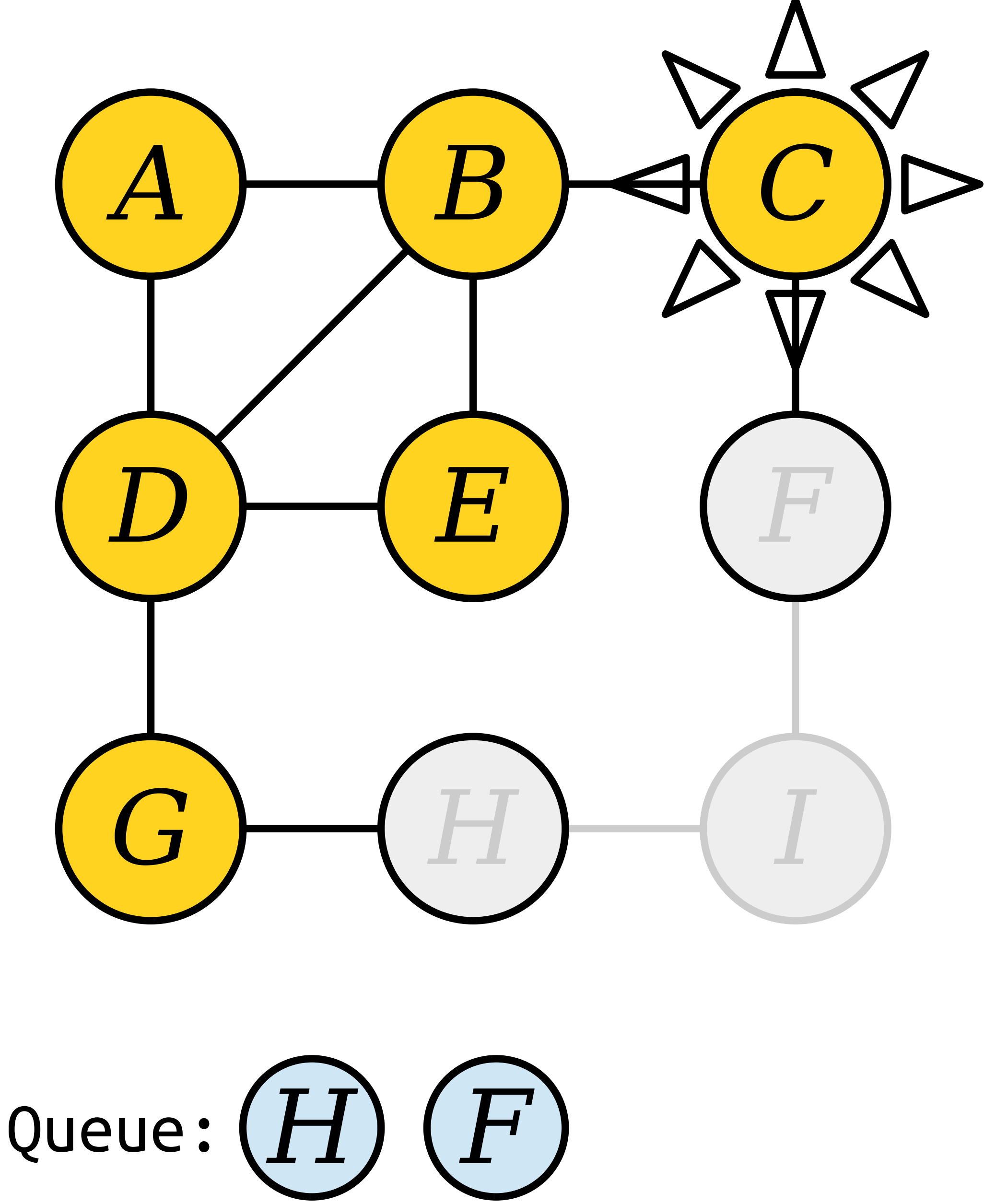
Visit nodes in ascending order of distance from the start node *E*.



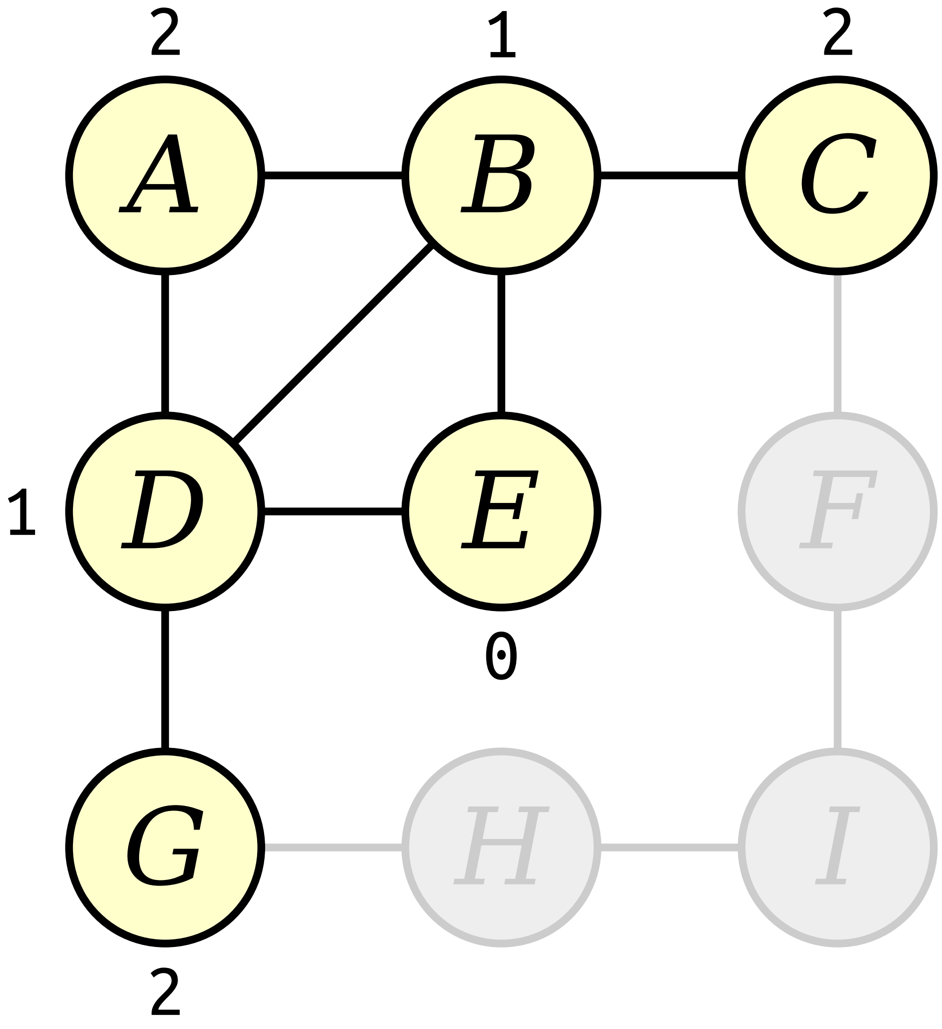
Load newly-discovered nodes into a queue.



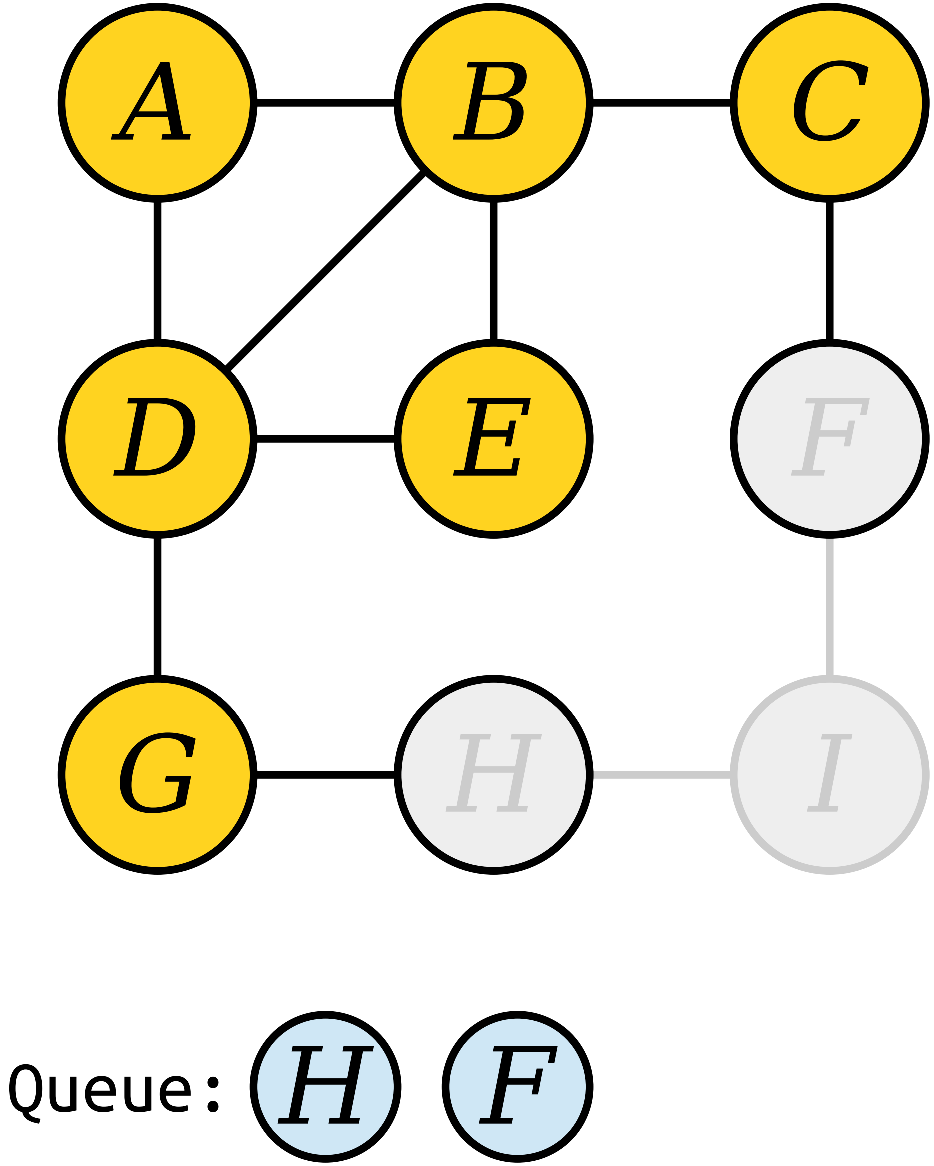
Visit nodes in ascending order of distance from the start node *E*.



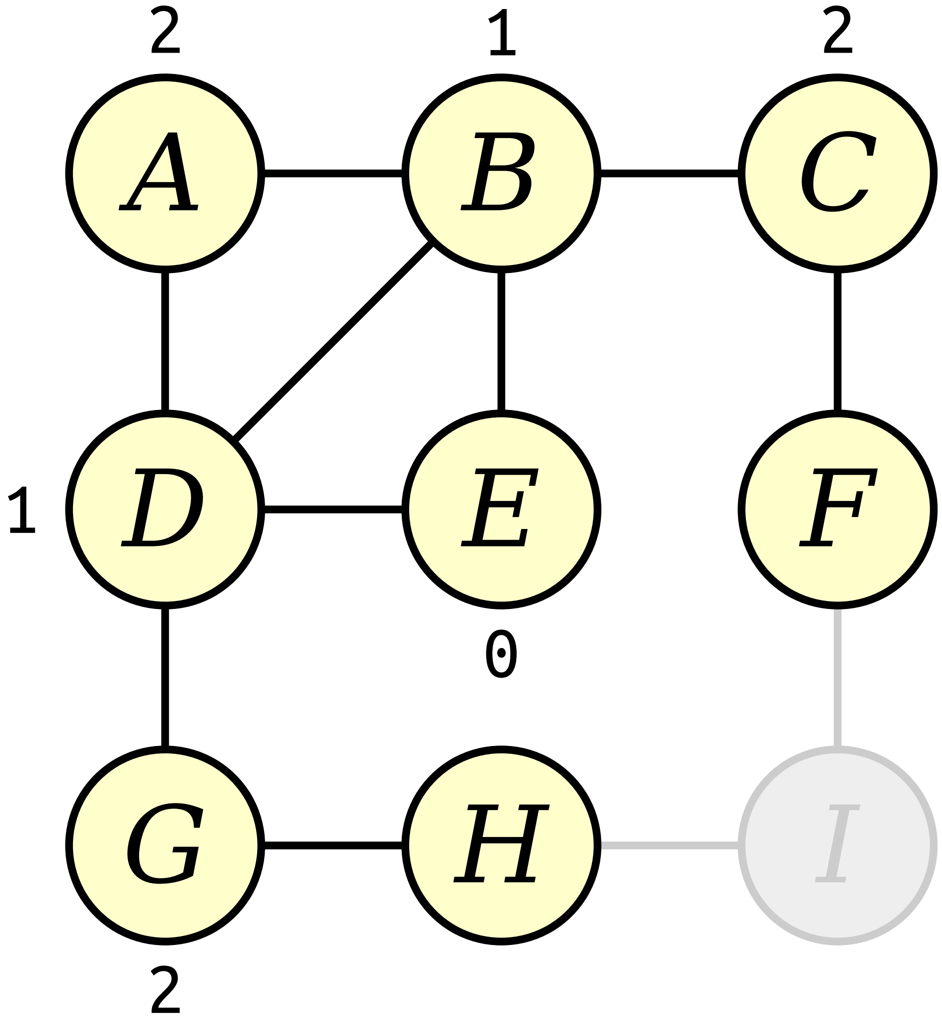
Load newly-discovered nodes into a queue.



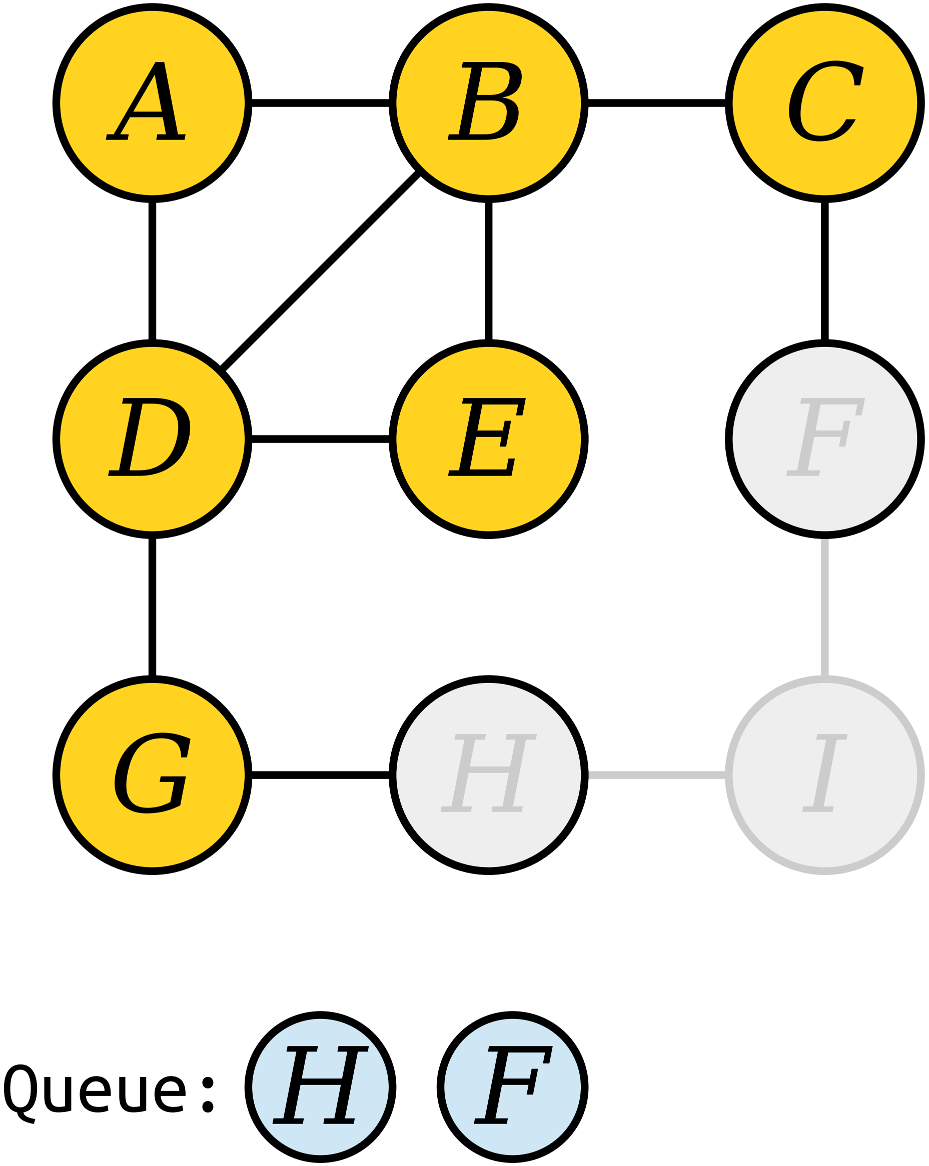
Visit nodes in ascending order of distance from the start node E.



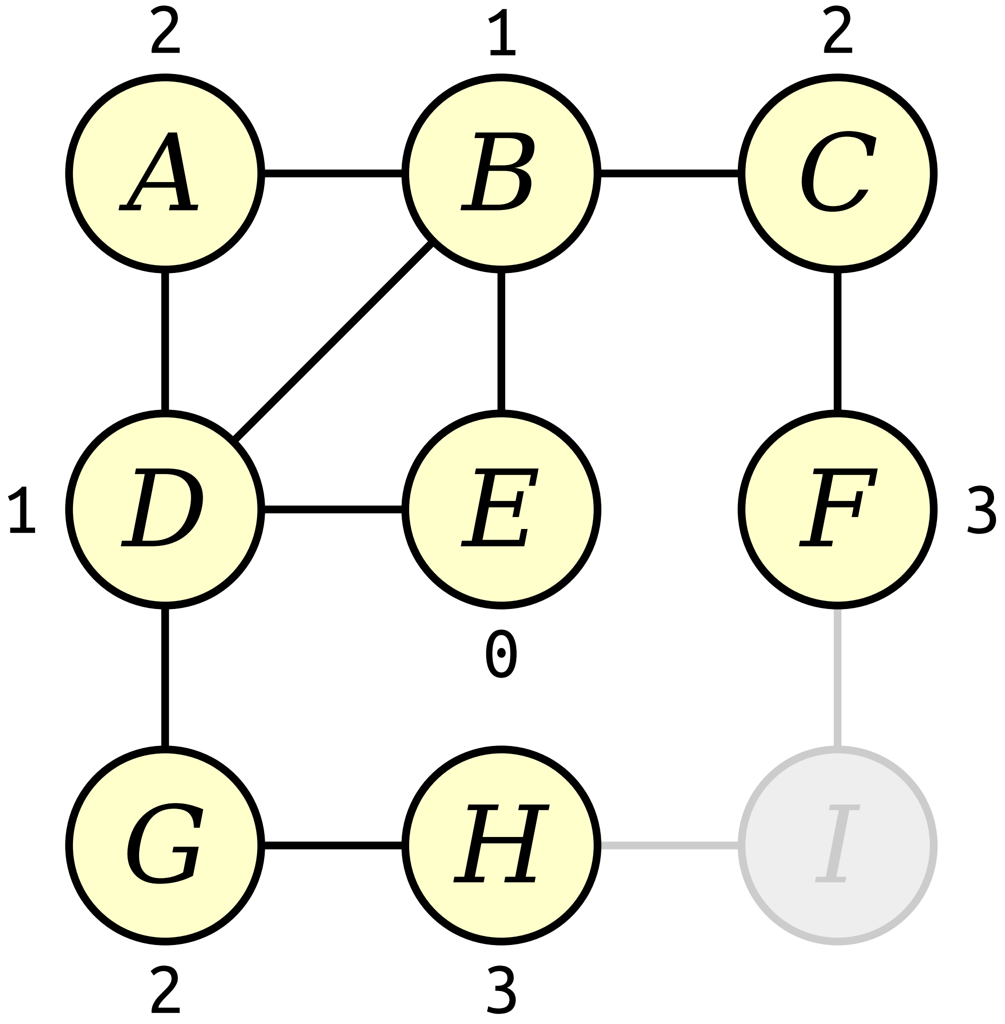
Load newly-discovered nodes into a queue.



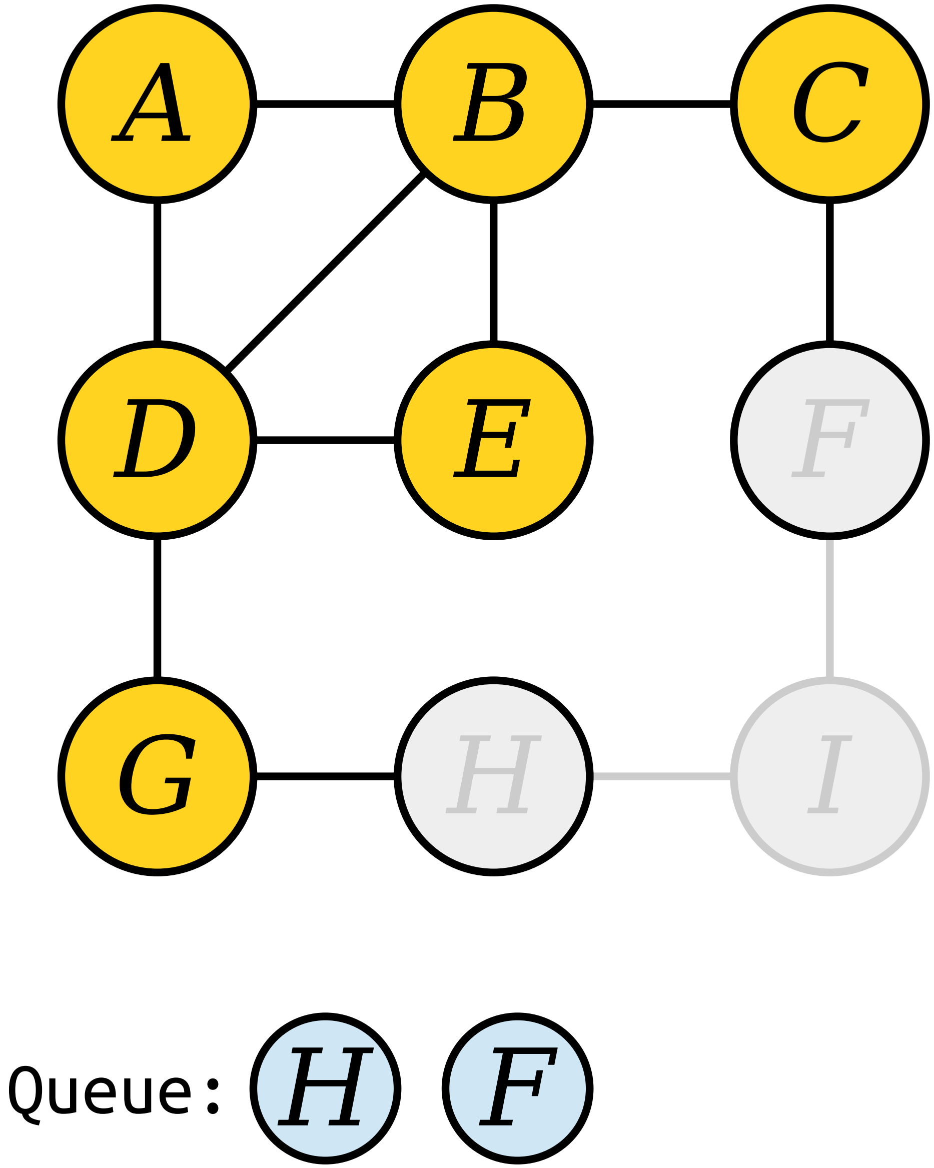
Visit nodes in ascending order of distance from the start node *E*.



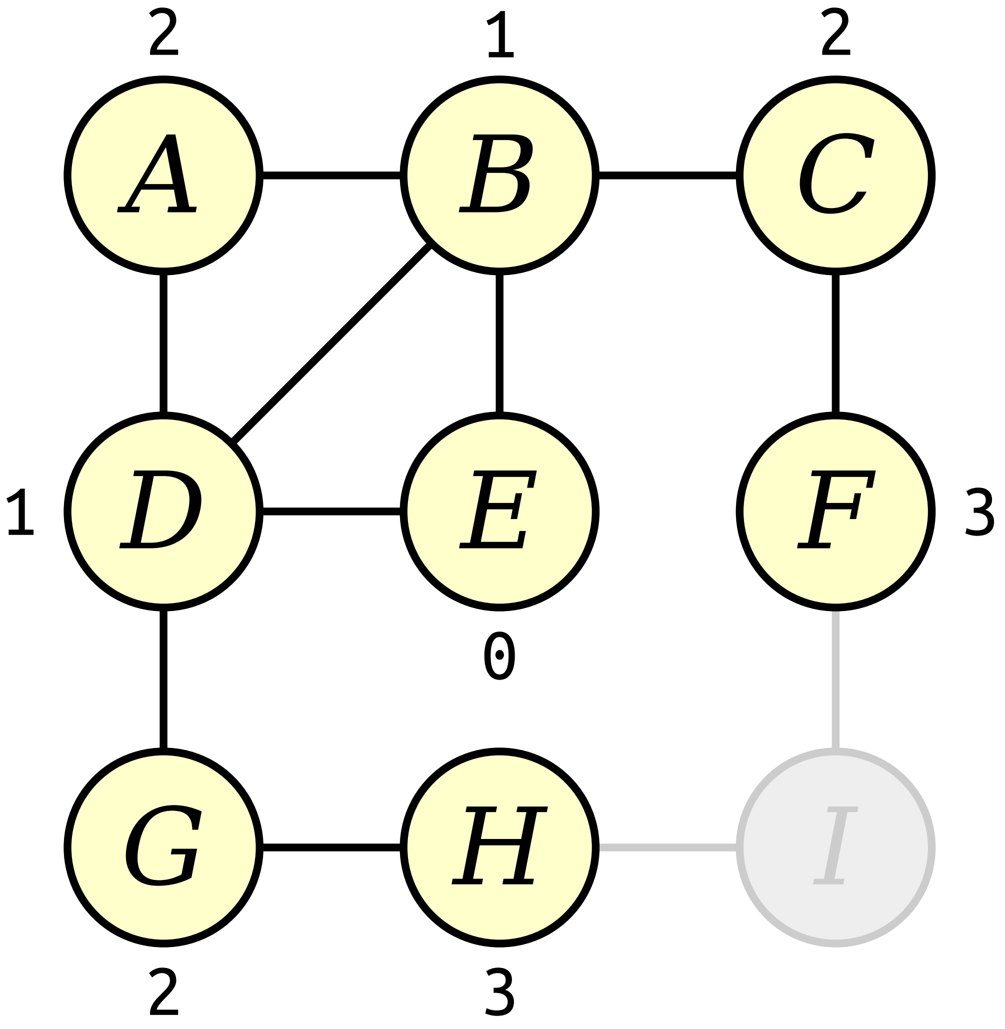
Load newly-discovered nodes into a queue.



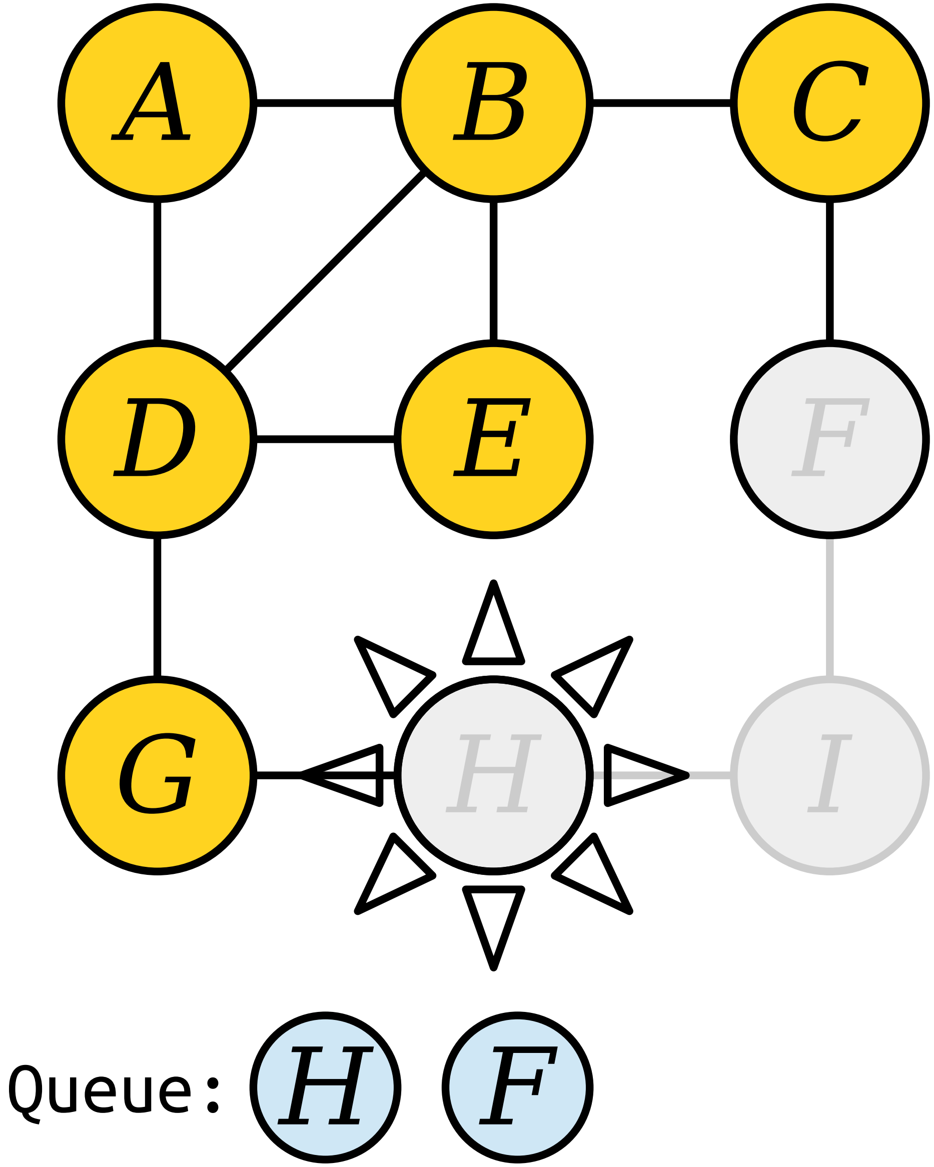
Visit nodes in ascending order of distance from the start node E.



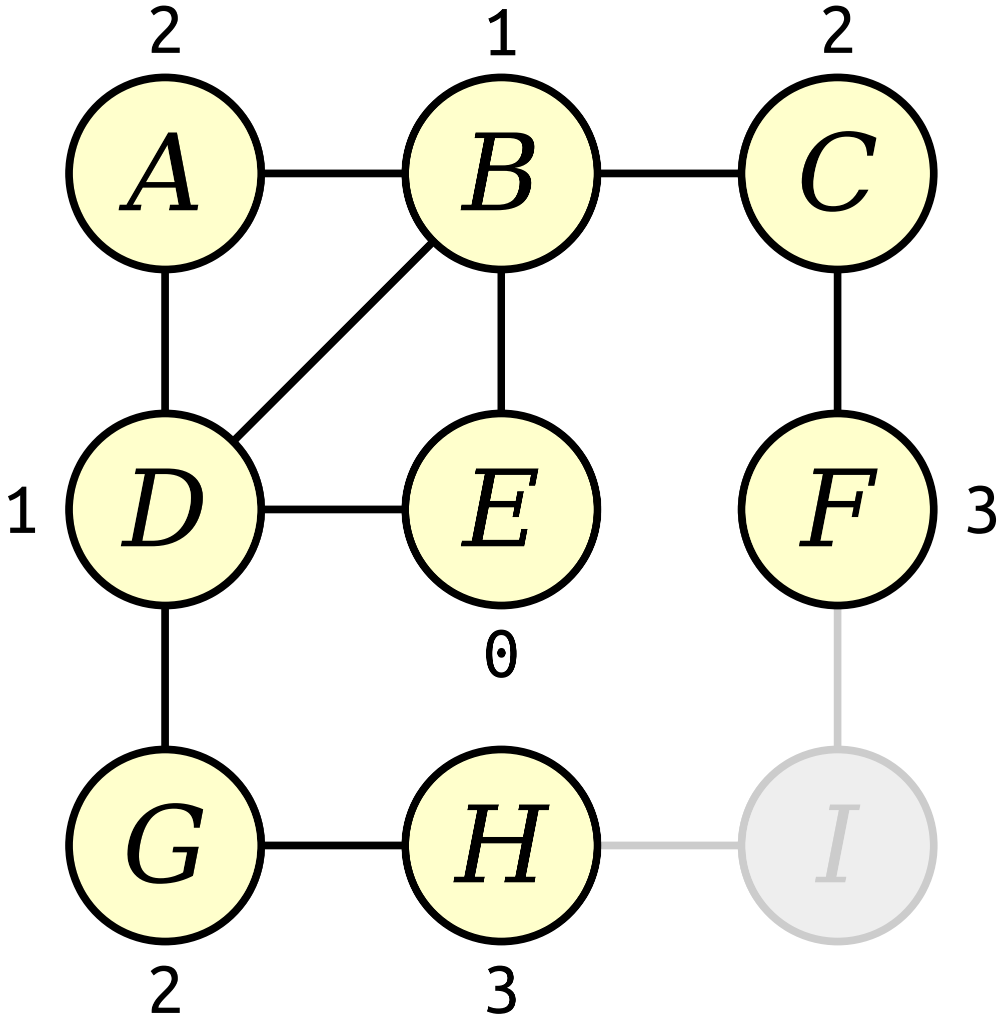
Load newly-discovered nodes into a queue.



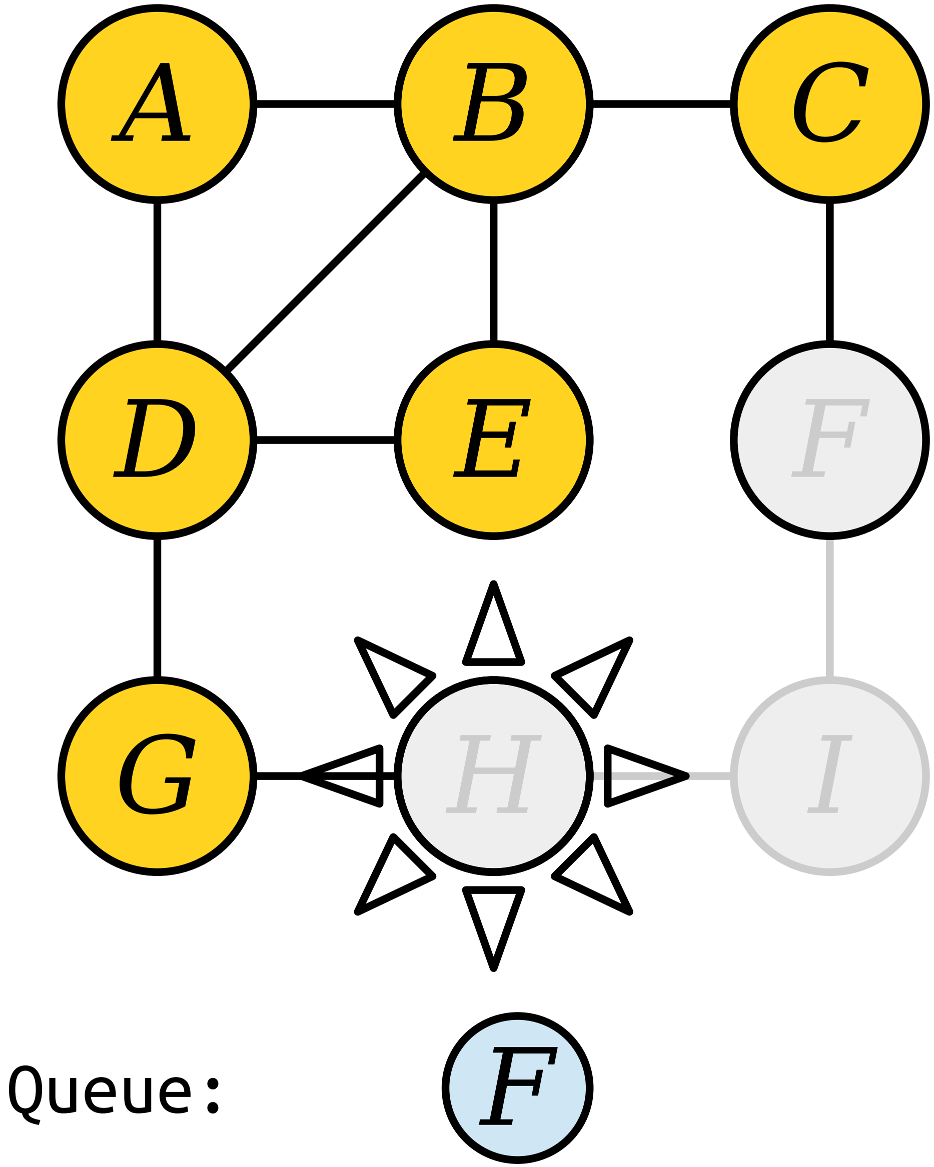
Visit nodes in ascending order of distance from the start node E.



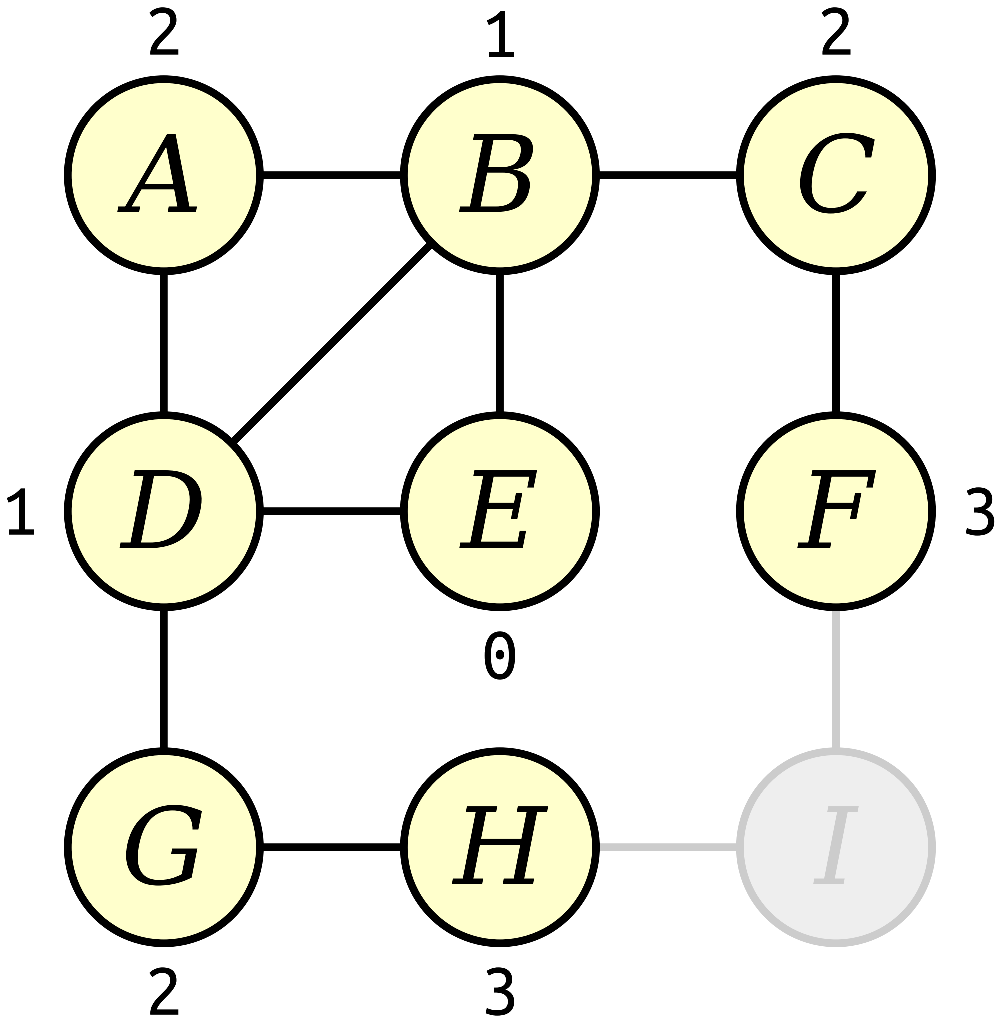
Load newly-discovered nodes into a queue.



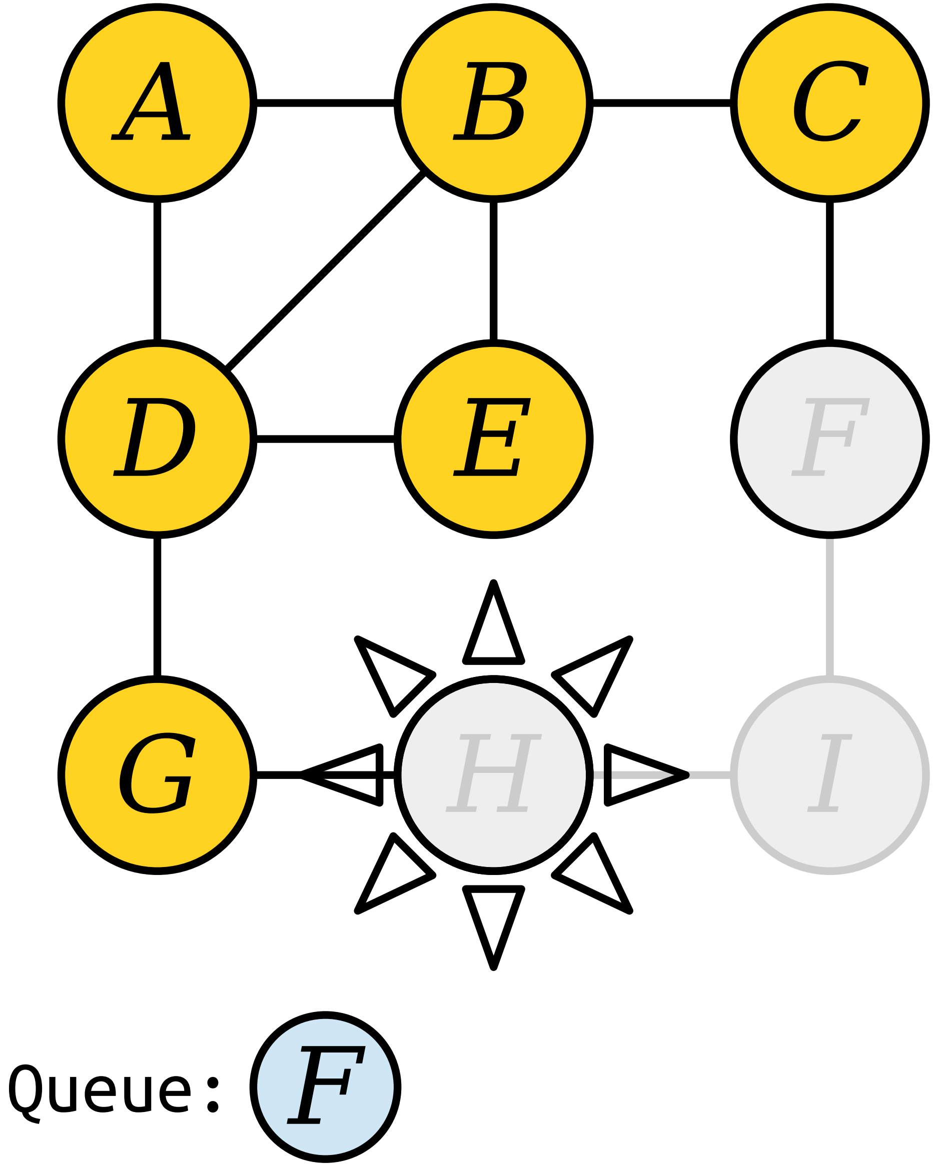
Visit nodes in ascending order of distance from the start node E.



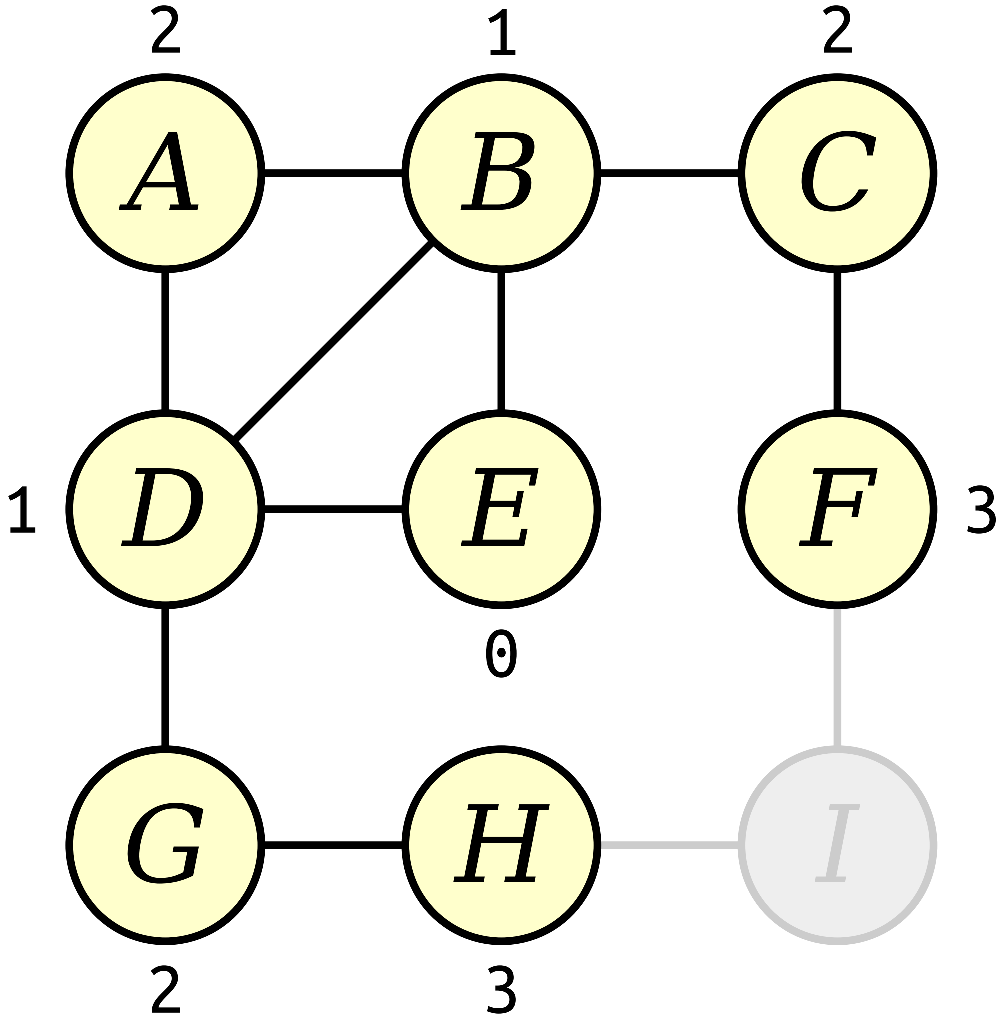
Load newly-discovered nodes into a queue.



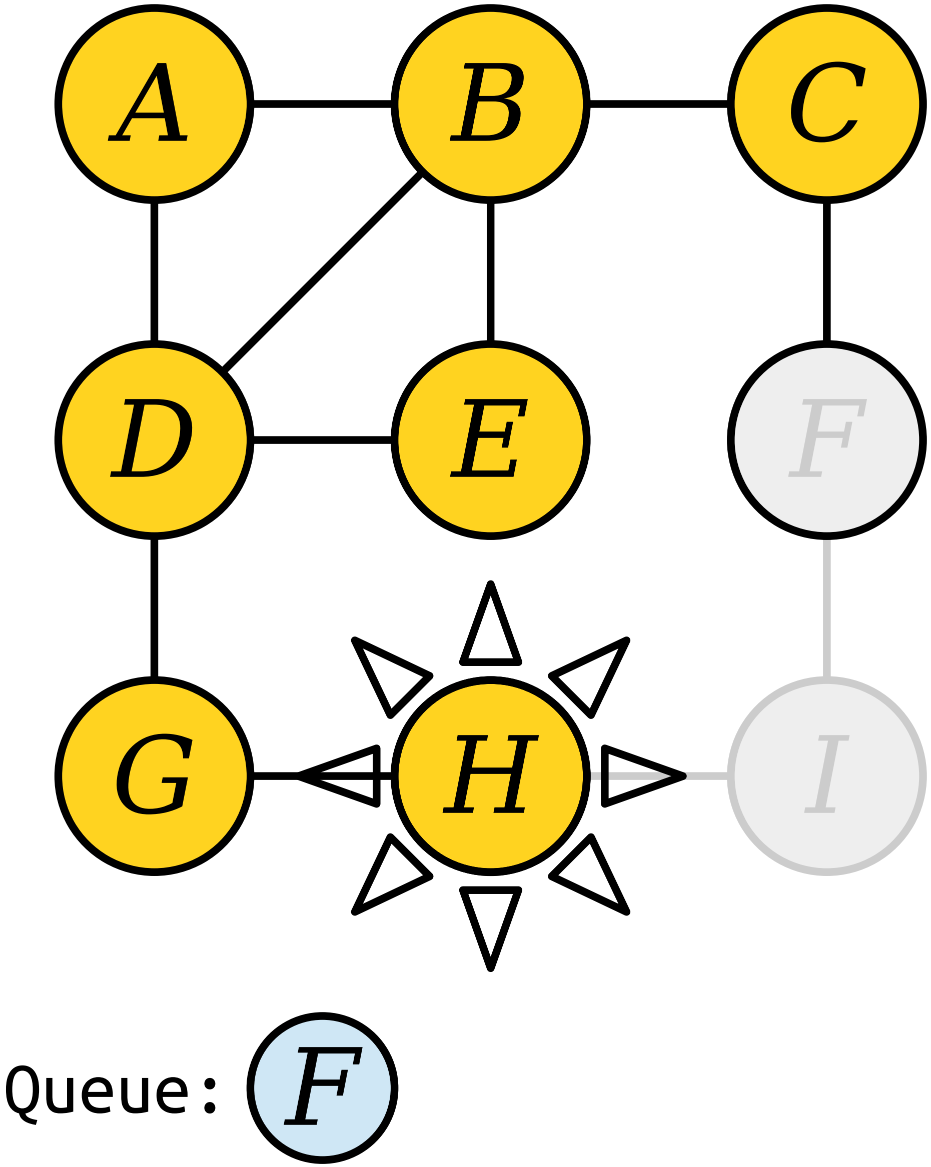
Visit nodes in ascending order of distance from the start node *E*.



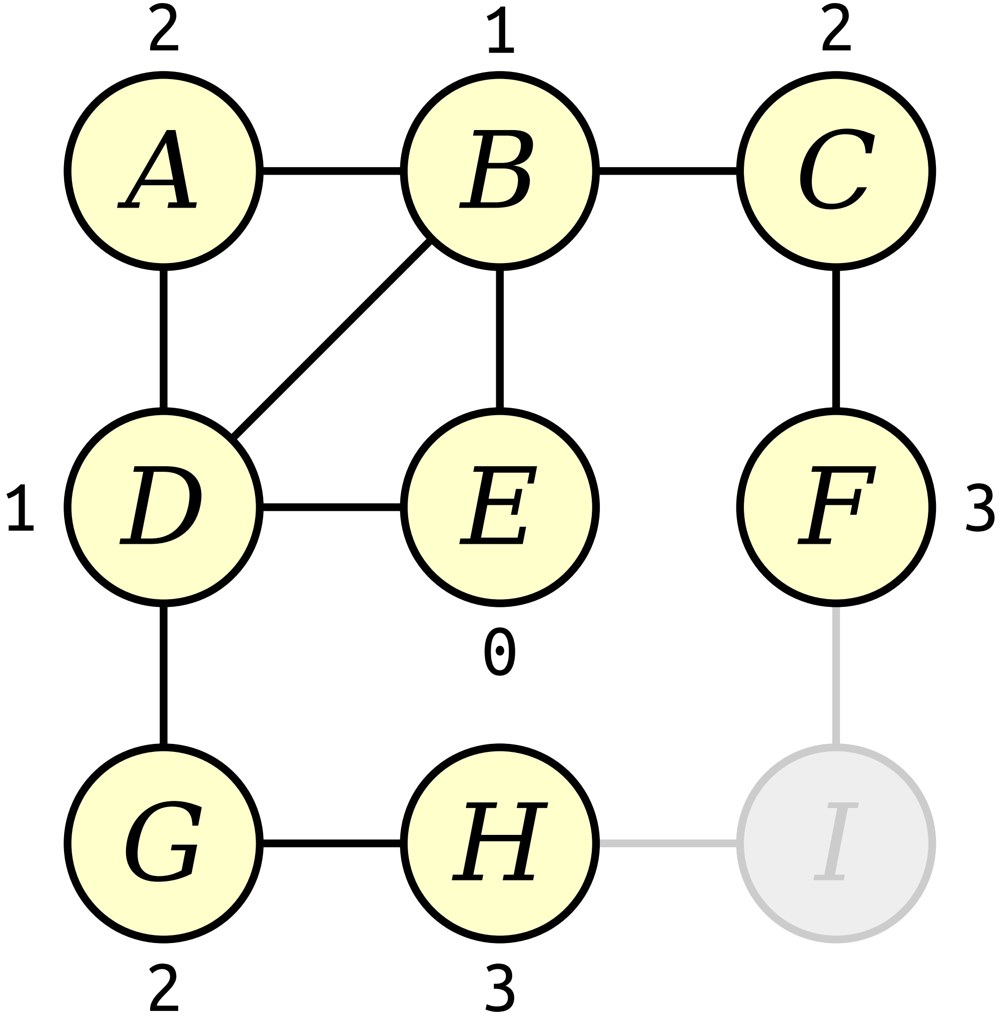
Load newly-discovered nodes into a queue.



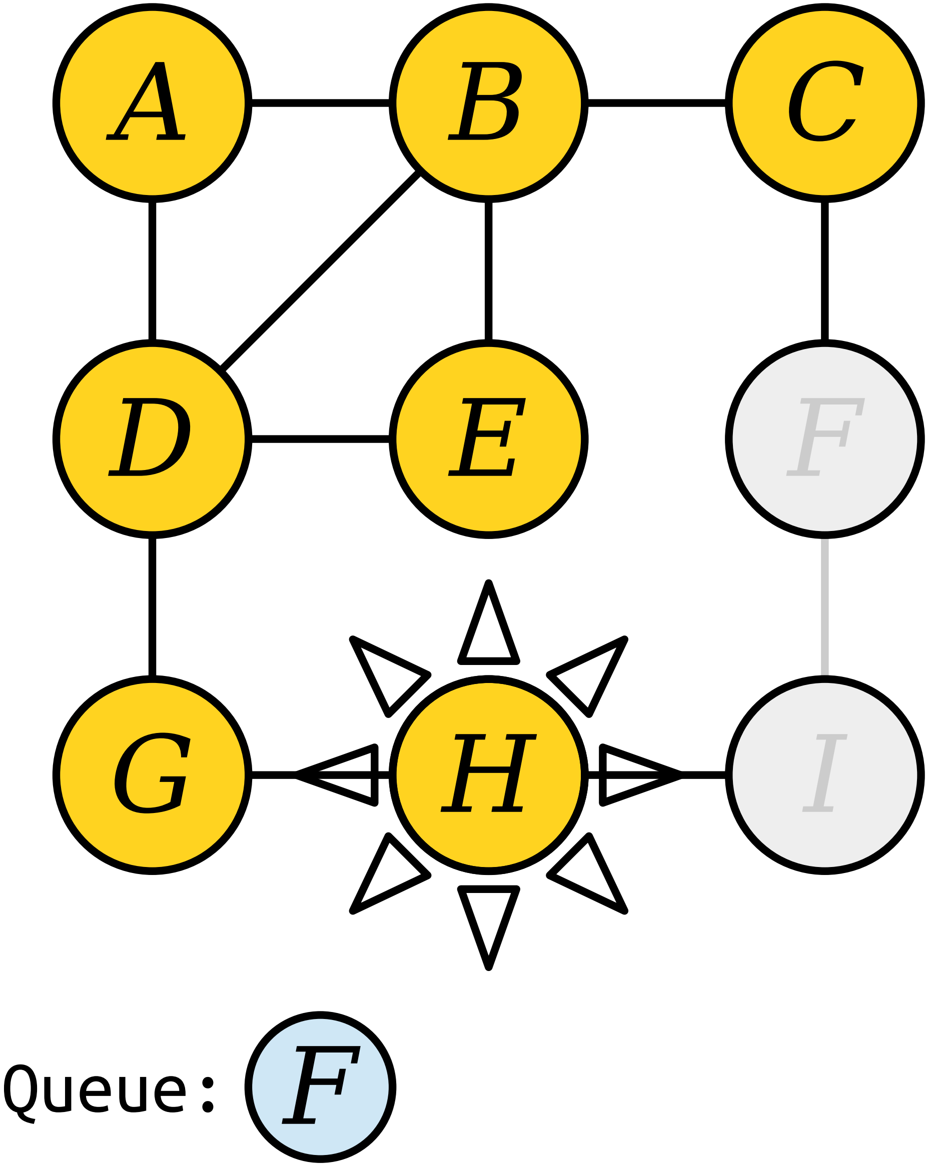
Visit nodes in ascending order of distance from the start node E.



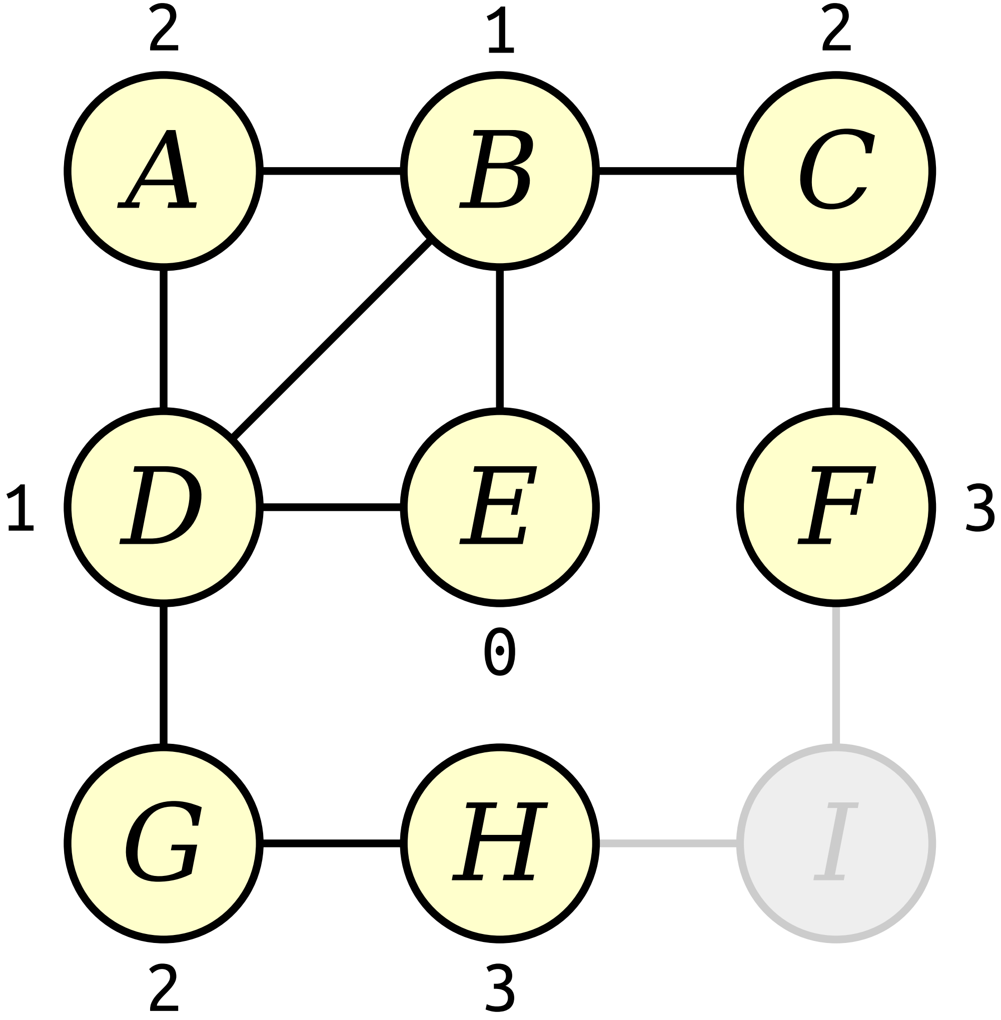
Load newly-discovered nodes into a queue.



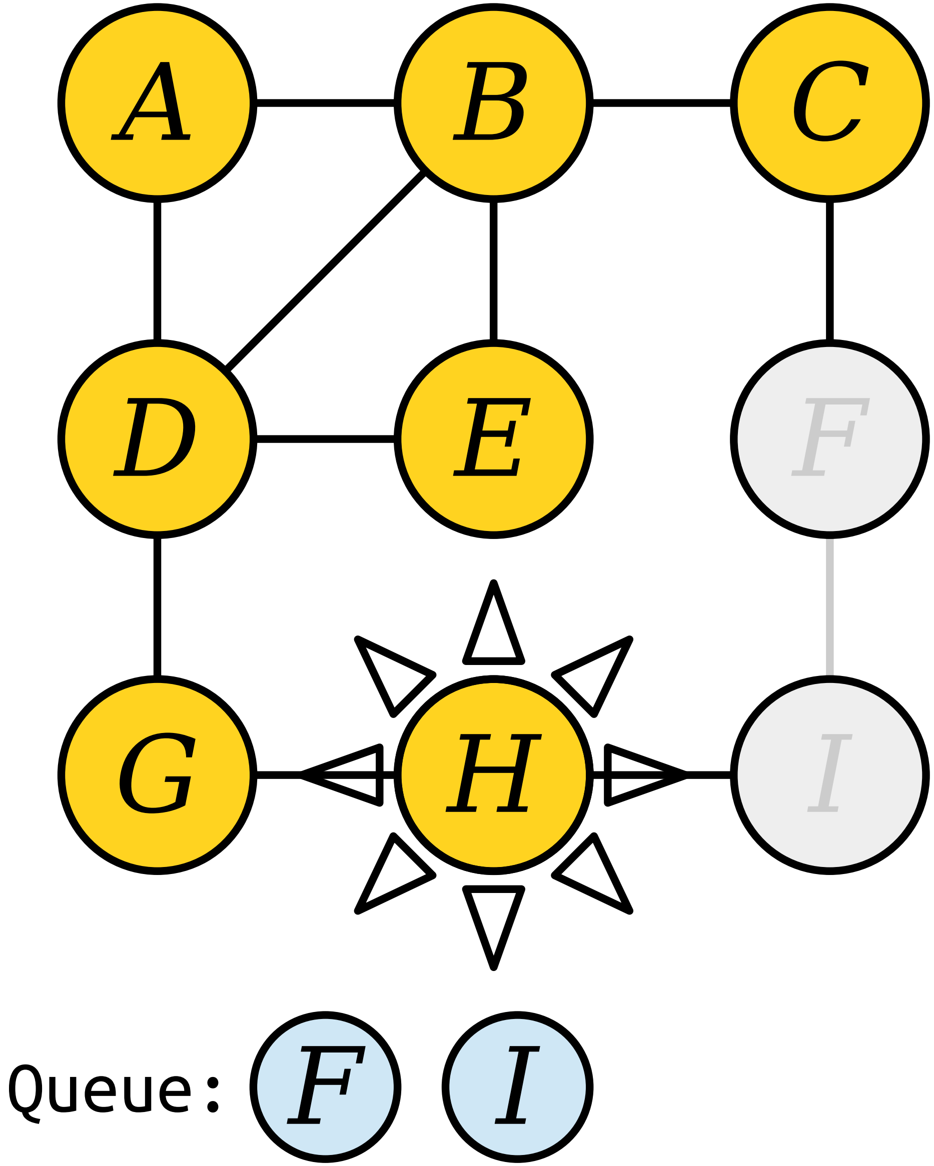
Visit nodes in ascending order of distance from the start node E.



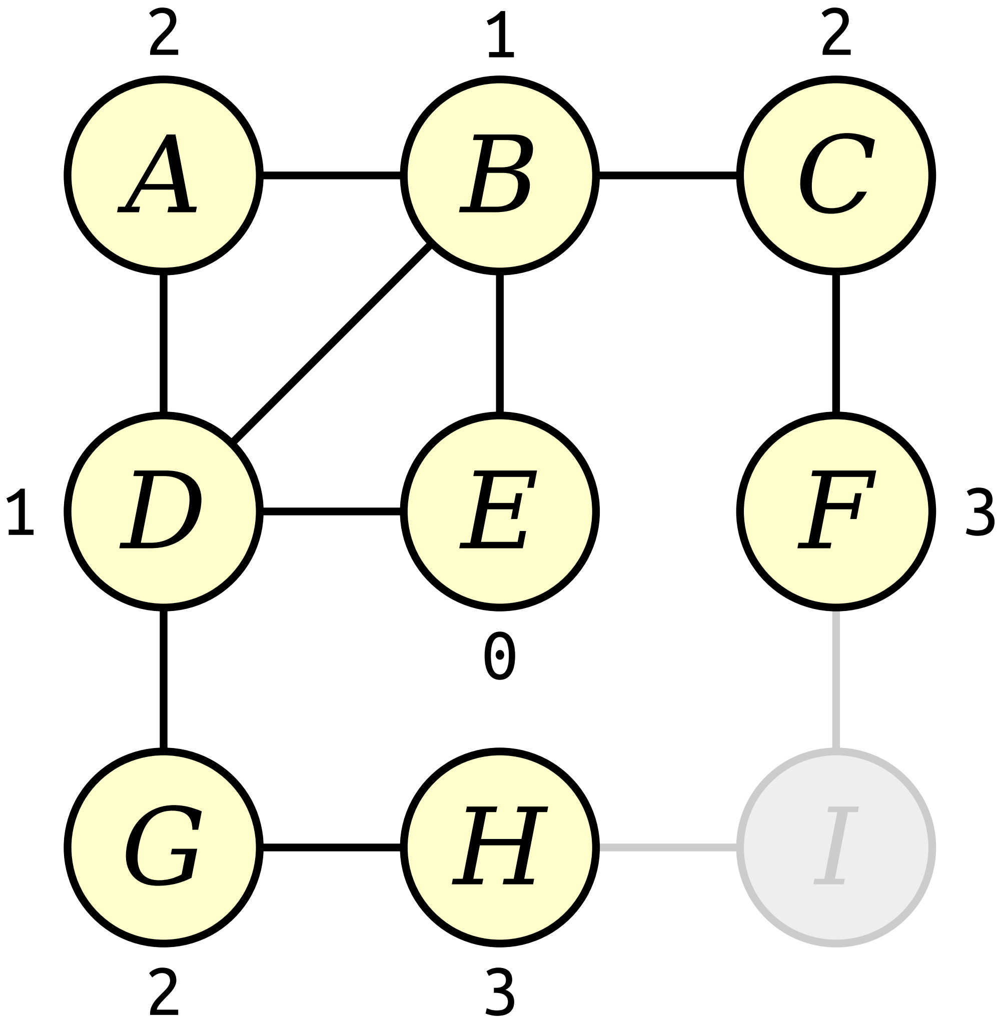
Load newly-discovered nodes into a queue.



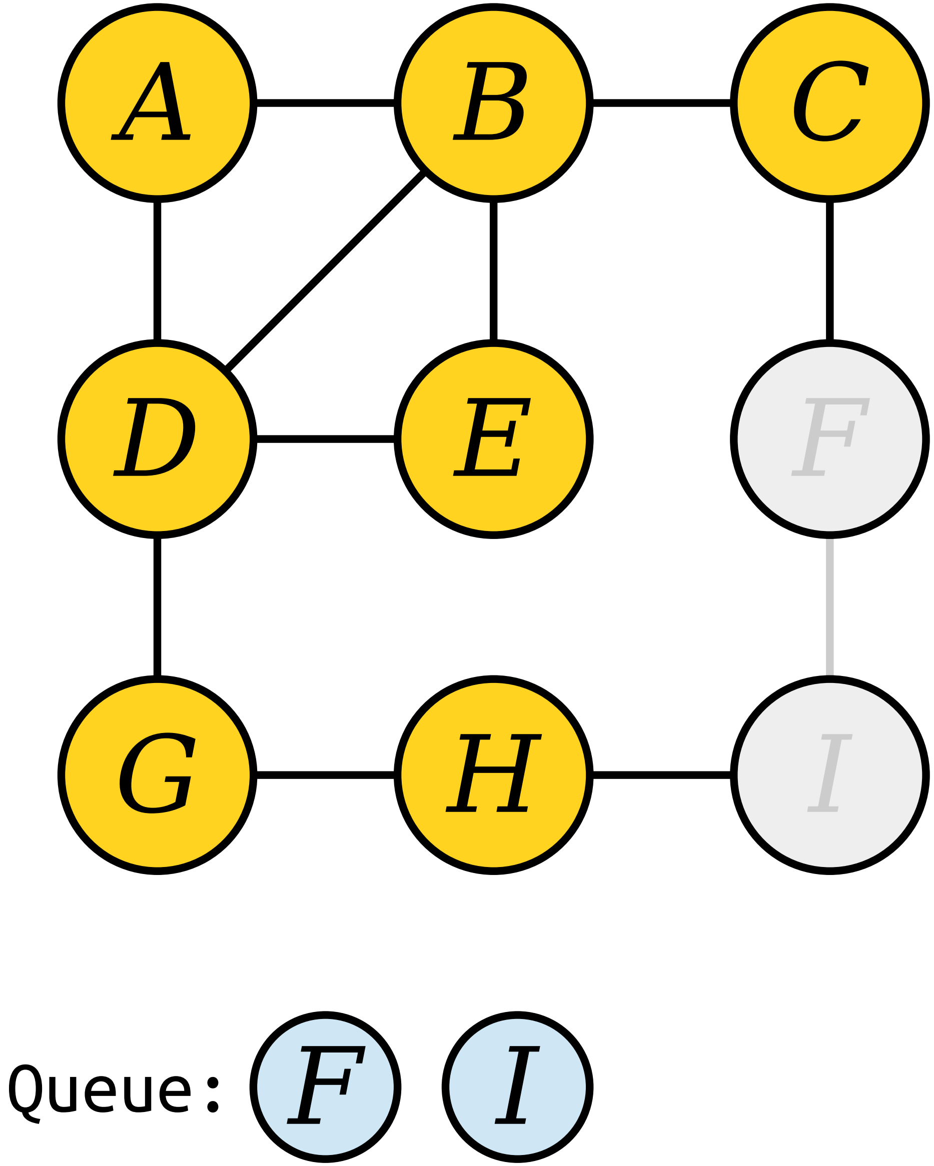
Visit nodes in ascending order of distance from the start node E.



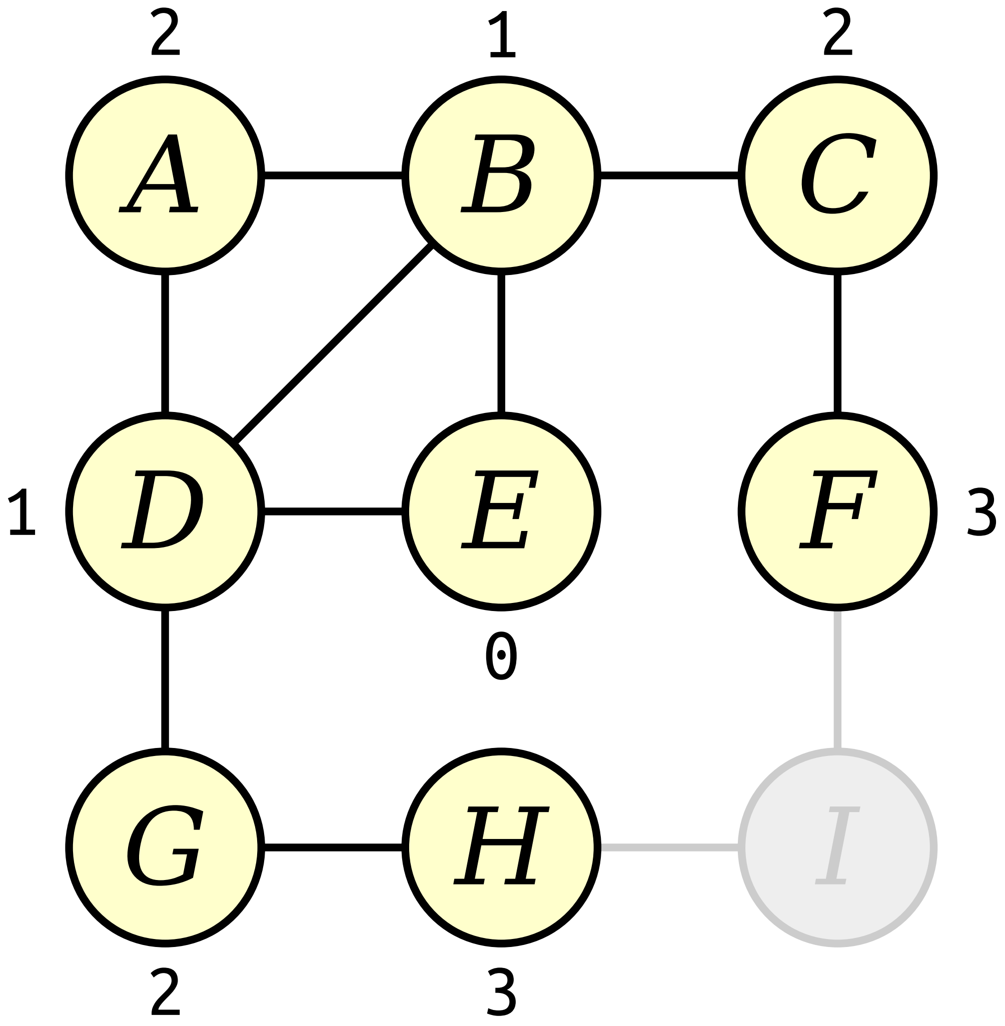
Load newly-discovered nodes into a queue.



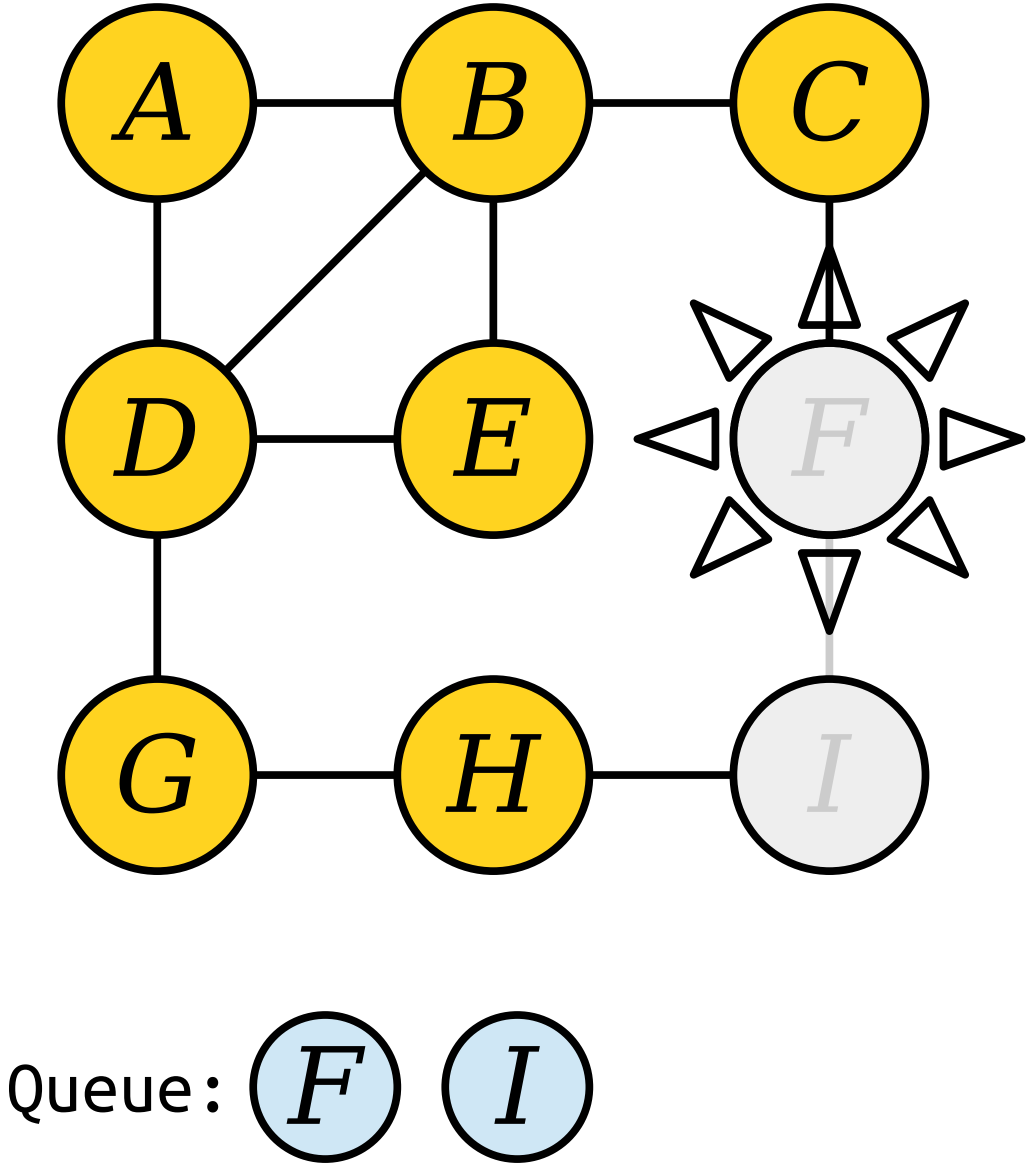
Visit nodes in ascending order of distance from the start node *E*.



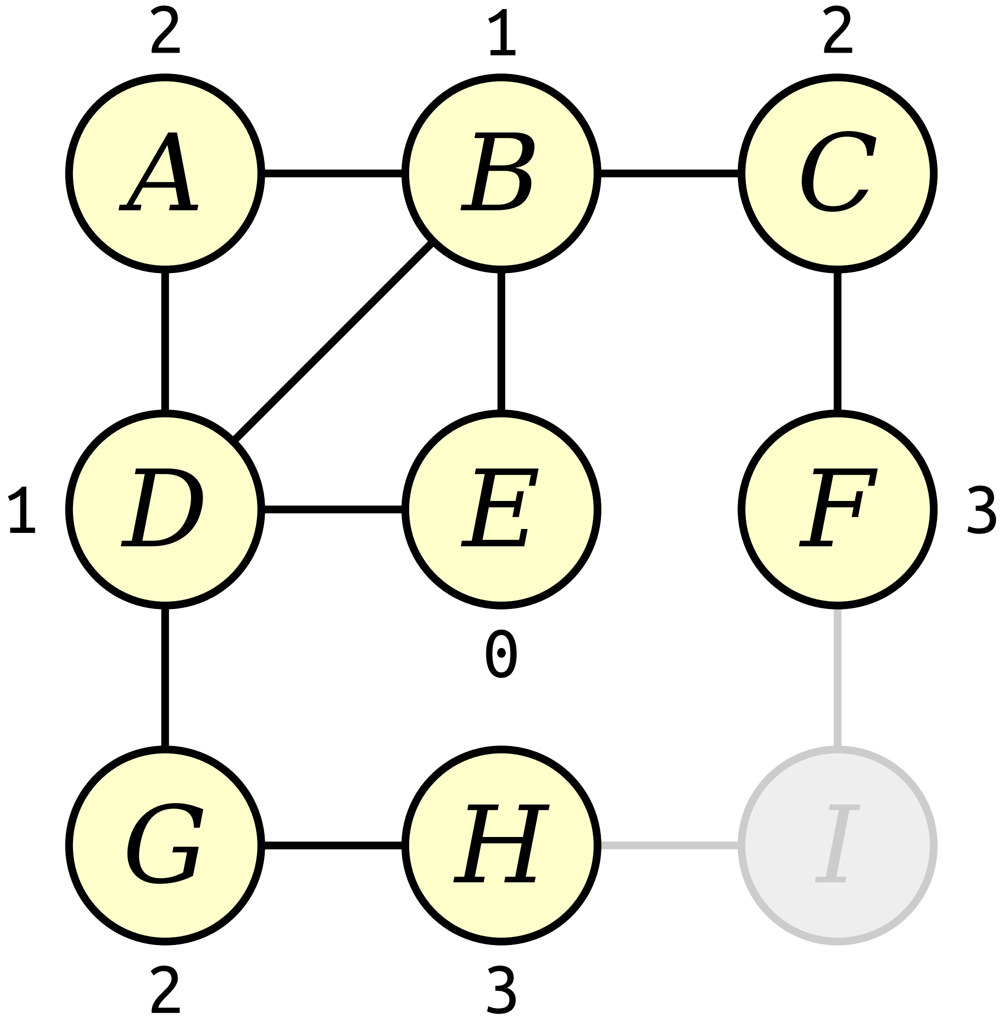
Load newly-discovered nodes into a queue.



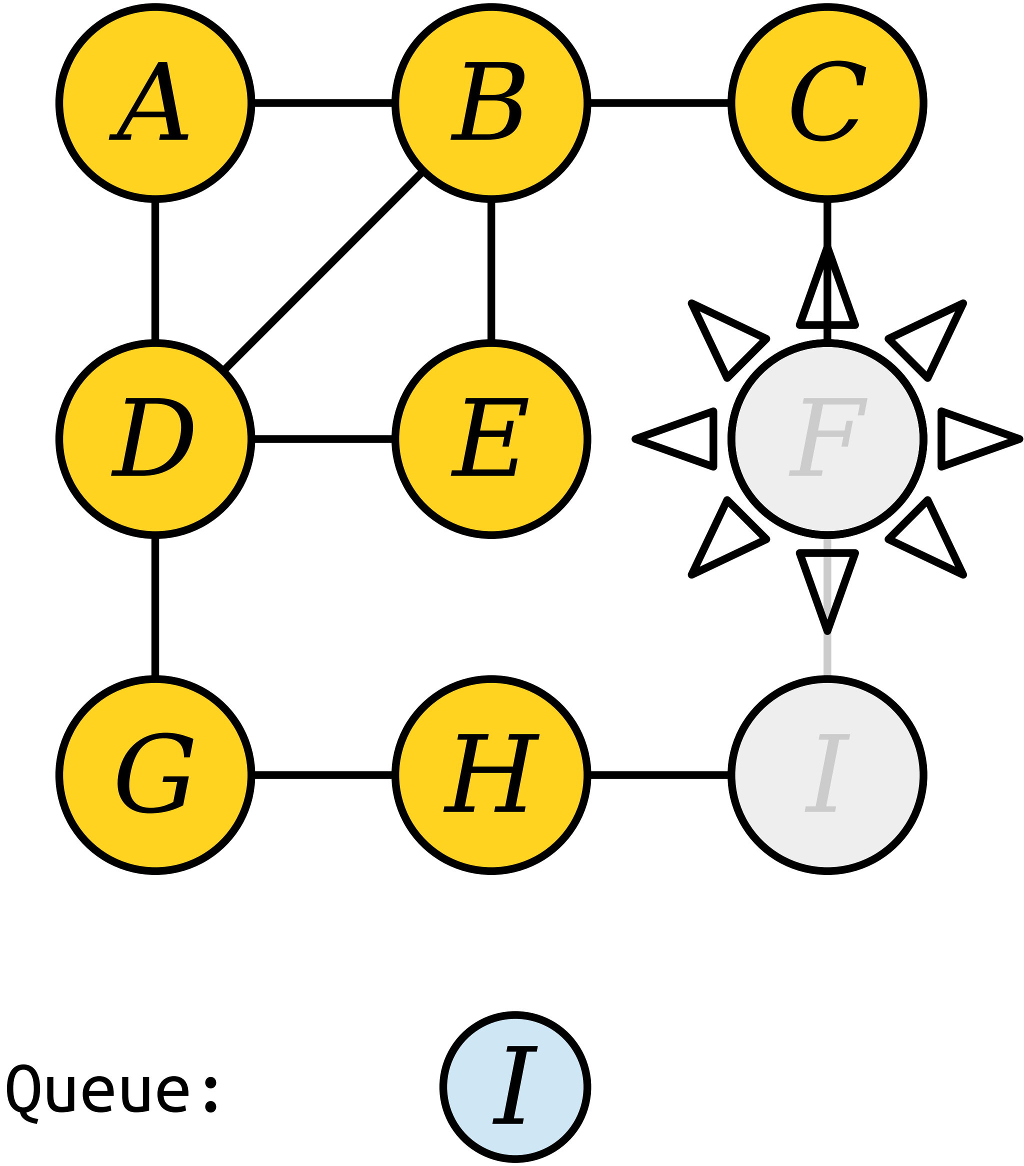
Visit nodes in ascending order of distance from the start node *E*.



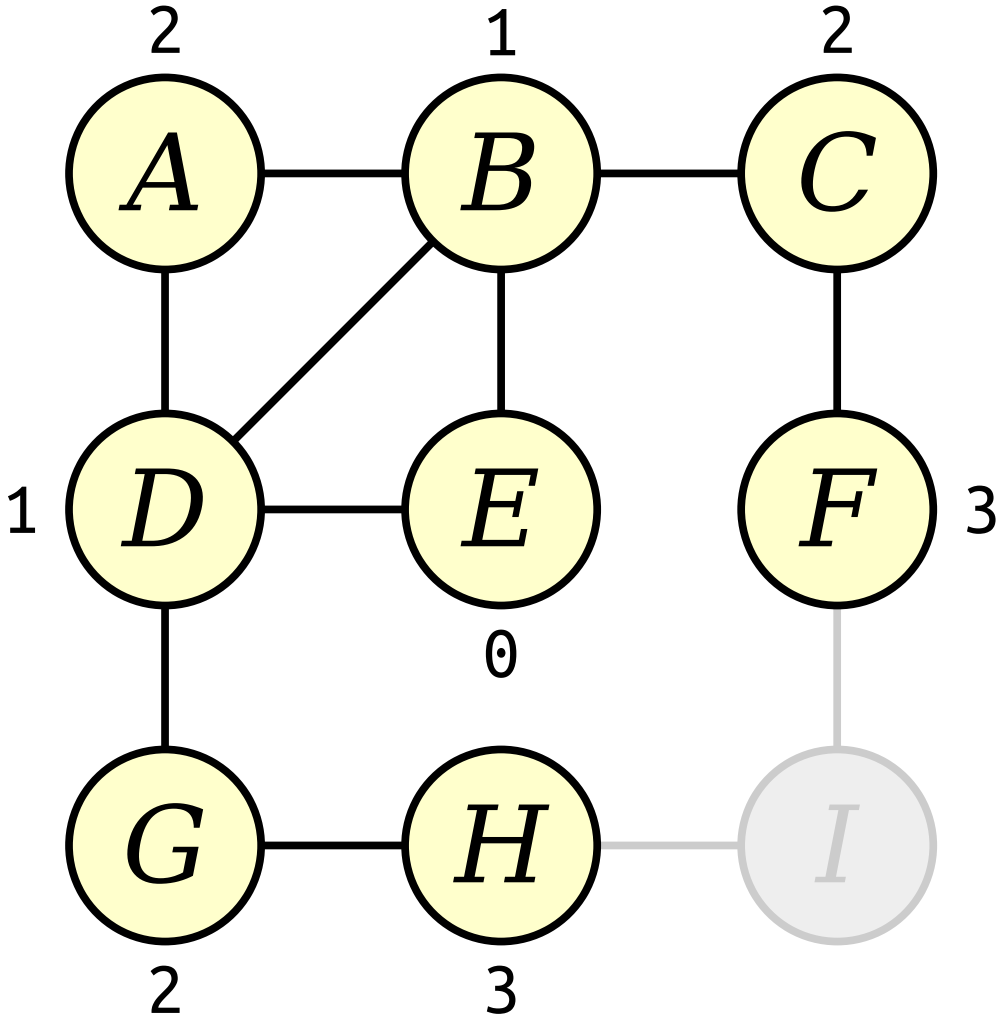
Load newly-discovered nodes into a queue.



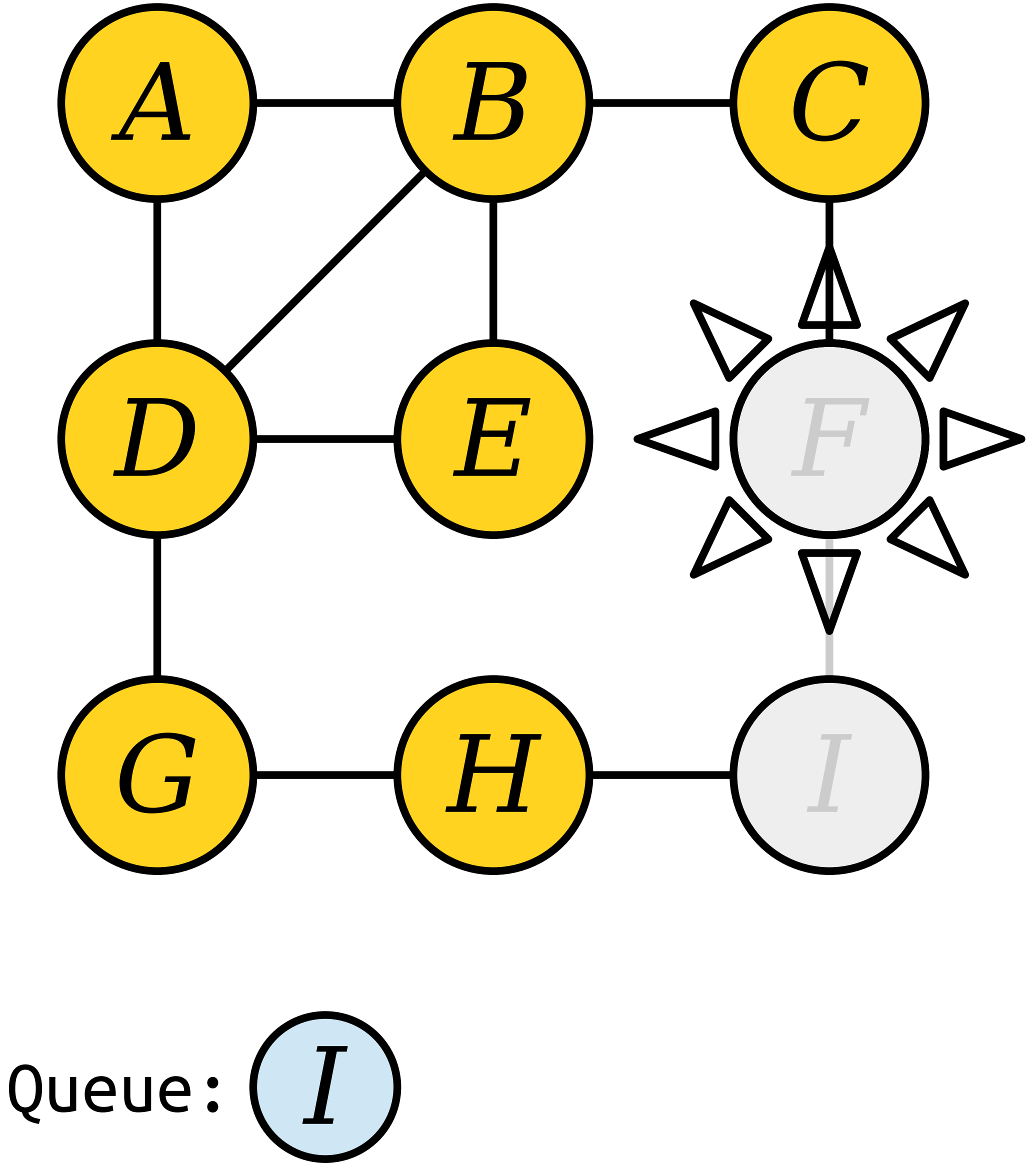
Visit nodes in ascending order of distance from the start node *E*.



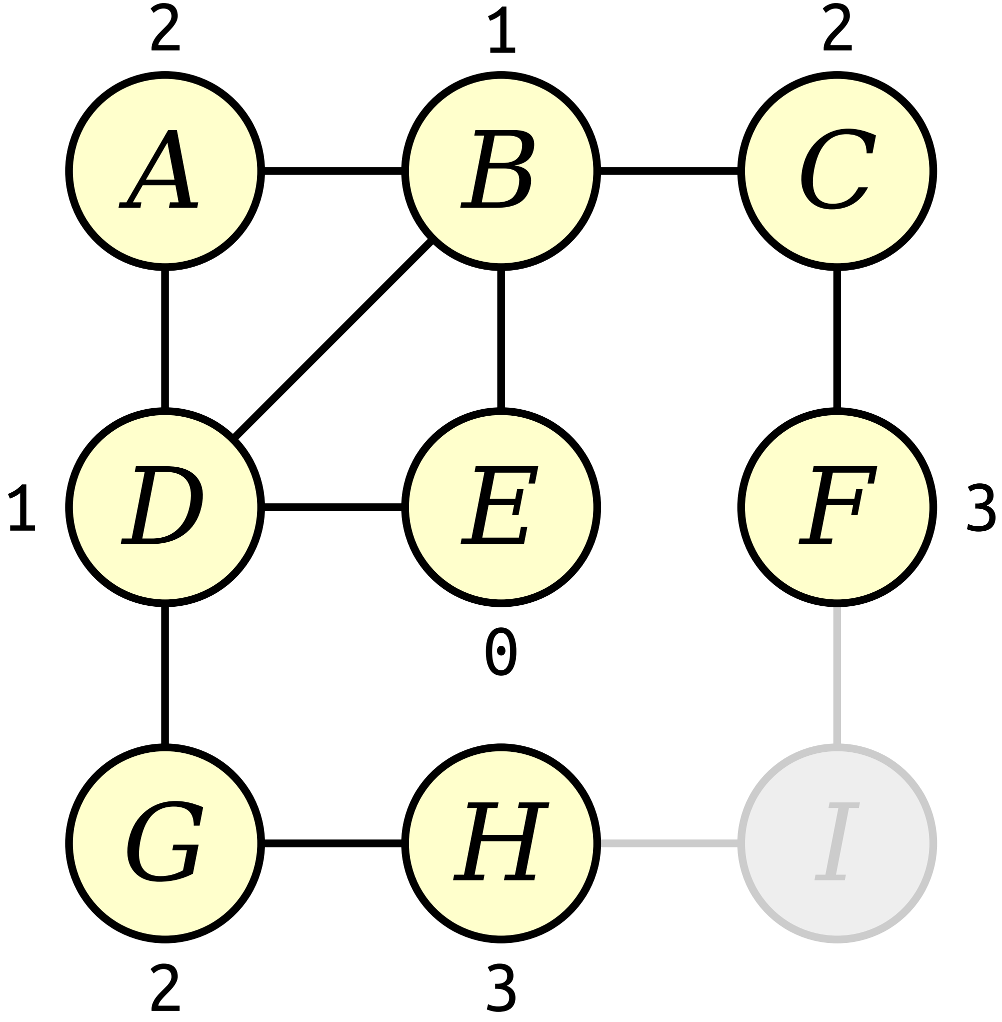
Load newly-discovered nodes into a queue.



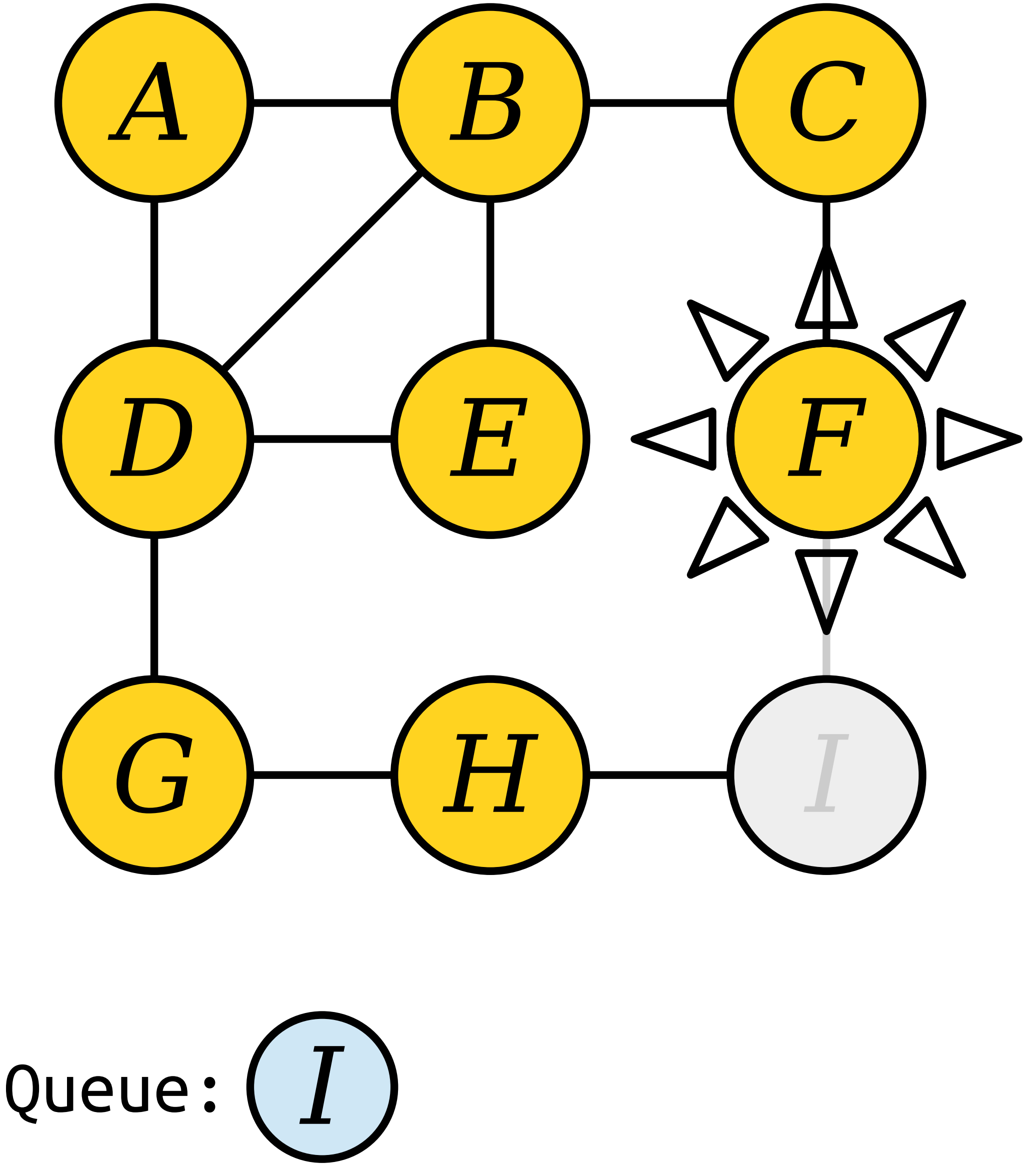
Visit nodes in ascending order of distance from the start node *E*.



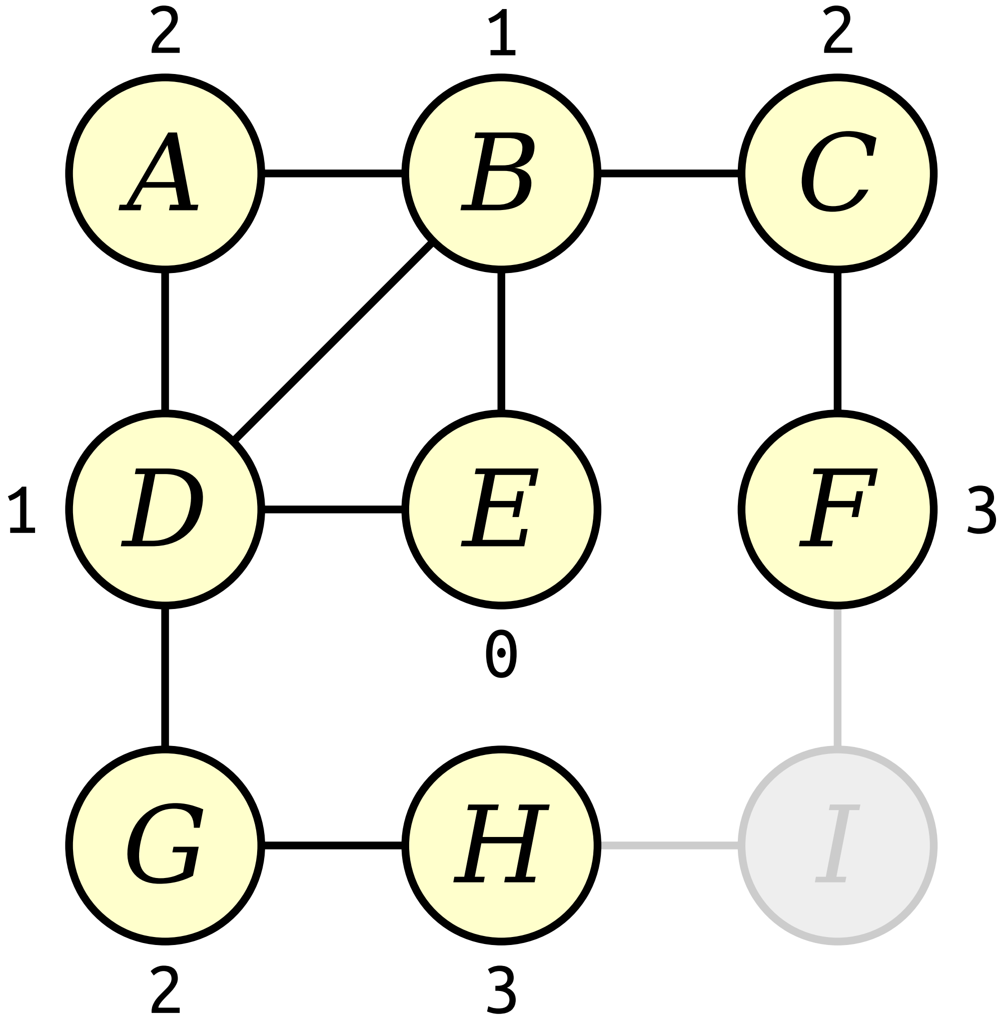
Load newly-discovered nodes into a queue.



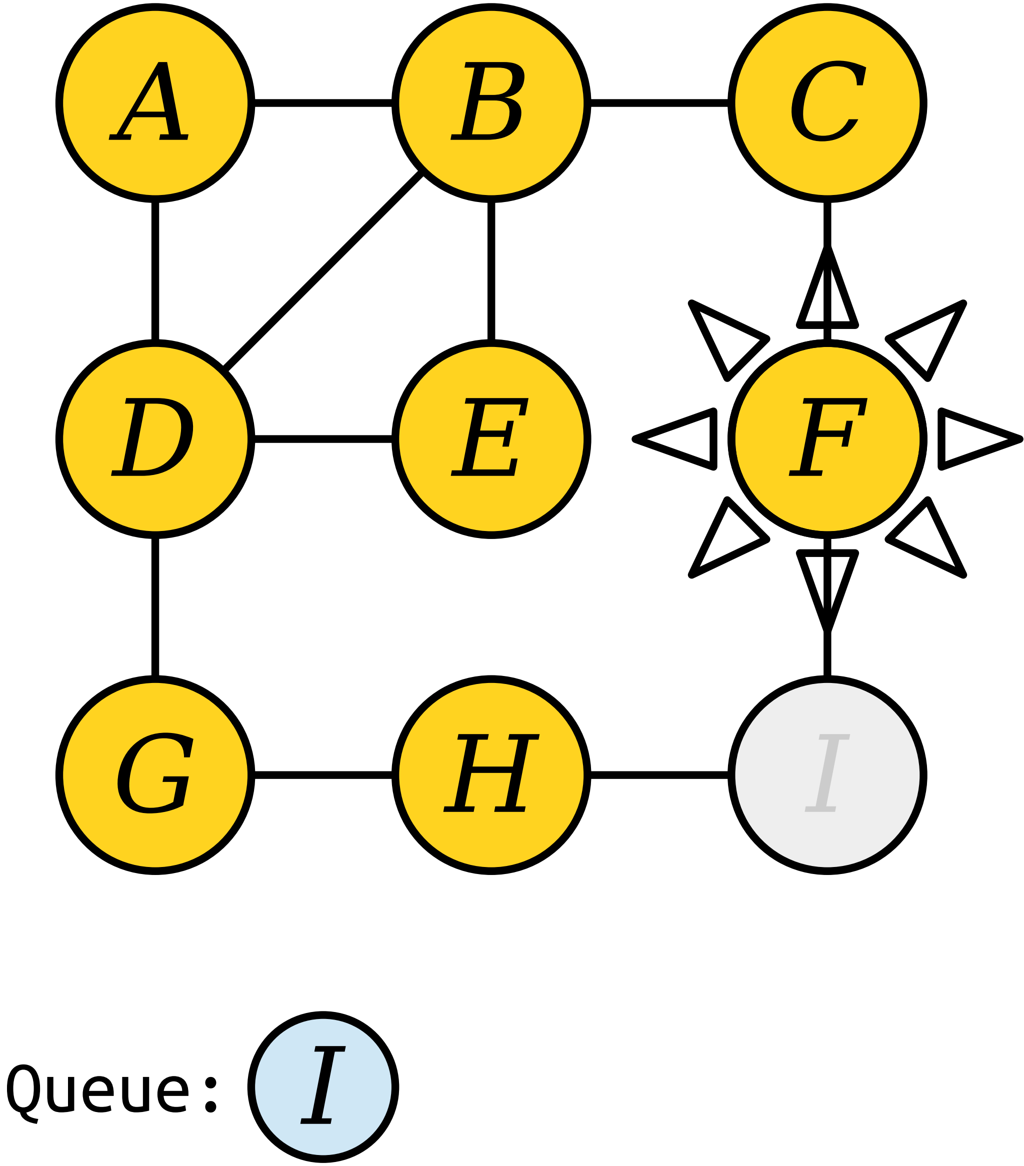
Visit nodes in ascending order of distance from the start node *E*.



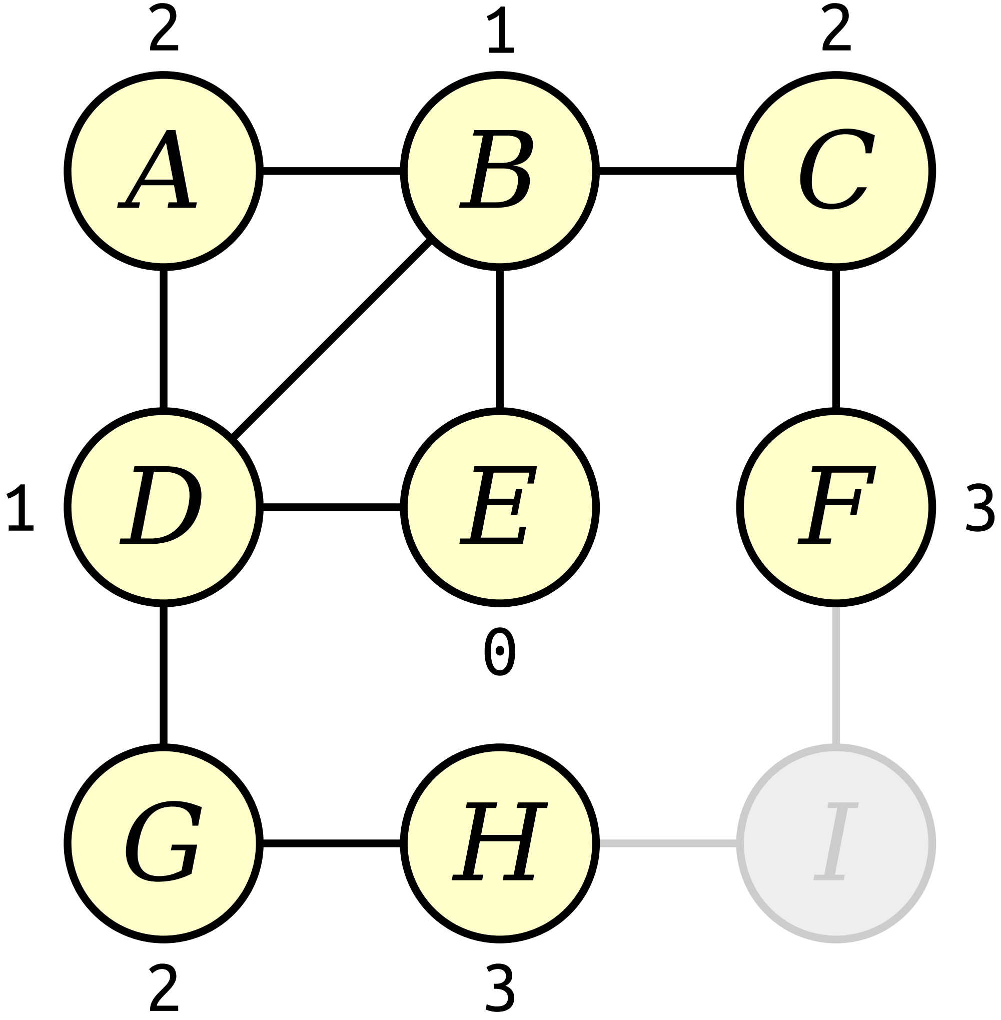
Load newly-discovered nodes into a queue.



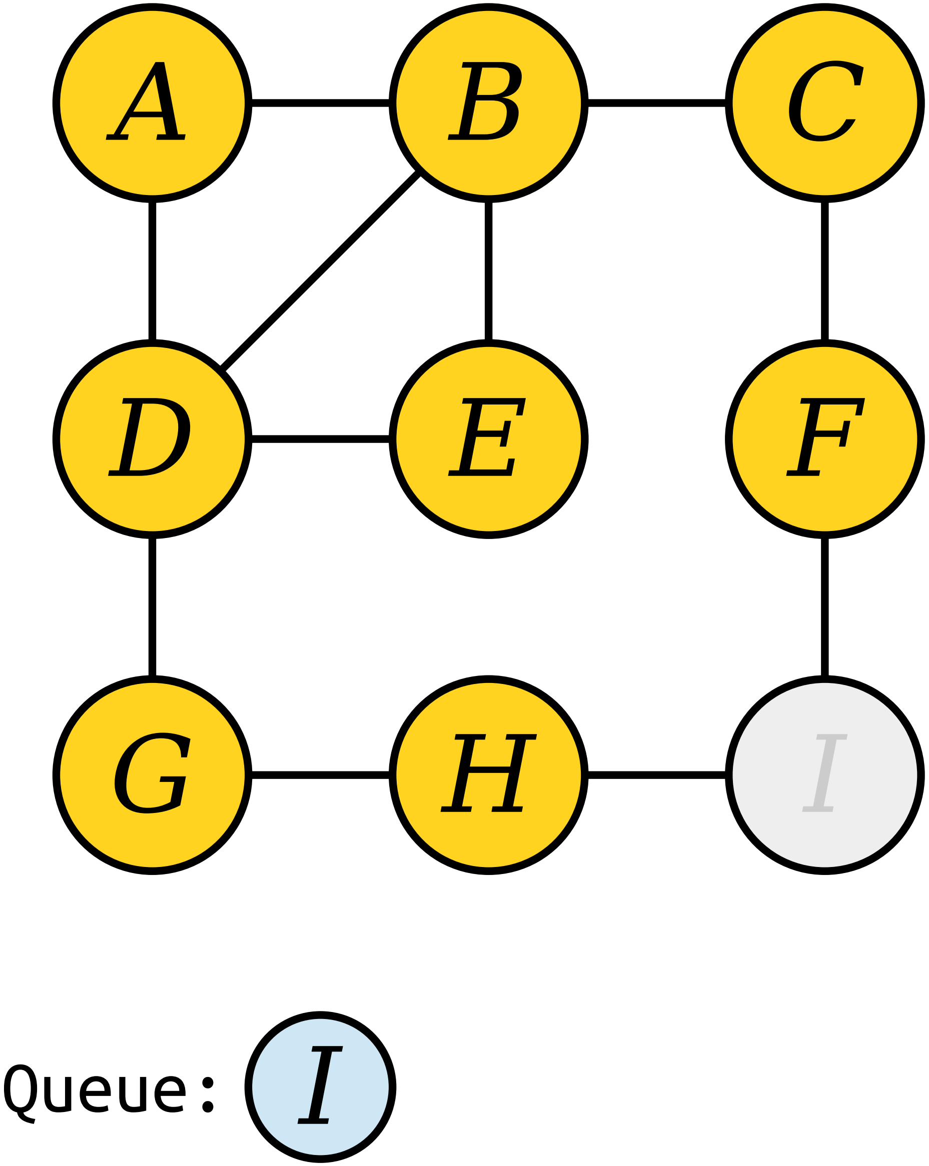
Visit nodes in ascending order of distance from the start node E.



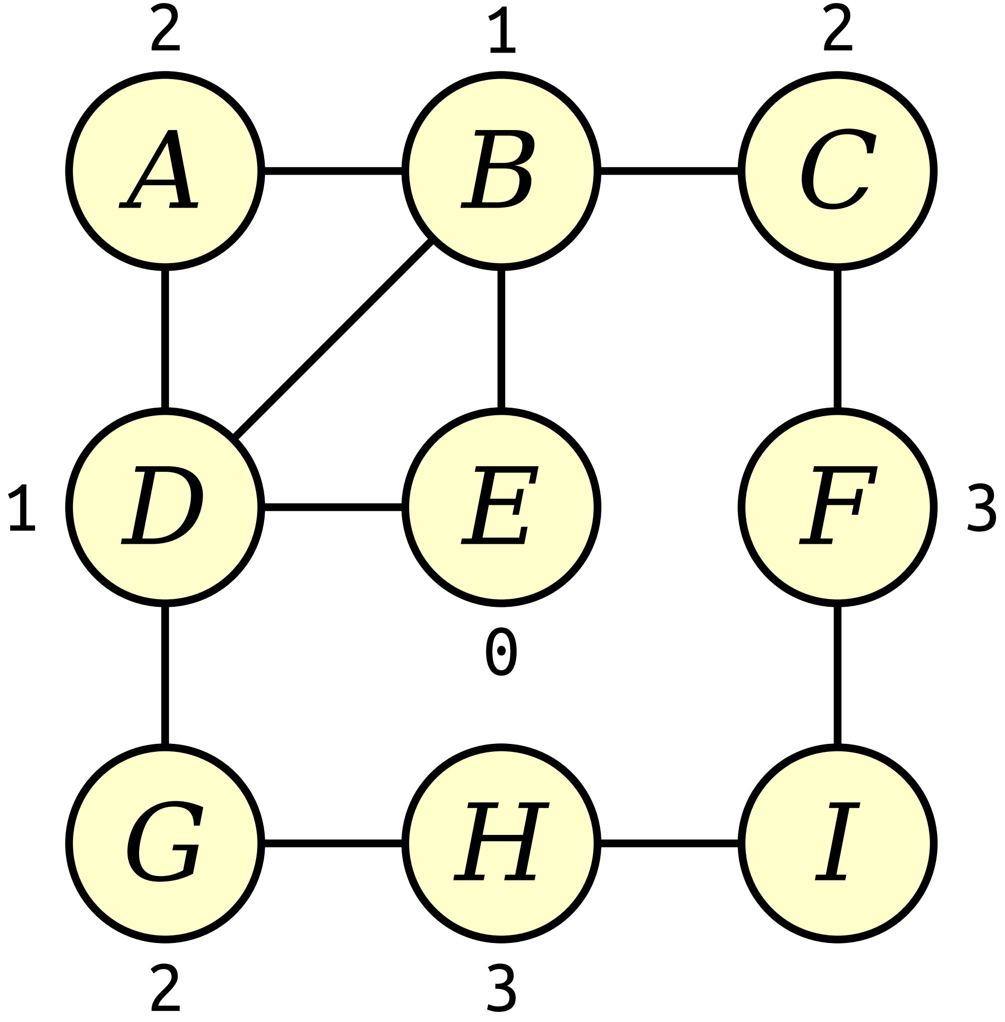
Load newly-discovered nodes into a queue.



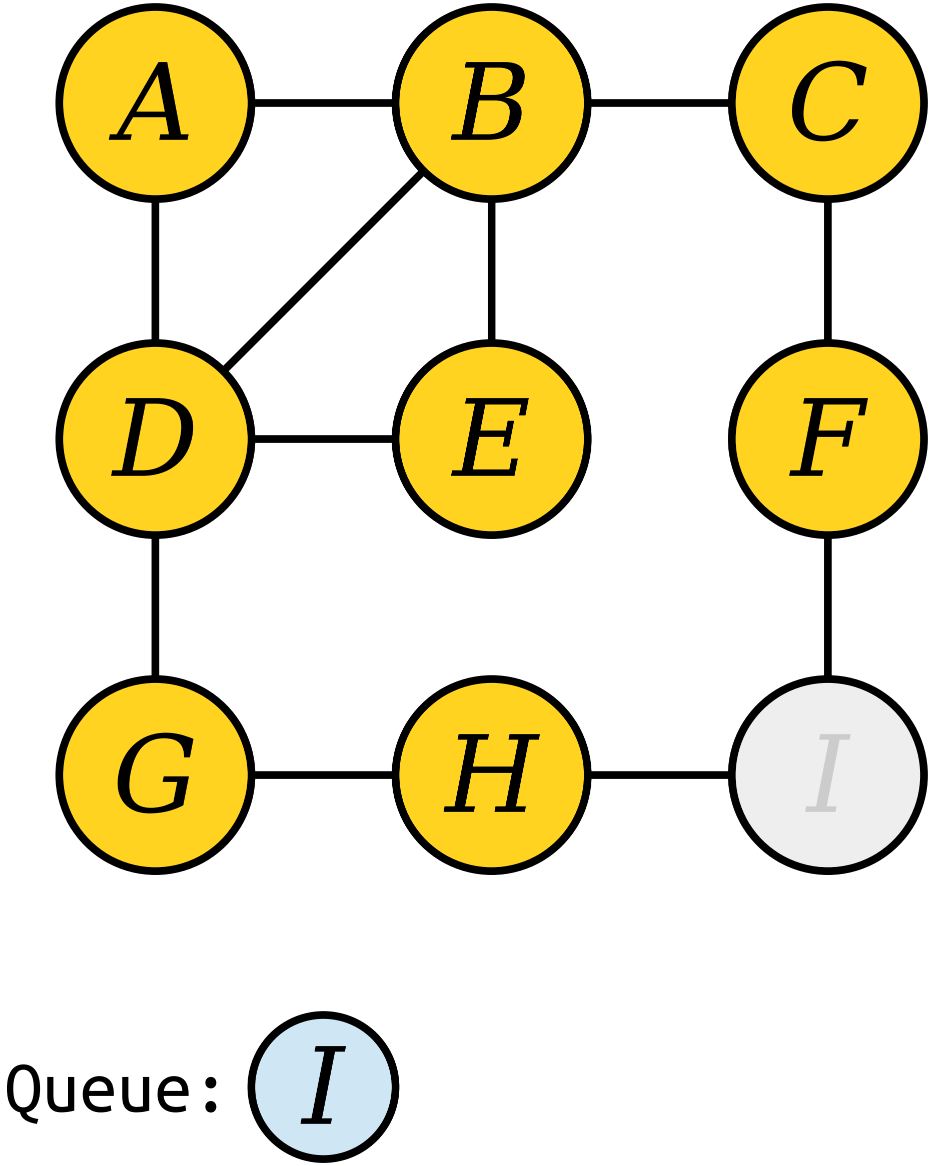
Visit nodes in ascending order of distance from the start node *E*.



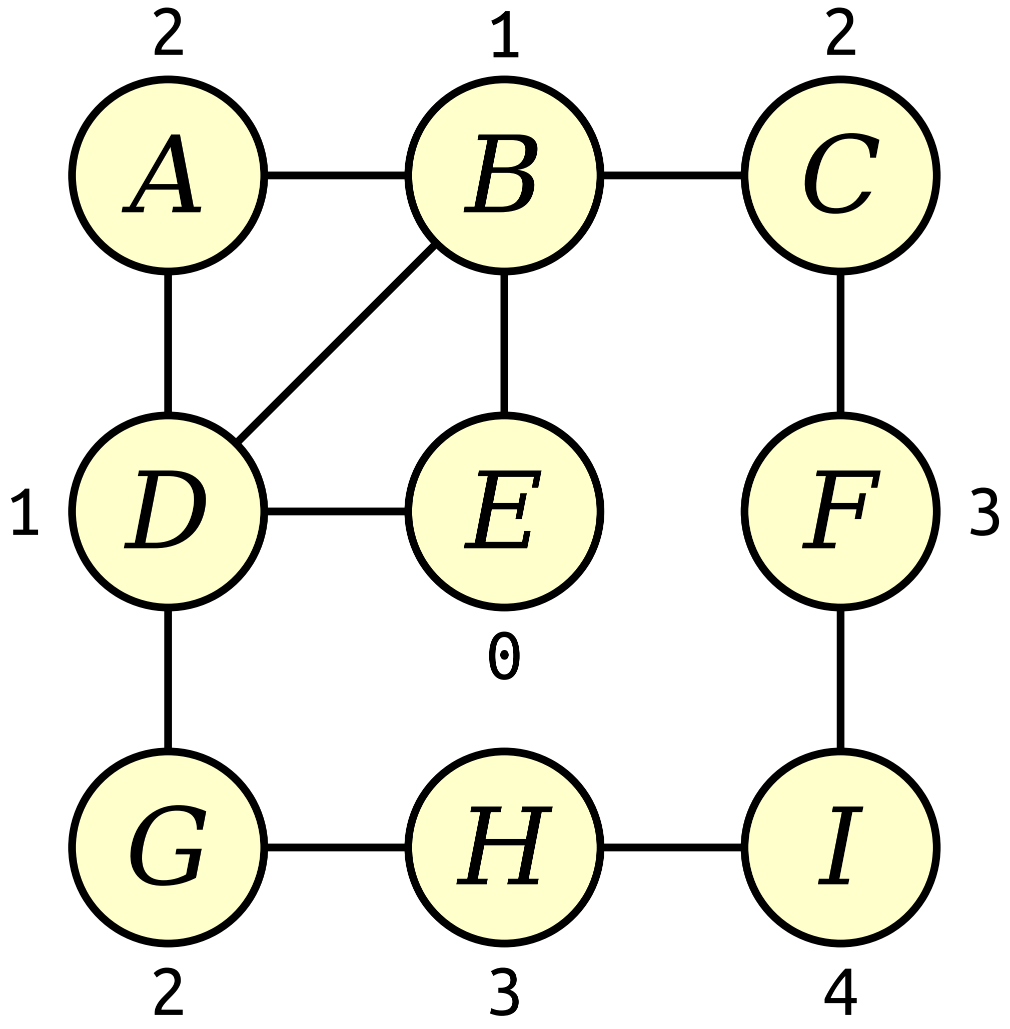
Load newly-discovered nodes into a queue.



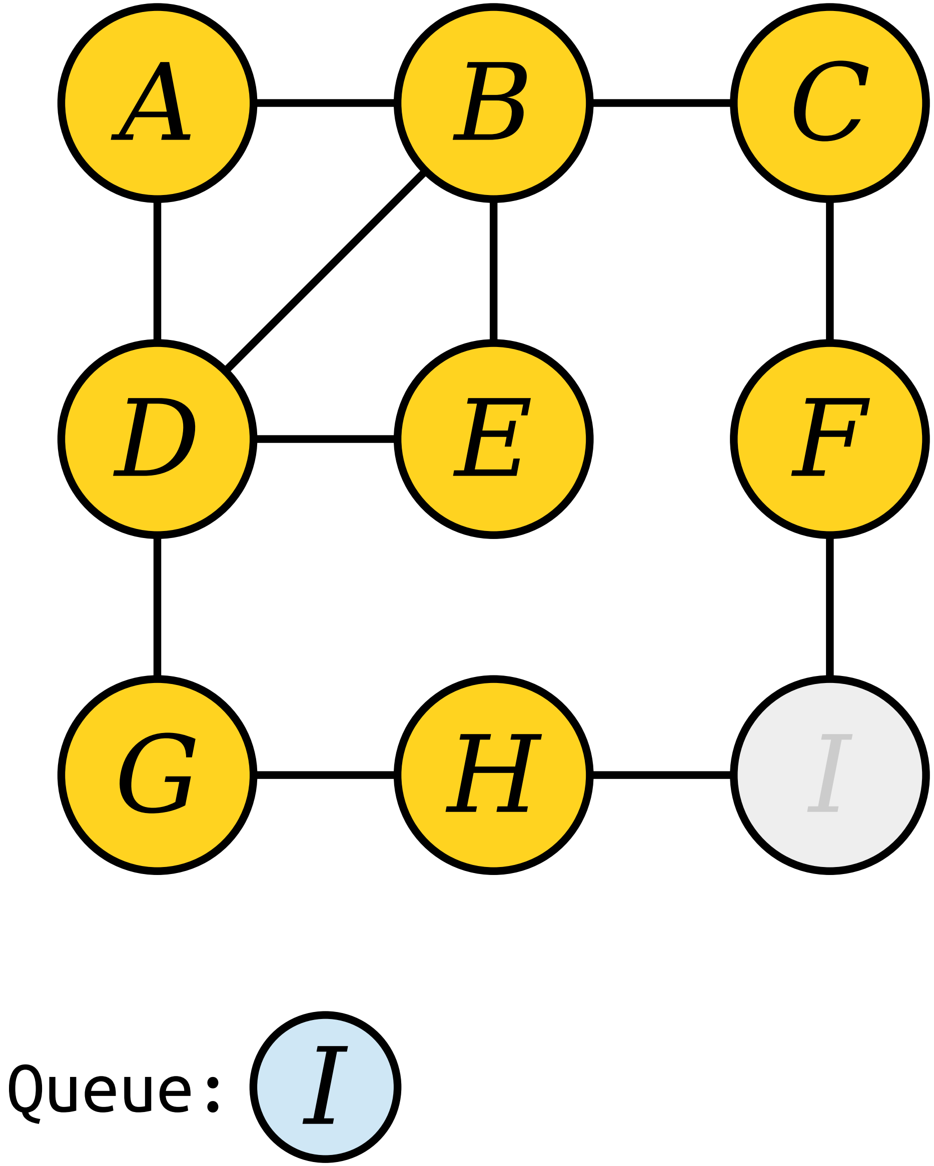
Visit nodes in ascending order of distance from the start node *E*.



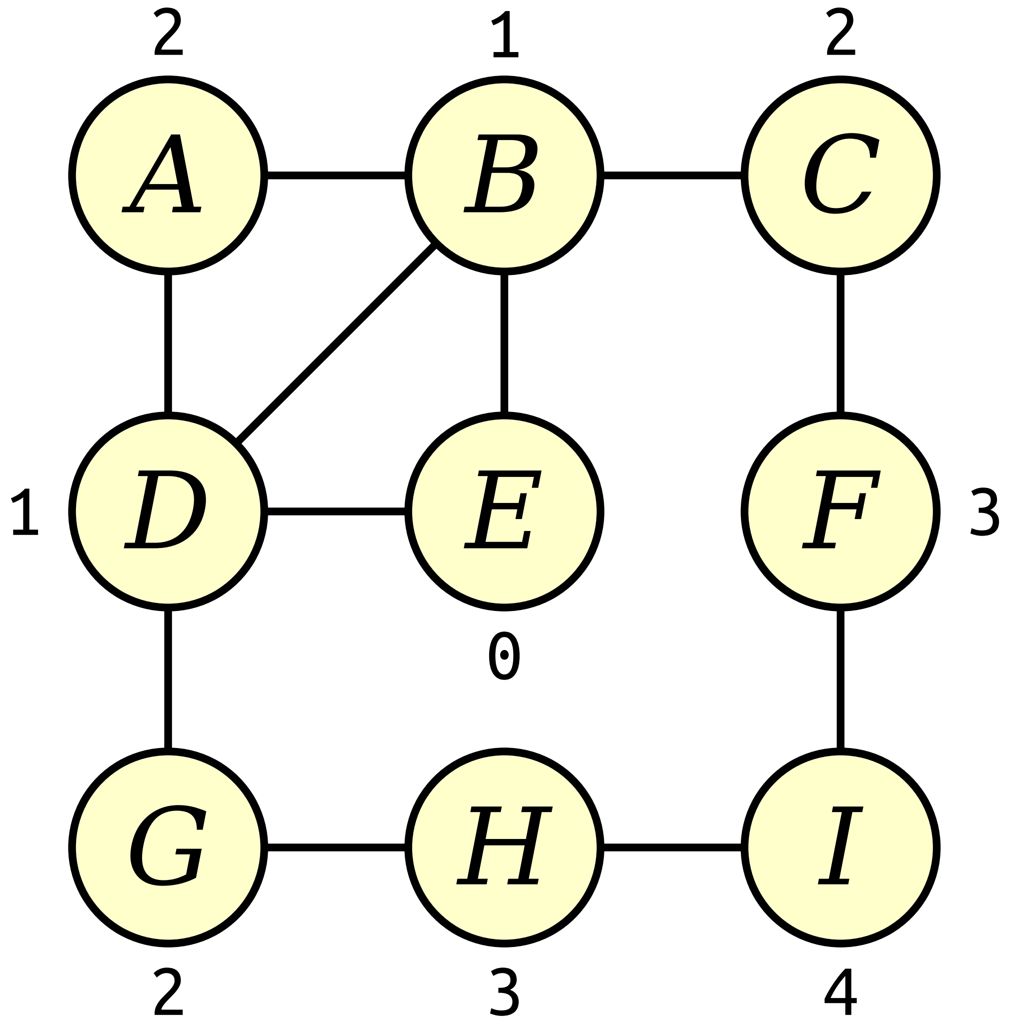
Load newly-discovered nodes into a queue.



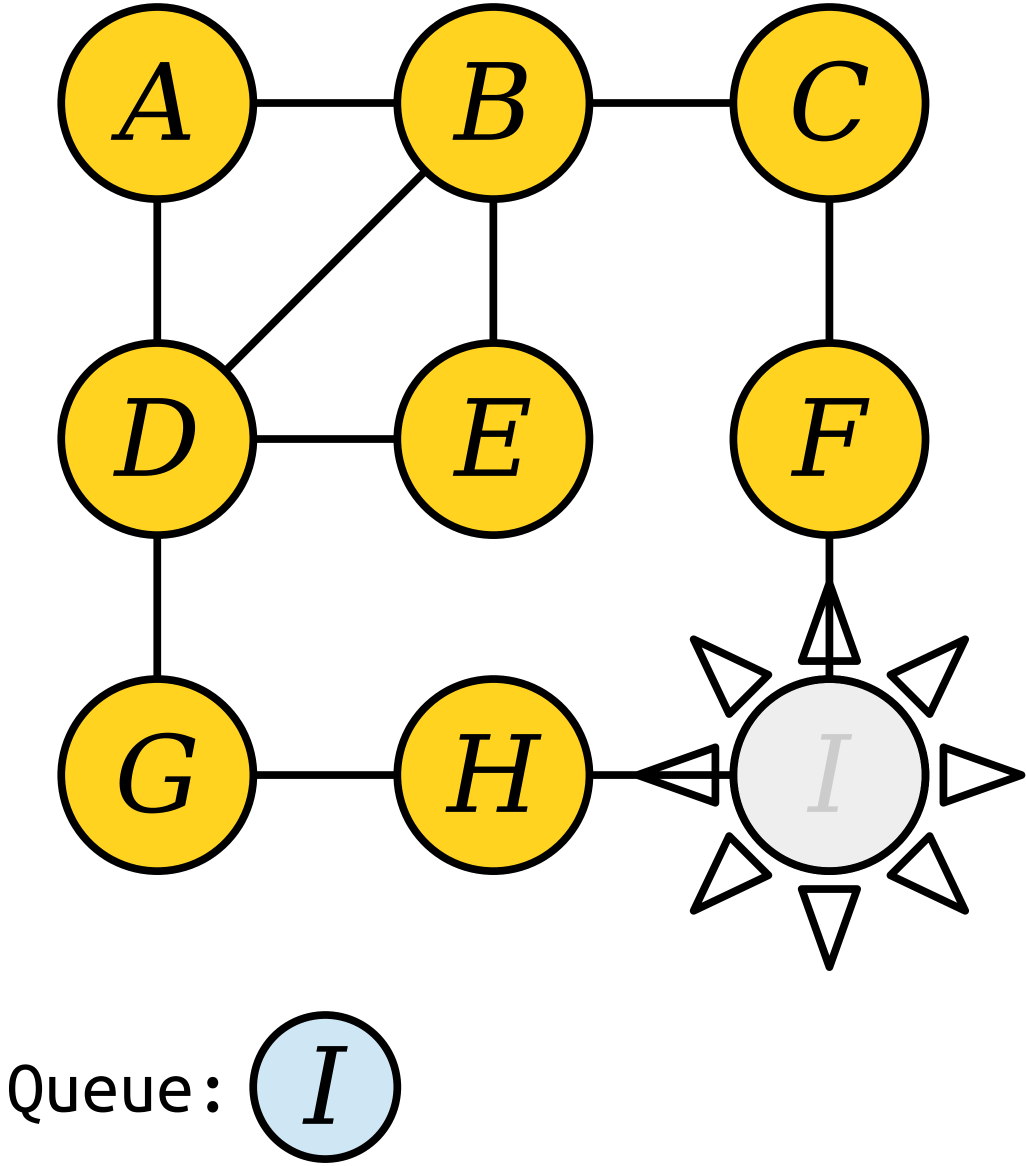
Visit nodes in ascending order of distance from the start node E.



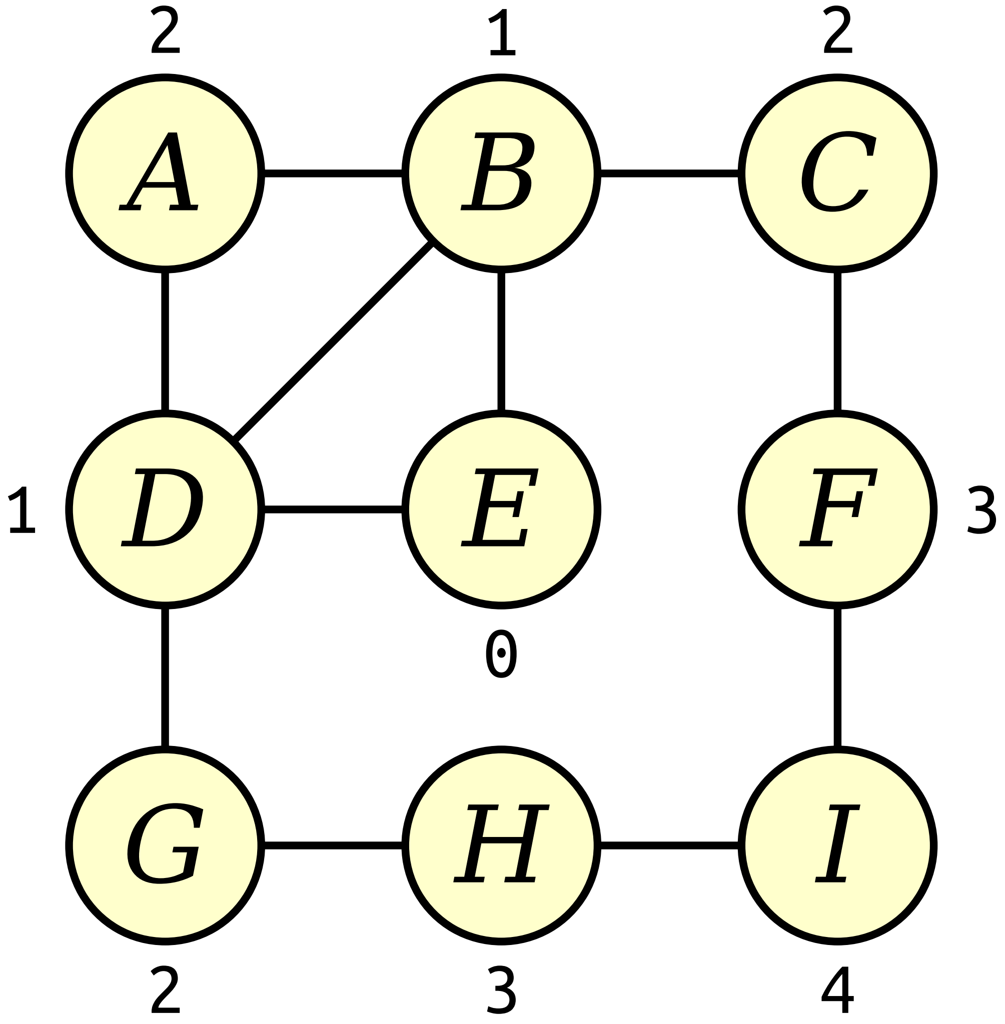
Load newly-discovered nodes into a queue.



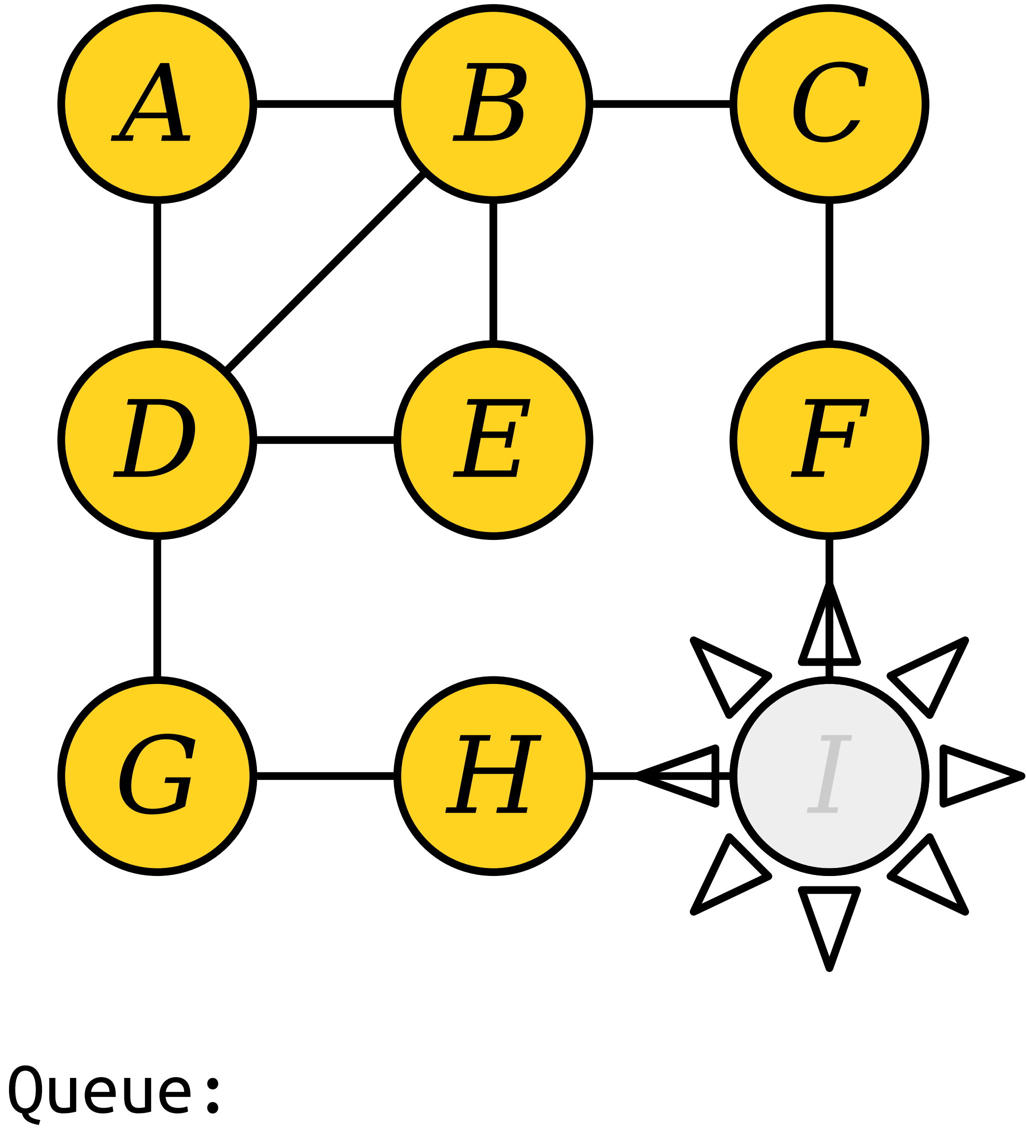
Visit nodes in ascending order of distance from the start node *E*.



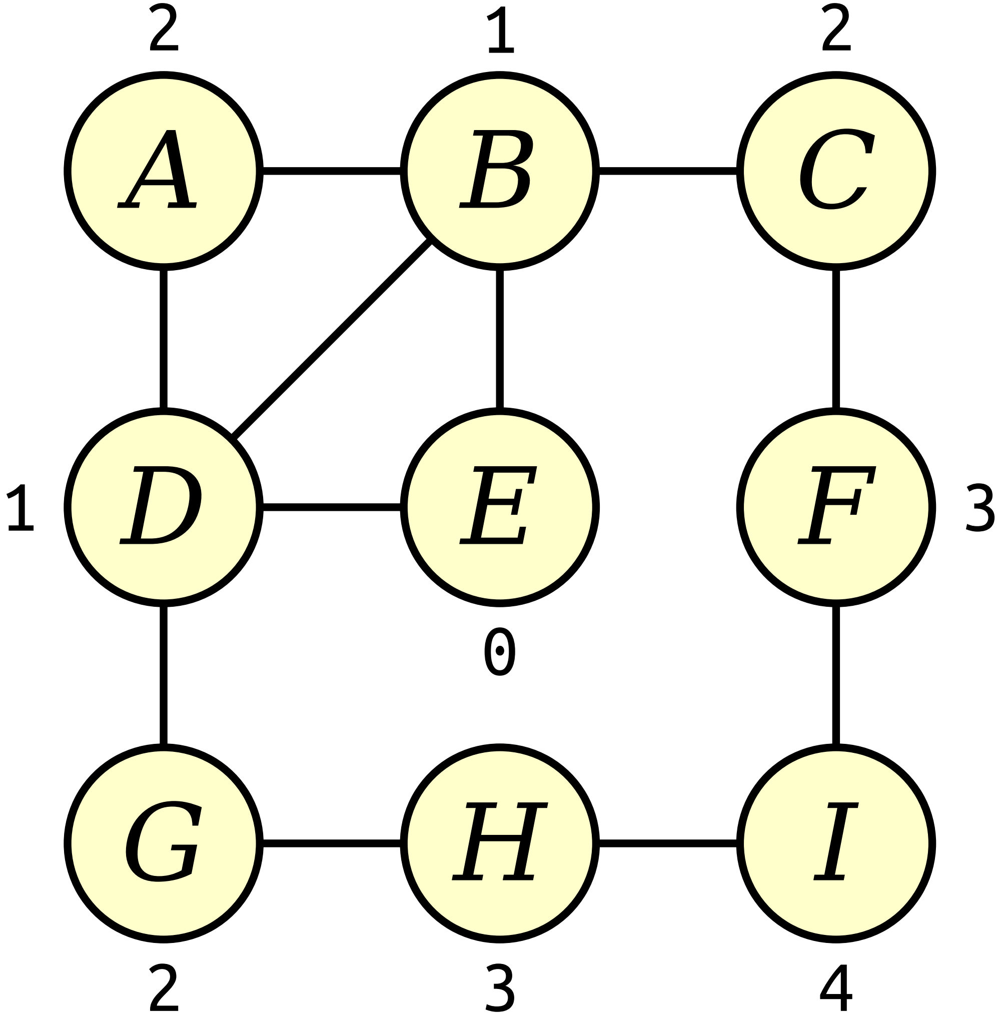
Load newly-discovered nodes into a queue.



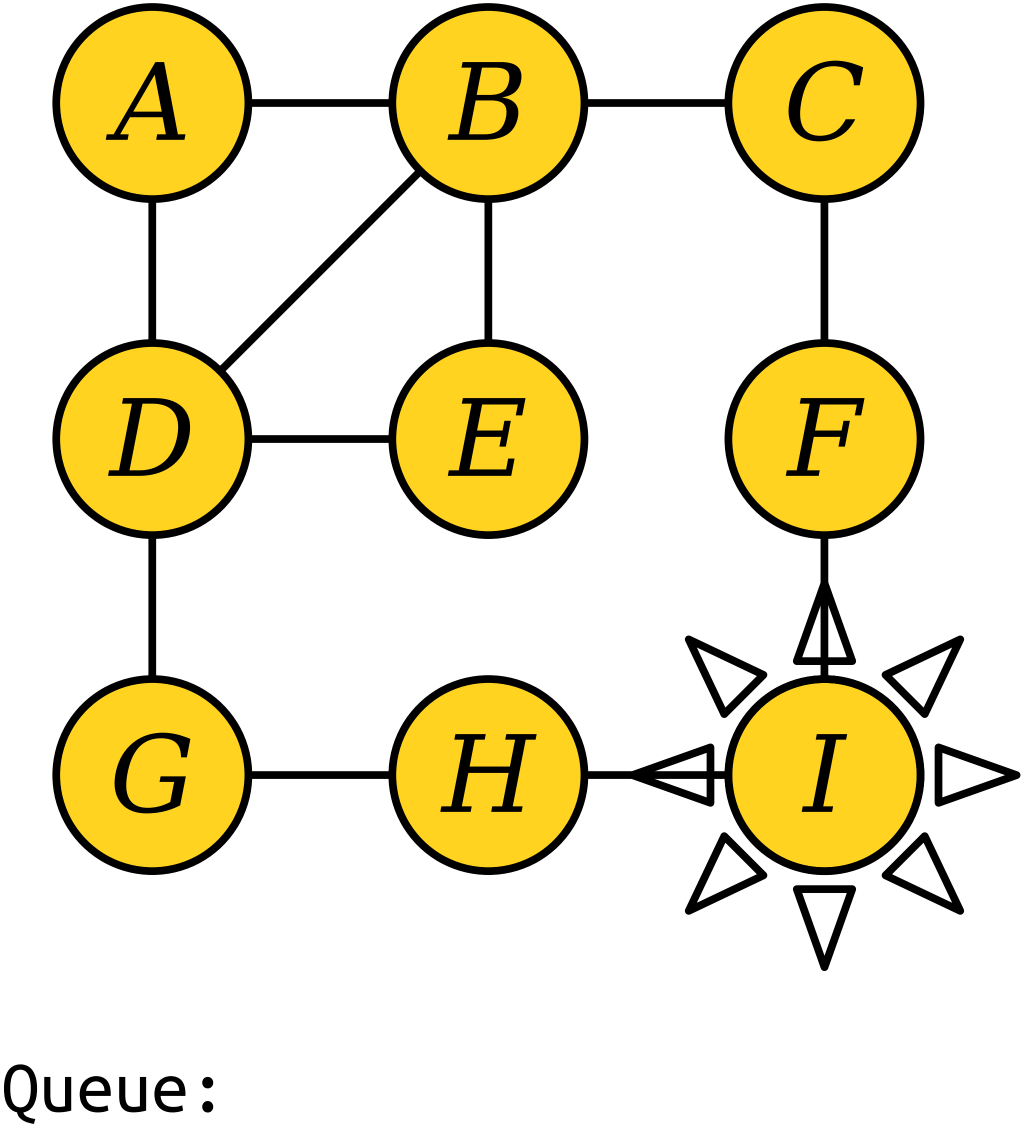
Visit nodes in ascending order of distance from the start node *E*.



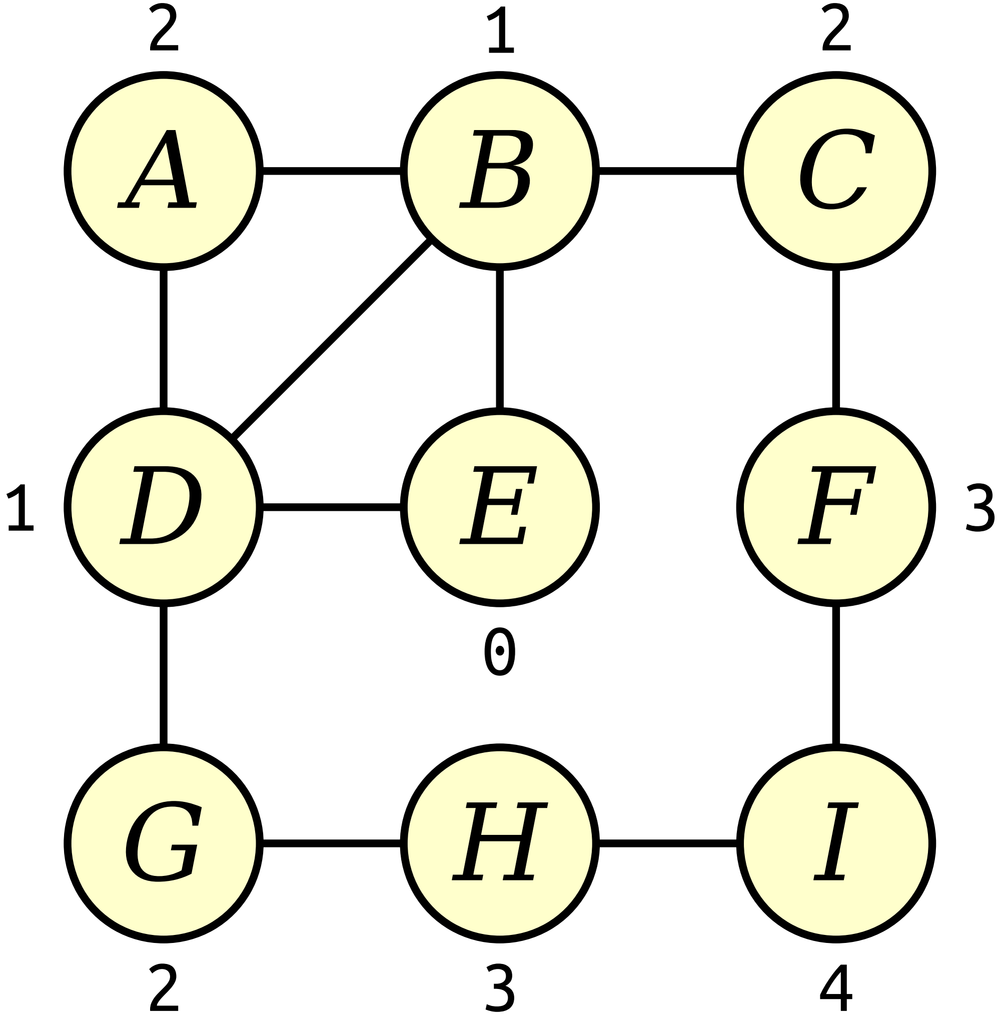
Load newly-discovered nodes into a queue.



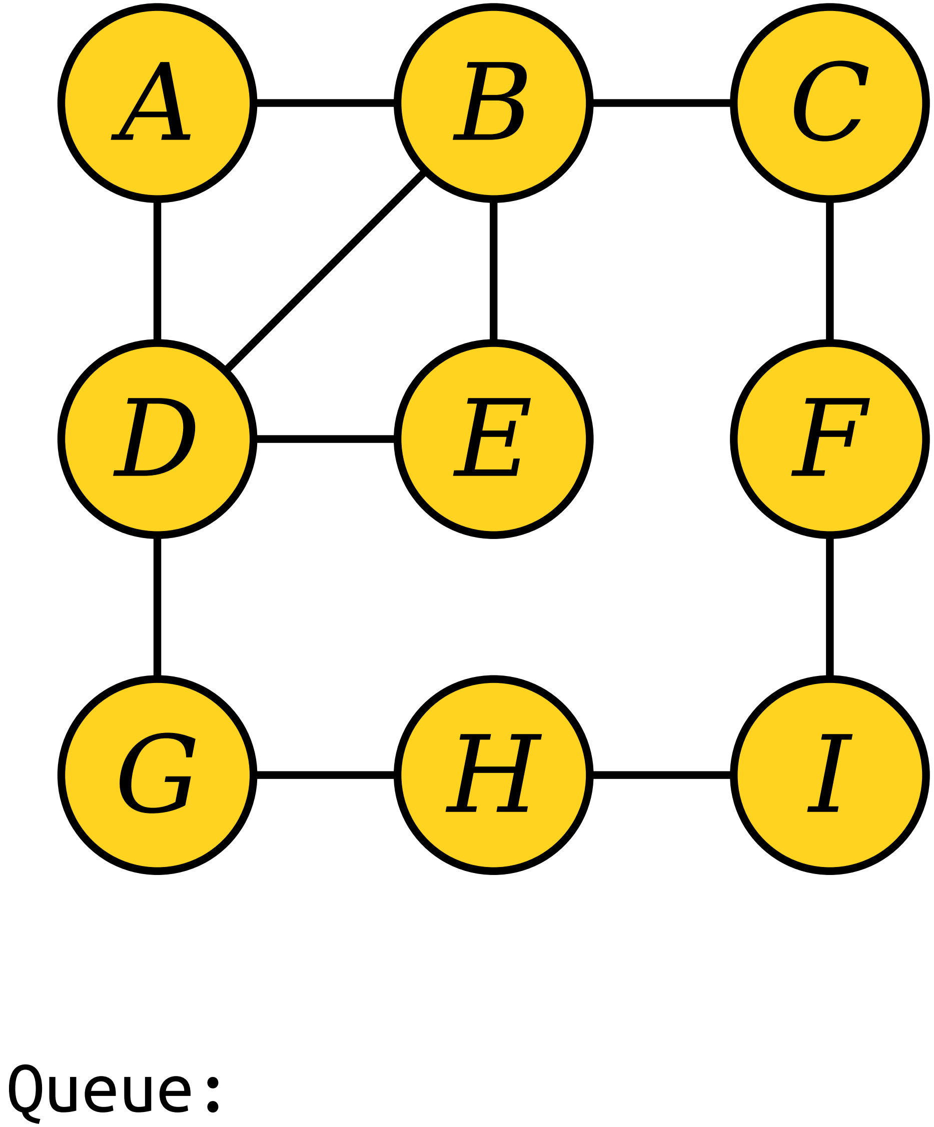
Visit nodes in ascending order of distance from the start node *E*.



Load newly-discovered nodes into a queue.



Visit nodes in ascending order of distance from the start node *E*.



Load newly-discovered nodes into a queue.

Breadth-First Search

- The `Queue`-based search strategy we just saw is called ***breadth-first search*** (or just ***BFS*** for short).
- In pseudocode:

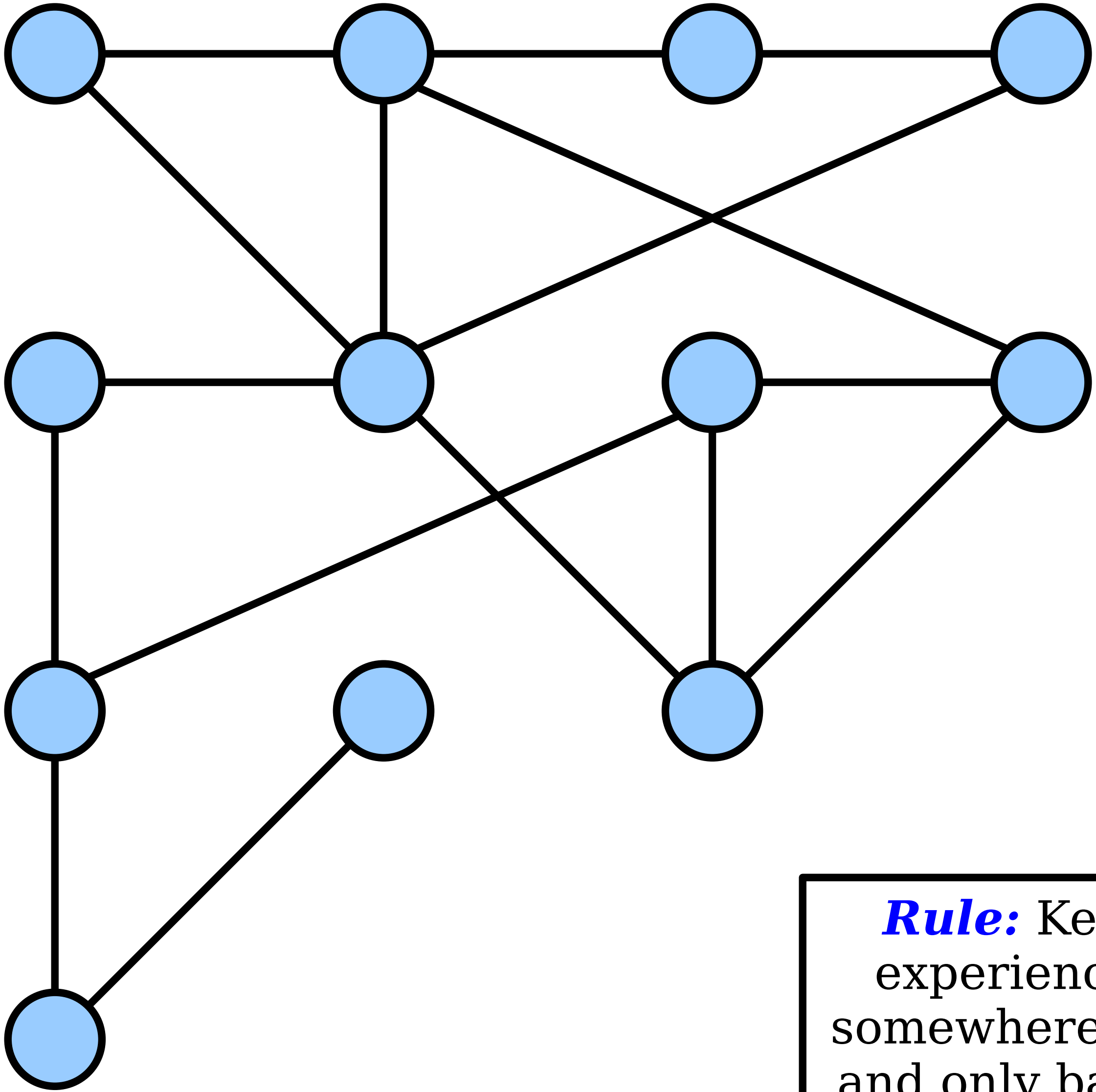
```
bfs-from(node v) {
    make a queue of nodes, initially seeded with v.

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

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

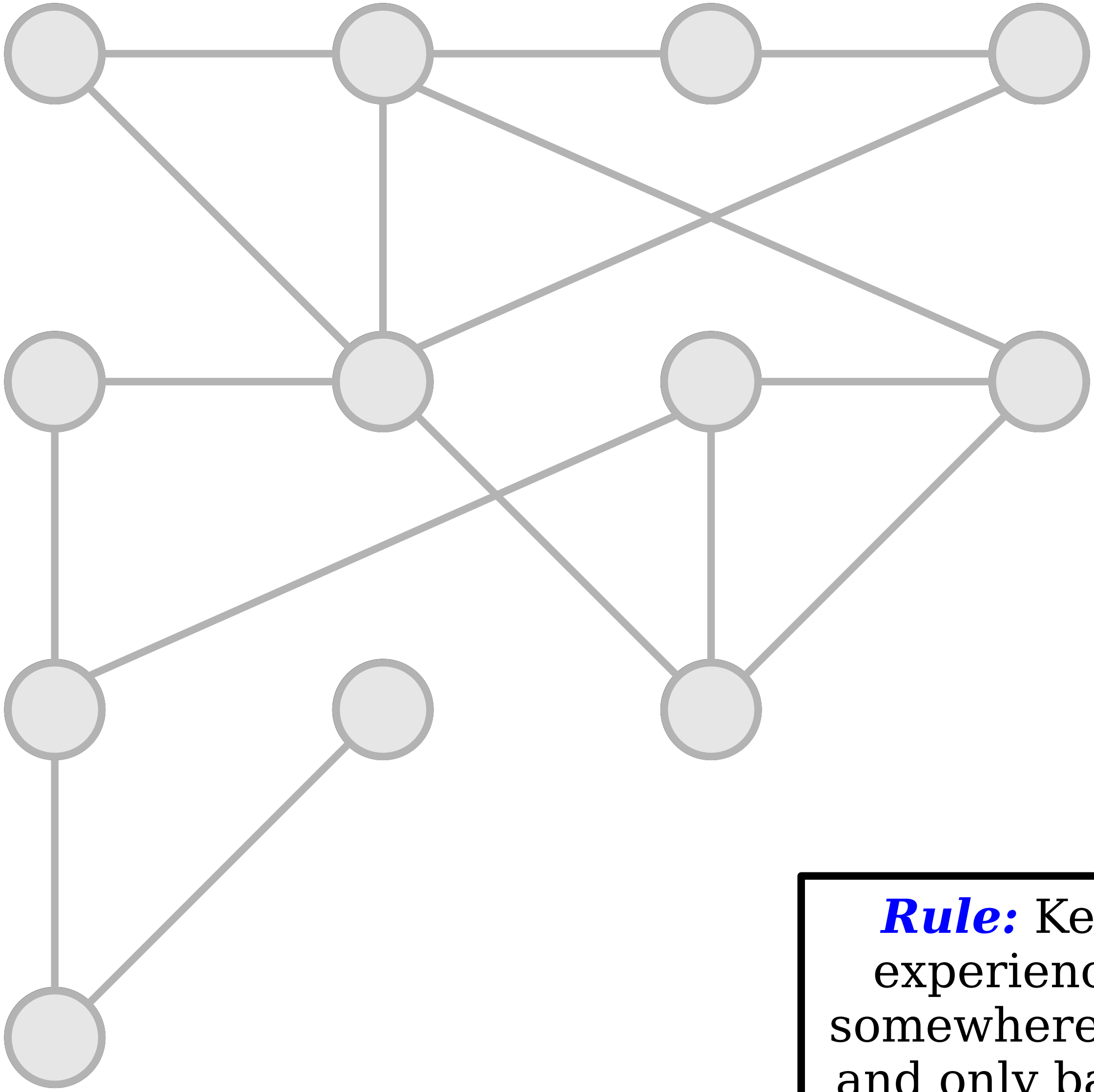
Depth-First Search

Depth-First Search



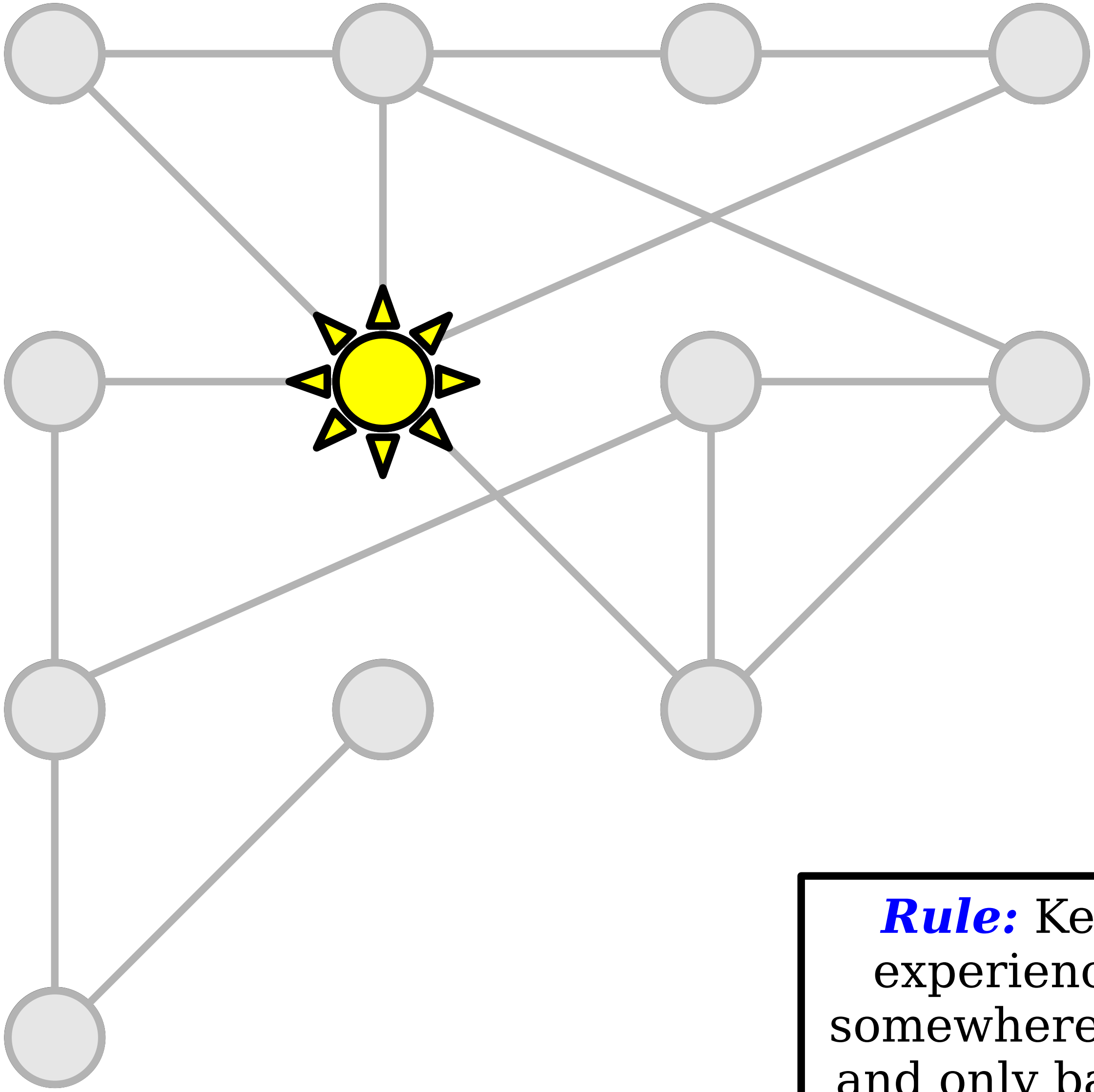
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



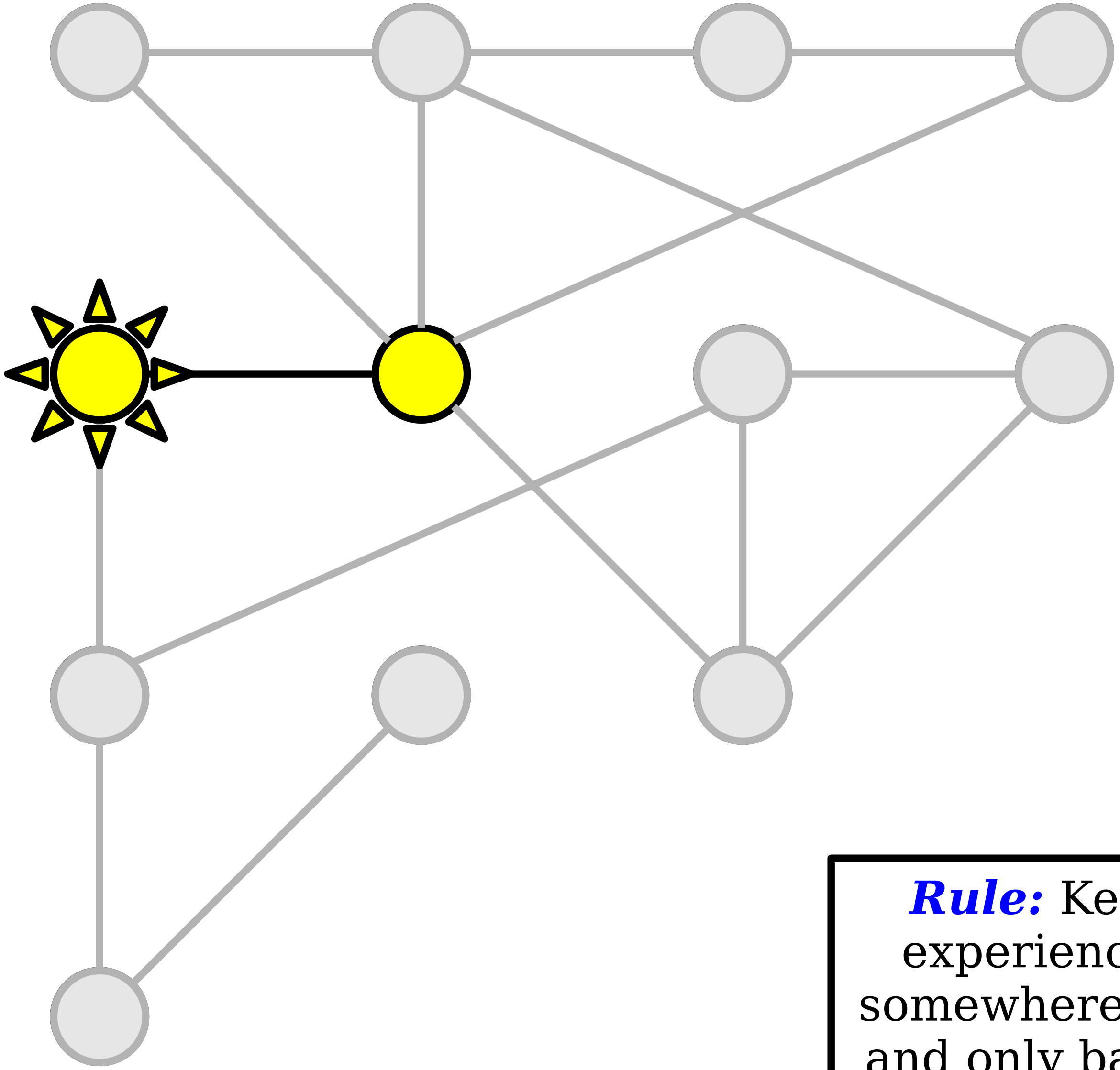
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



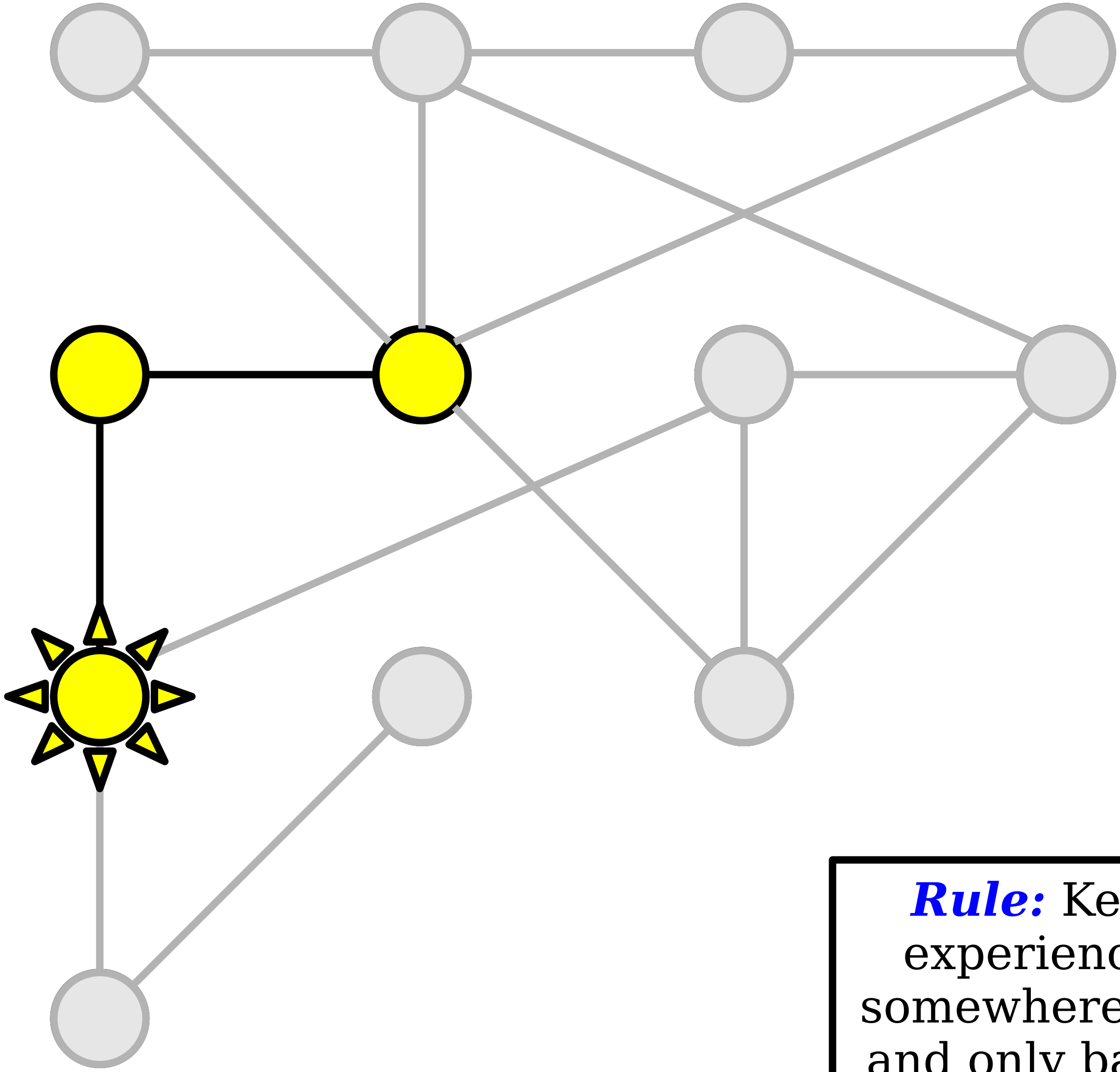
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



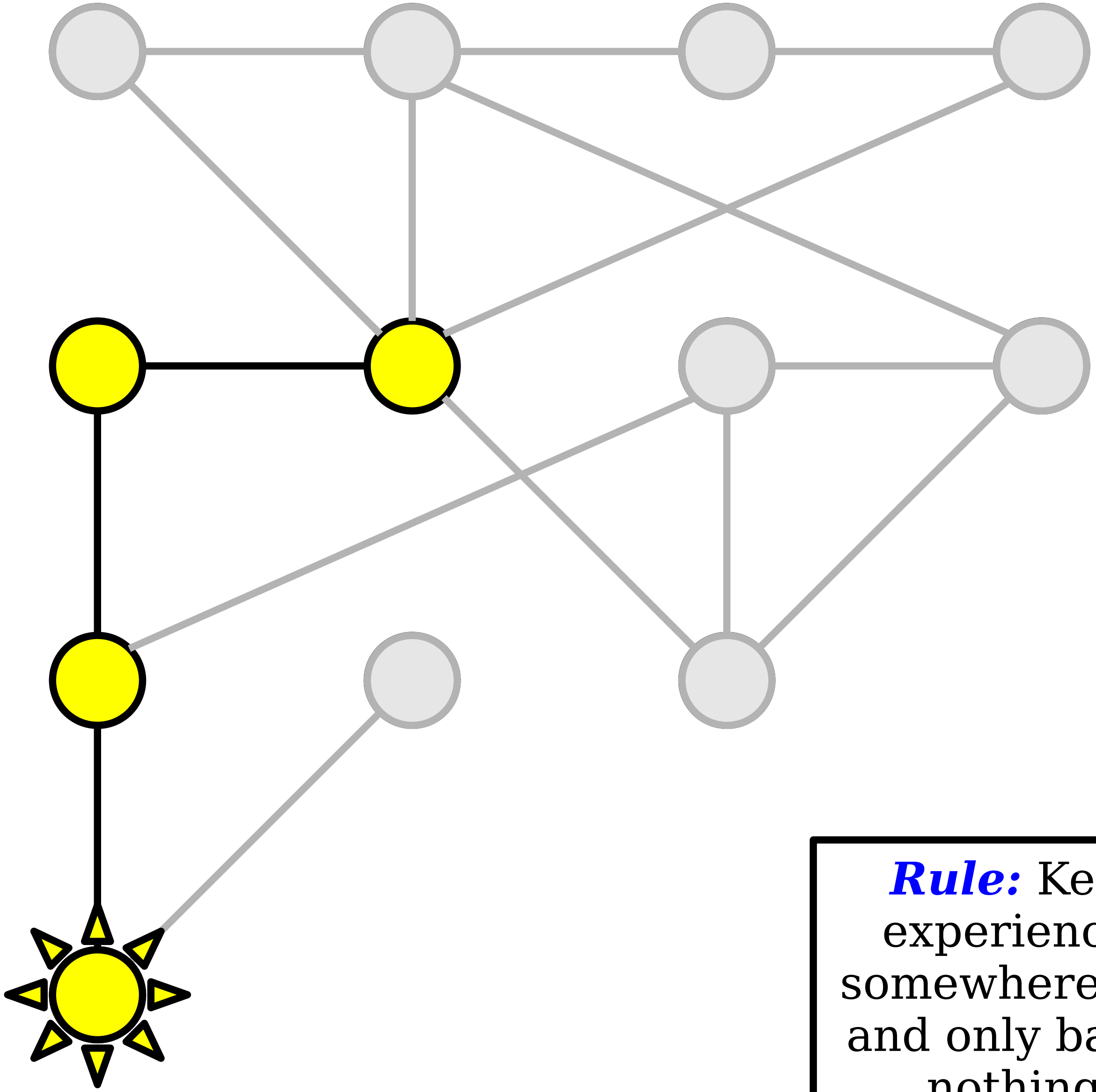
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



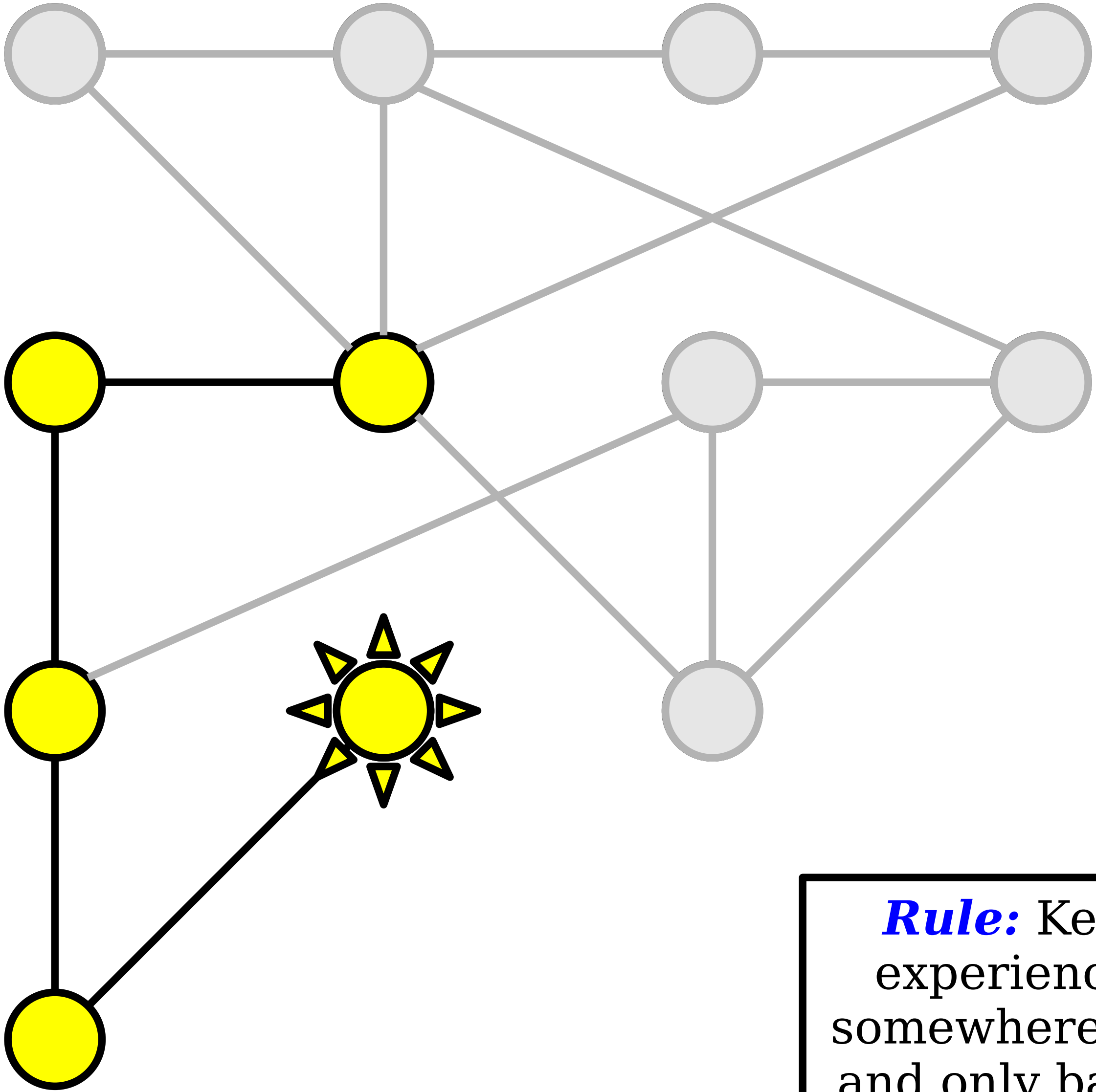
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



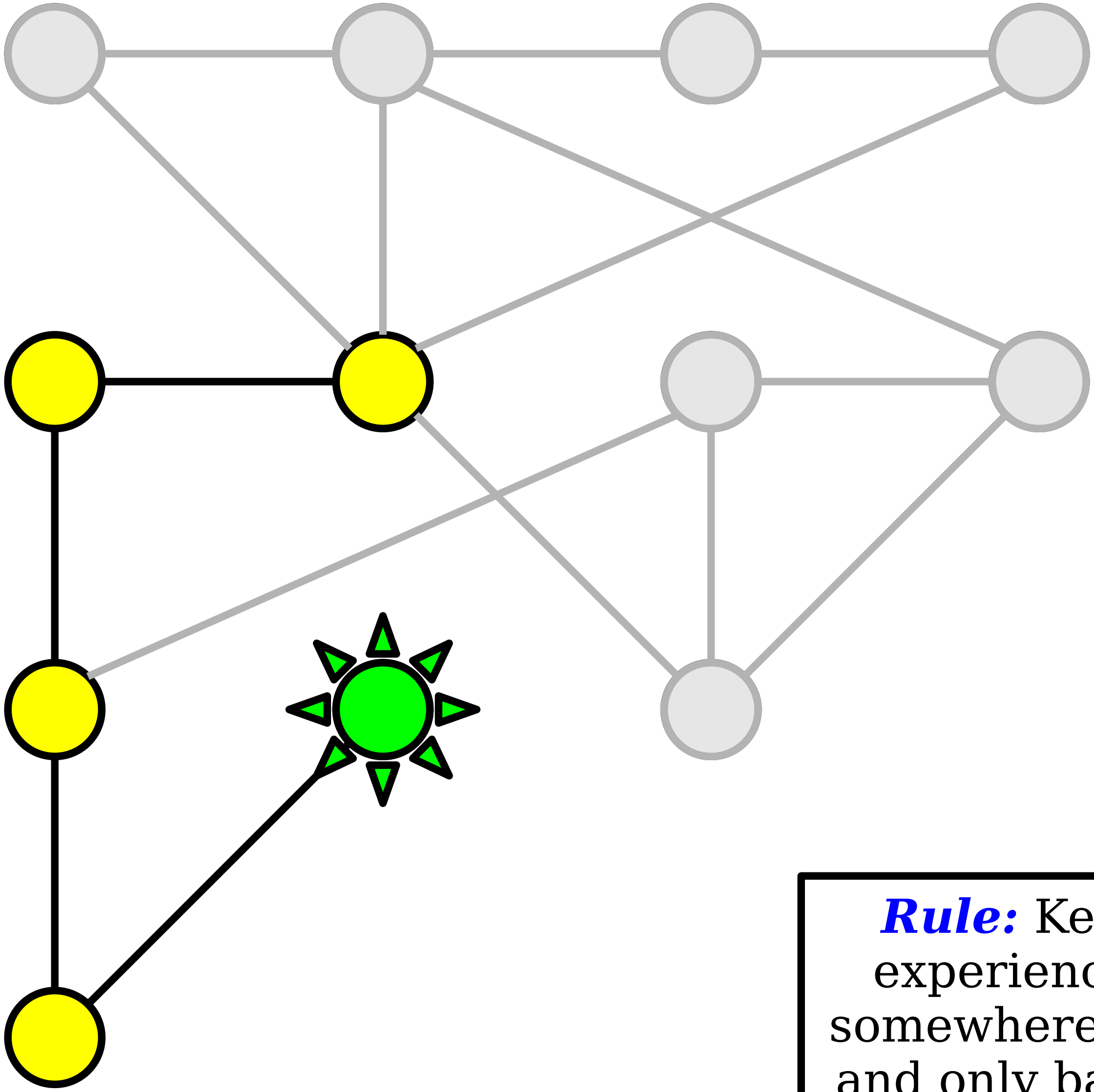
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



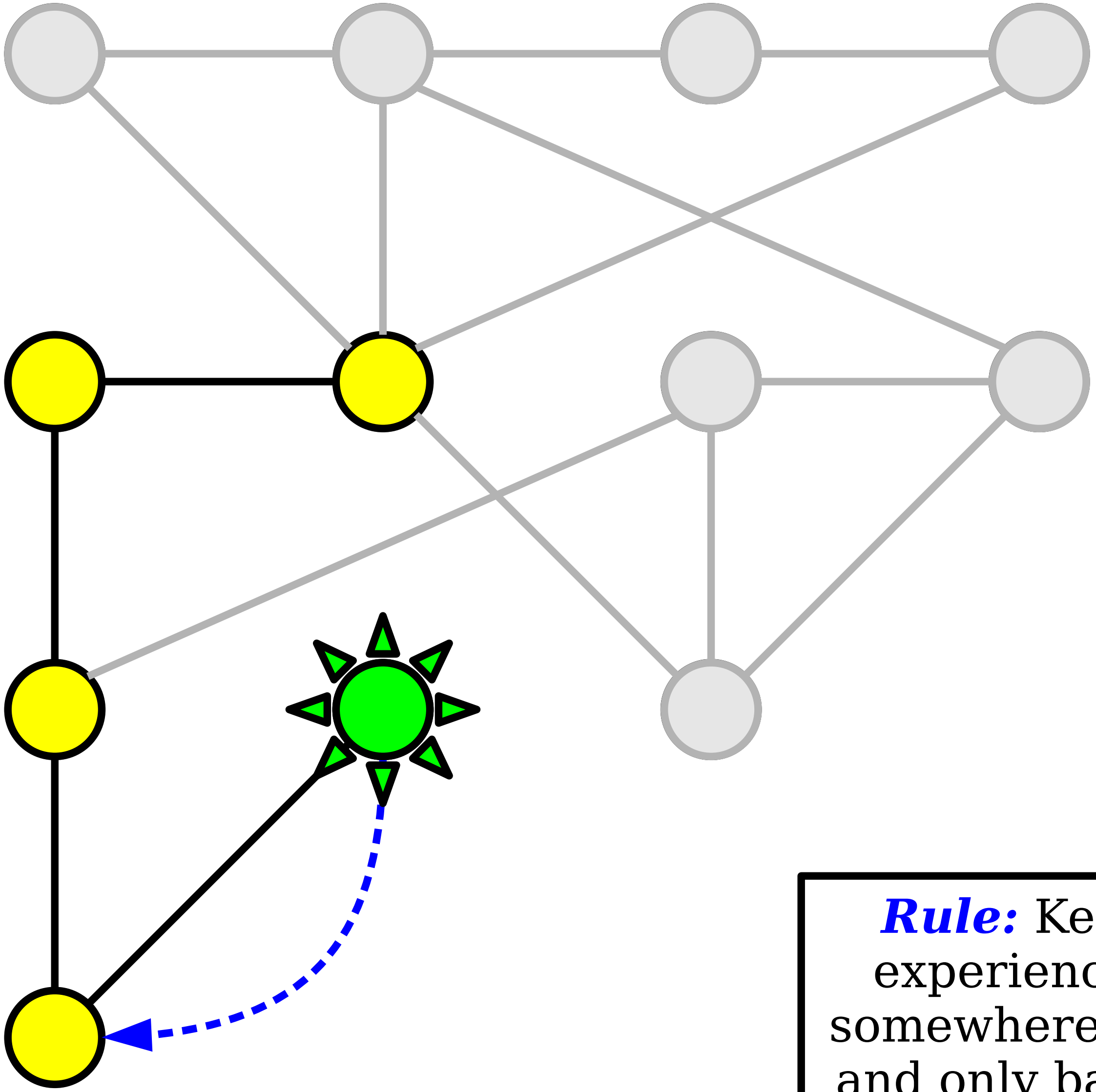
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



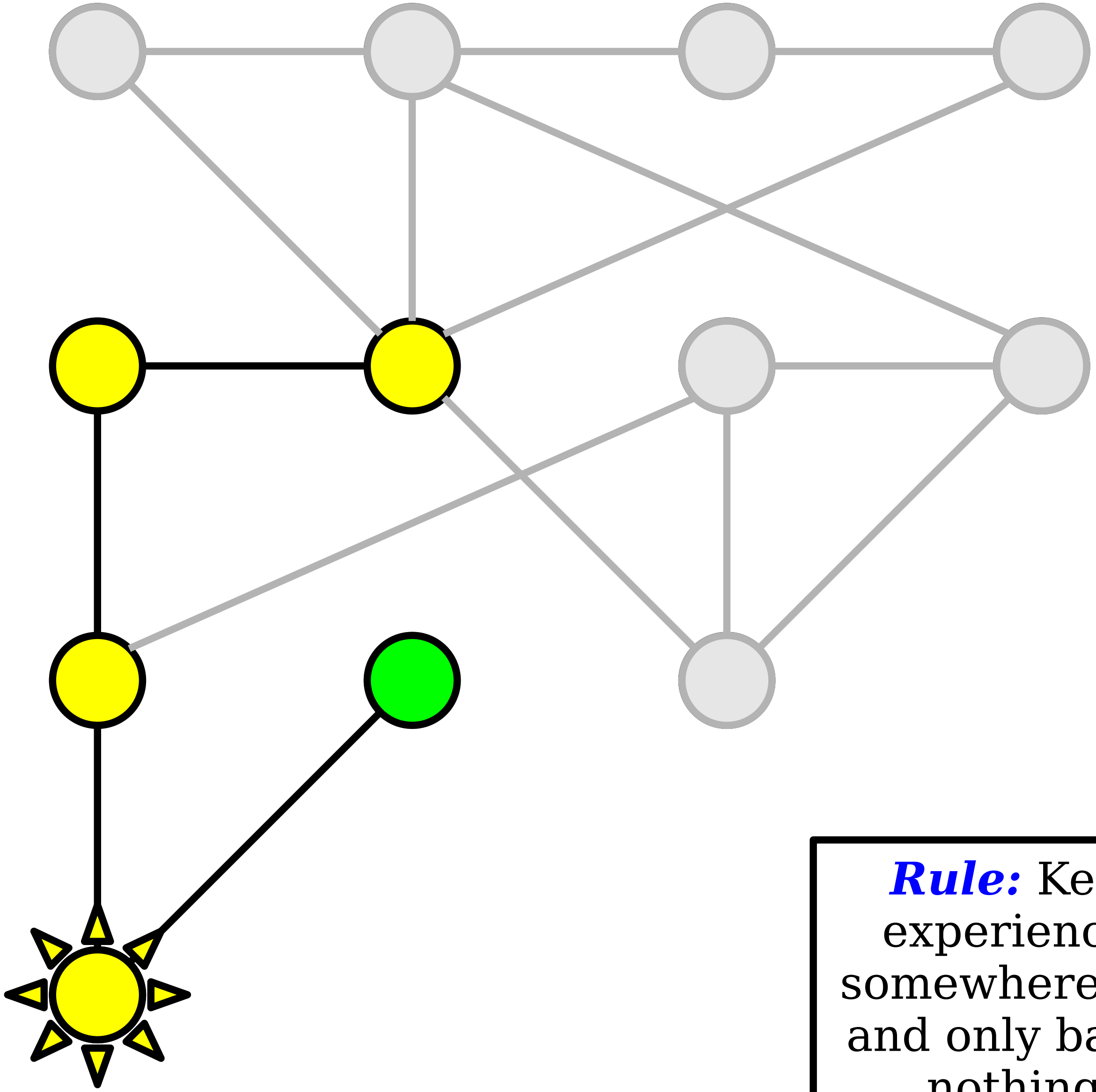
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



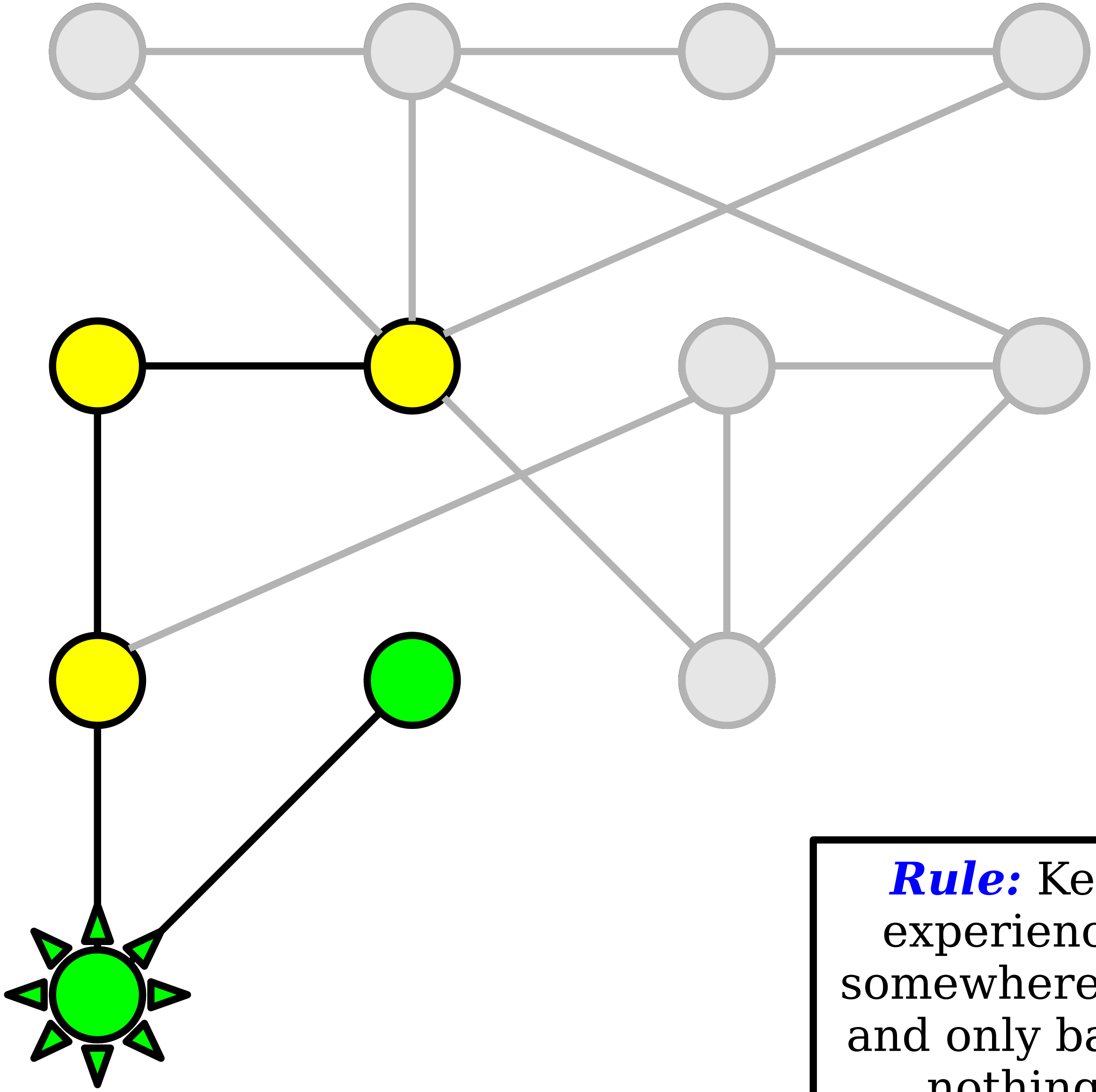
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



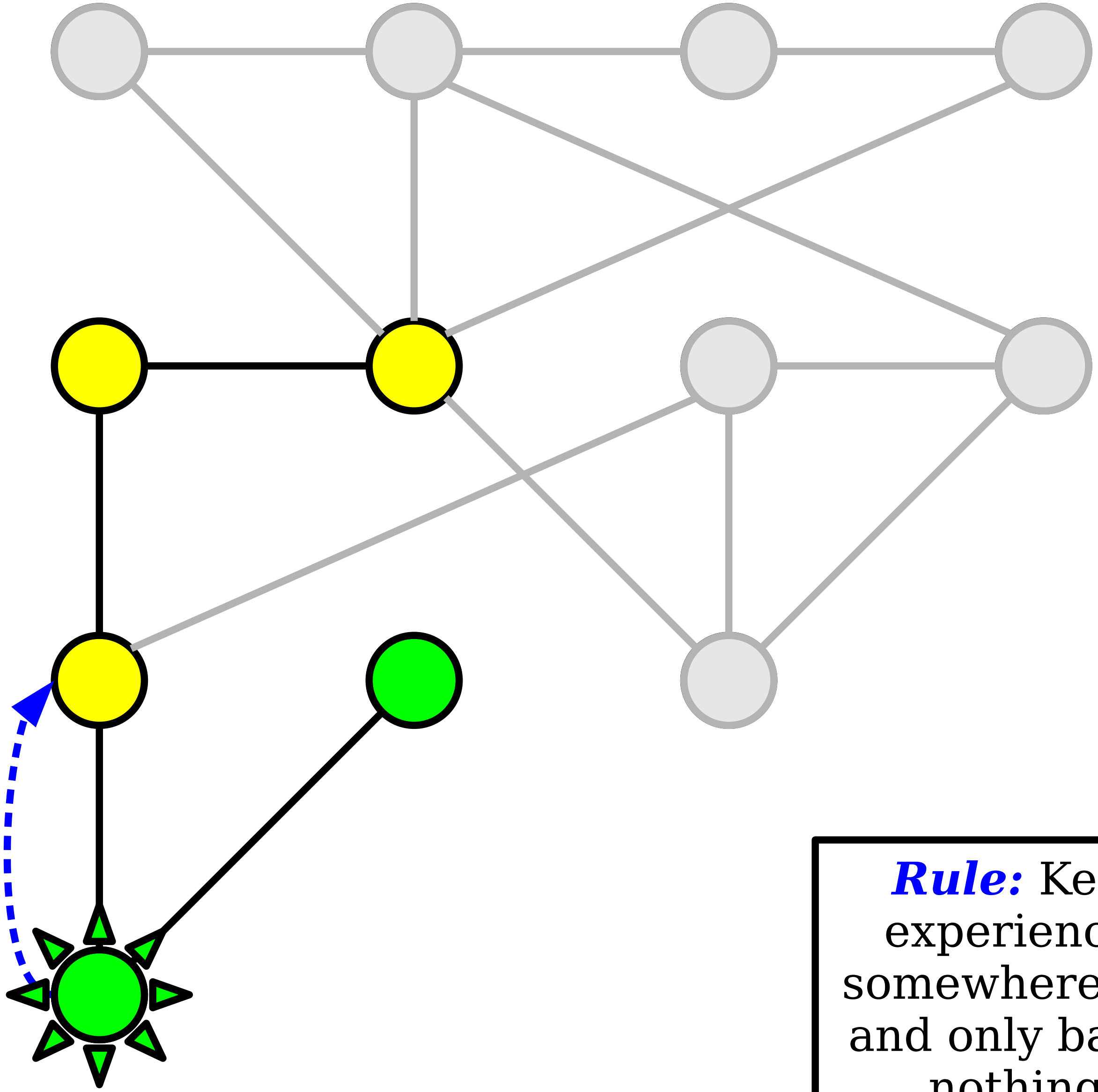
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



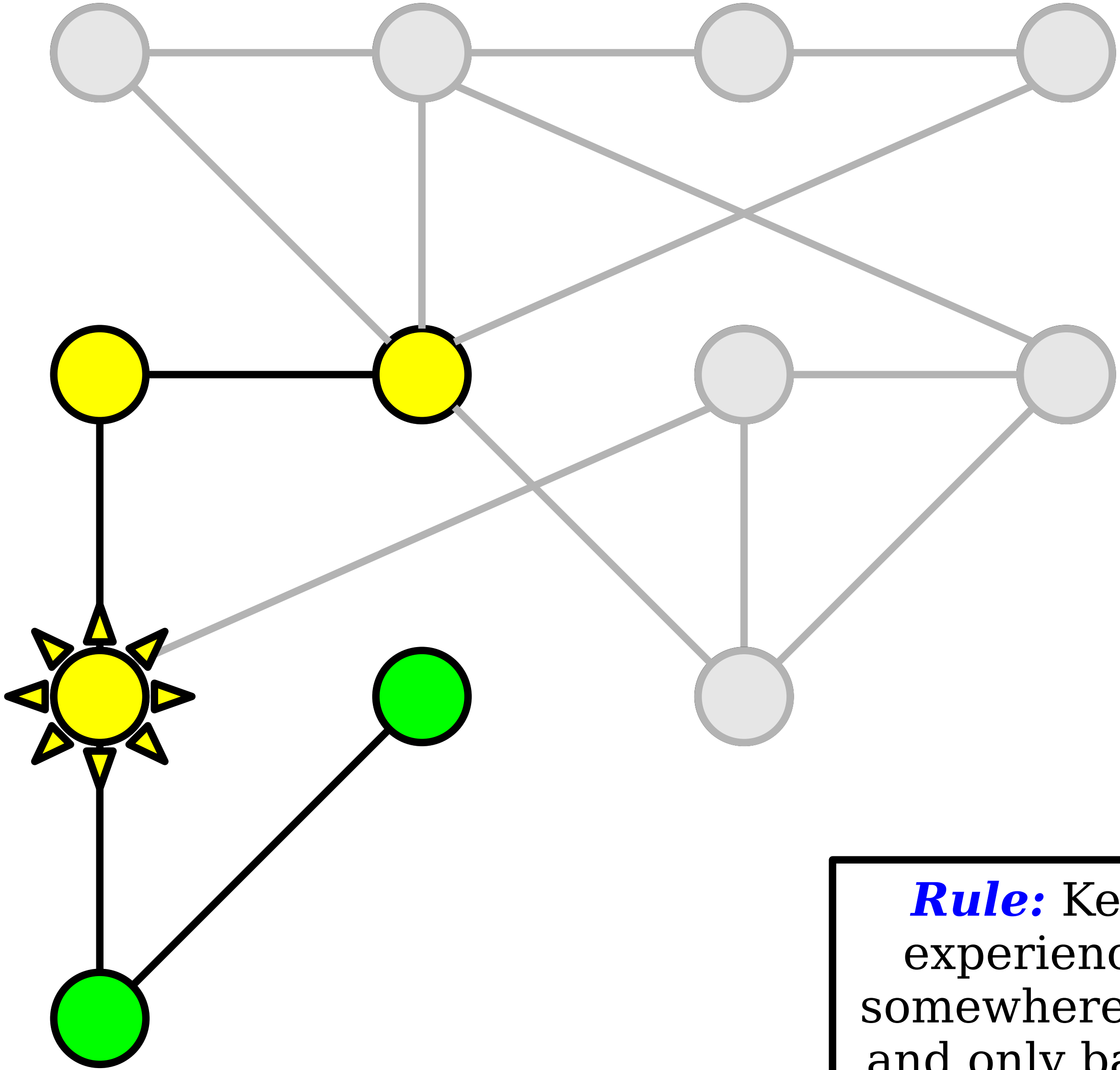
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there’s nothing new to see.

Depth-First Search



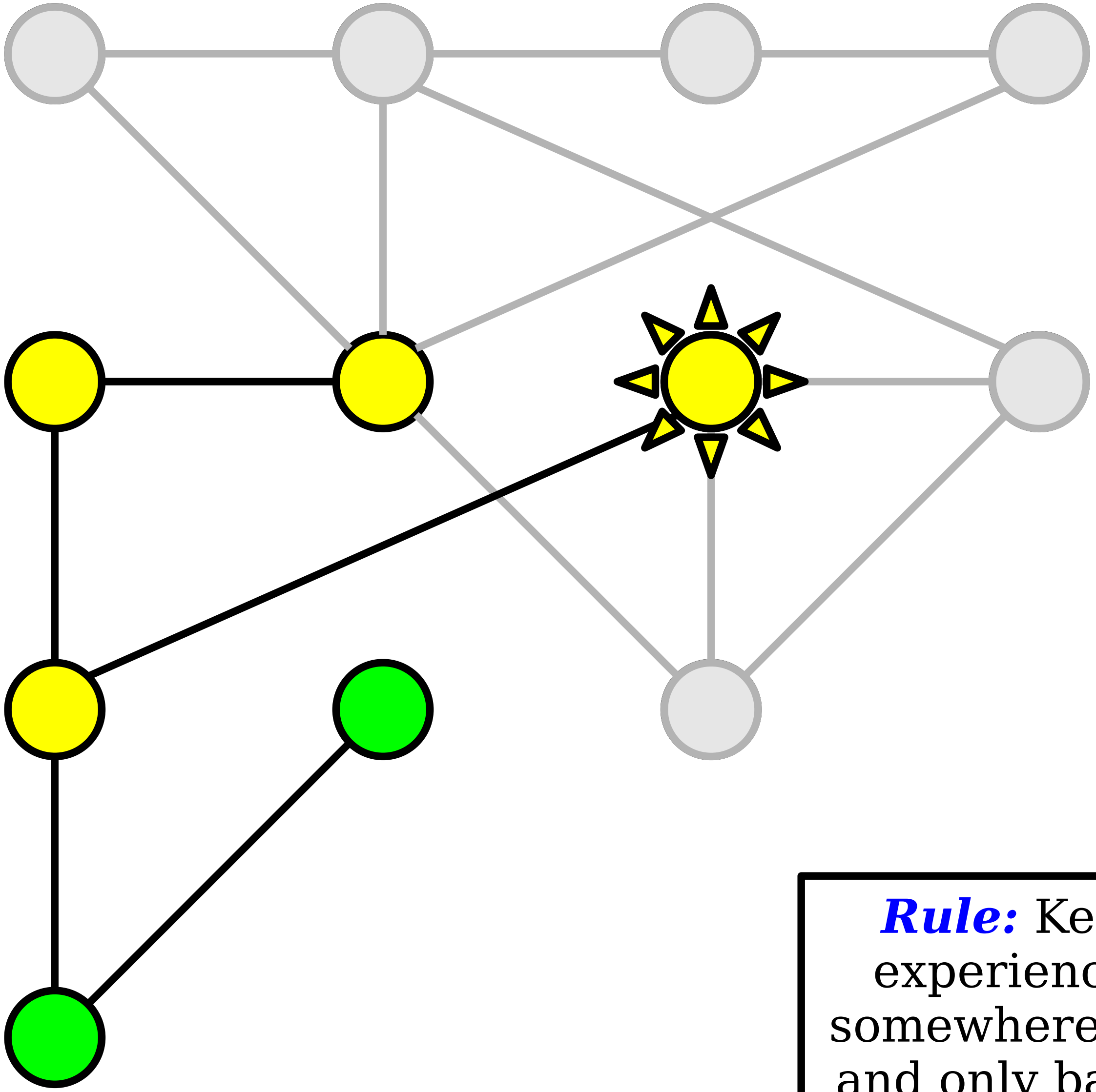
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



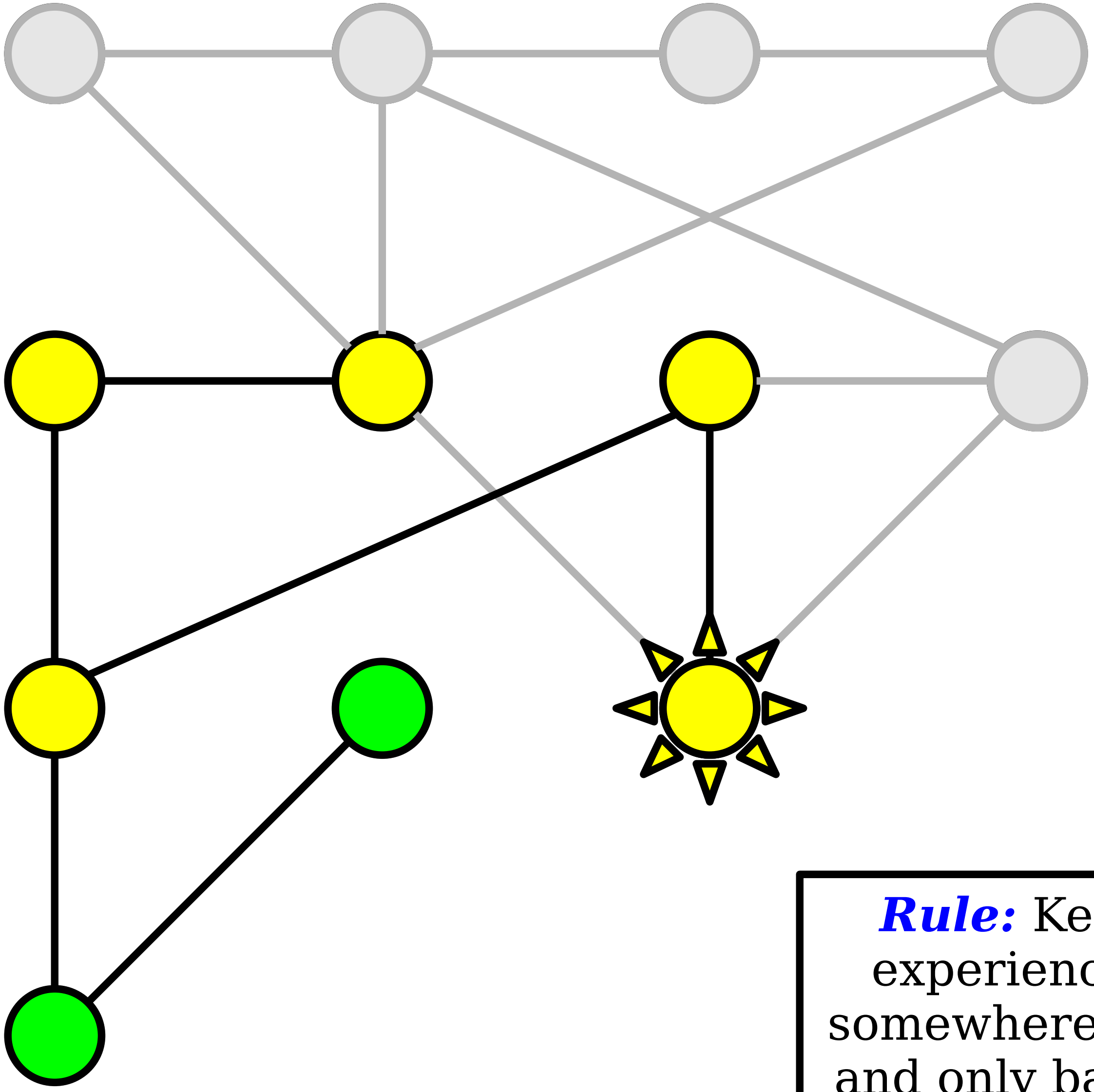
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



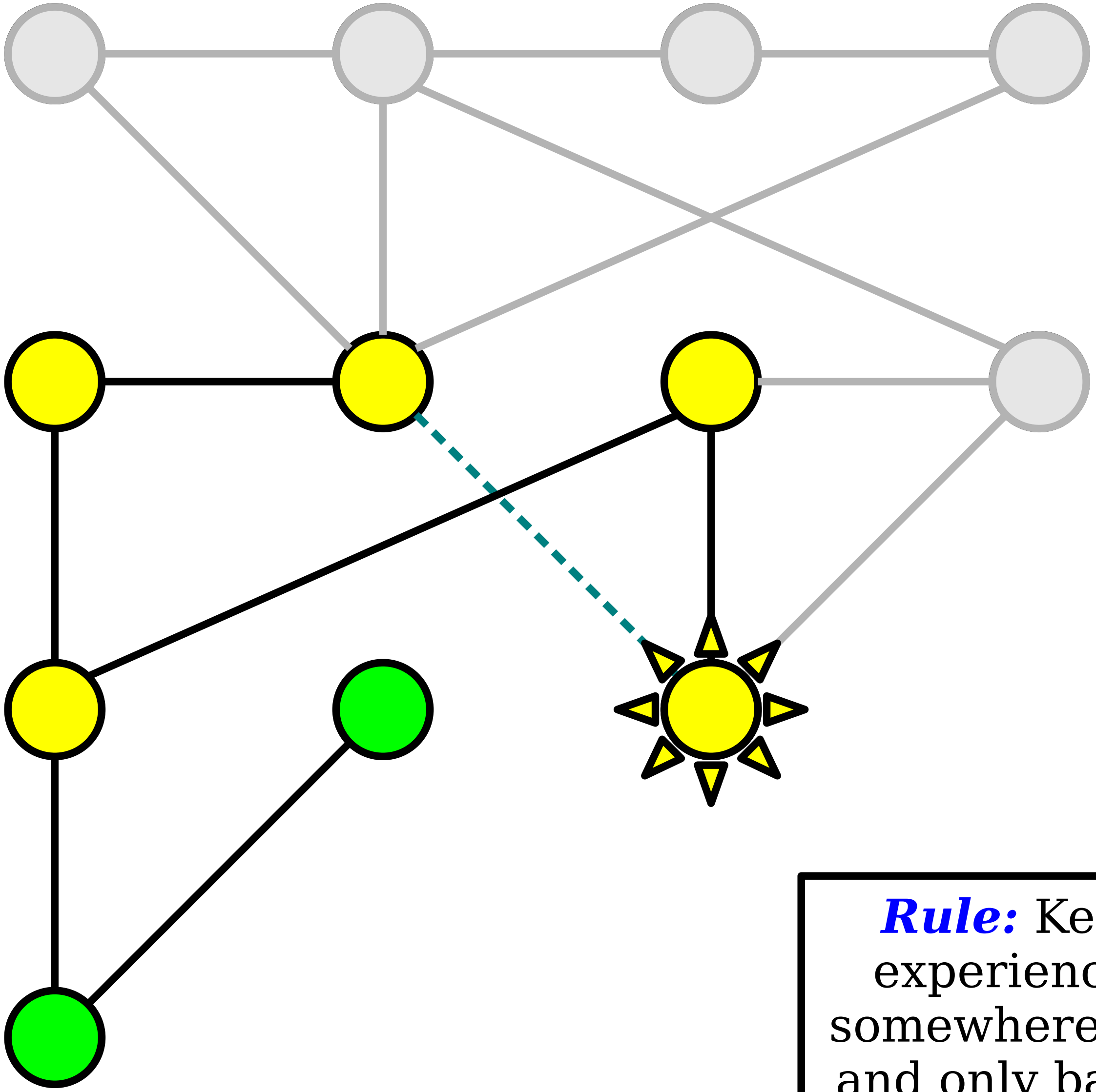
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



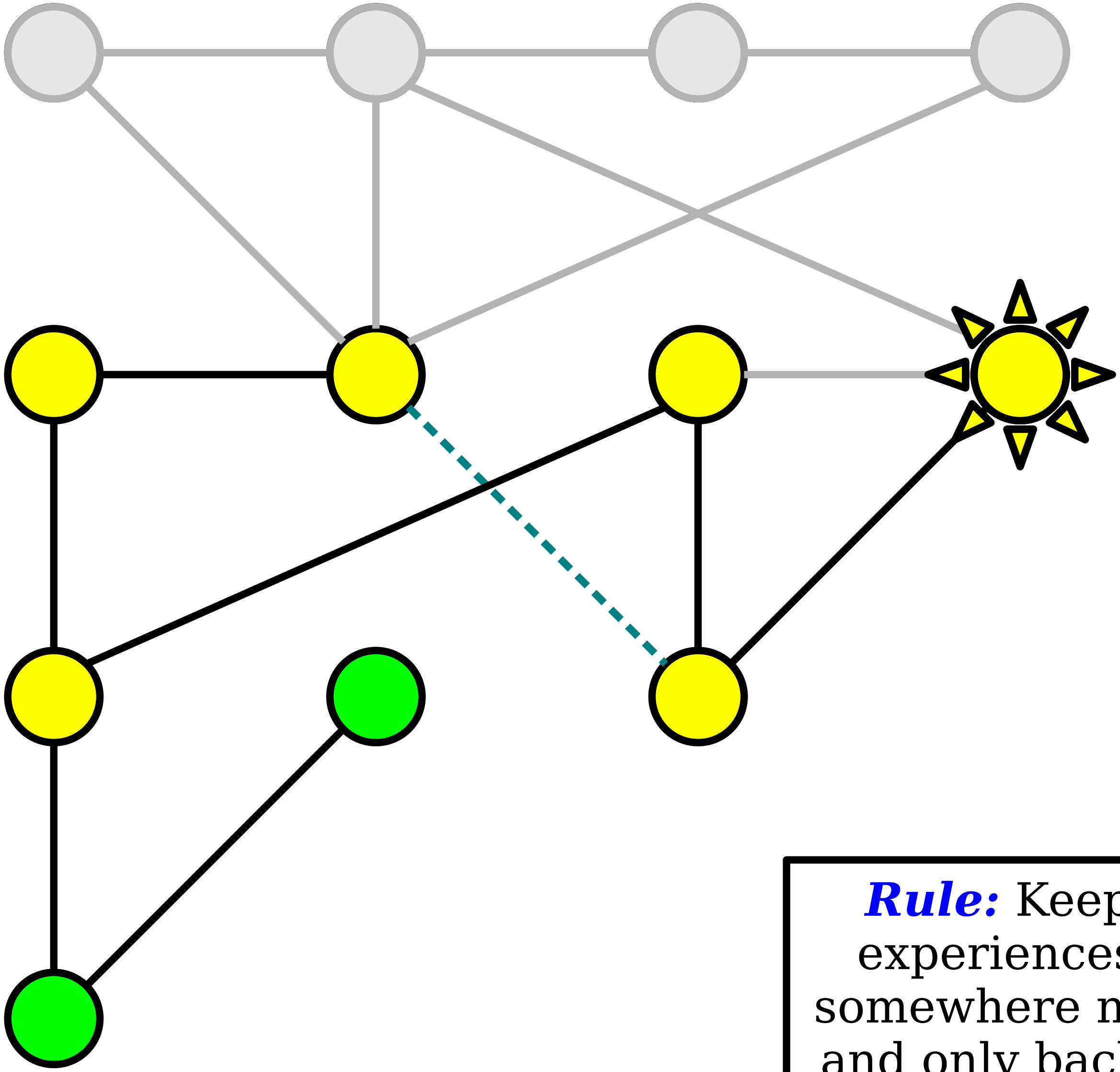
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



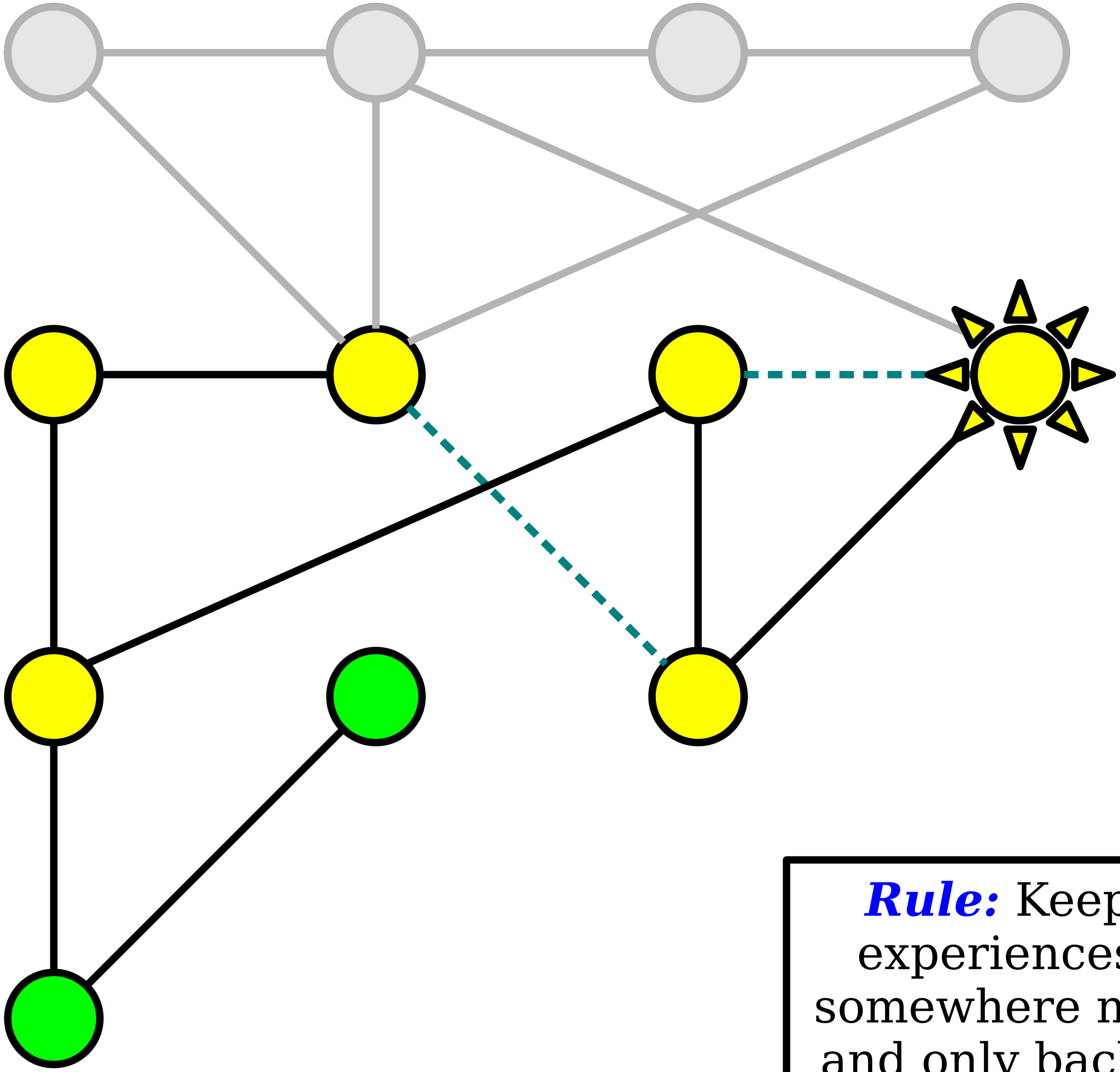
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



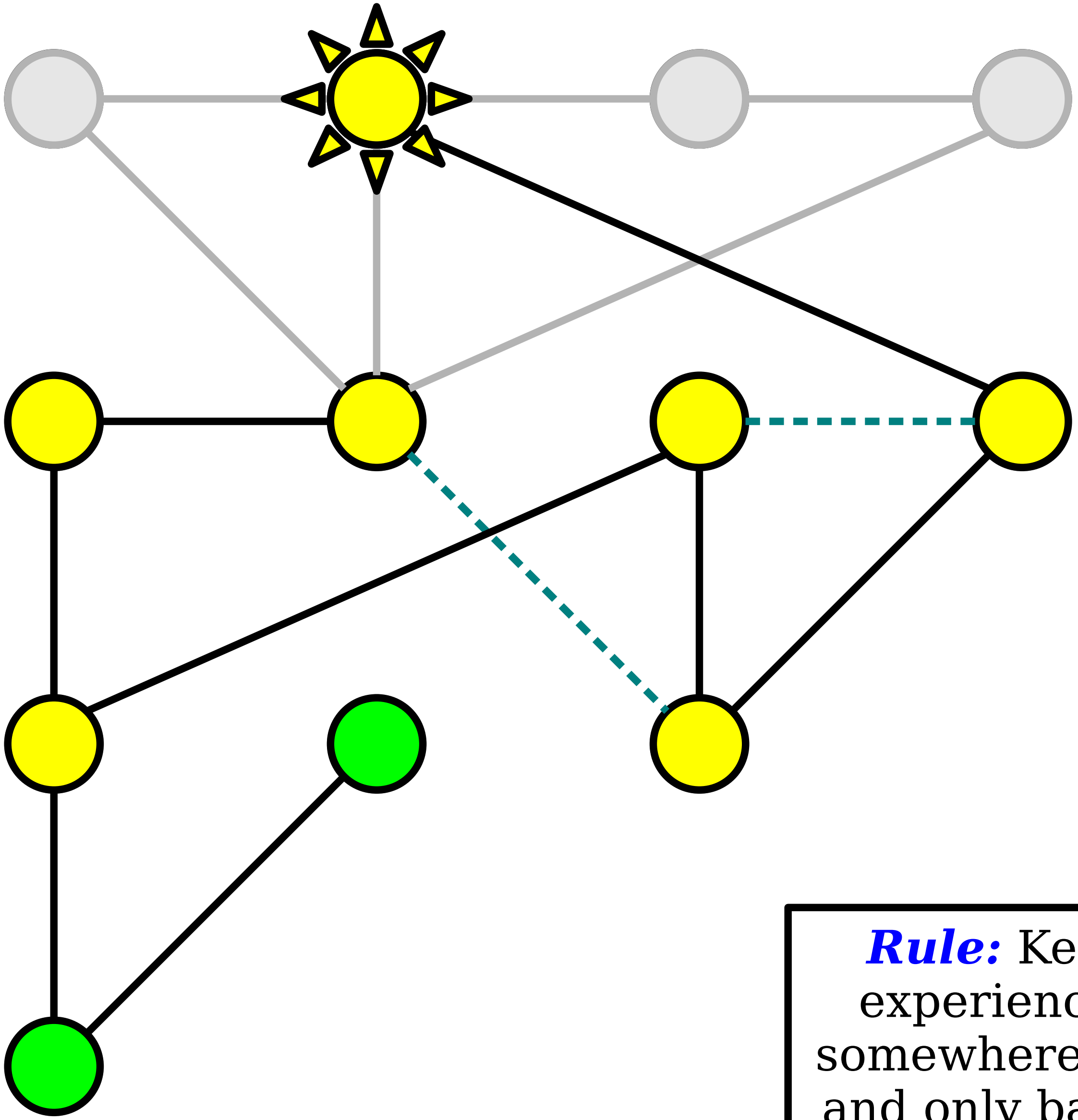
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



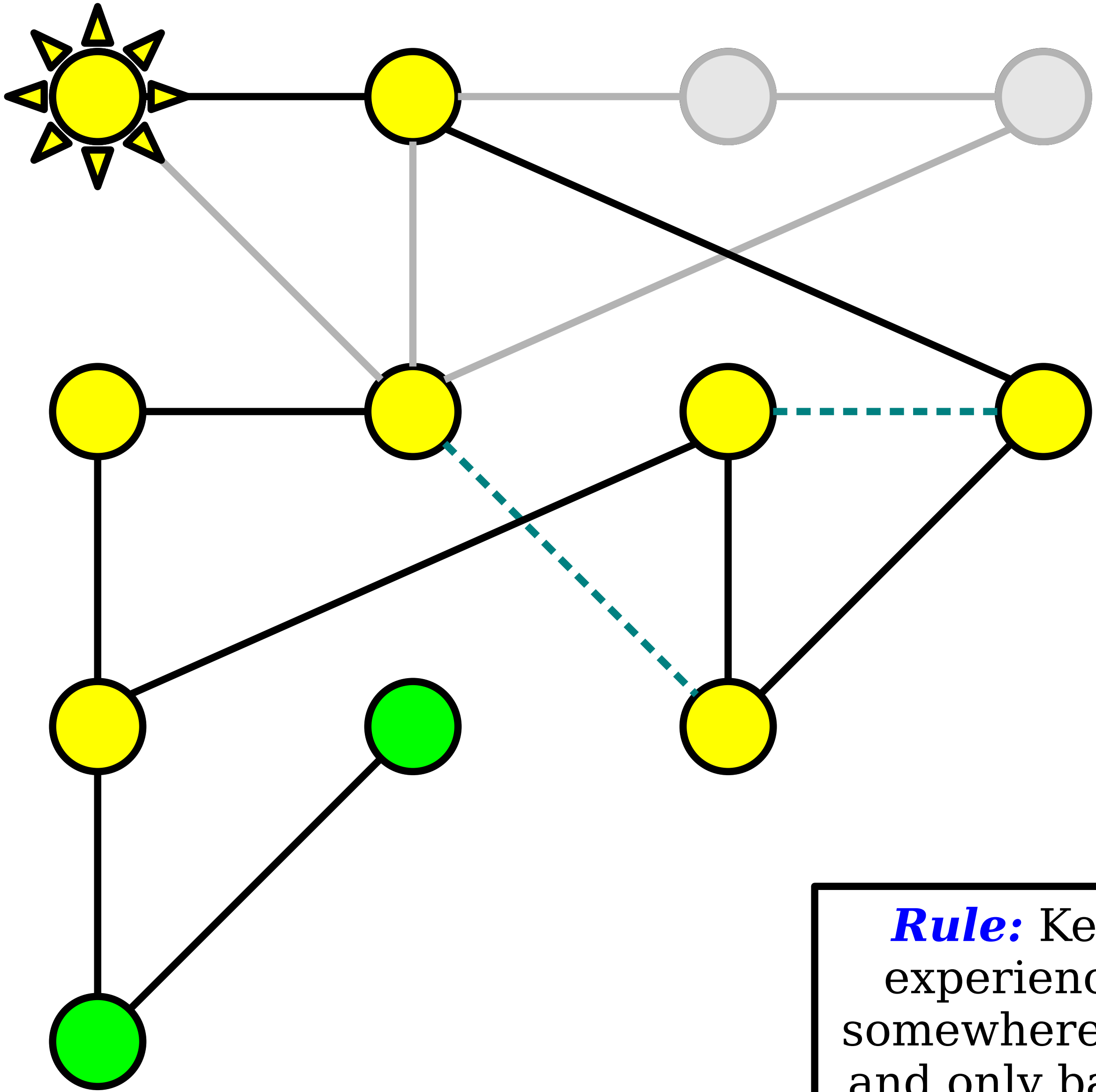
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



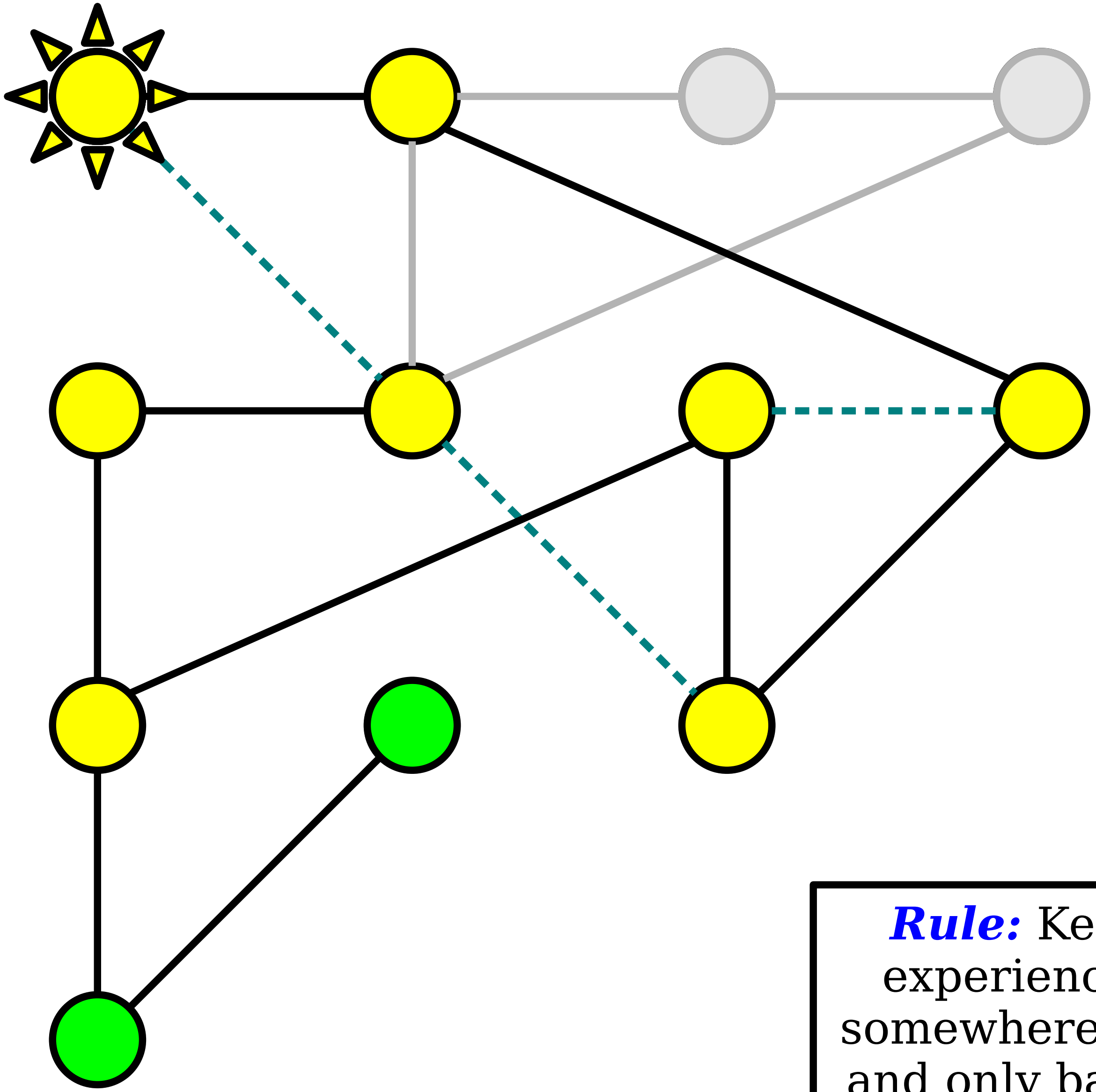
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there’s nothing new to see.

Depth-First Search



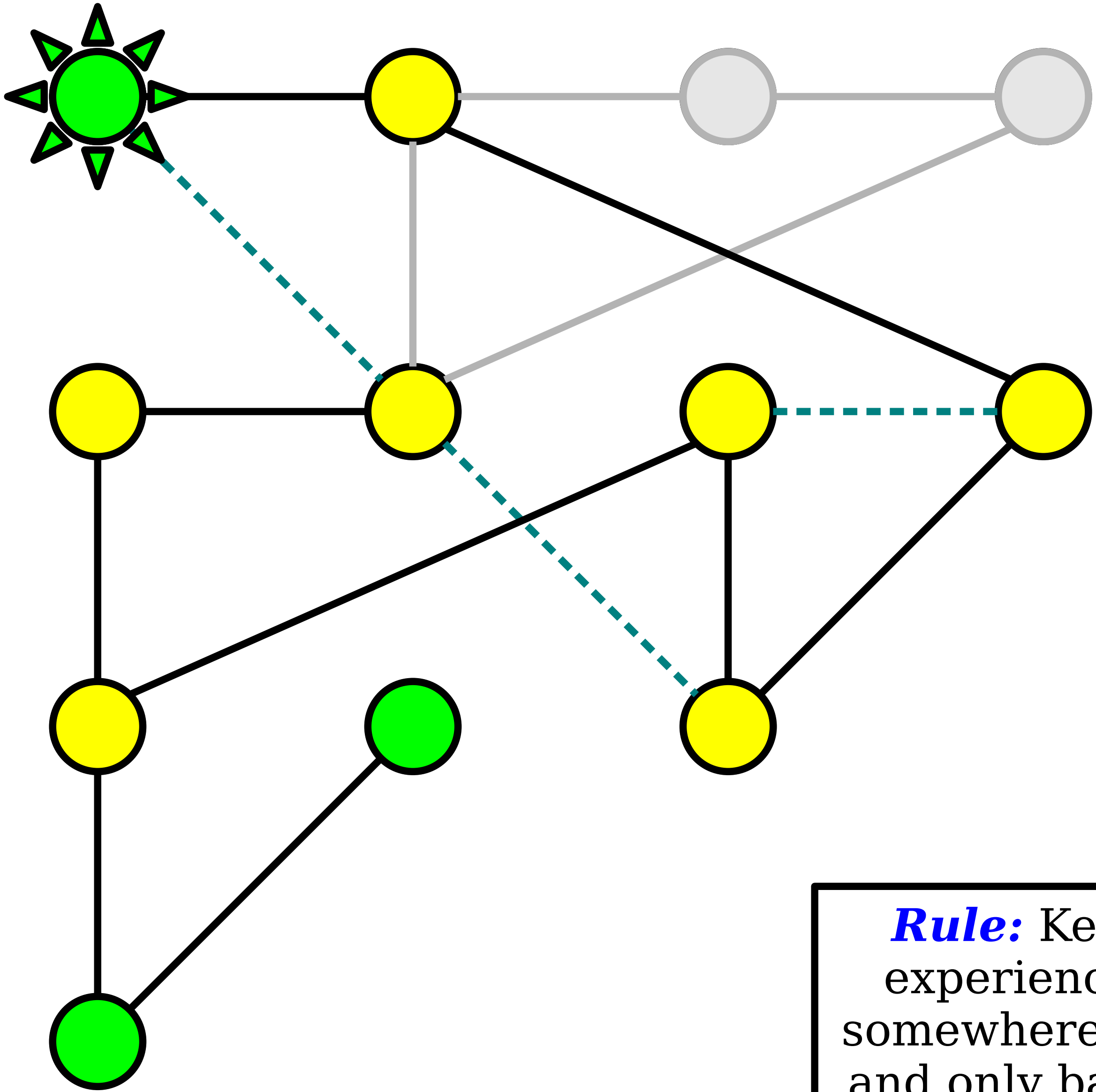
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there’s nothing new to see.

Depth-First Search



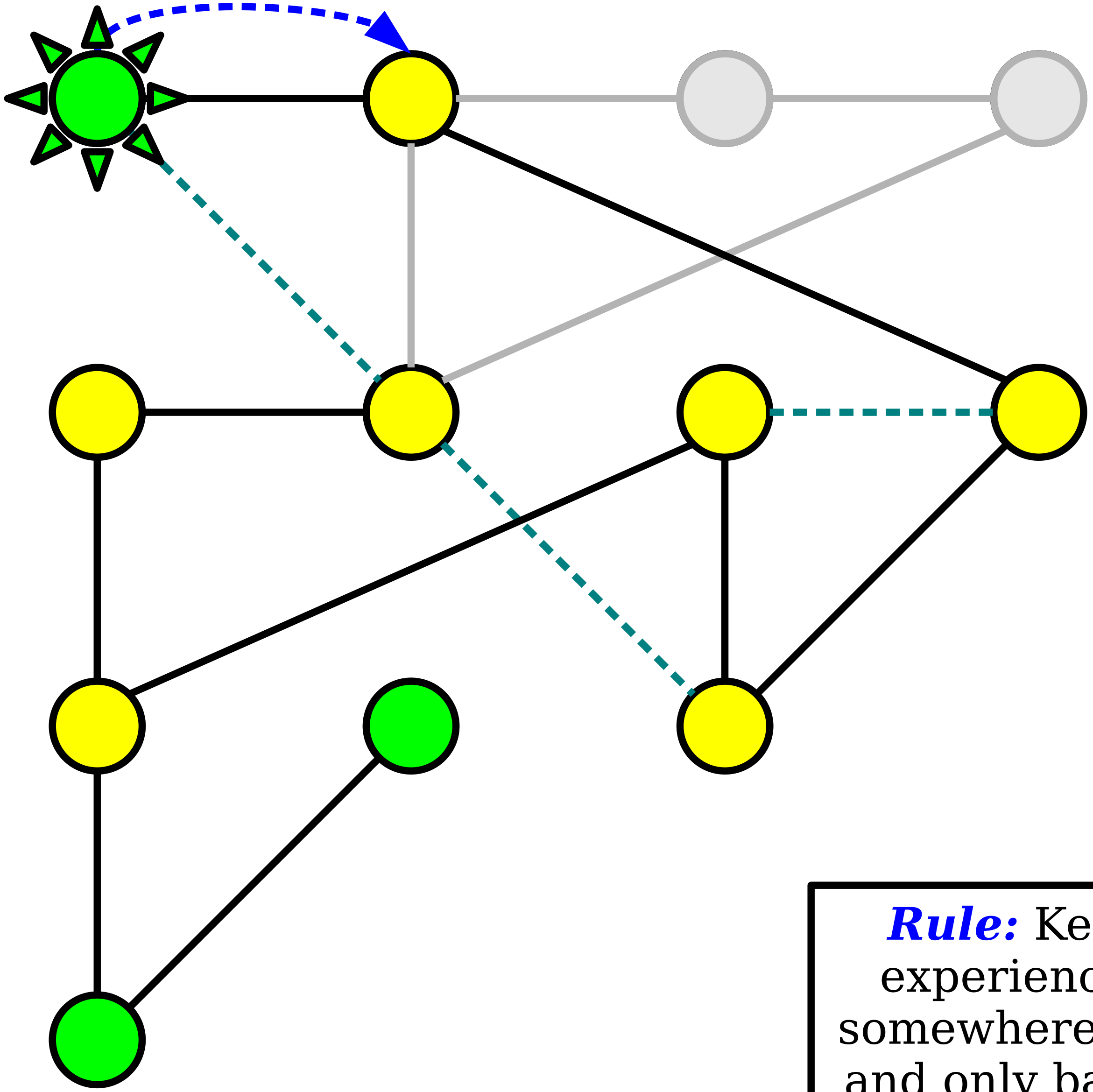
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



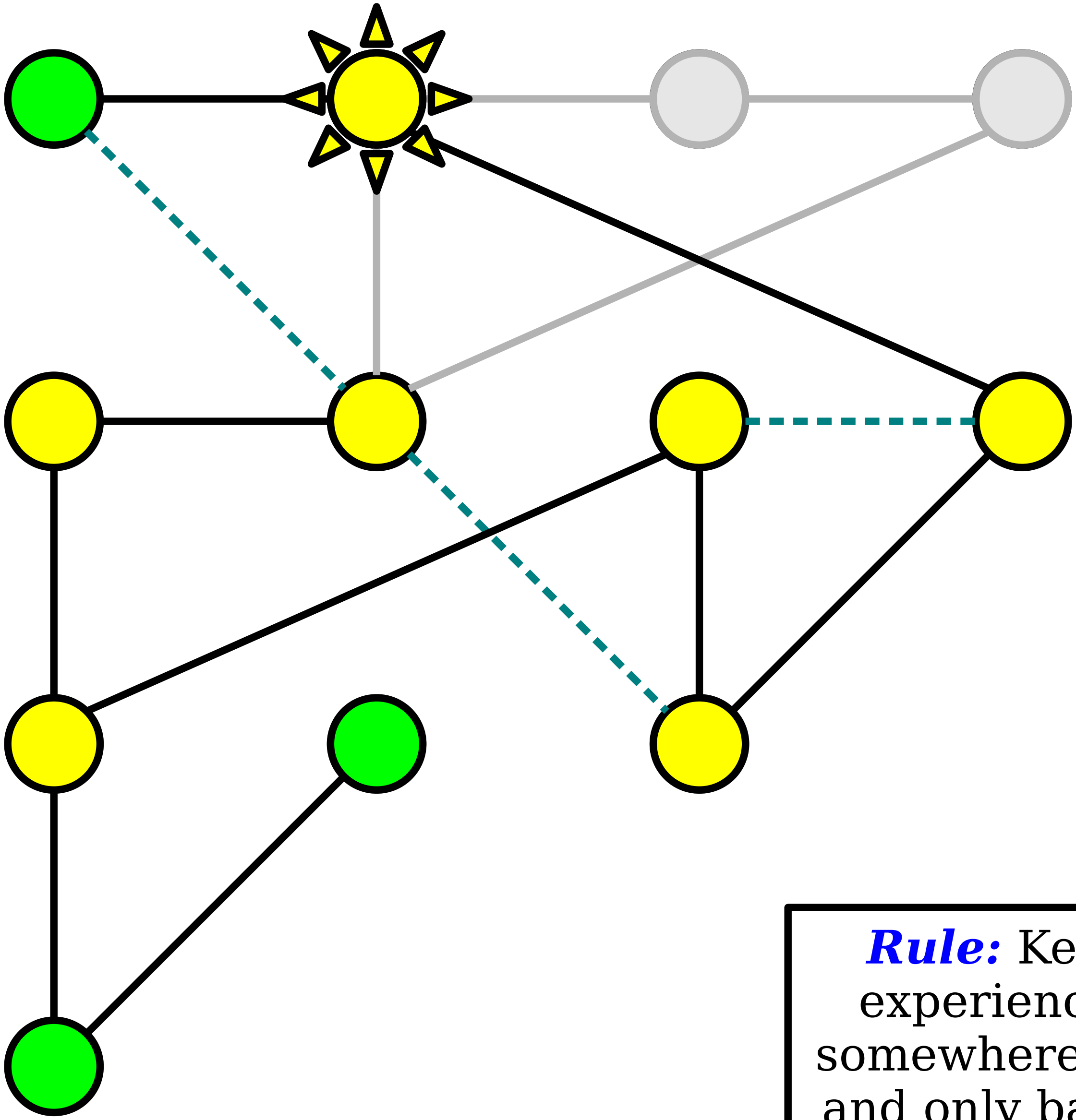
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



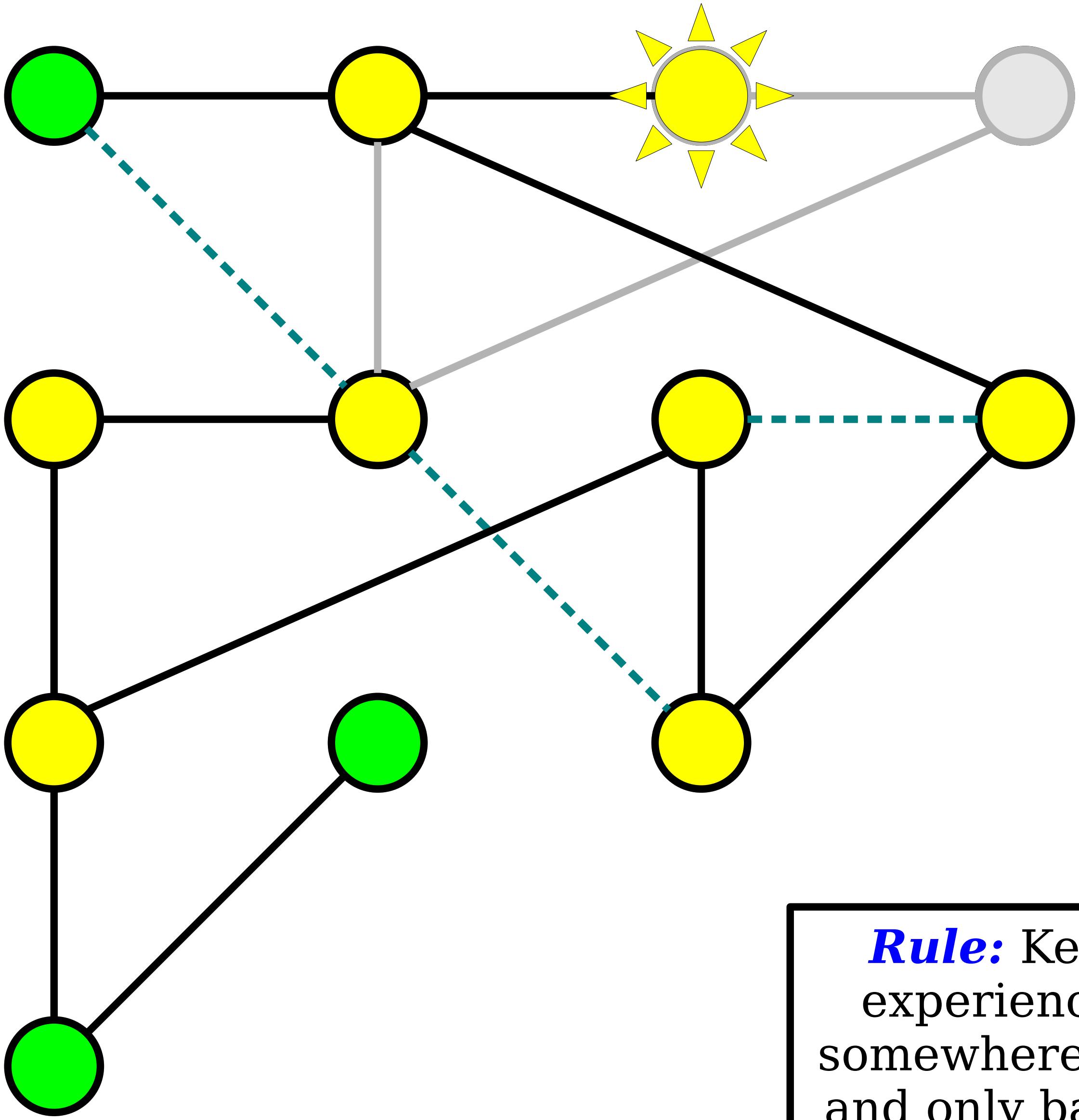
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there’s nothing new to see.

Depth-First Search



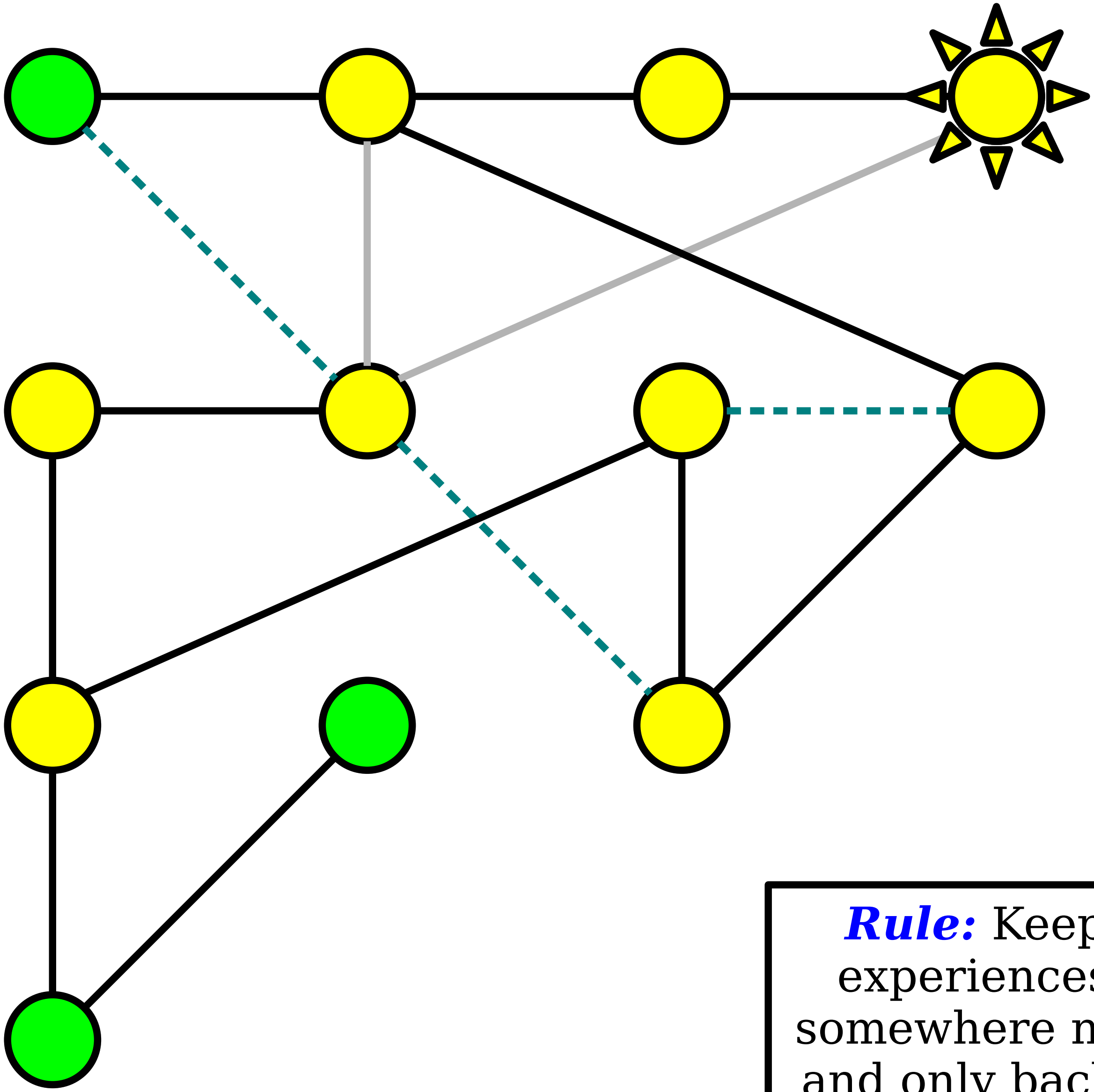
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



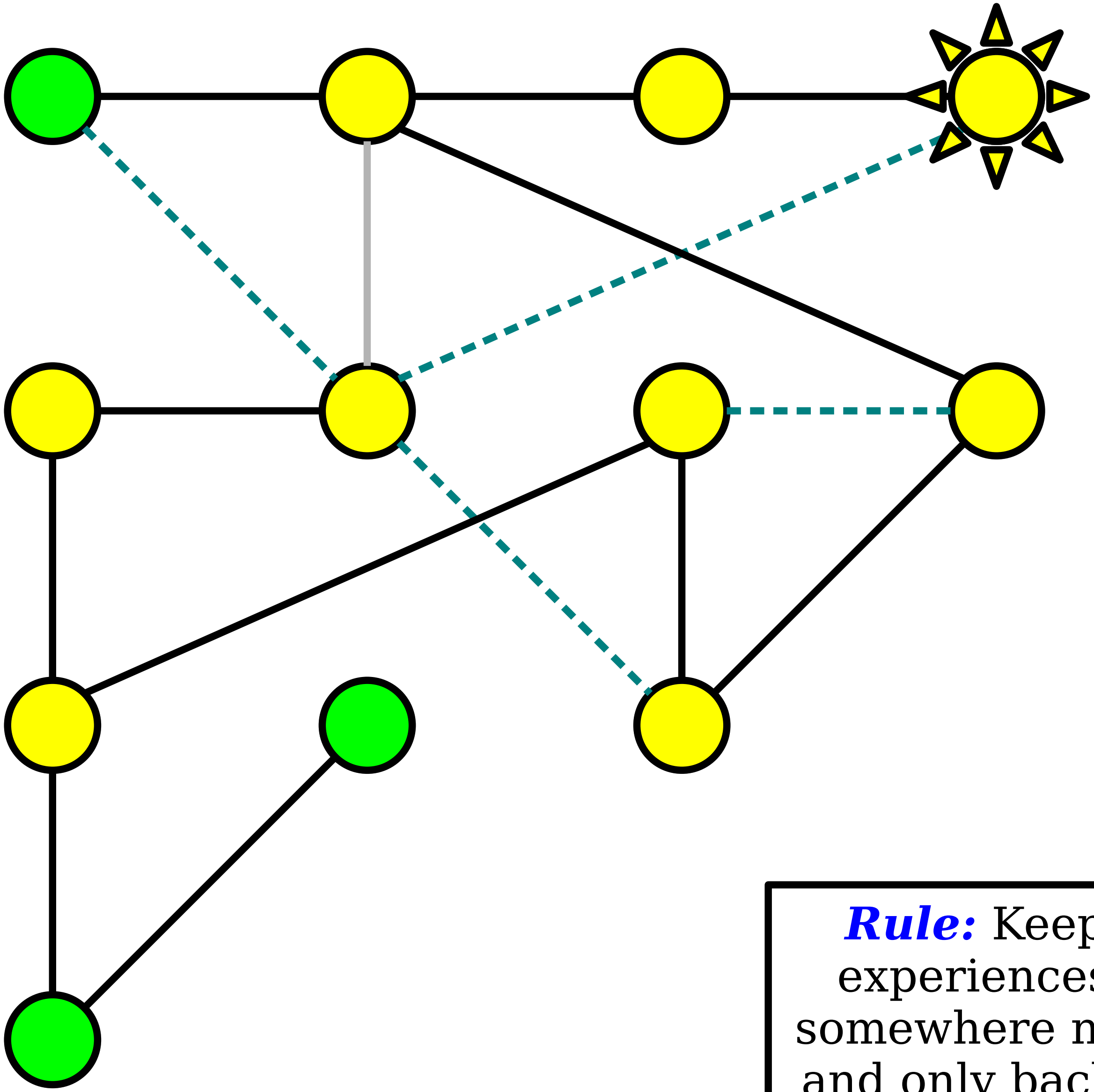
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



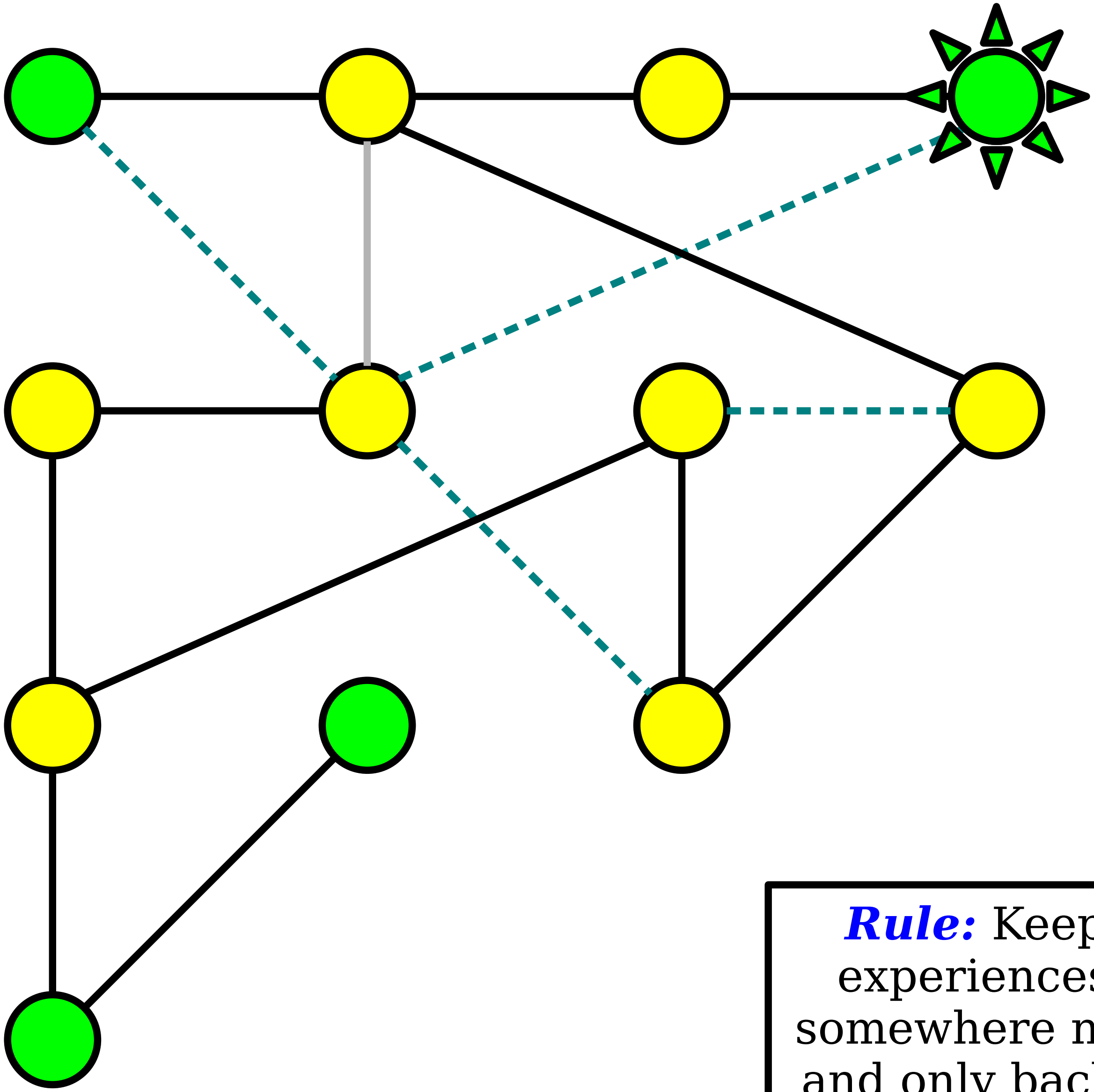
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



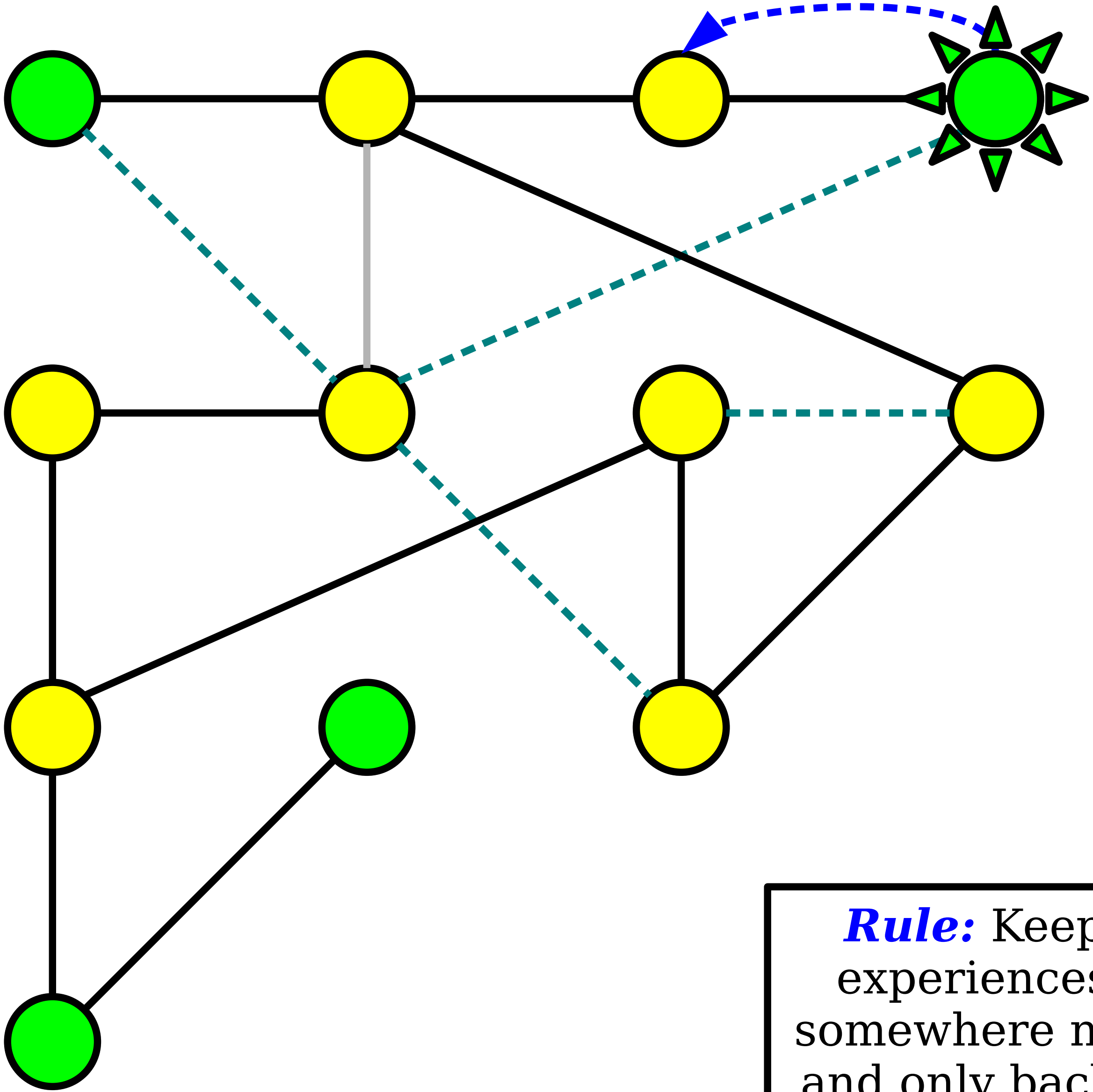
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



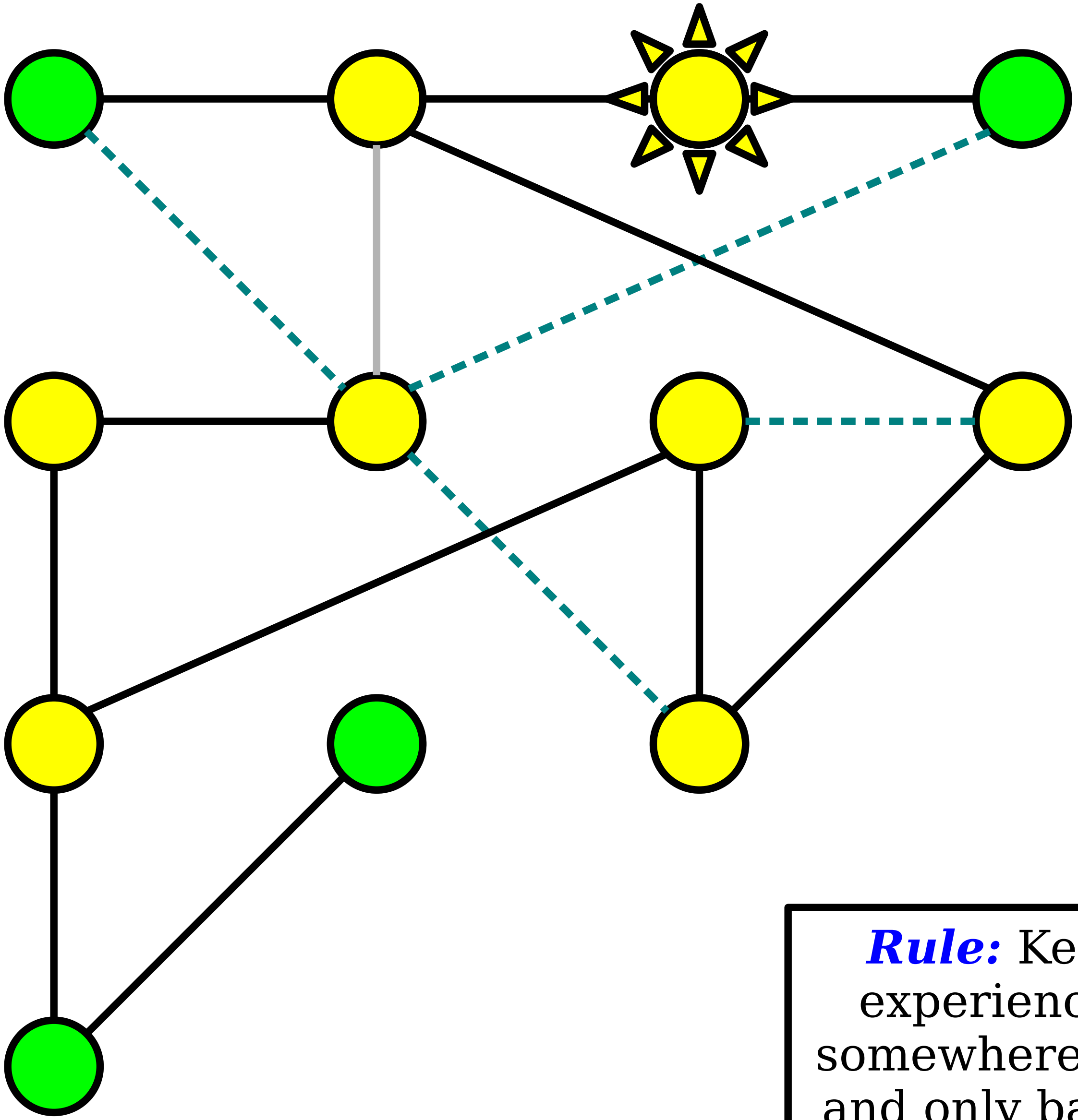
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



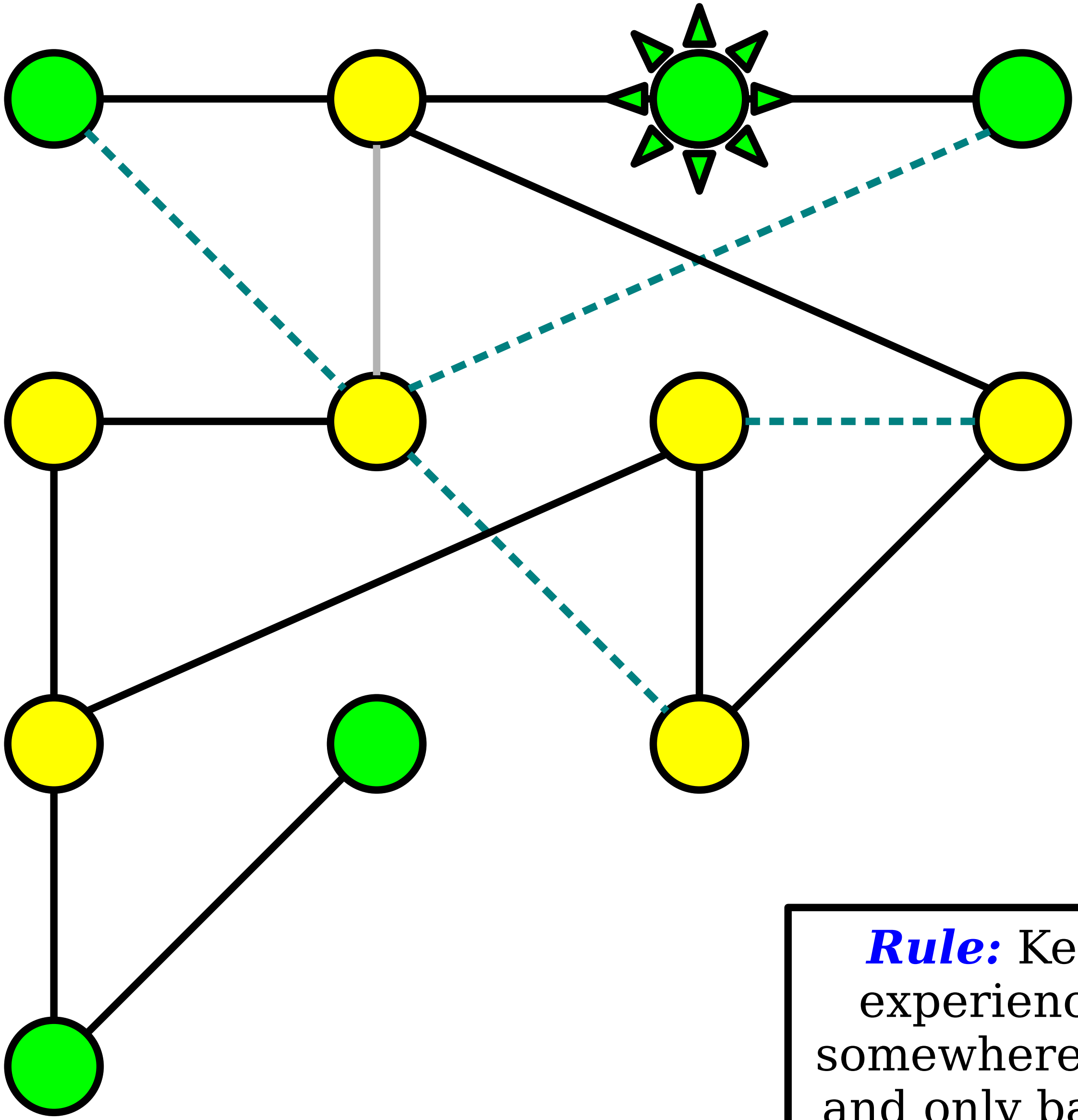
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



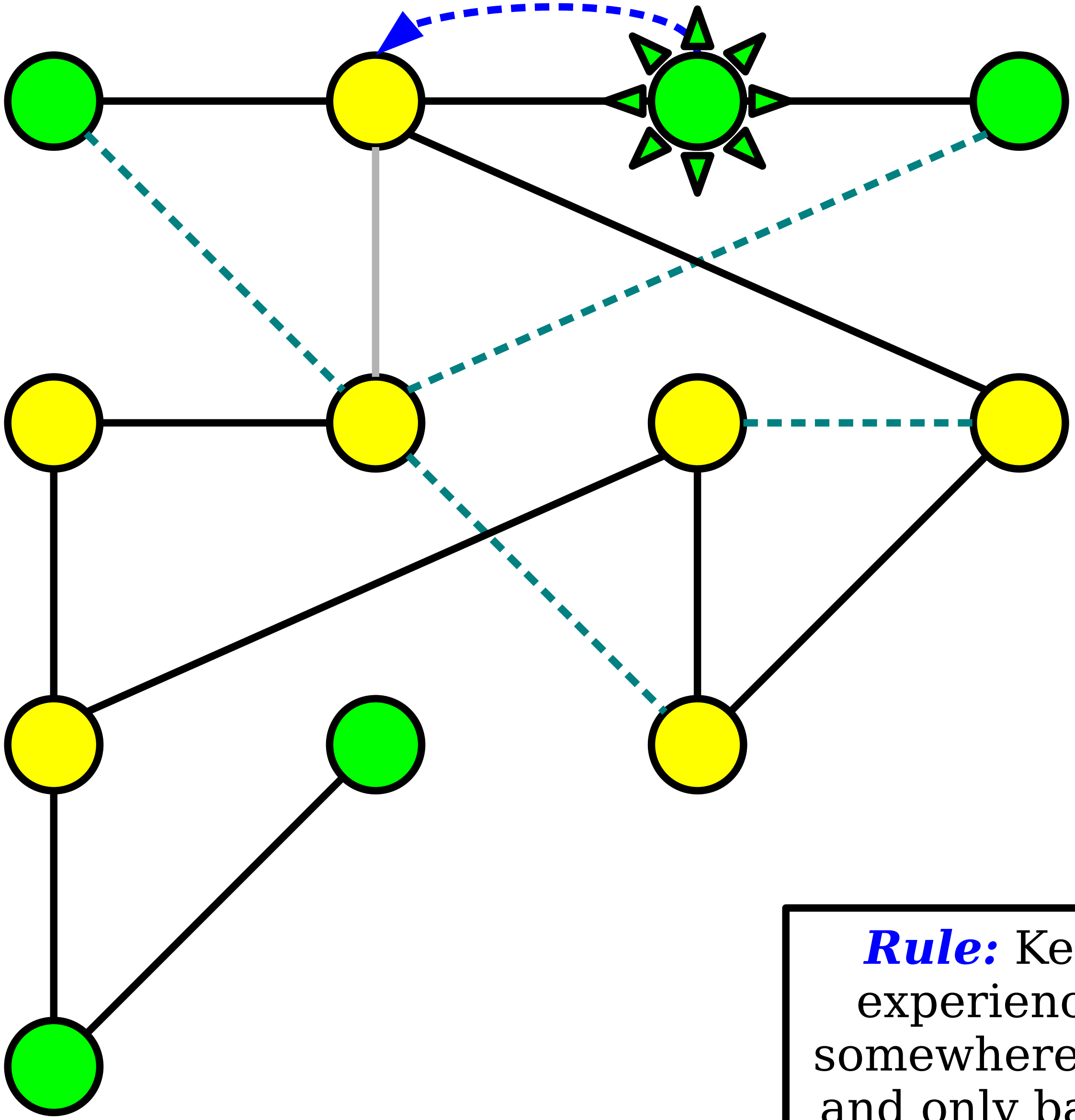
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



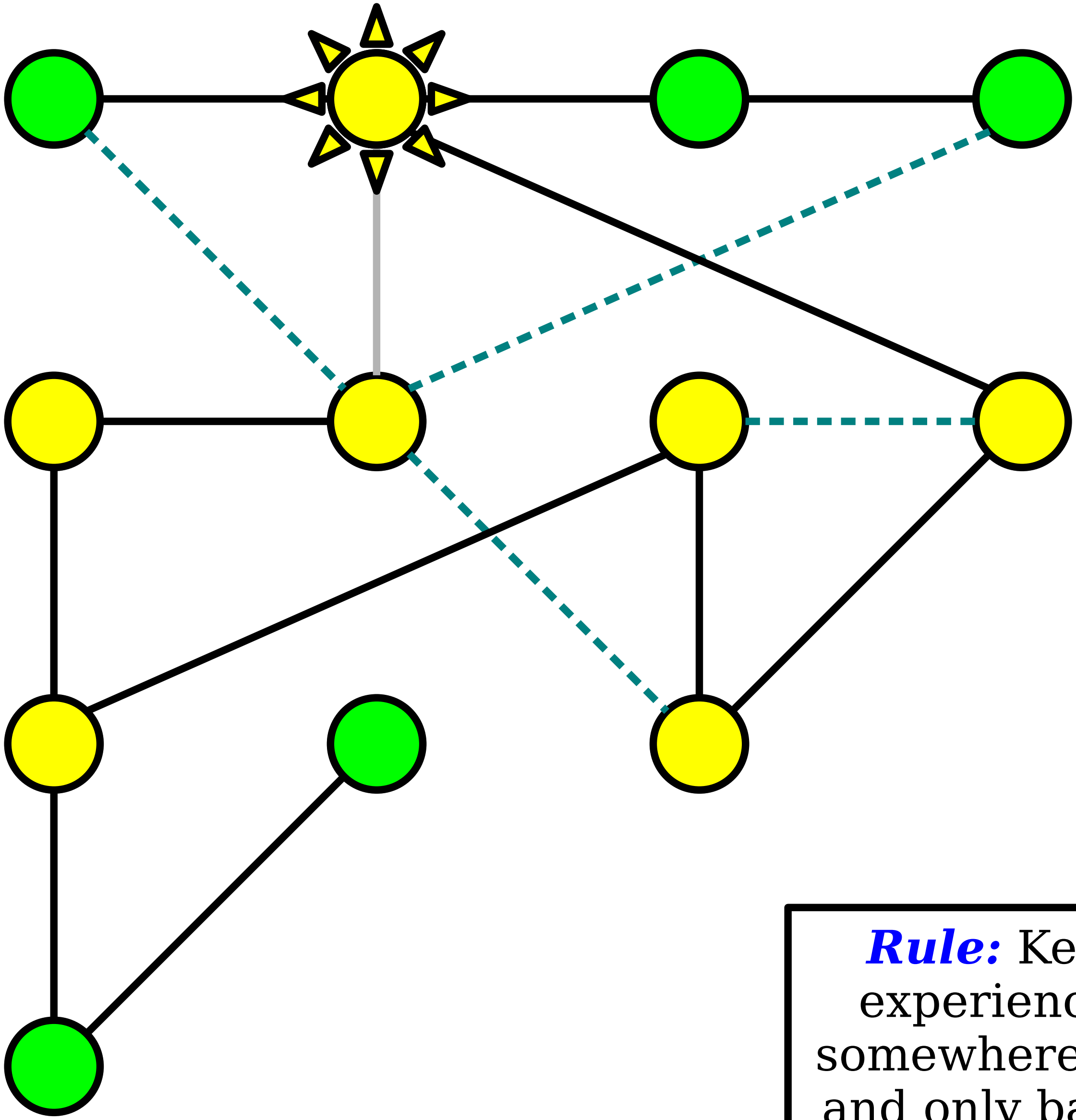
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



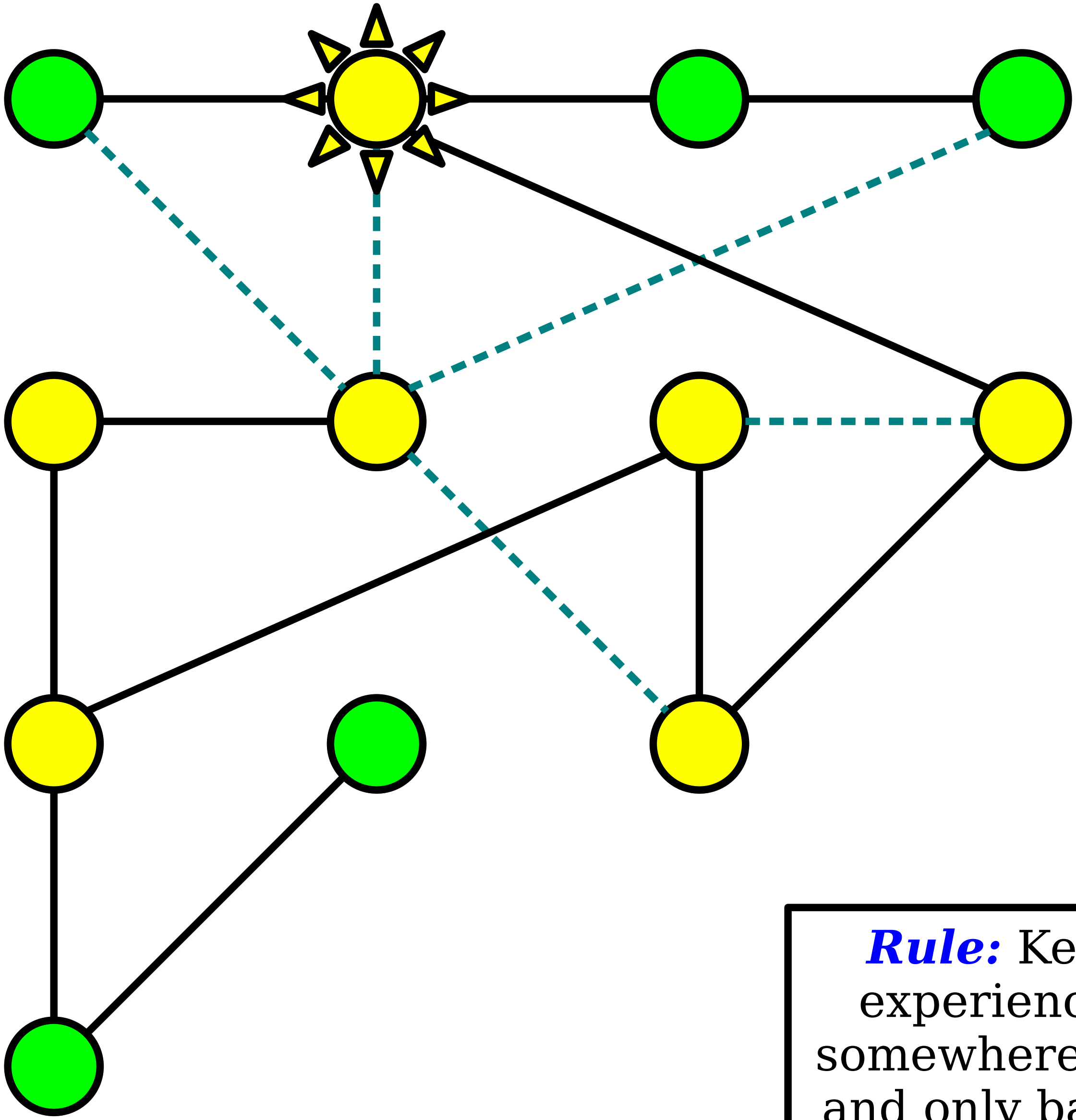
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



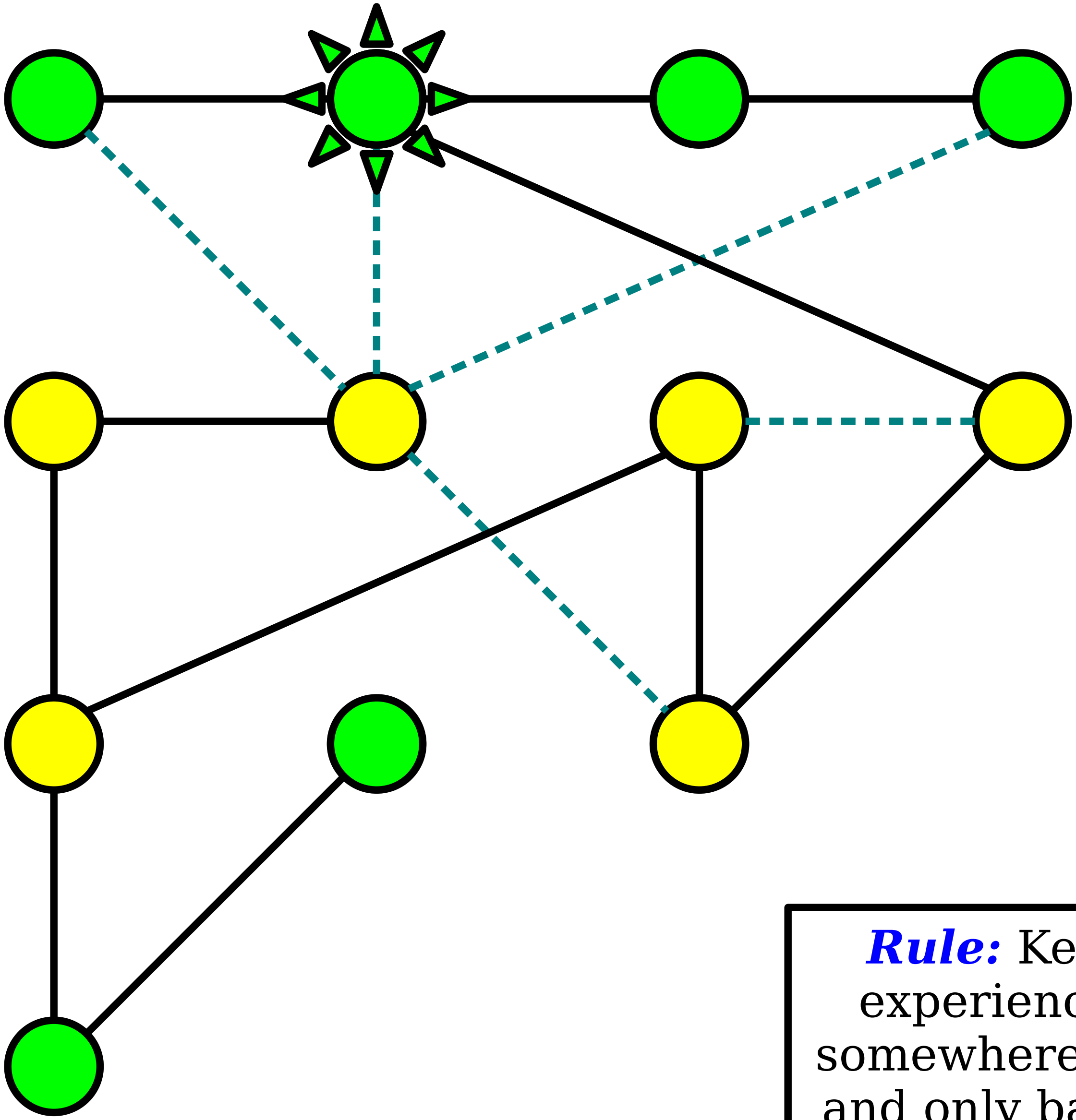
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



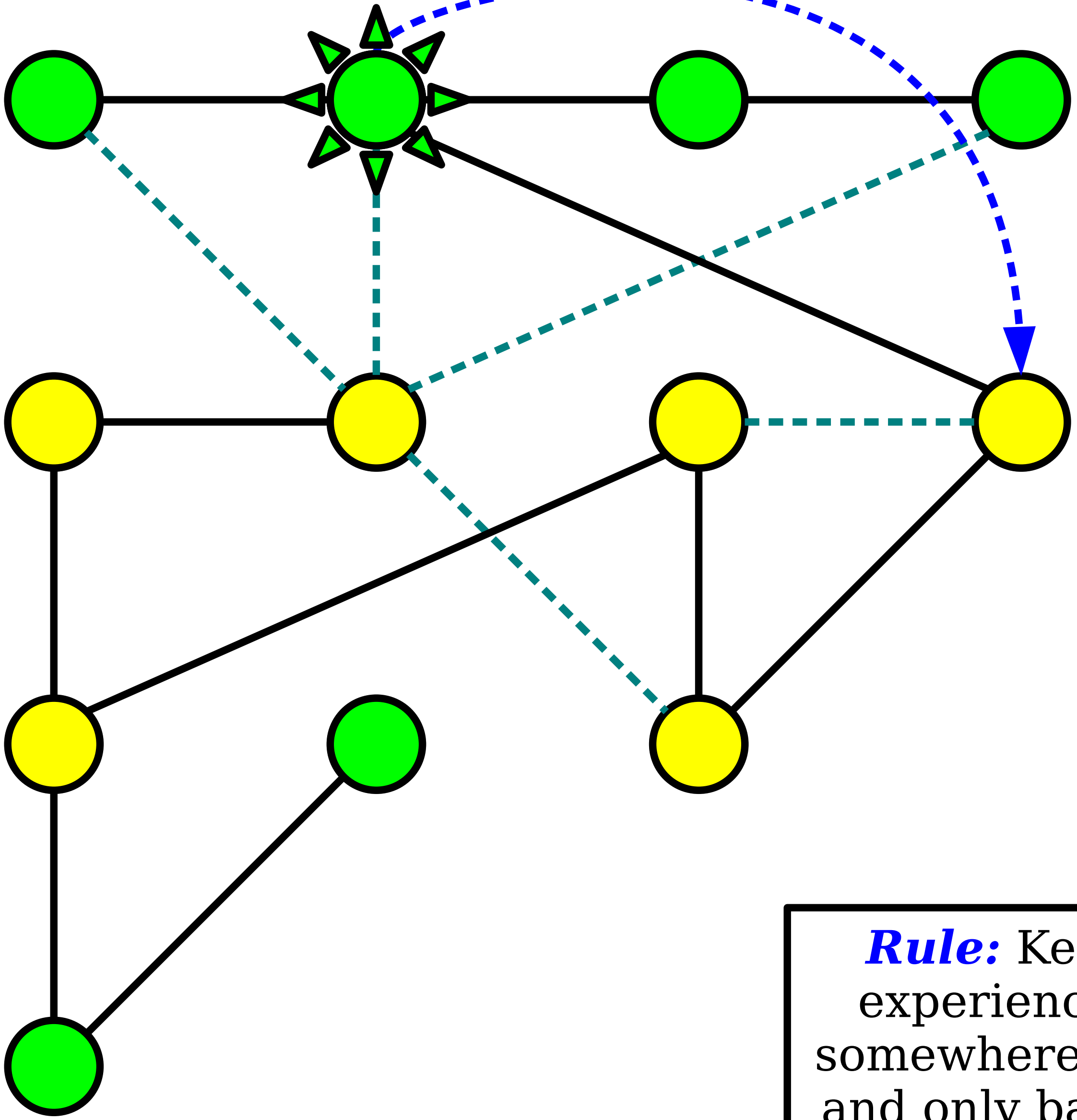
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



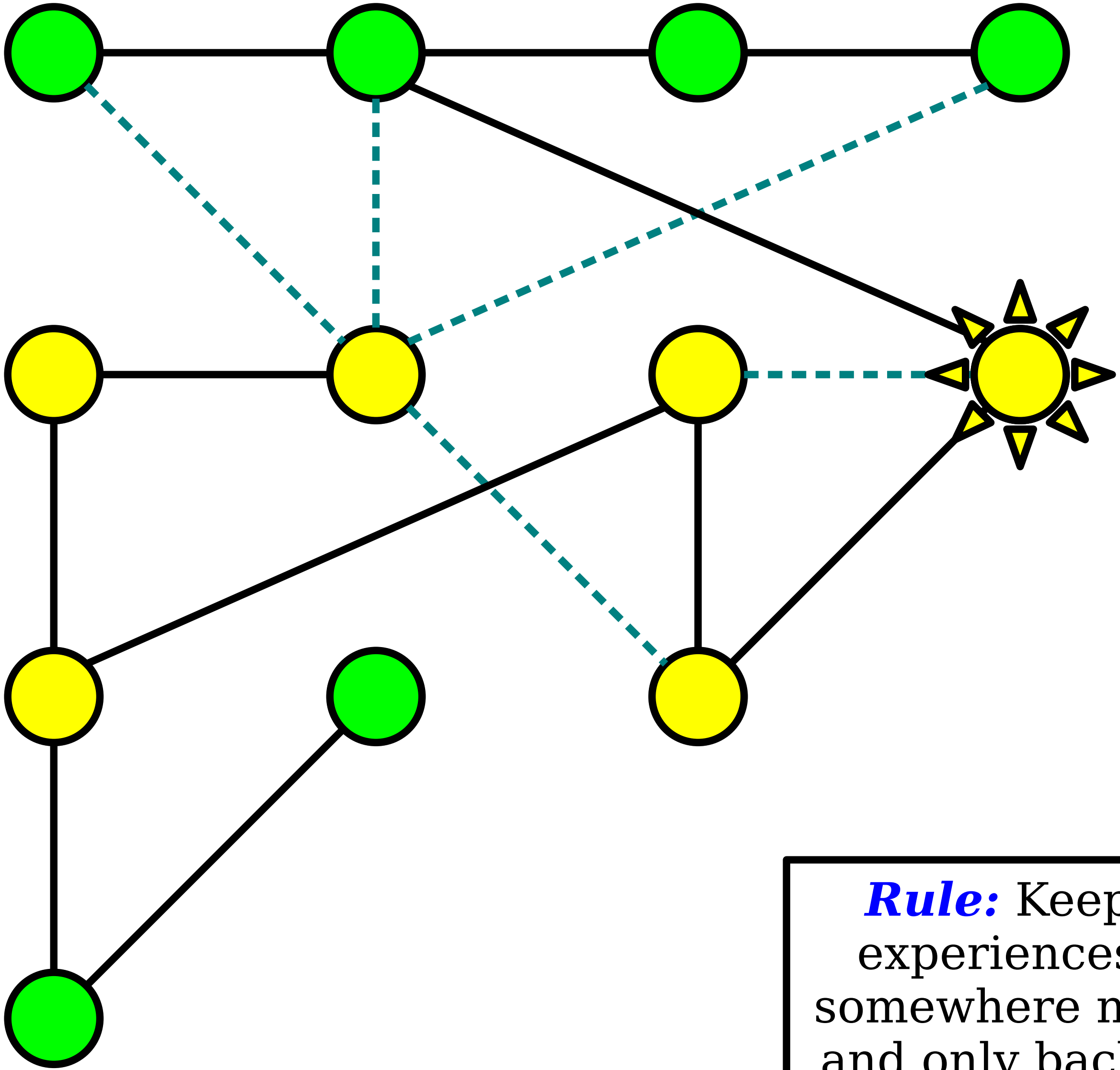
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



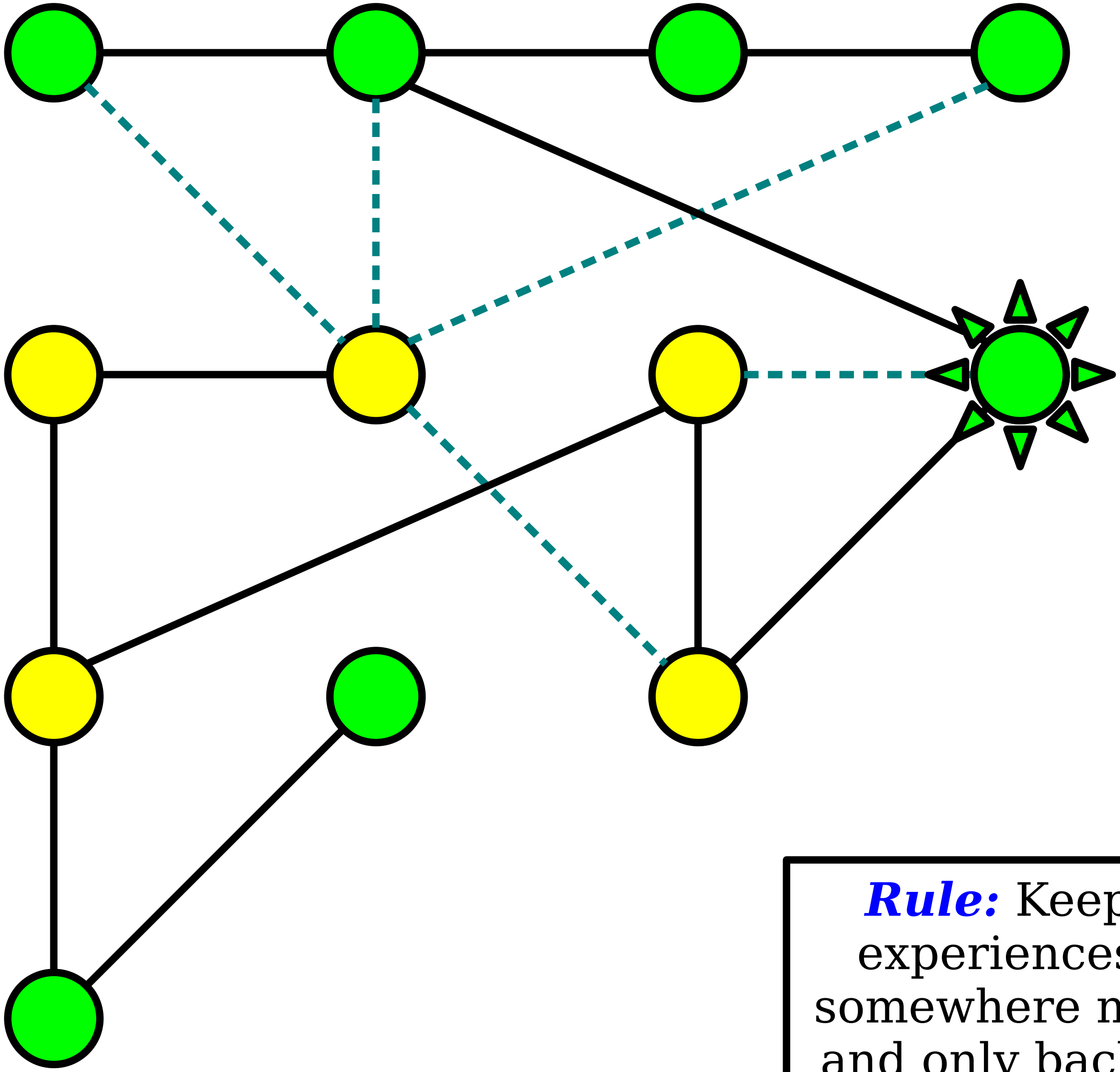
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



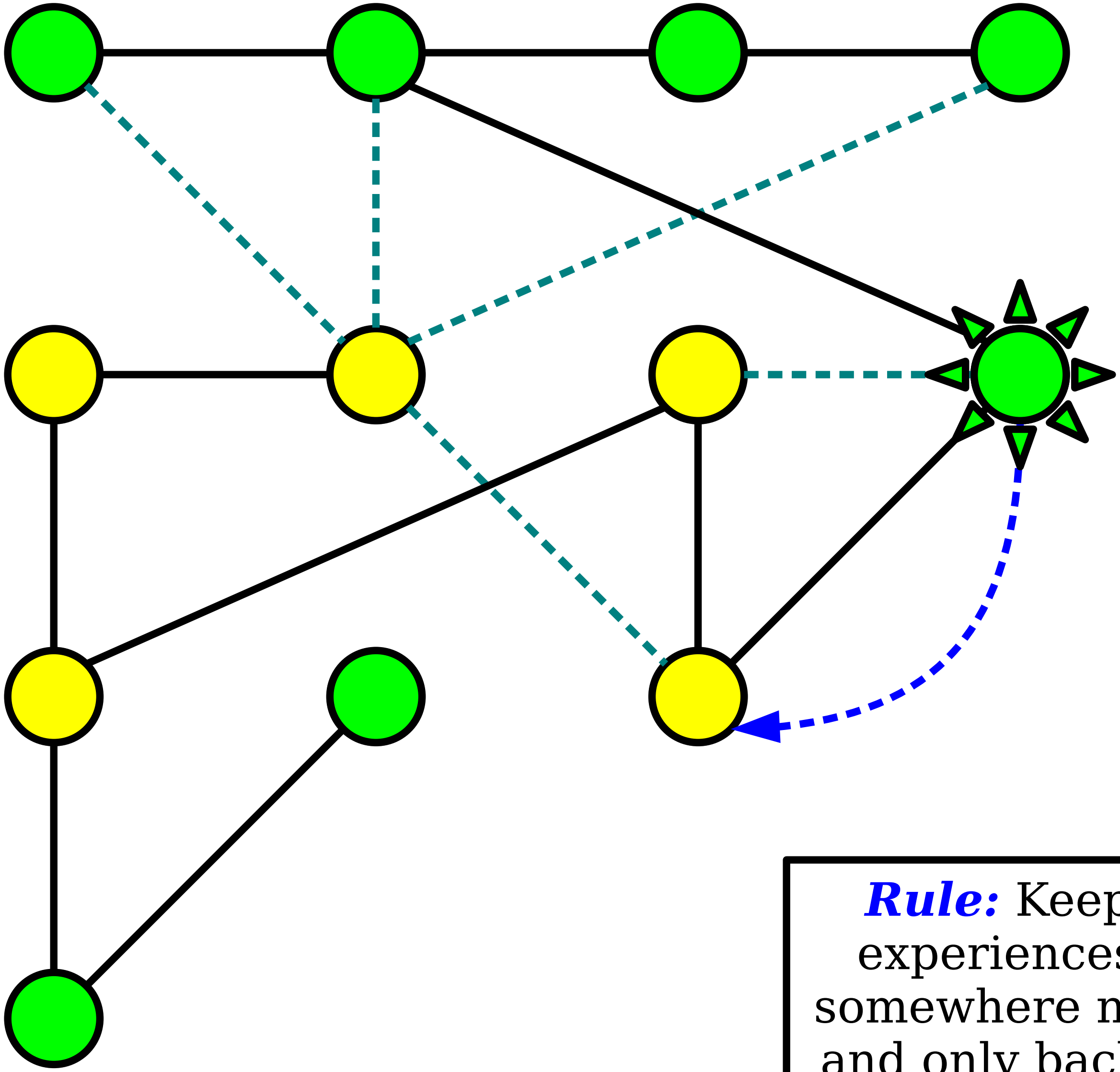
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



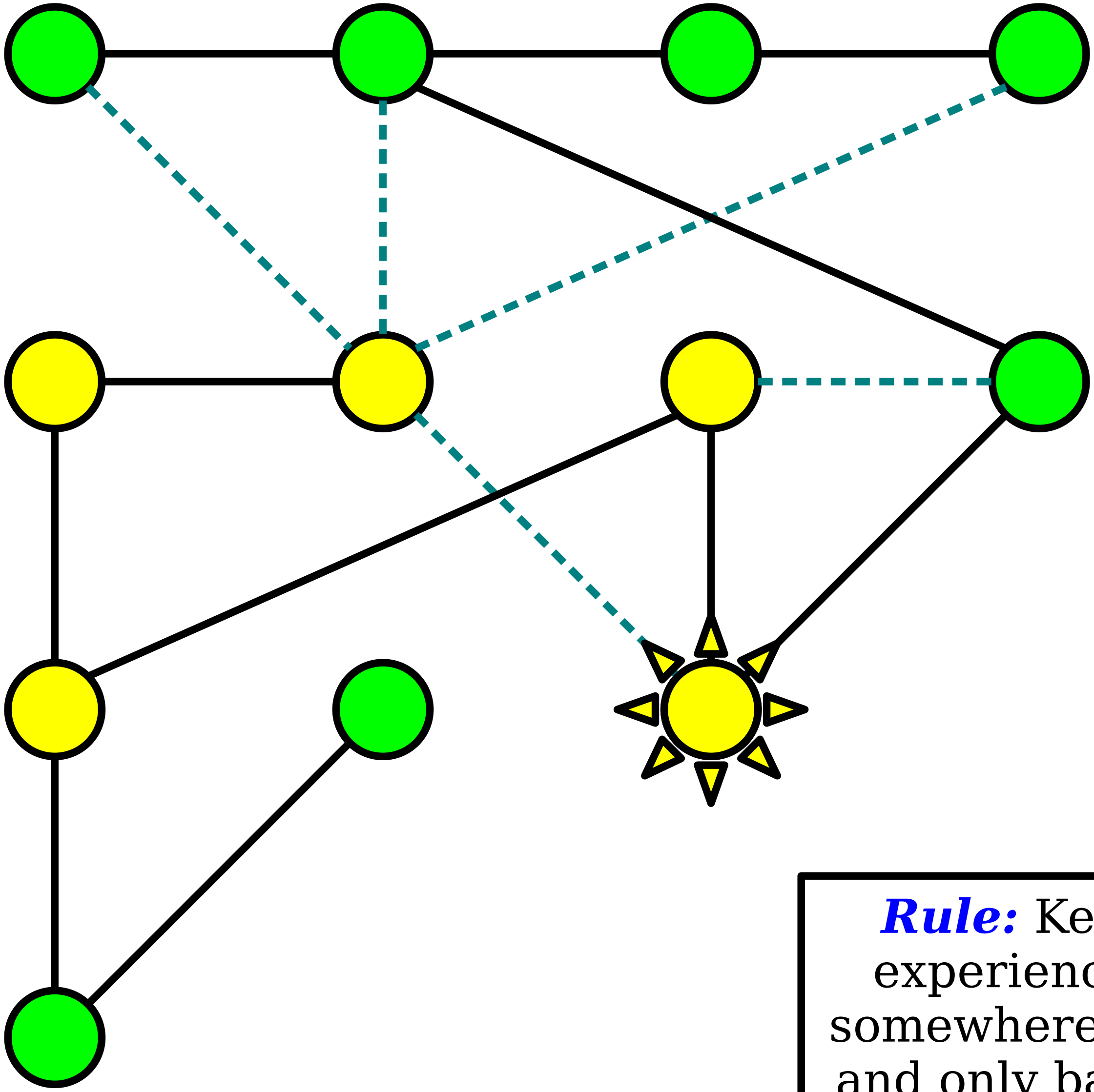
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



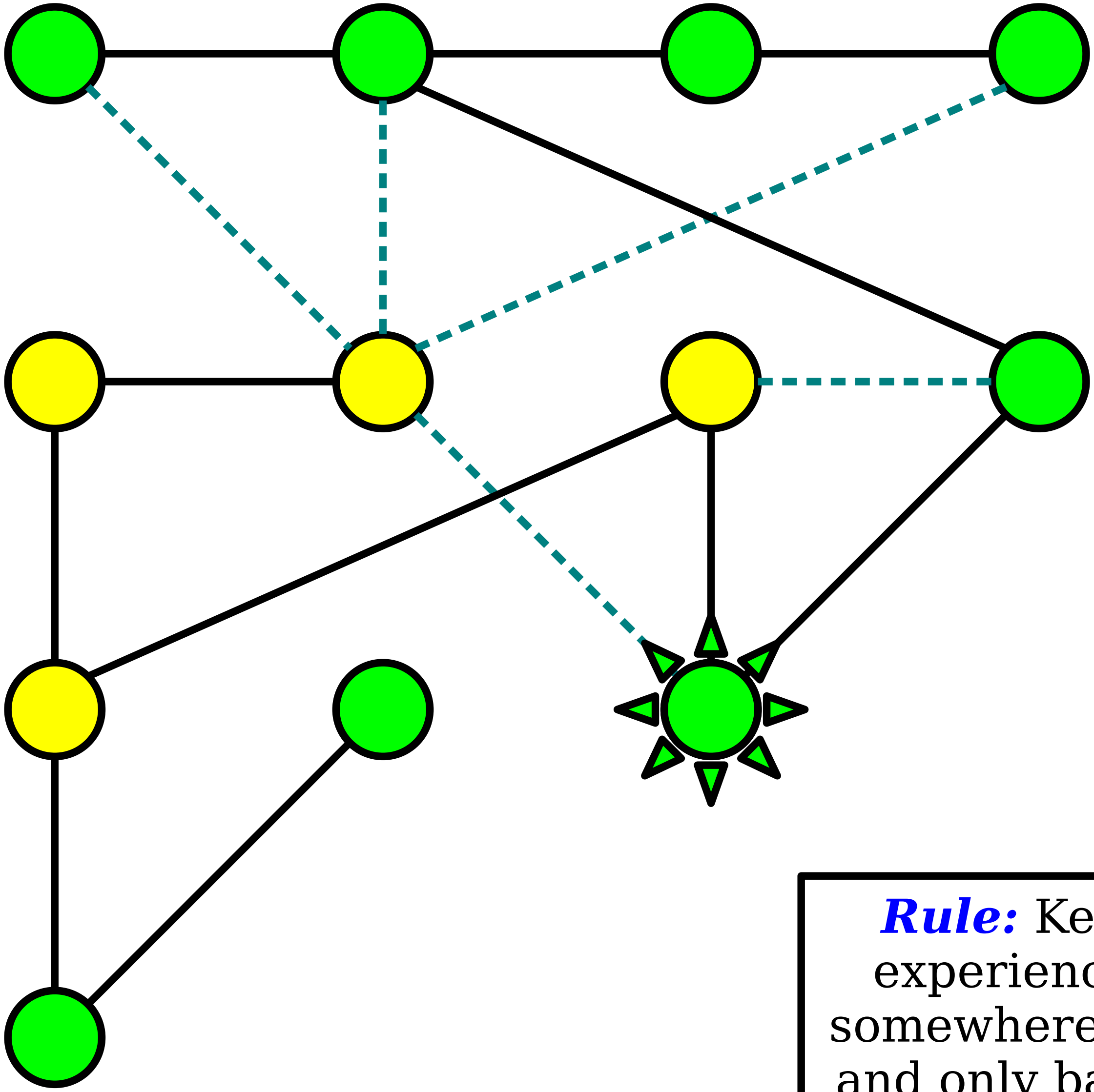
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



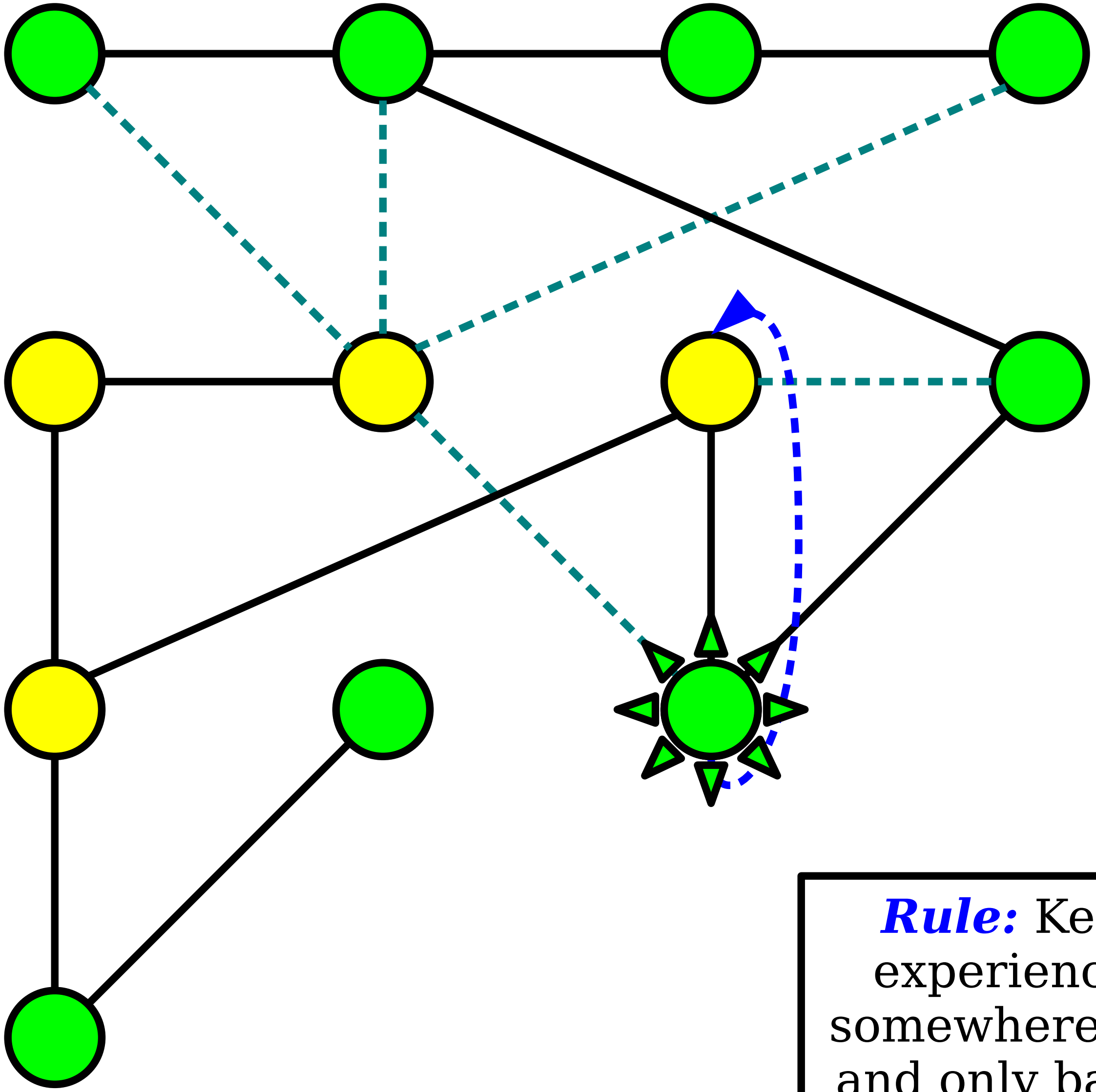
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there’s nothing new to see.

Depth-First Search



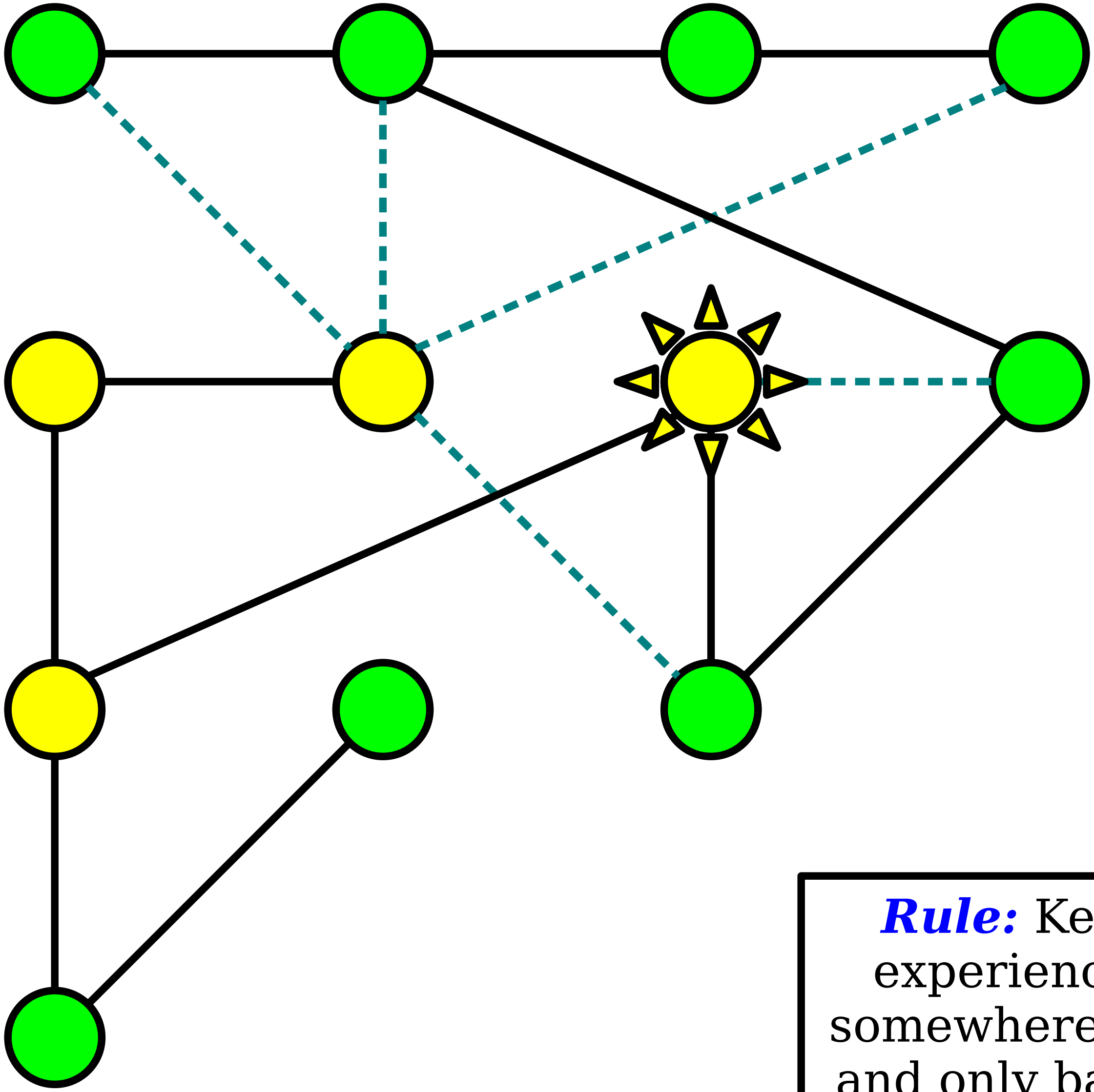
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there’s nothing new to see.

Depth-First Search



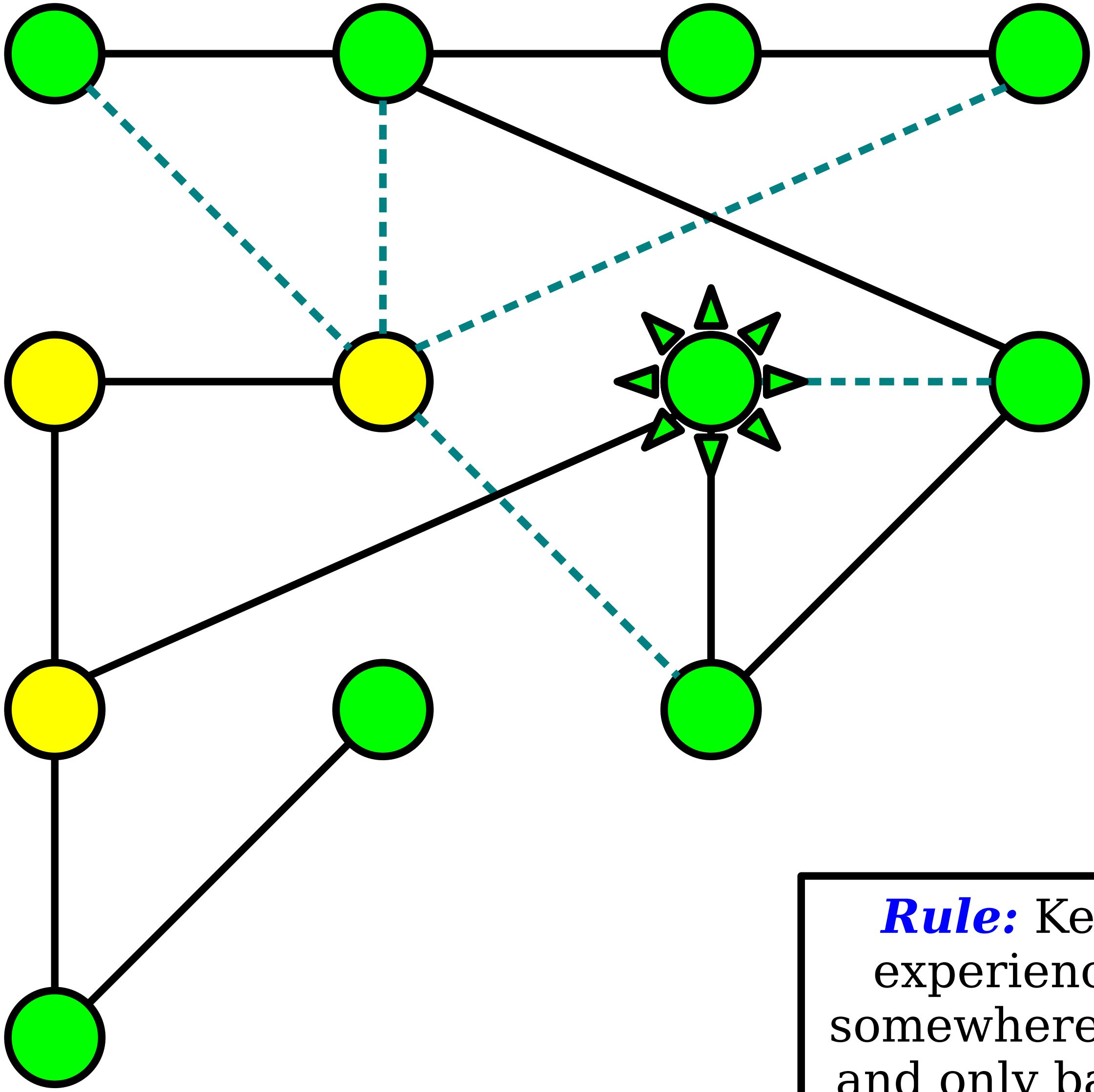
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



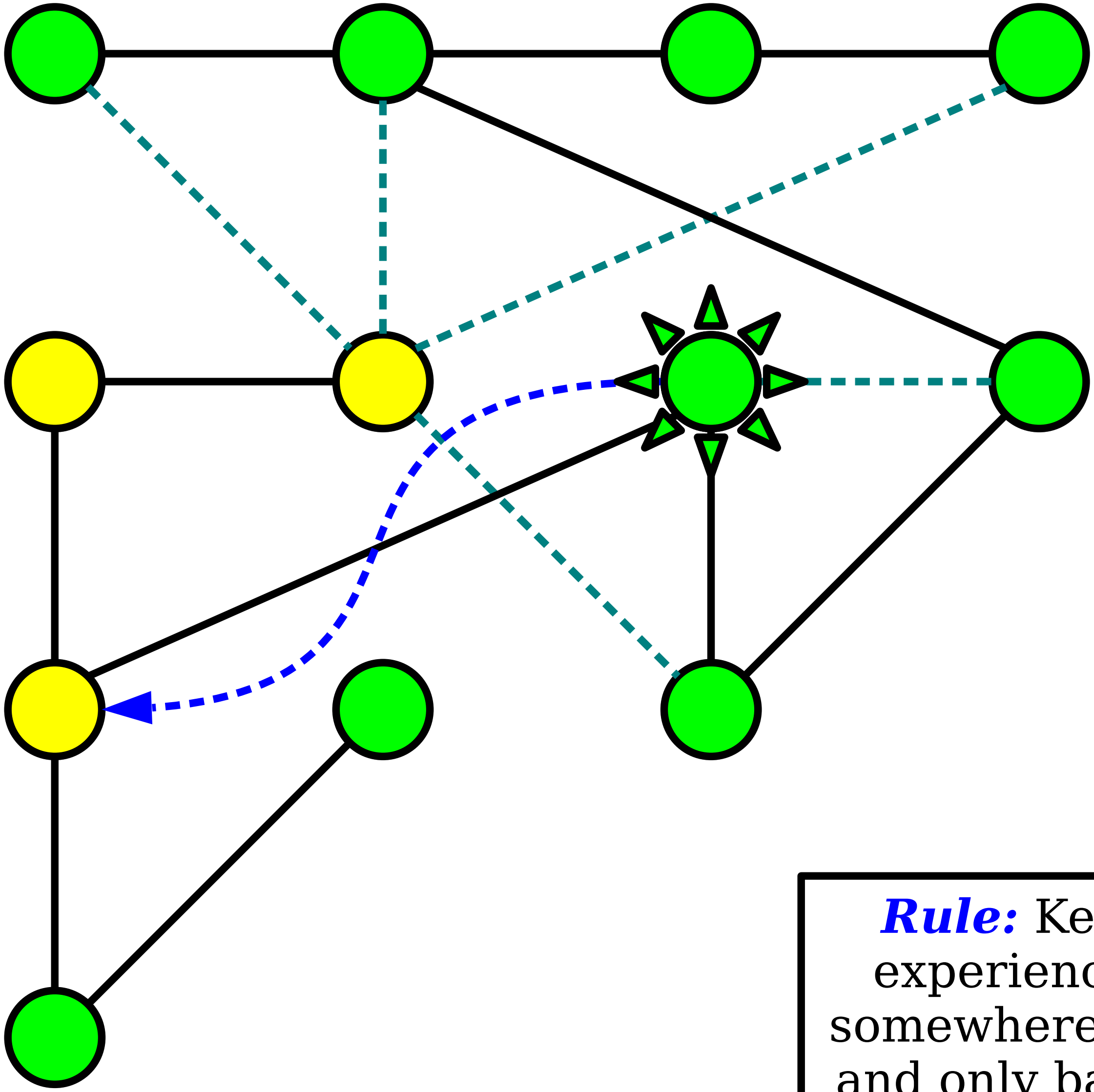
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there’s nothing new to see.

Depth-First Search



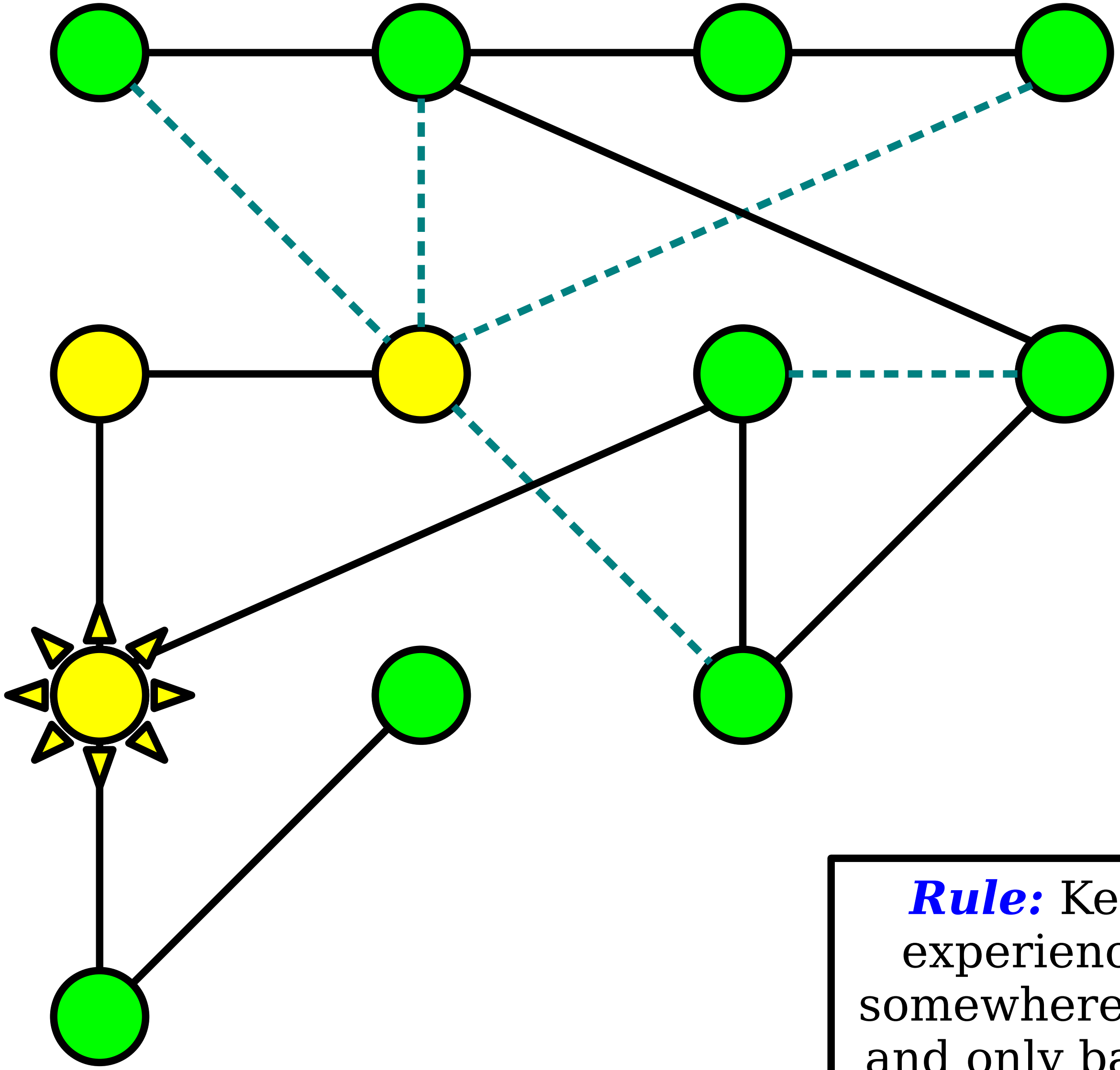
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



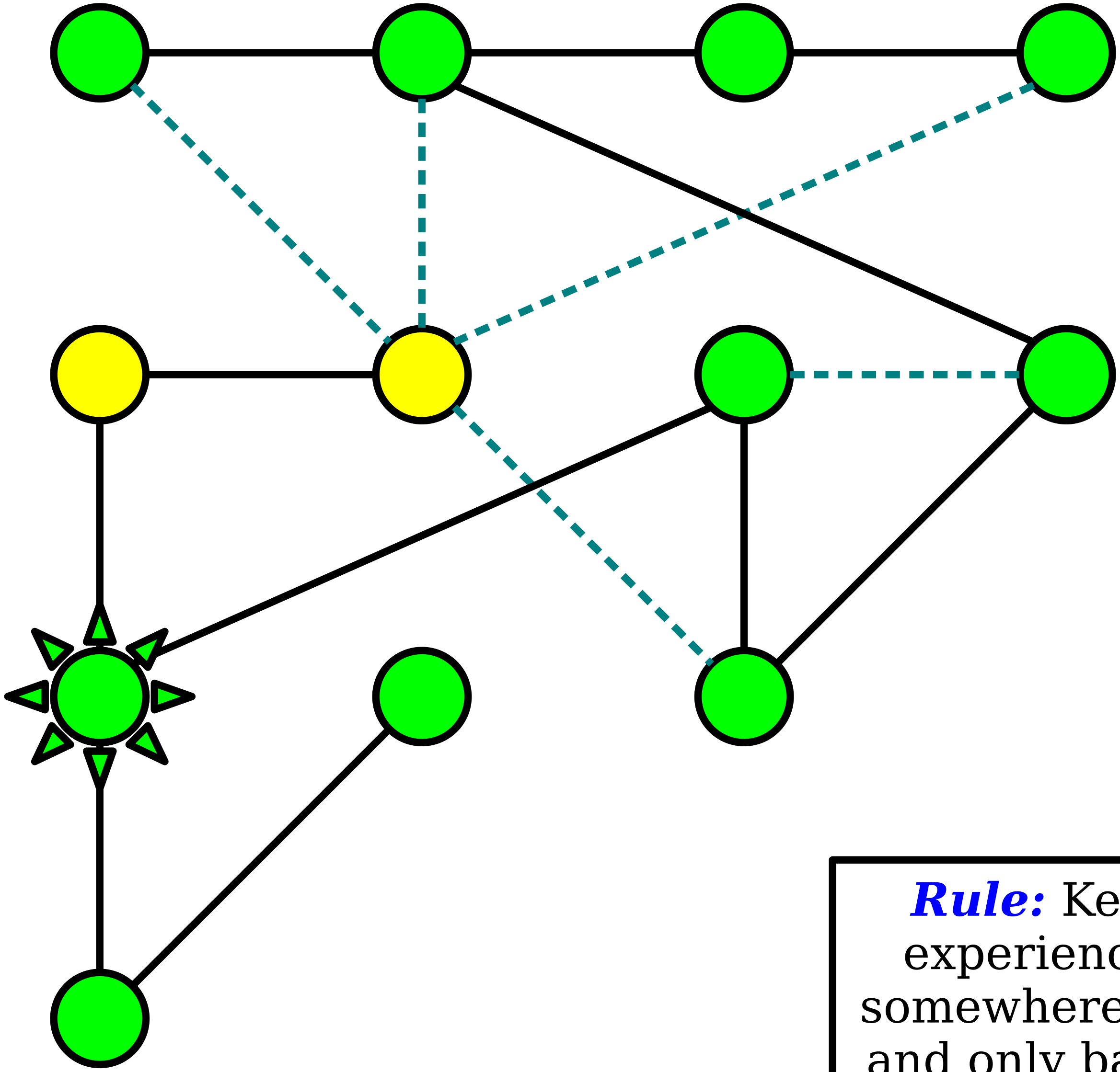
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



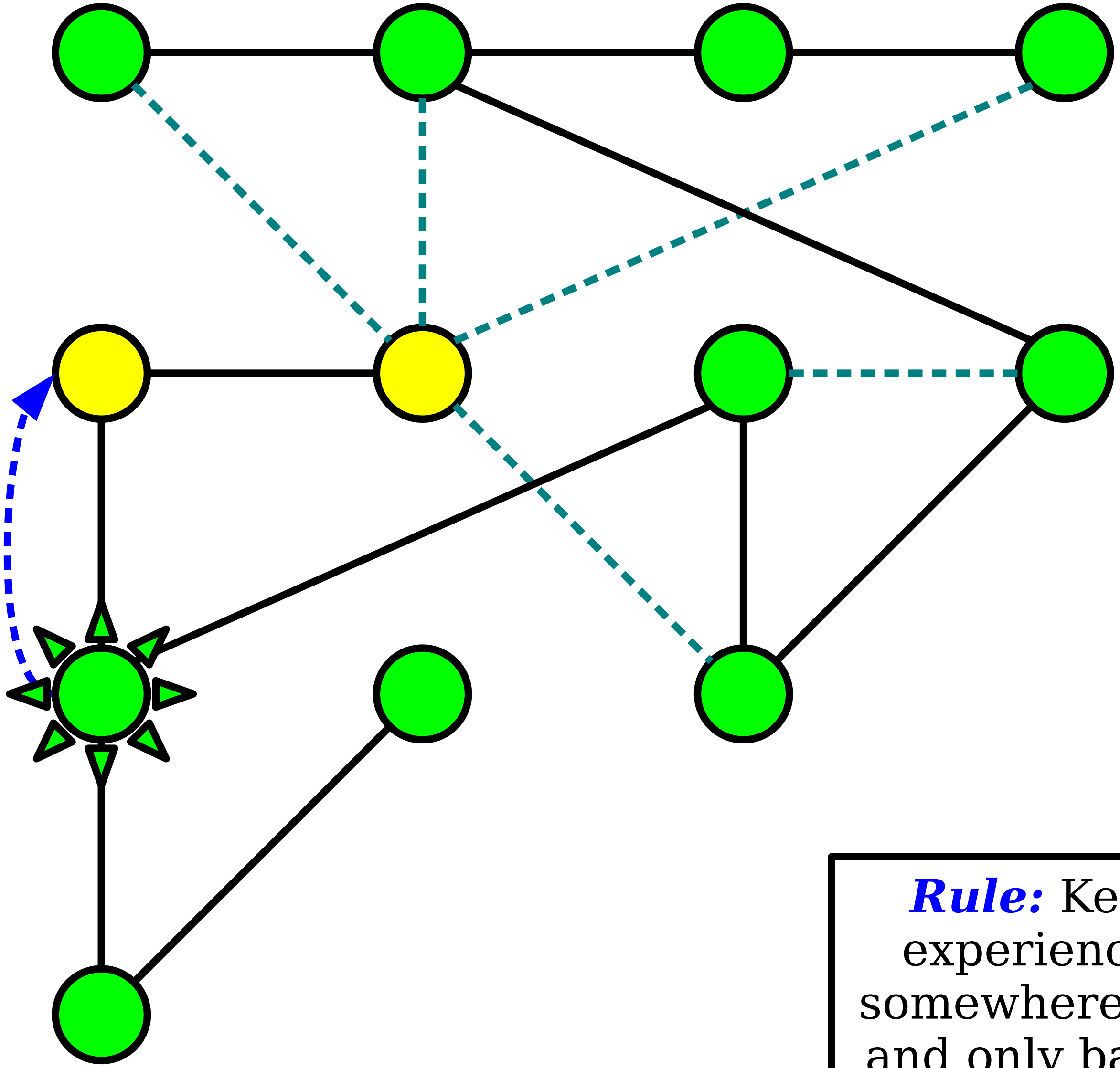
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



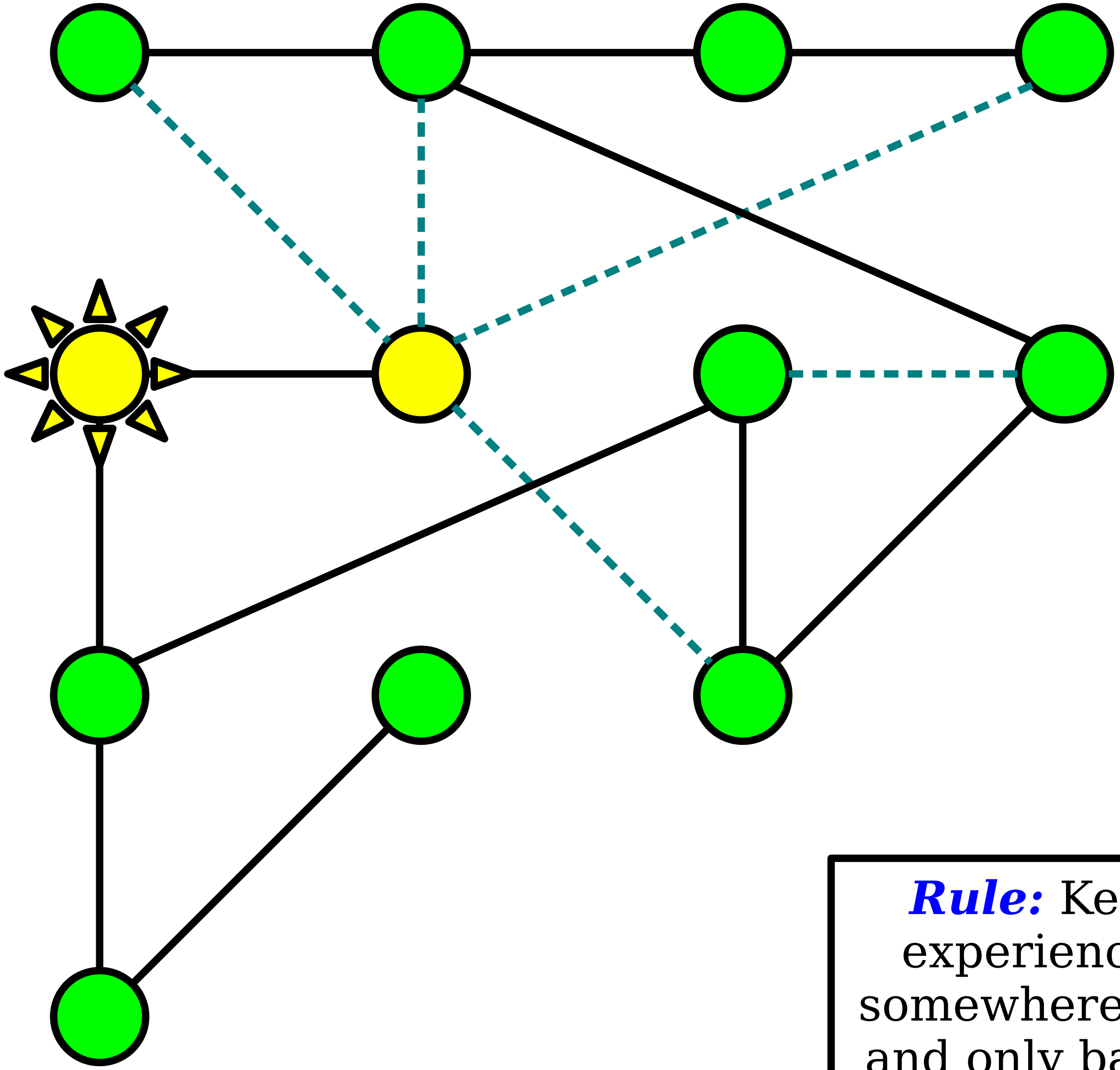
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



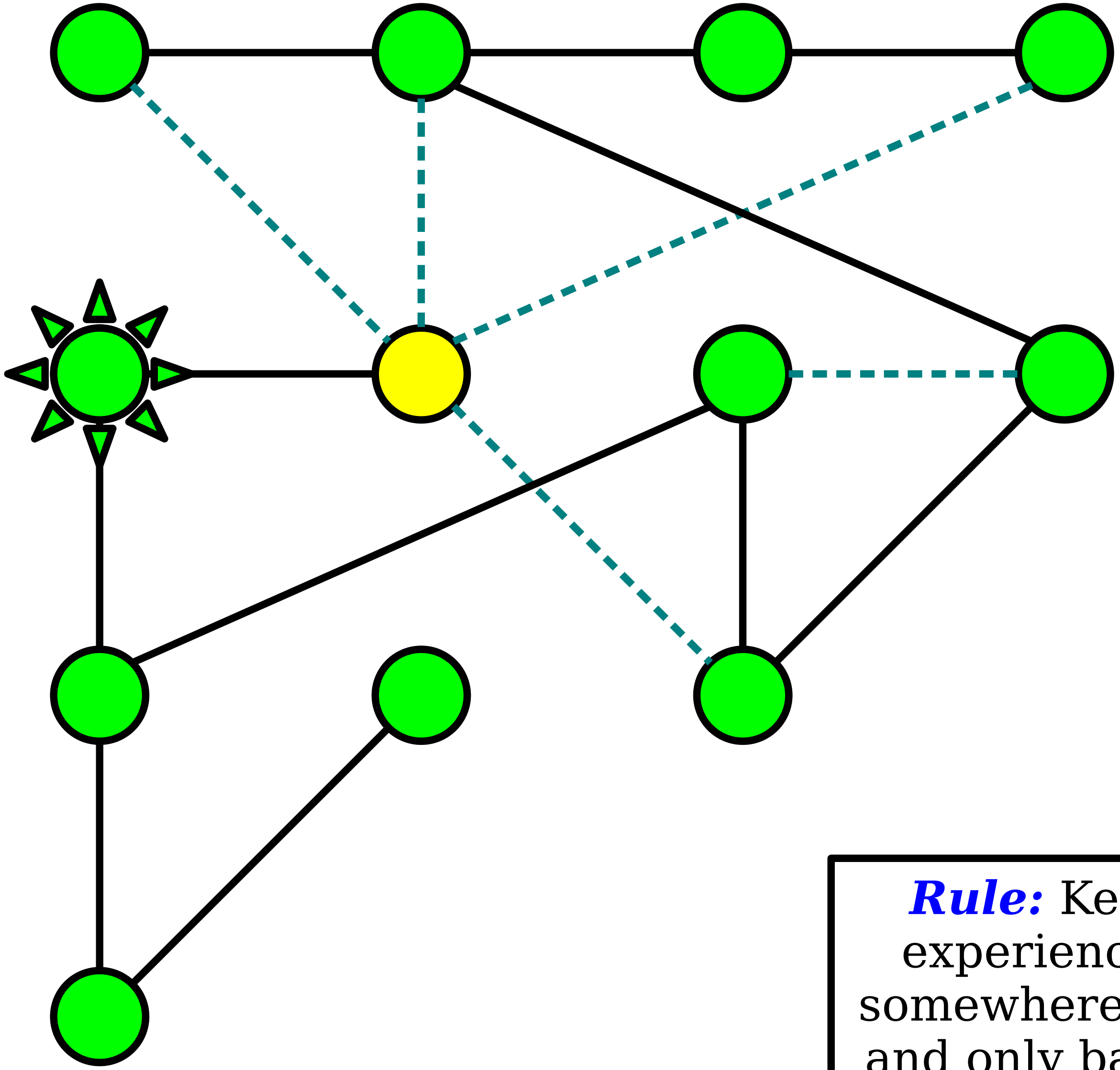
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



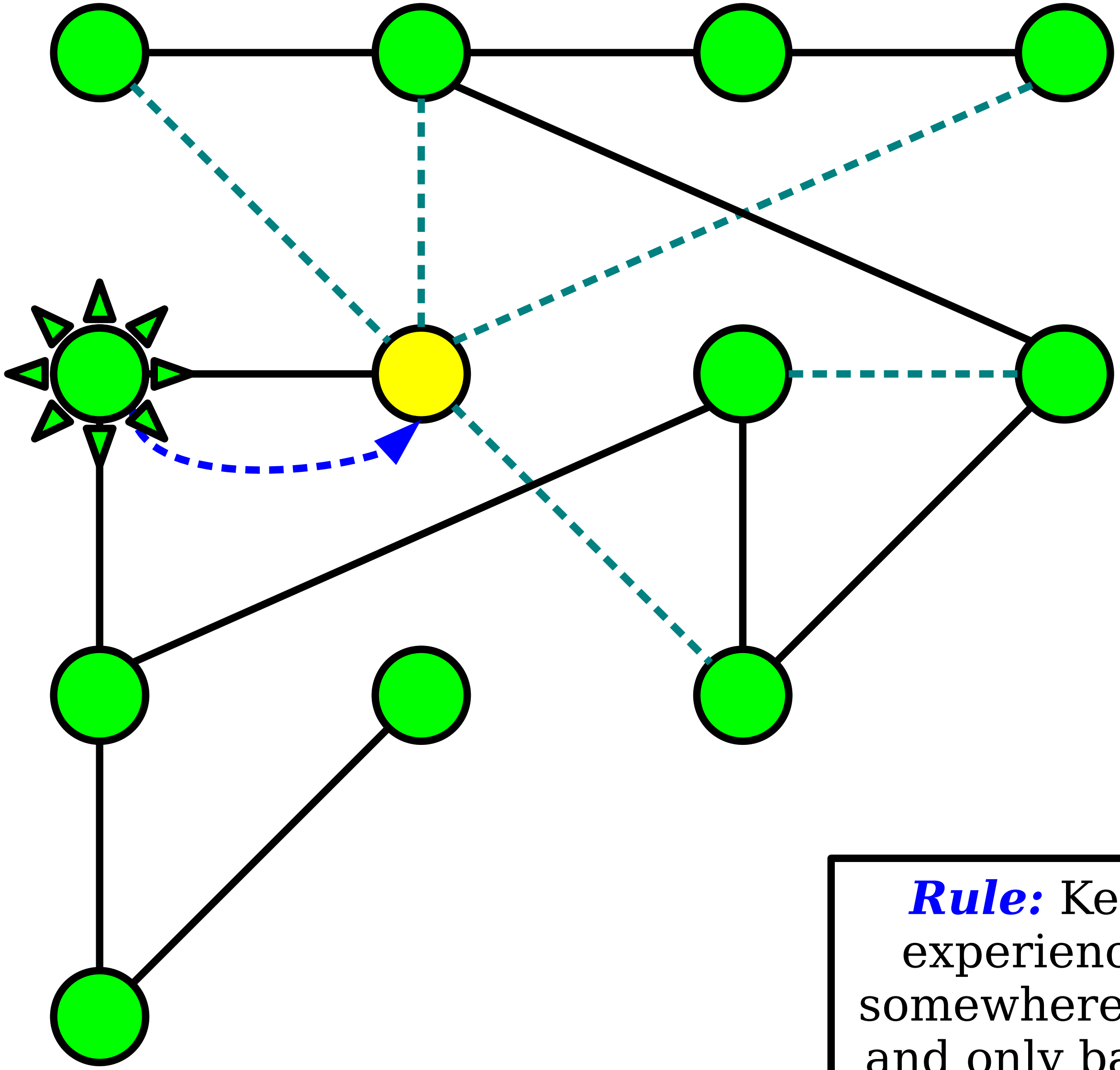
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



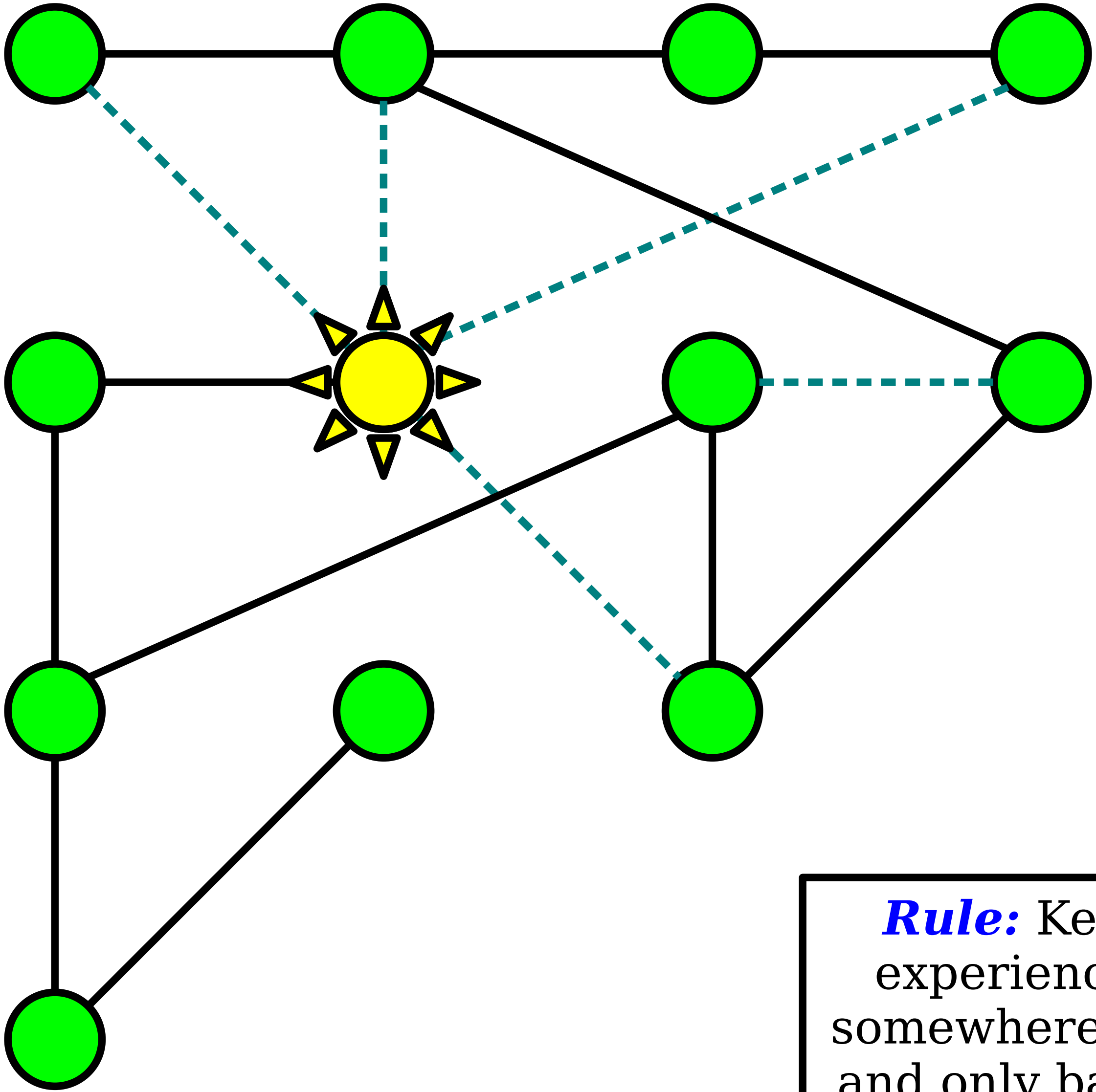
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



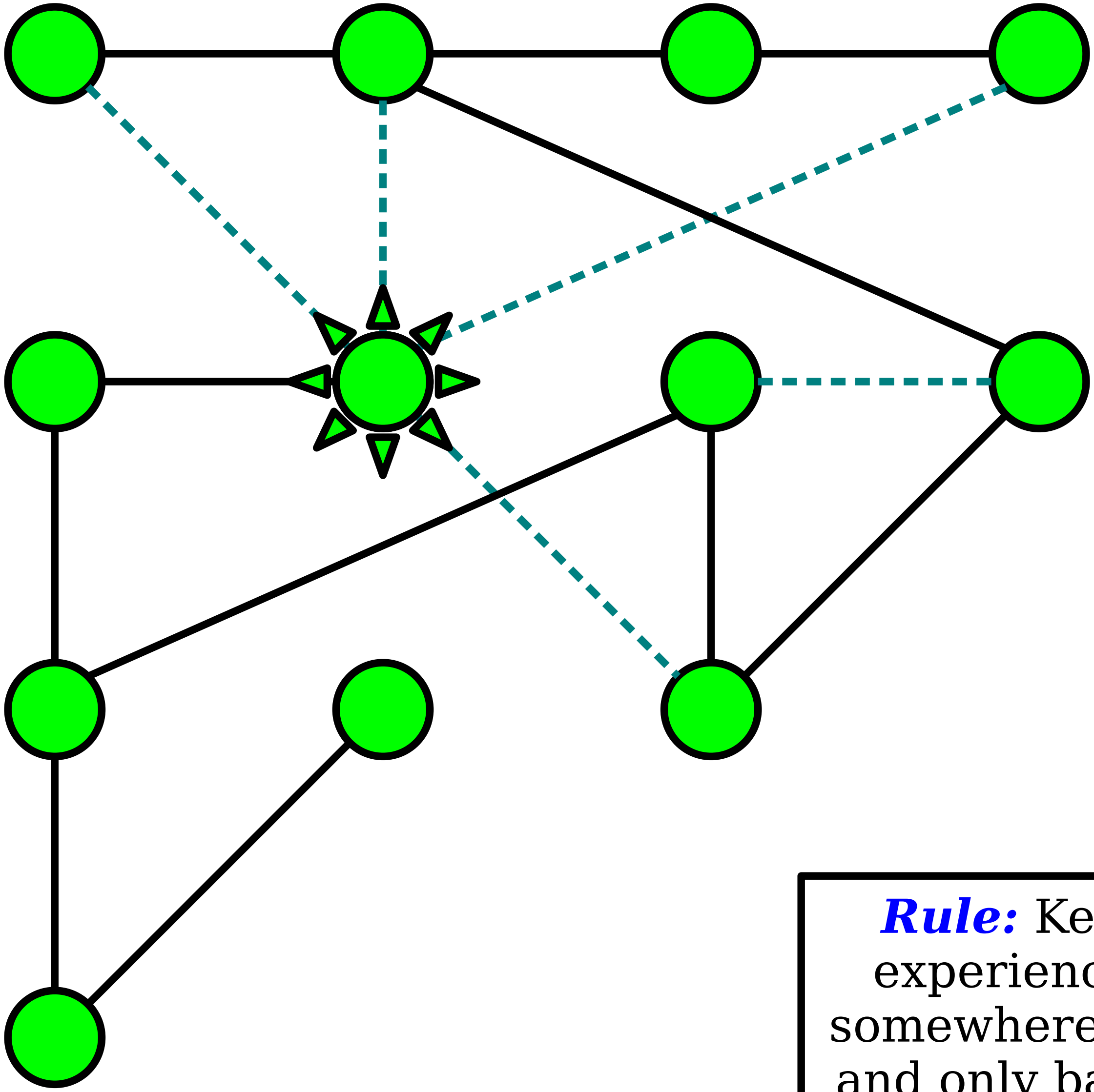
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



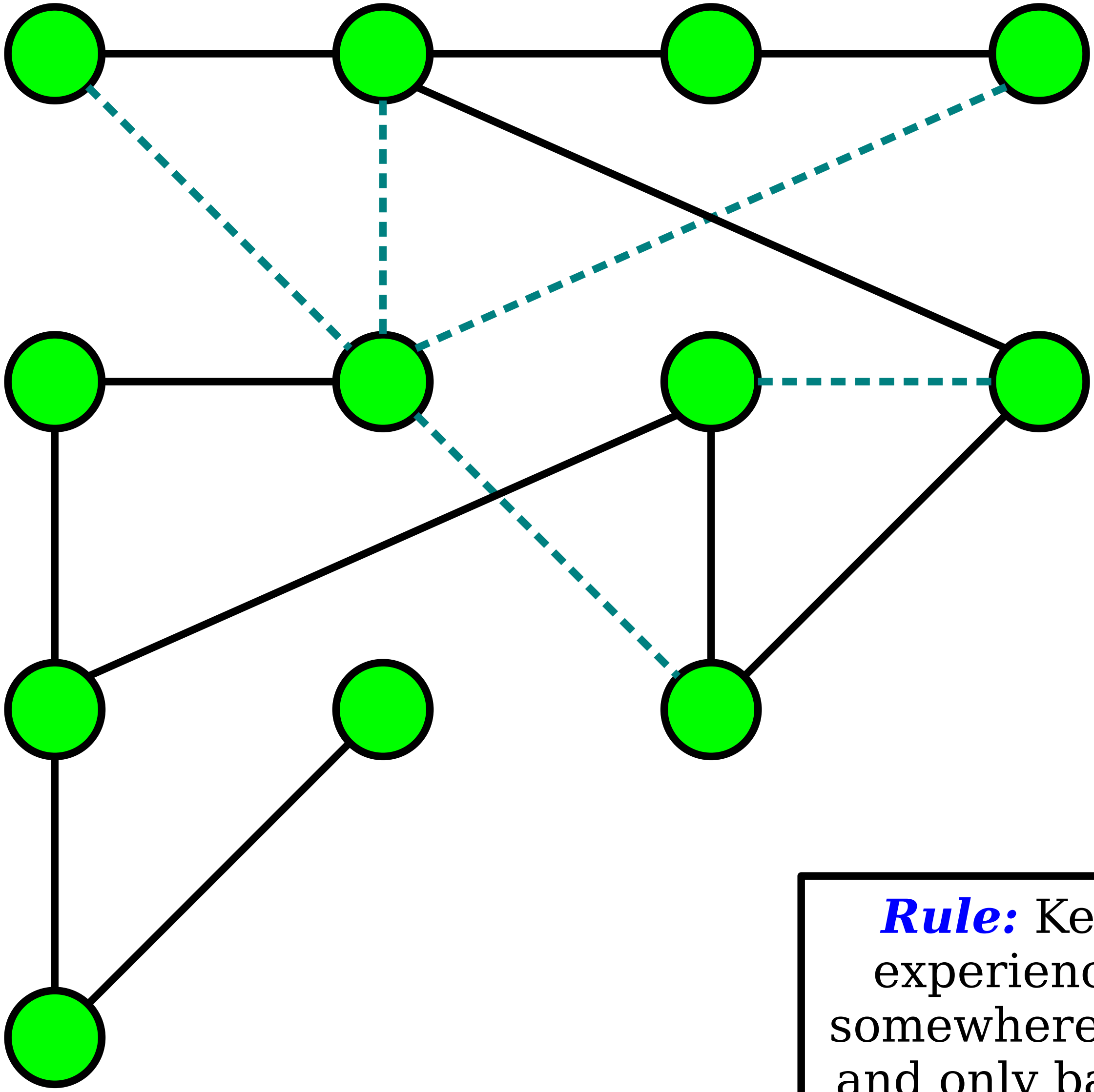
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

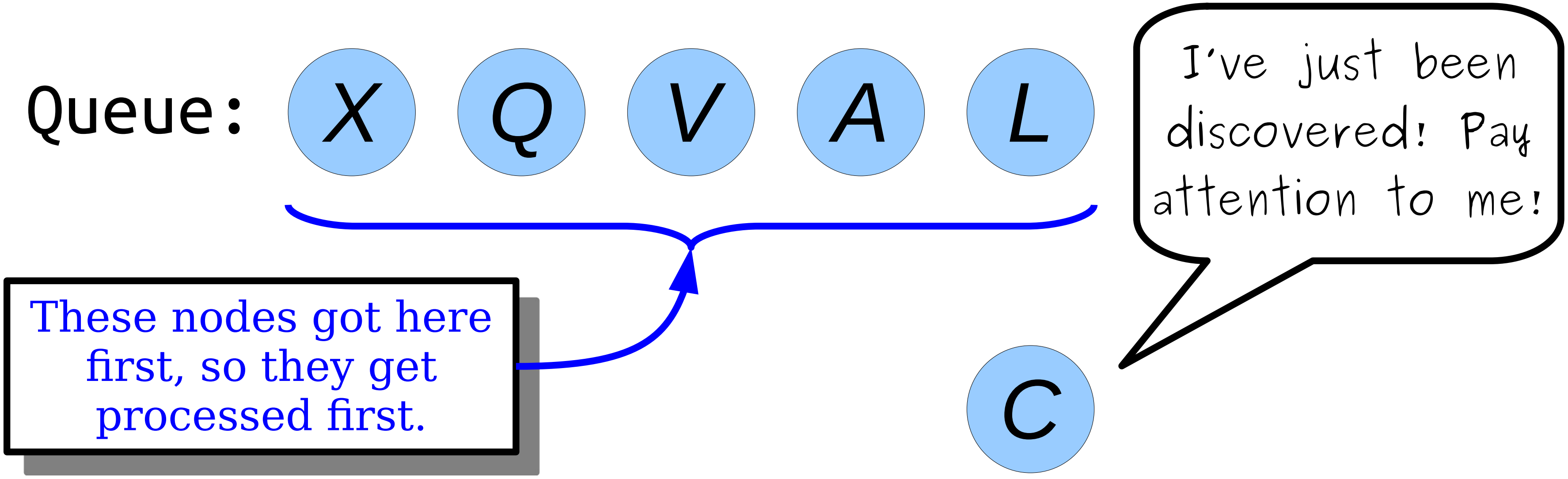
Depth-First Search



Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

How can we implement this?

Breadth-First Search



These nodes got here first, so they get processed first.

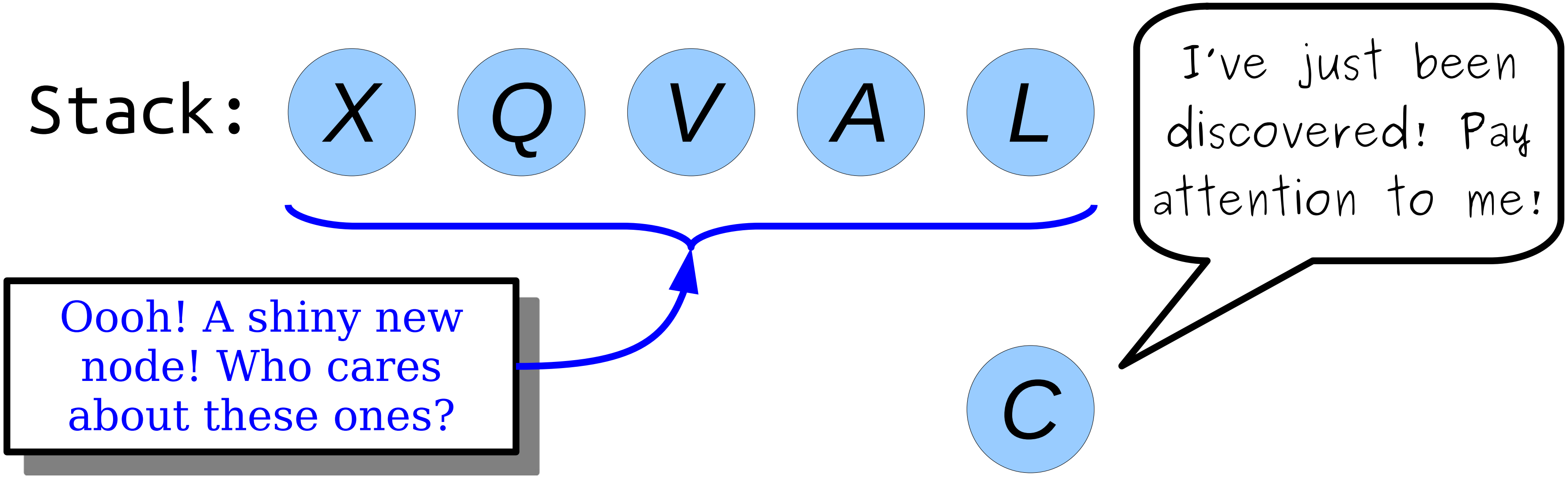
I've just been discovered! Pay attention to me!

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

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

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

Depth-First Search



Oooh! A shiny new node! Who cares about these ones?

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

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

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

For Comparison

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

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

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

For Comparison

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

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

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

```
dfs-from(node v) {
    if this is first time we've called dfs-from(v):
        process node v
        for each node adjacent to v:
            call dfs-from on that node
}
```

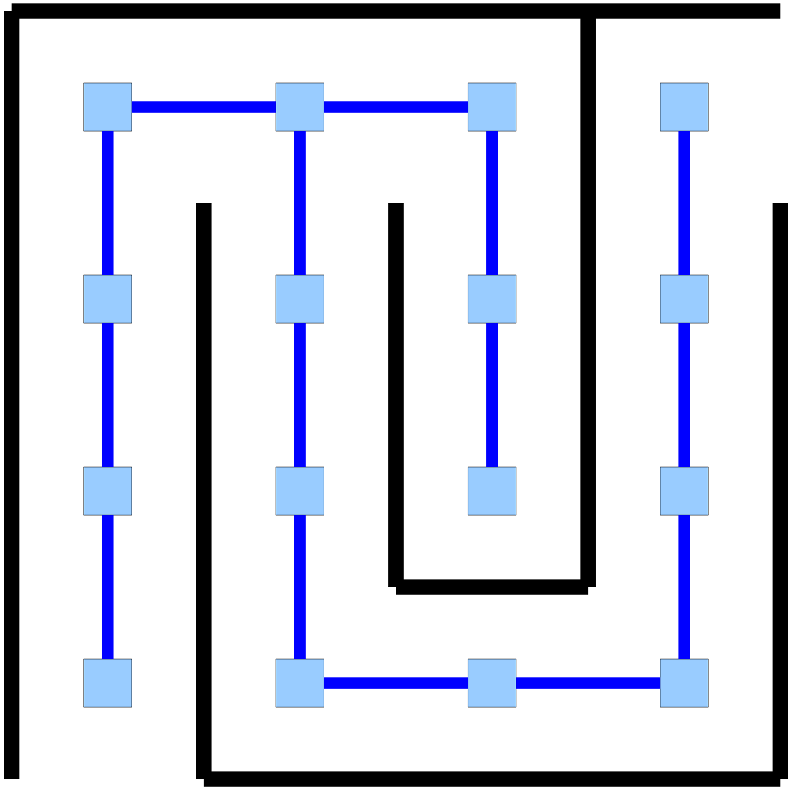
When you see a stack-based algorithm,
think recursion!

BFS and DFS

- Running BFS or DFS from a node in a graph will visit the same set of nodes, but probably in a different order.
- BFS will visit nodes in increasing order of distance.
- DFS does visit nodes in *some* interesting order, but not order of distance.
    - Take CS161 for more details!

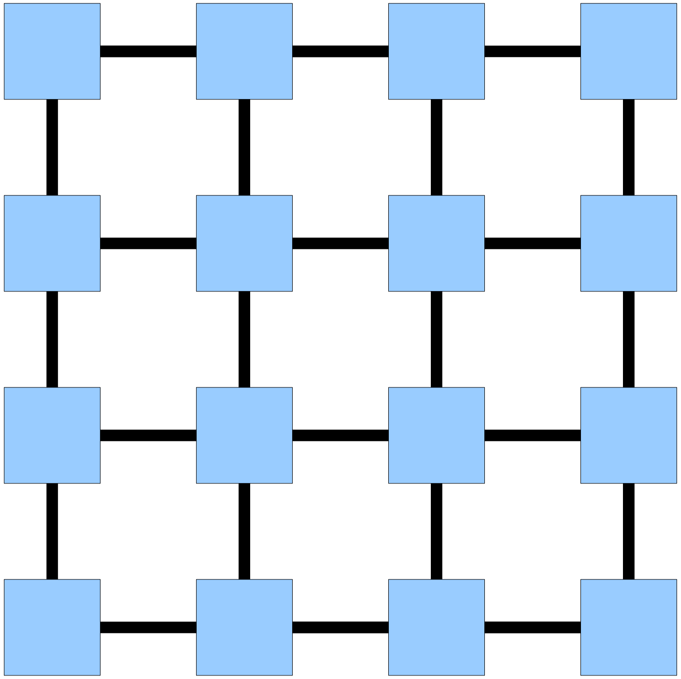
A Whimsical Application

Mazes as Graphs



Creating a Maze with DFS

- Create a ***grid graph*** of the appropriate size.



- Starting at any node, run a depth-first search, choosing neighbor orderings at random.
- The resulting DFS tree is a maze with one solution.

Next Time

- ***Minimum Spanning Trees***
  - How to wire an electrical grid cheaply.
- ***Applications of MSTs***
  - Data clustering, computational biology, and more!