

Graphs

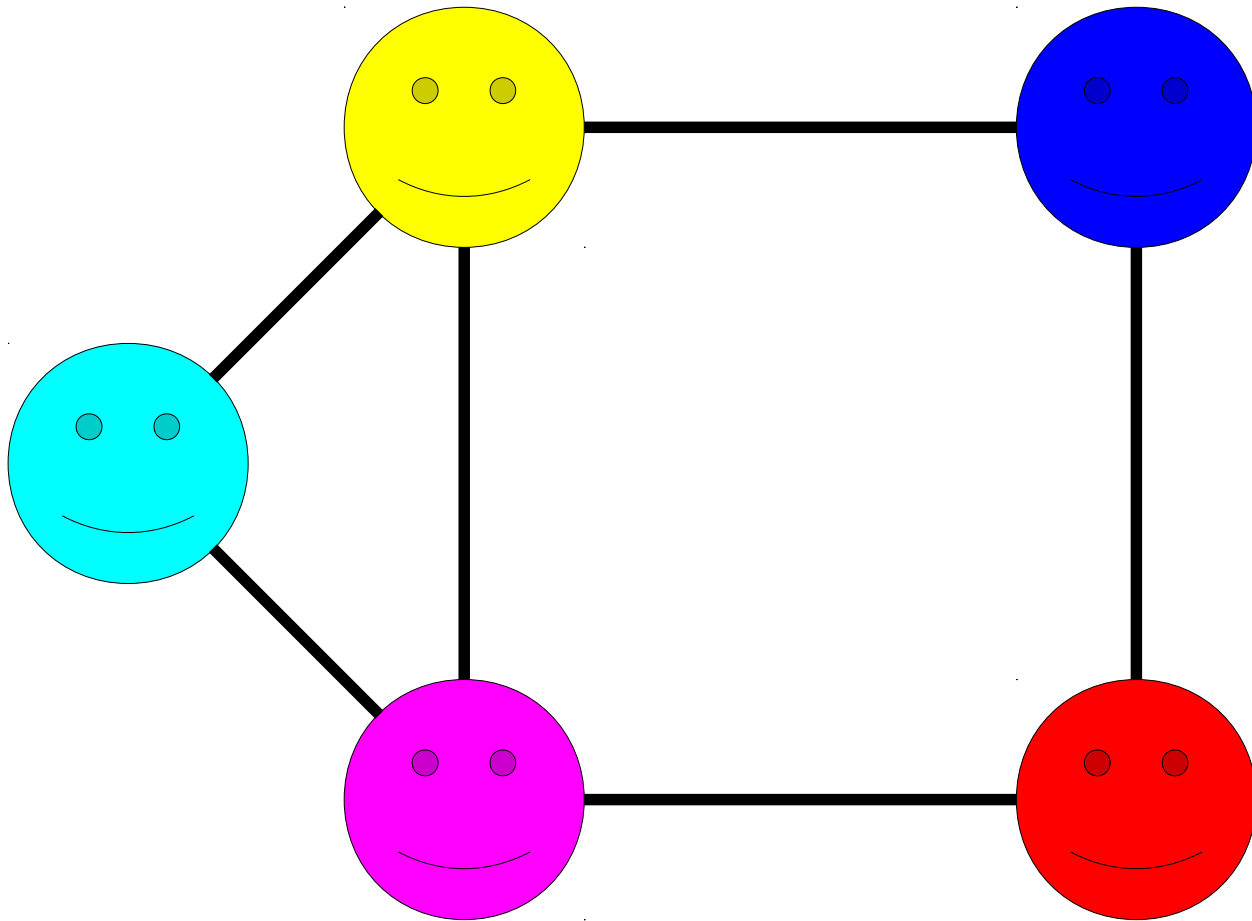
Part Two

Outline for Today

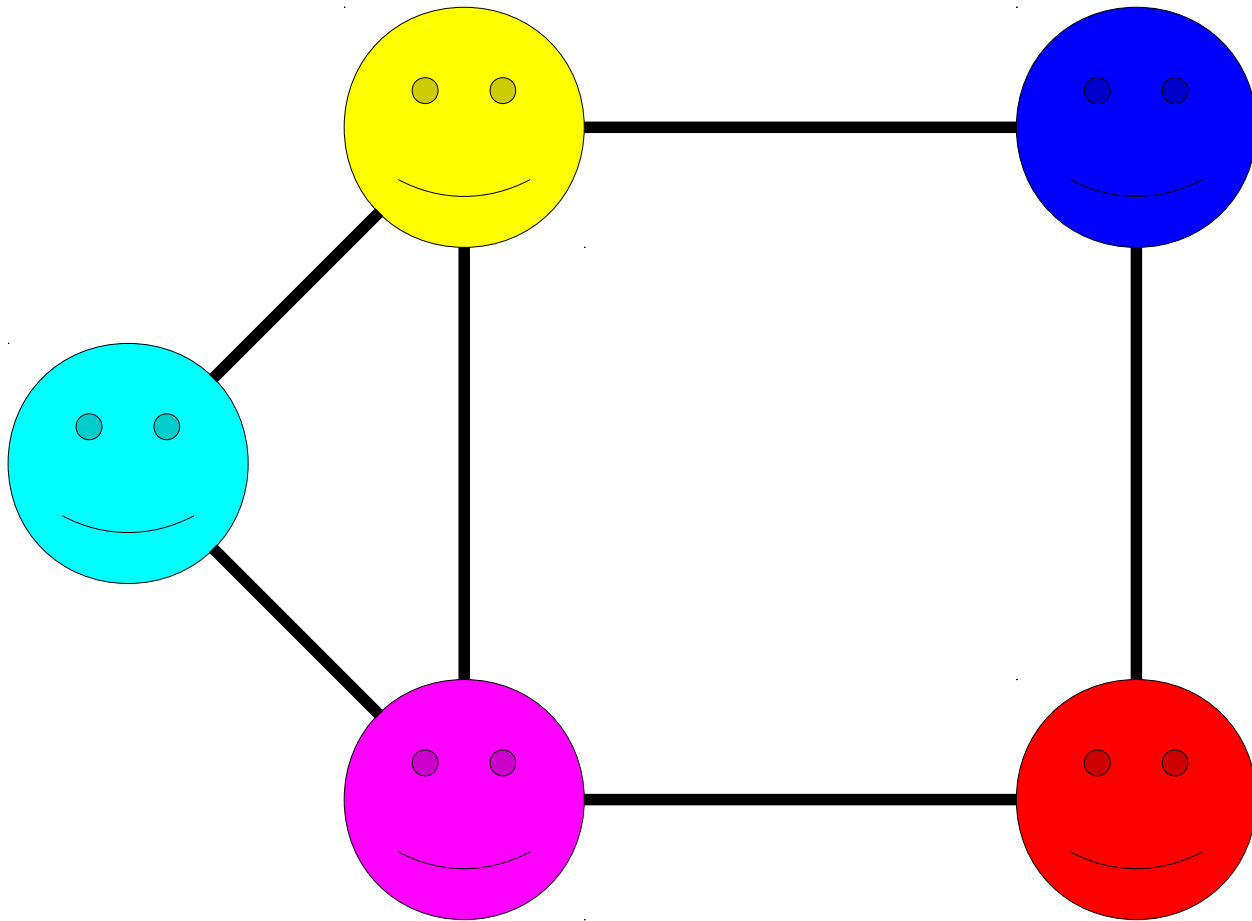
- ***Recap from Last Time***
 - Where are we, again?
- ***Depth-First Search***
 - A different way to explore a graph.
- ***DAGs***
 - A useful type of directed graph.
- ***Topological Sorting***
 - Getting things done in the right order.

Recap from Last Time

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

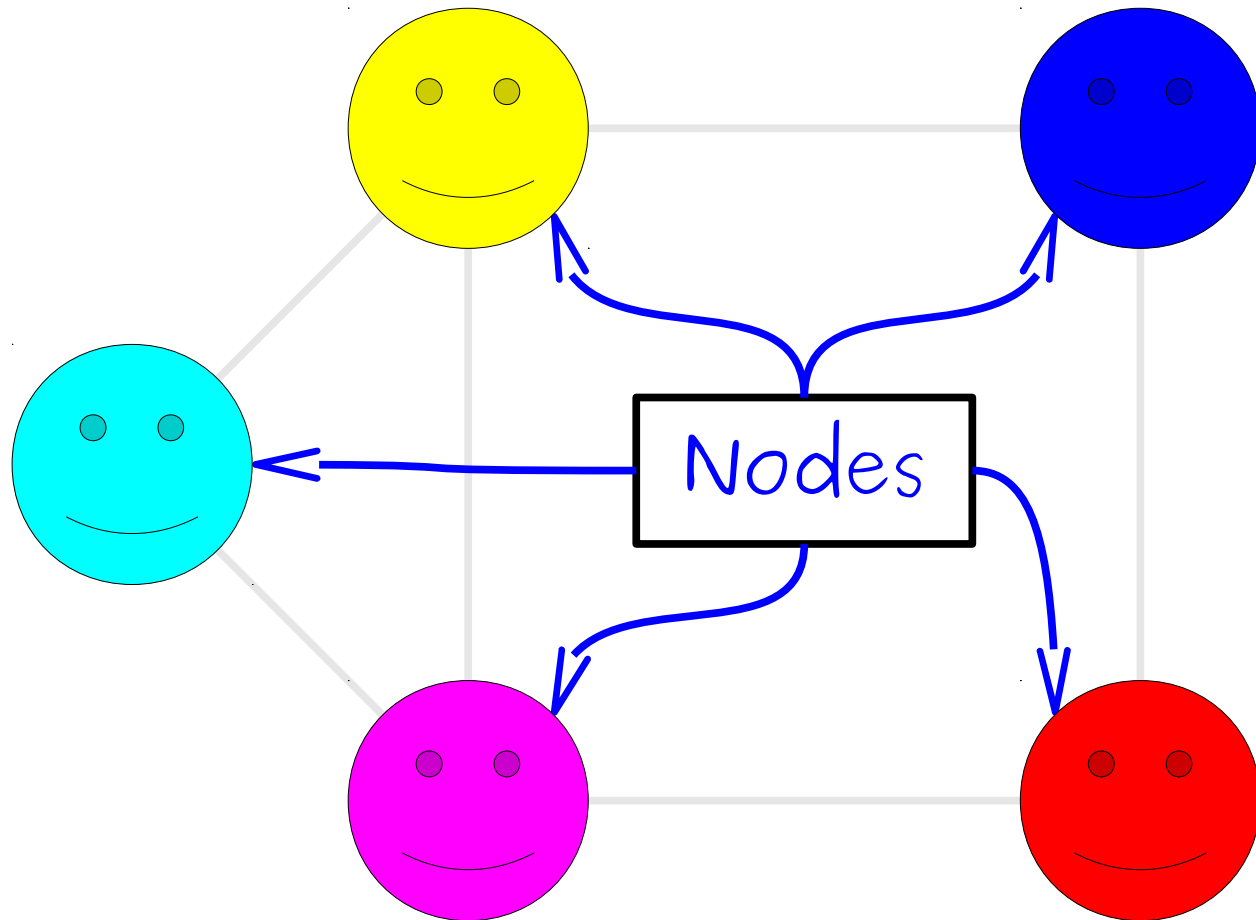


A **graph** is a mathematical structure for representing relationships.



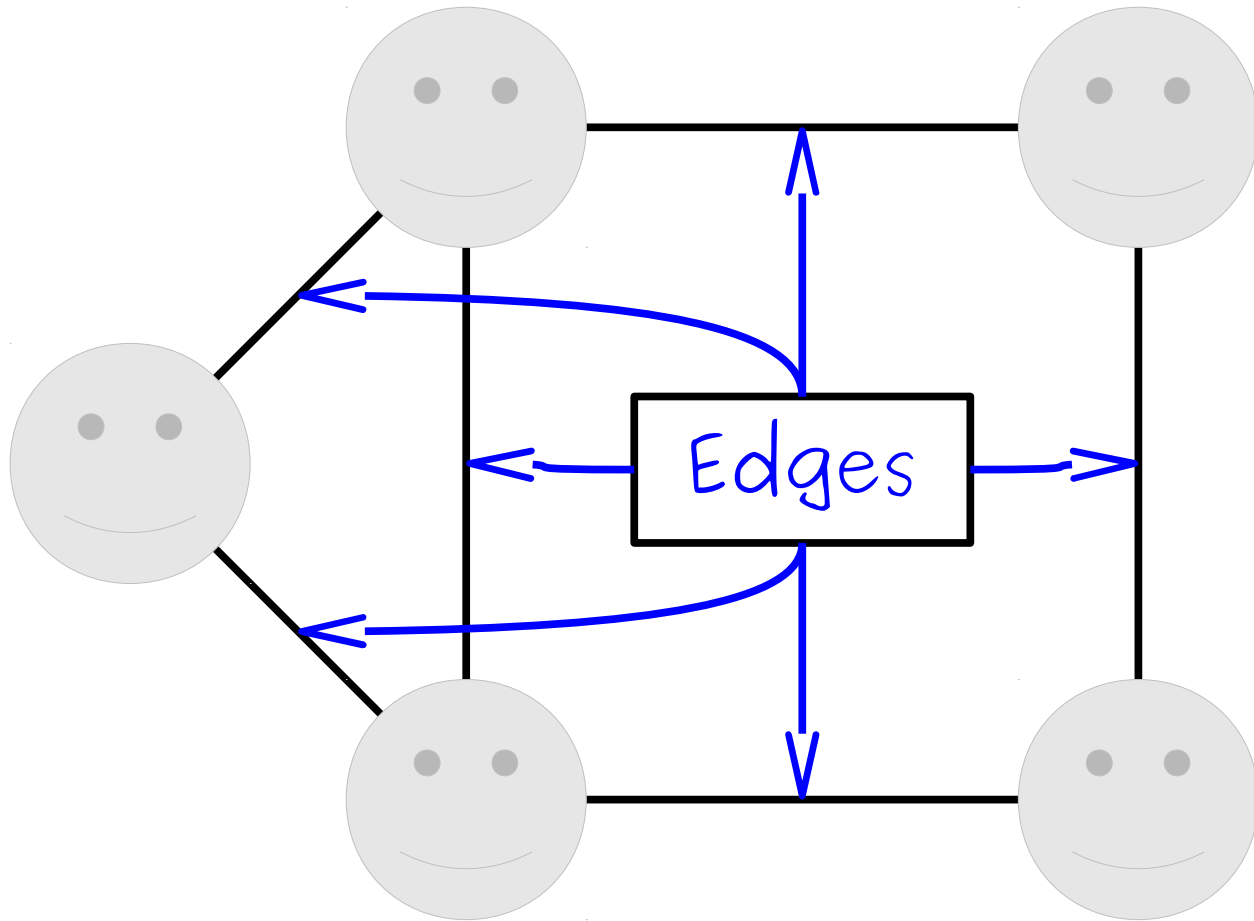
A graph consists of a set of **nodes** connected by **edges**.

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



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

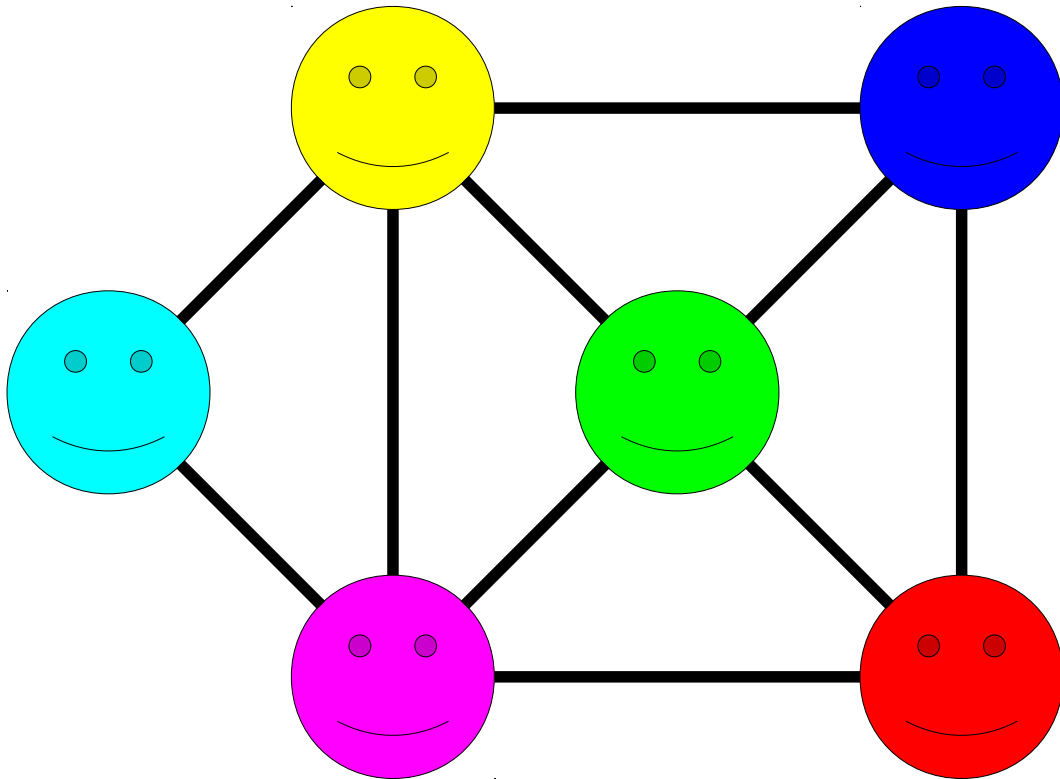
A **graph** is a mathematical structure for representing relationships.



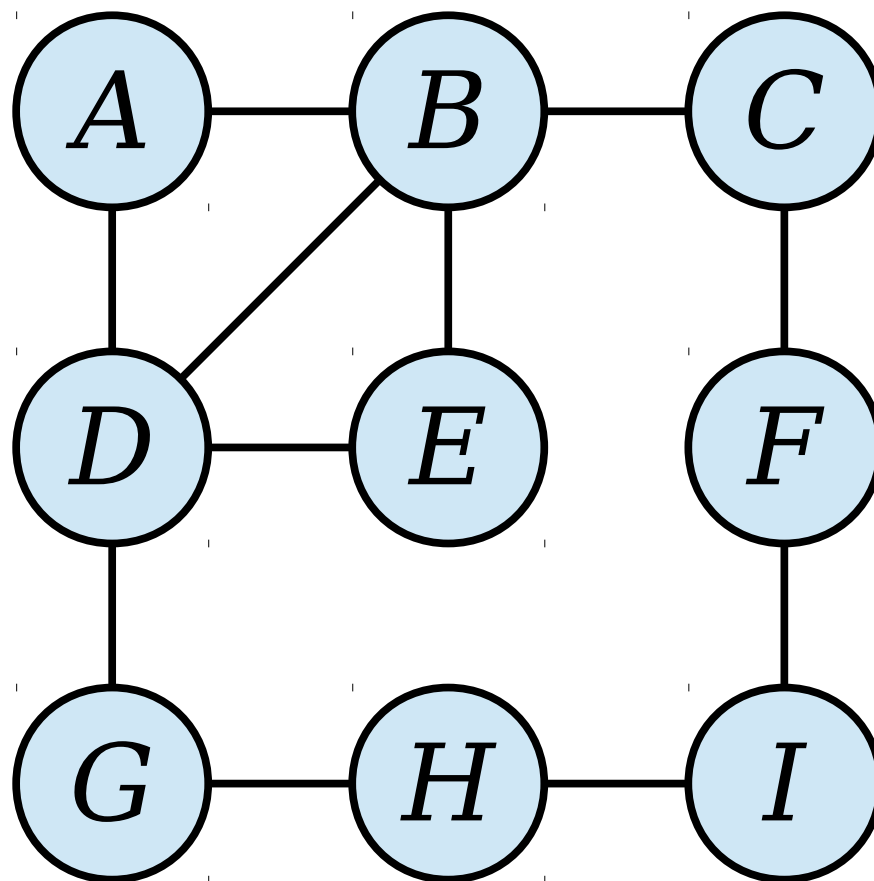
A graph consists of a set of **nodes** connected by **edges**.

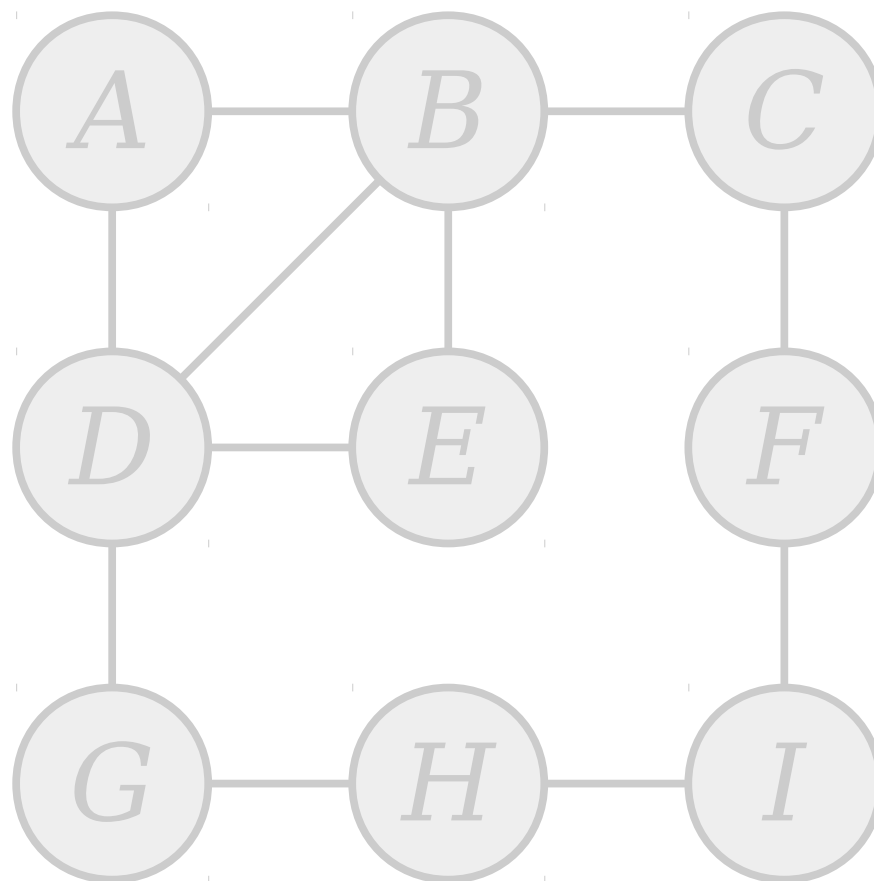
Representing Graphs

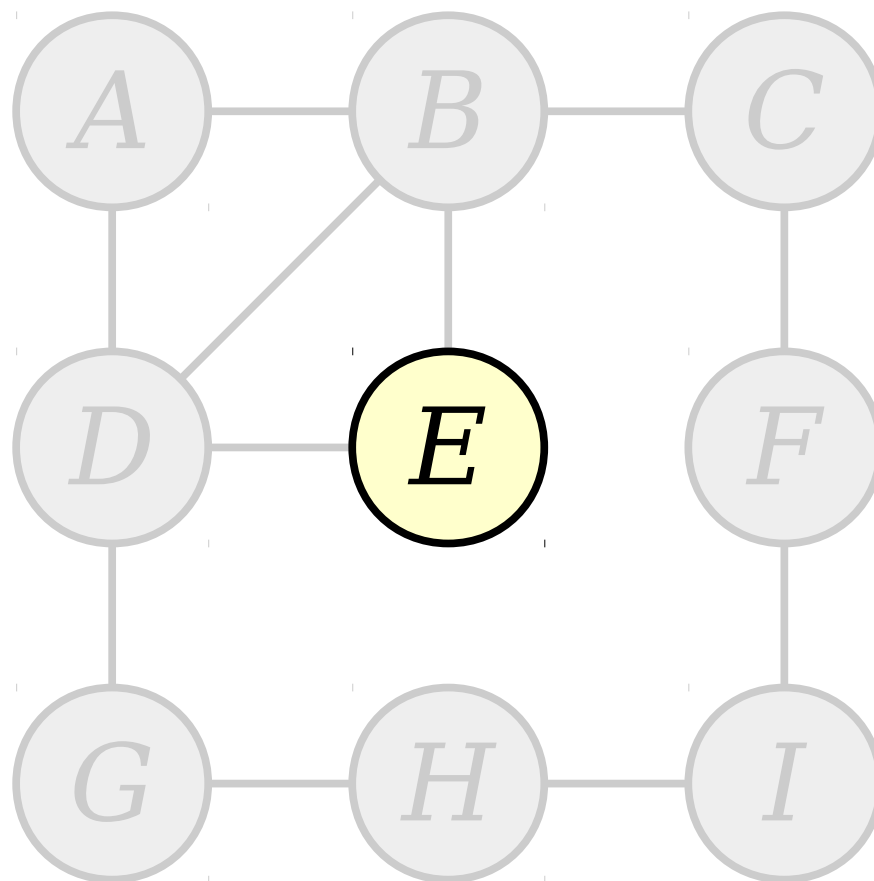
We can represent a graph as a map from nodes to the list of nodes each node is connected to.

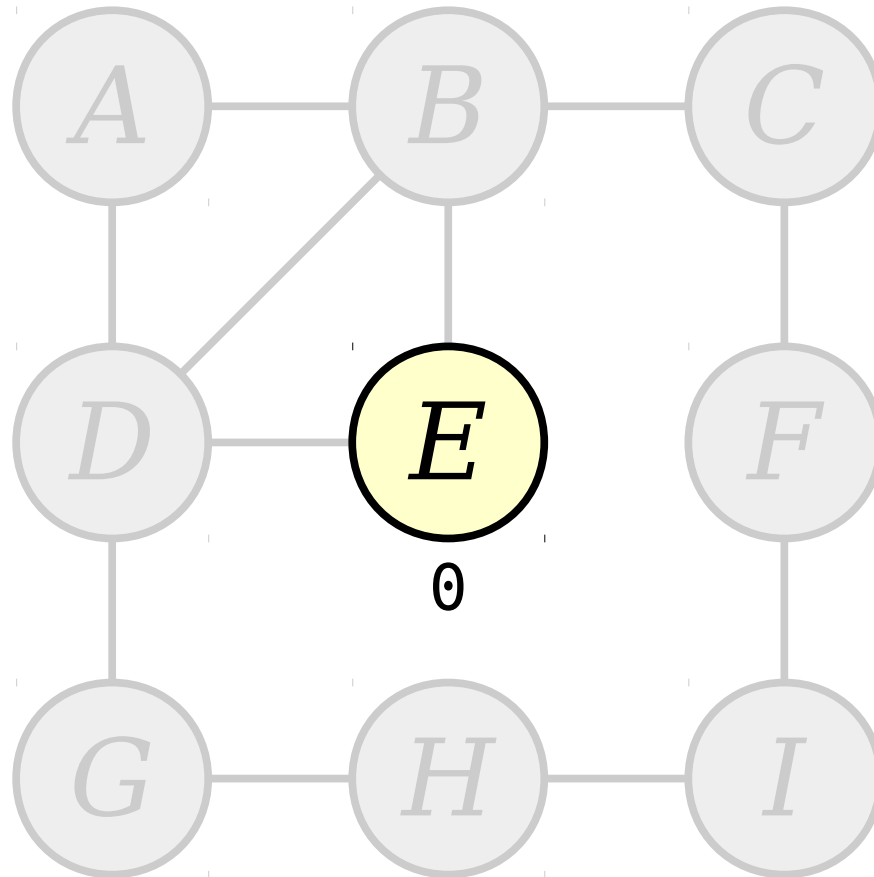


Node	Adjacent To

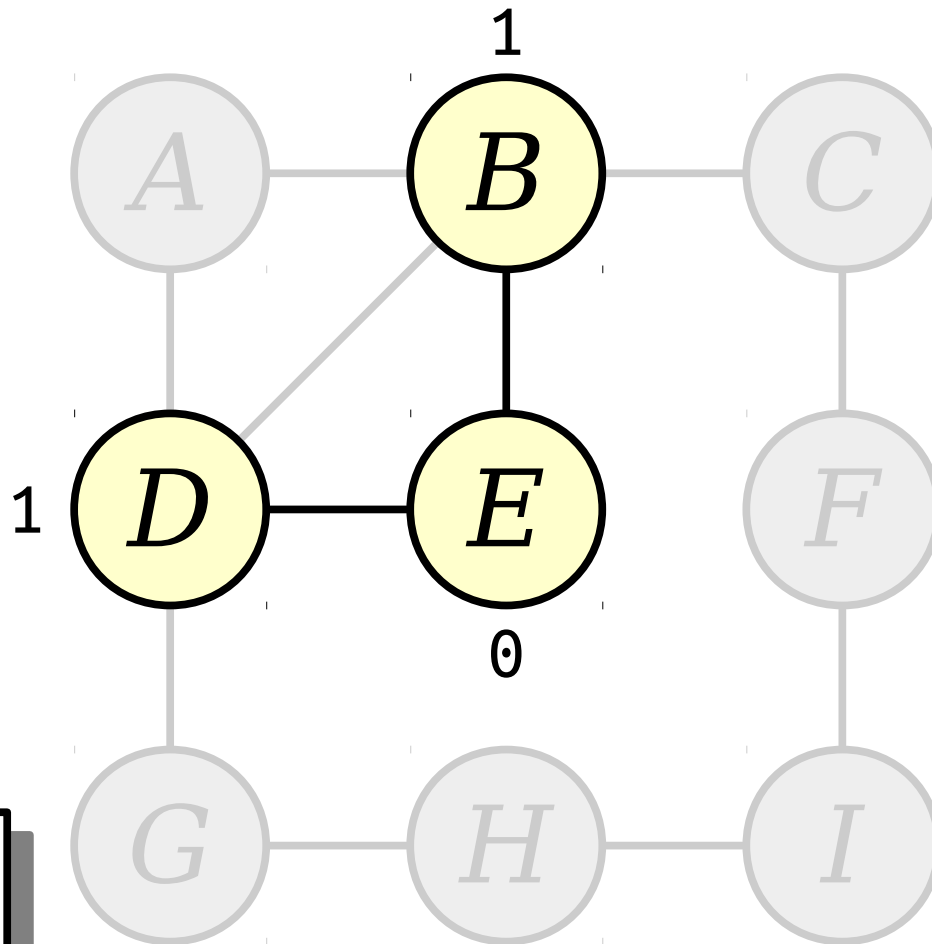




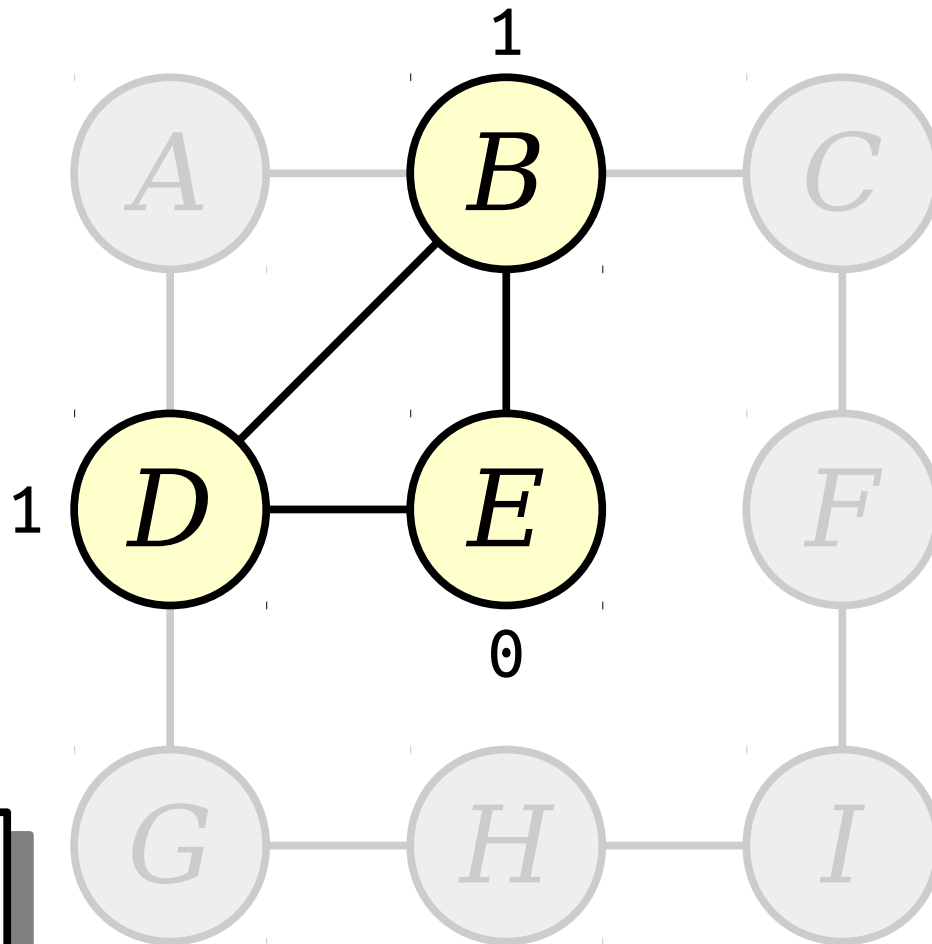




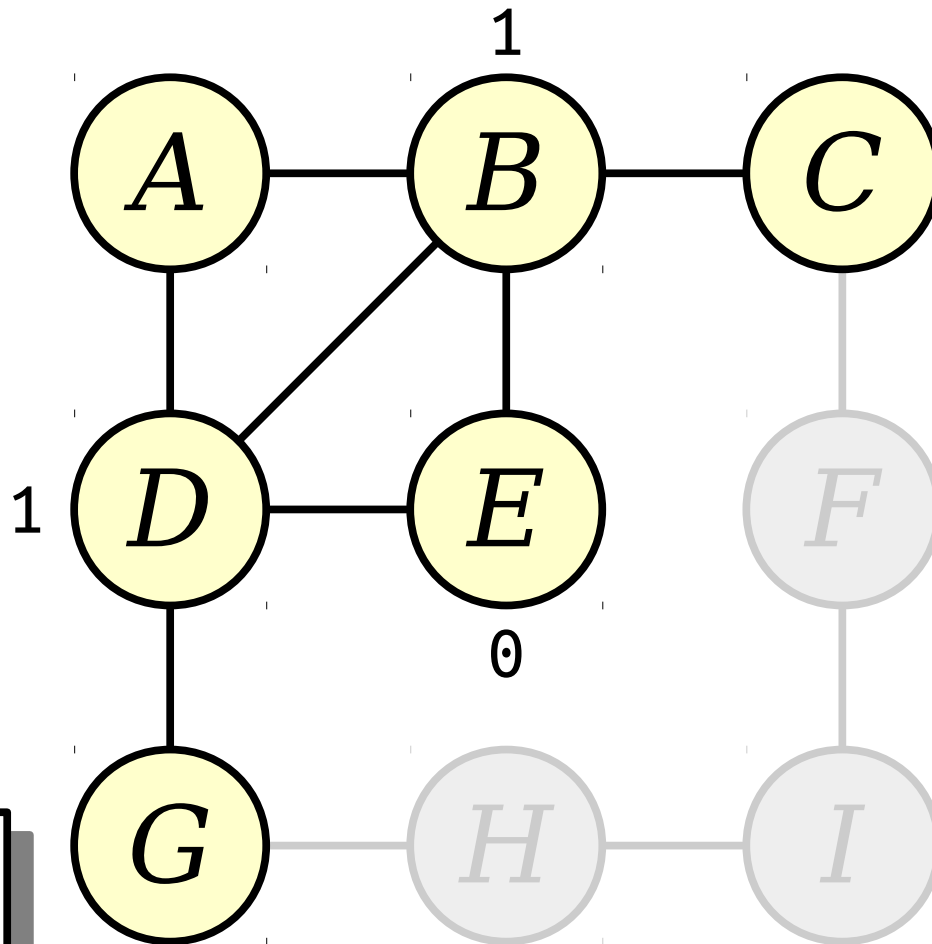
Core idea: Find everything one hop away from the start, then two hops away, then three hops away, etc.



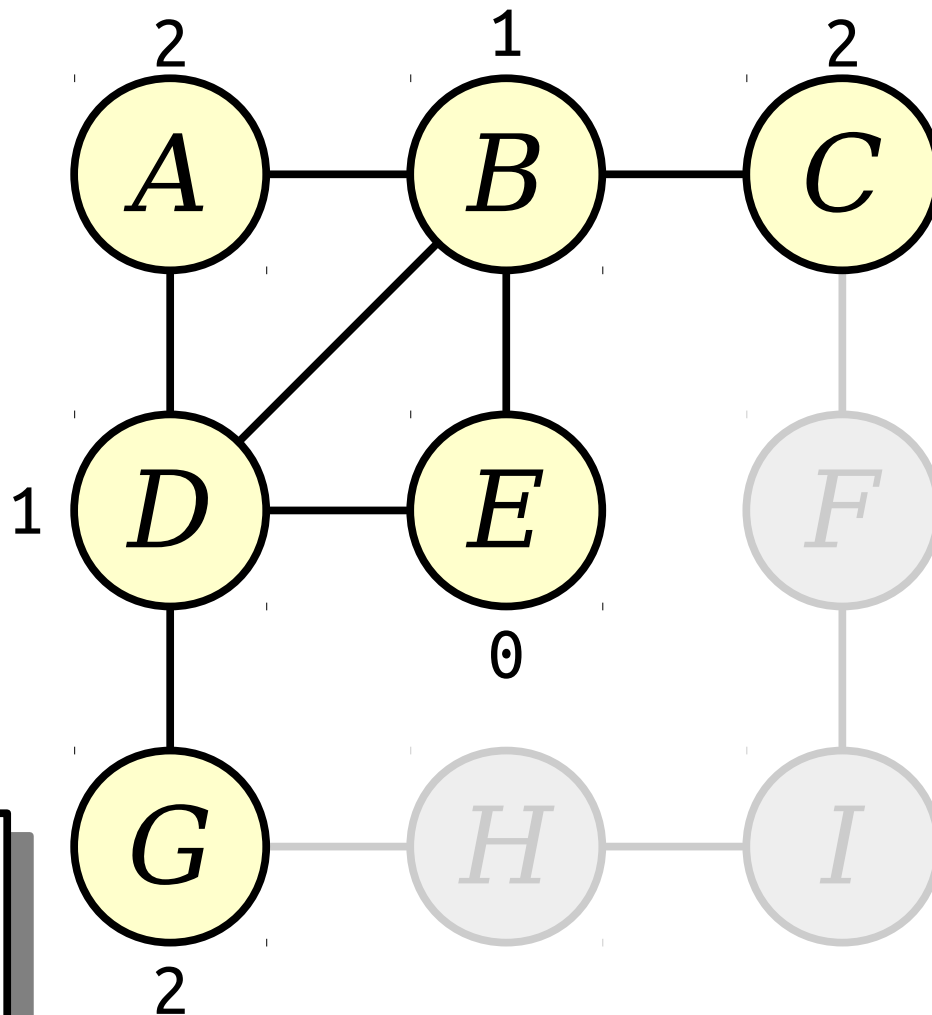
Core idea: Find everything one hop away from the start, then two hops away, then three hops away, etc.



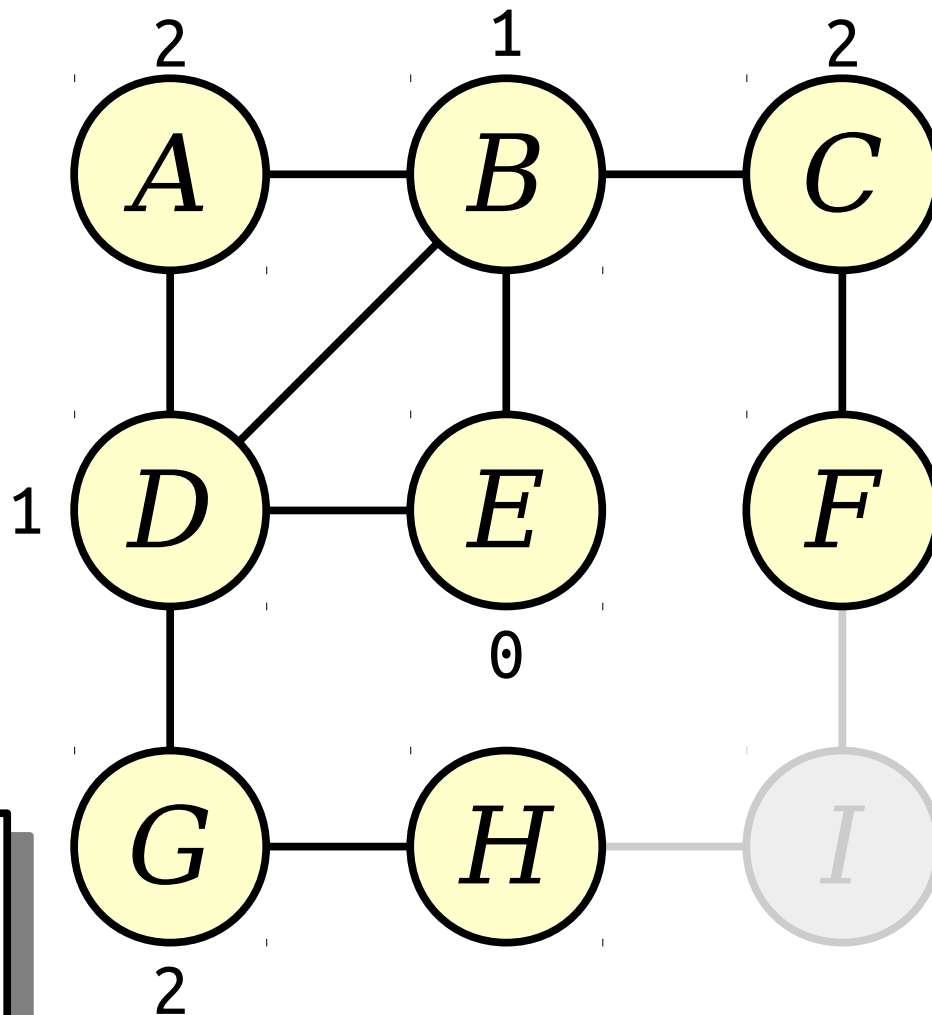
Core idea: Find everything one hop away from the start, then two hops away, then three hops away, etc.



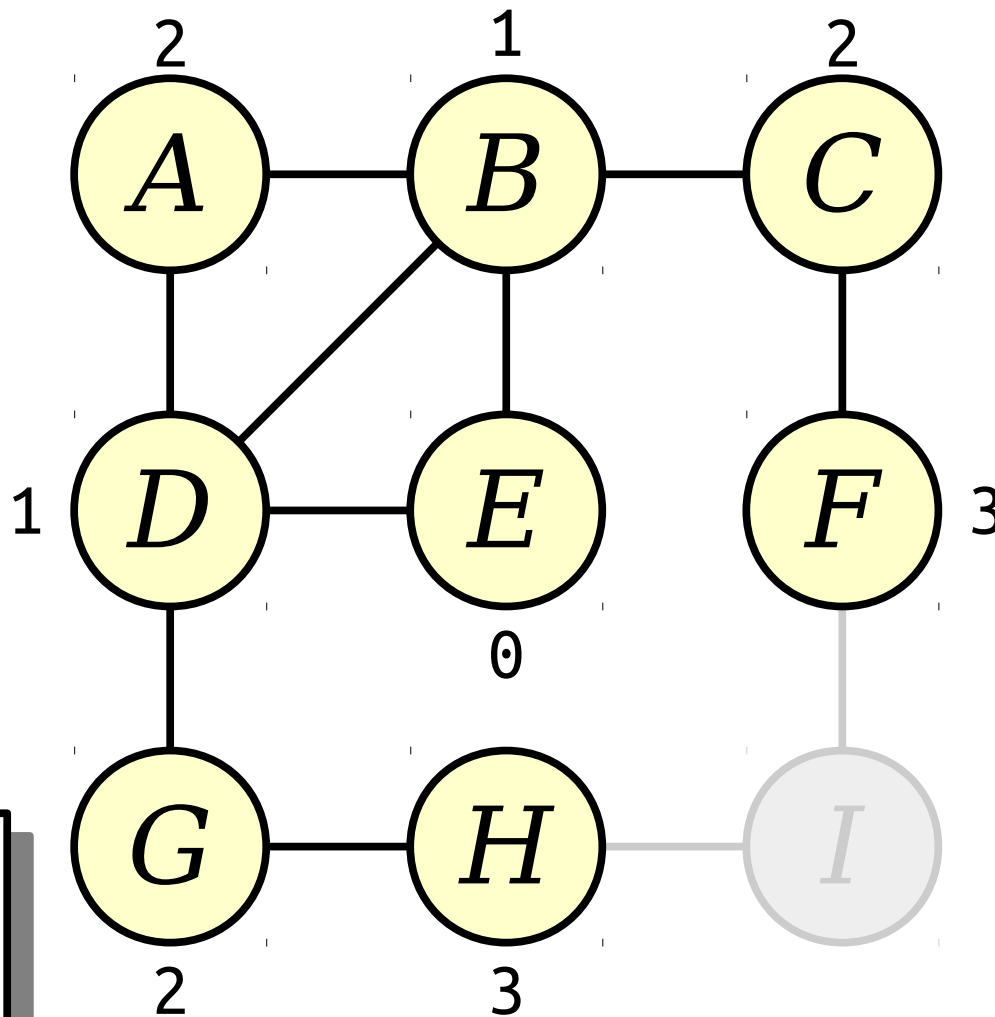
Core idea: Find everything one hop away from the start, then two hops away, then three hops away, etc.



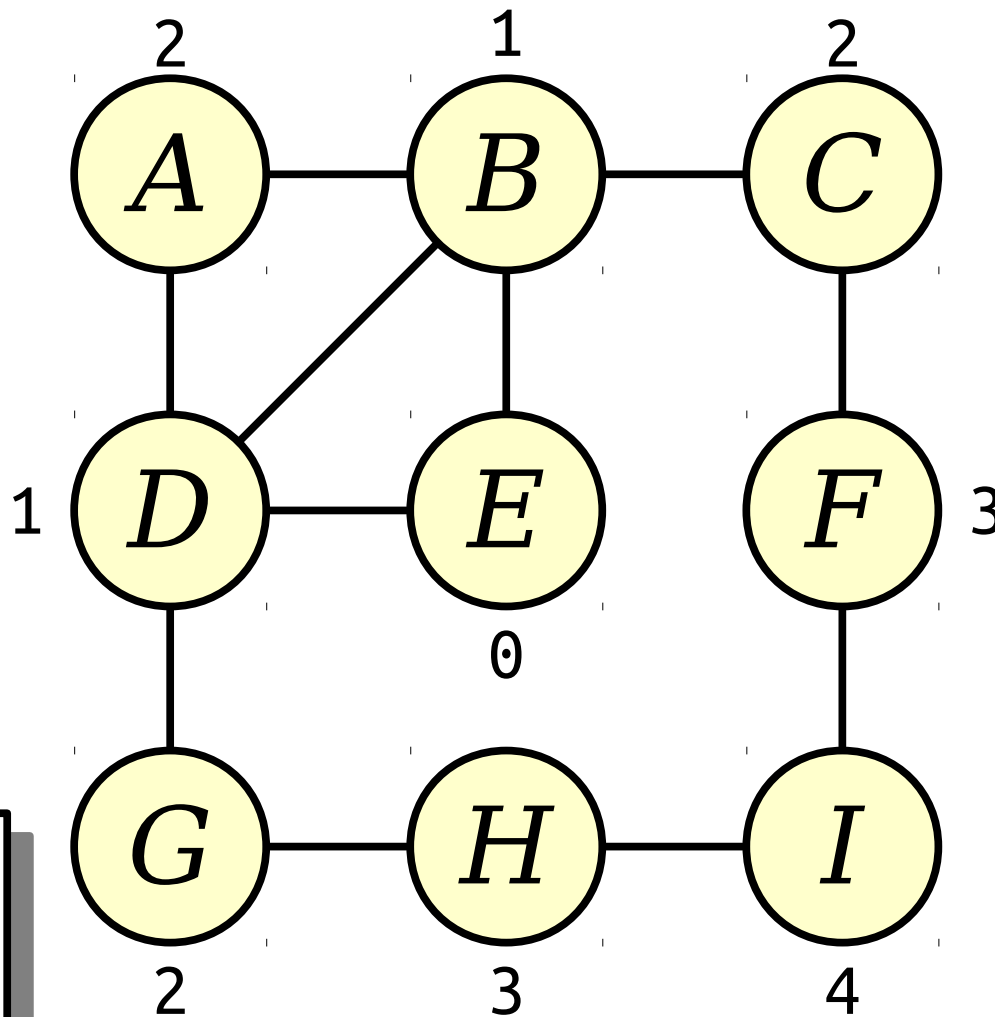
Core idea: Find everything one hop away from the start, then two hops away, then three hops away, etc.



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.

Breadth-First Search

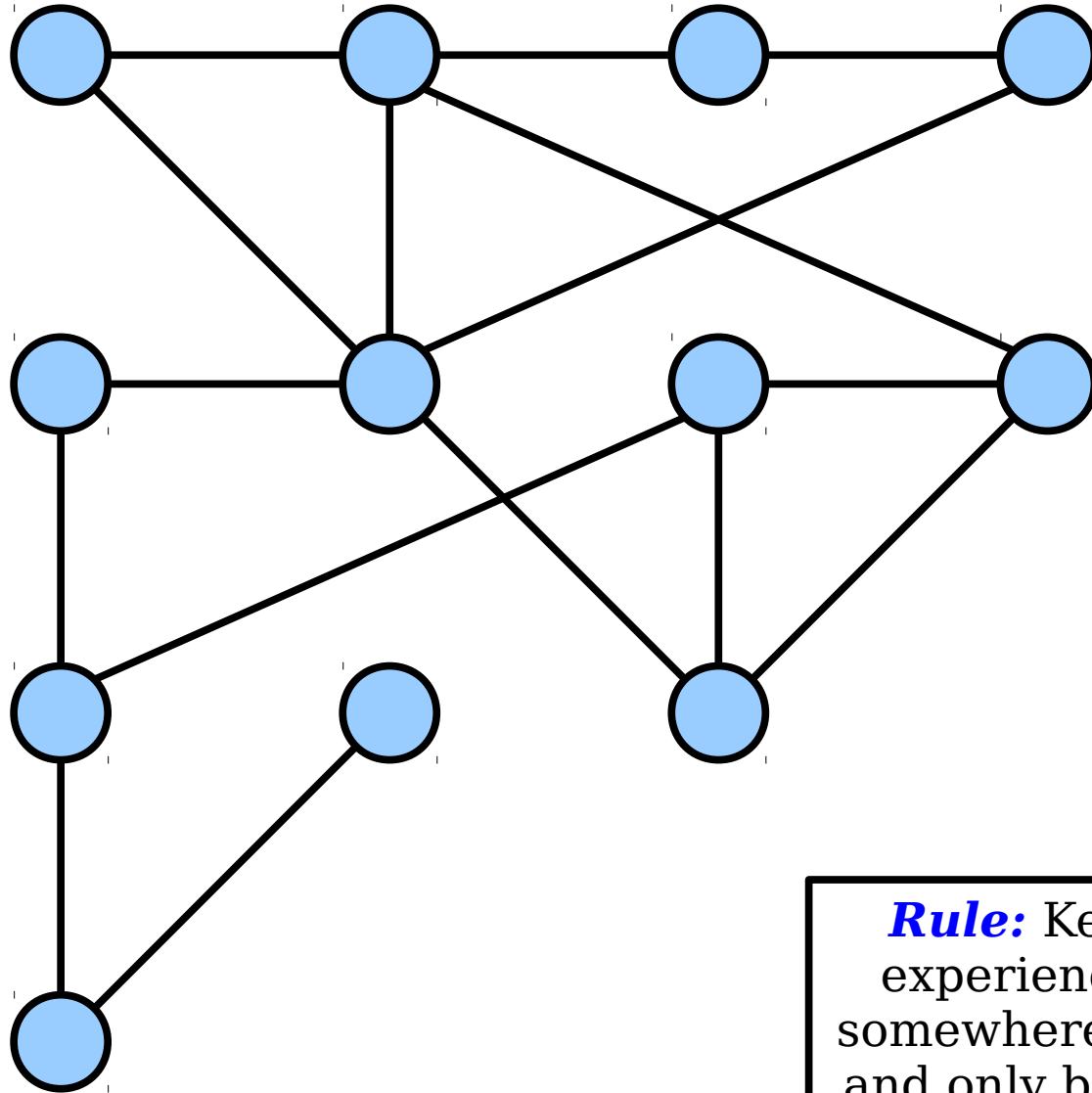
- ***Breadth-first search*** (or just ***BFS*** for short) is an algorithm for exploring the nodes in a graph in ascending order of distance from some start node.
- In pseudocode:

```
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.  
}
```

New Stuff!

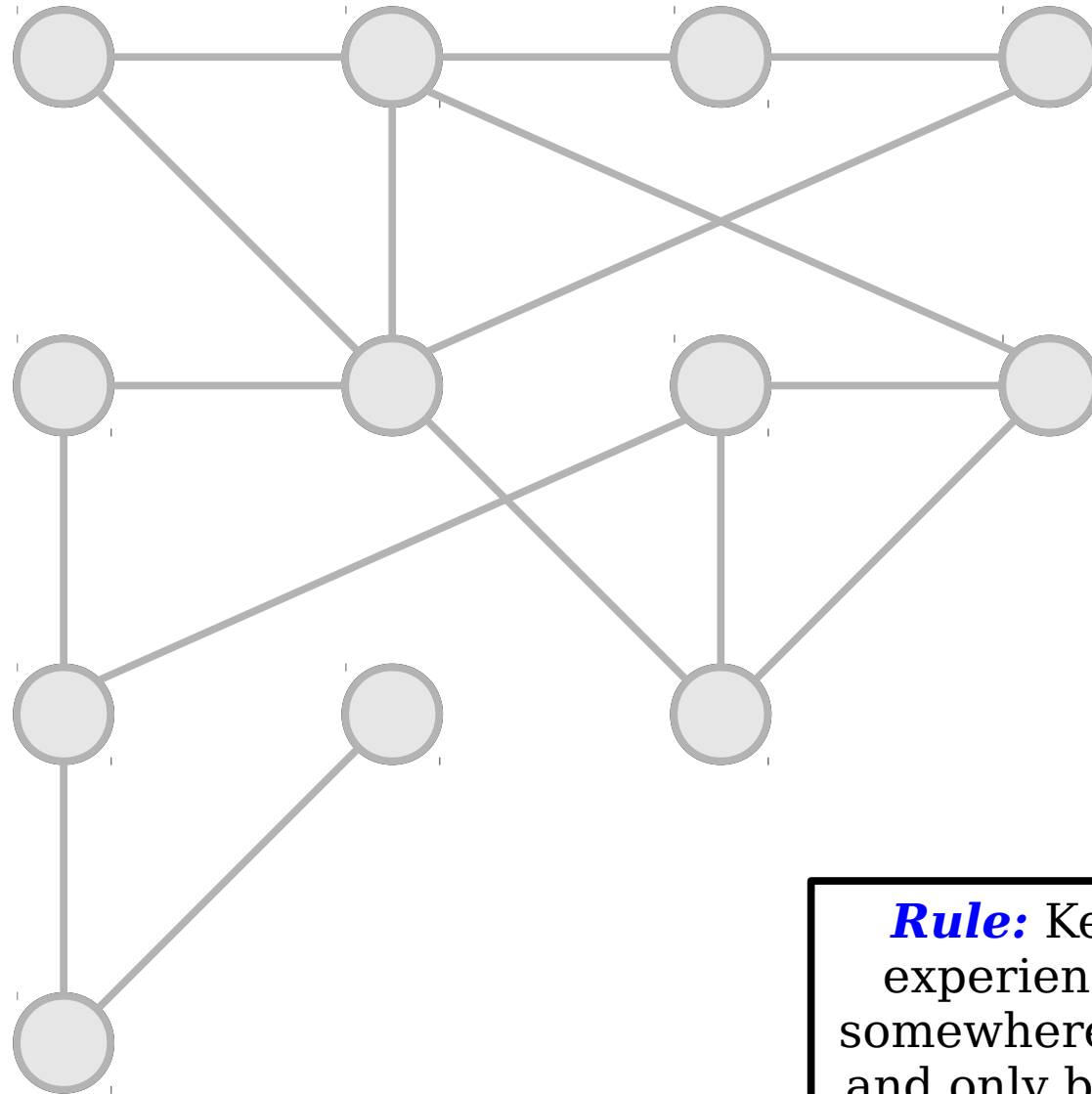
Depth-First Search

Depth-First Search



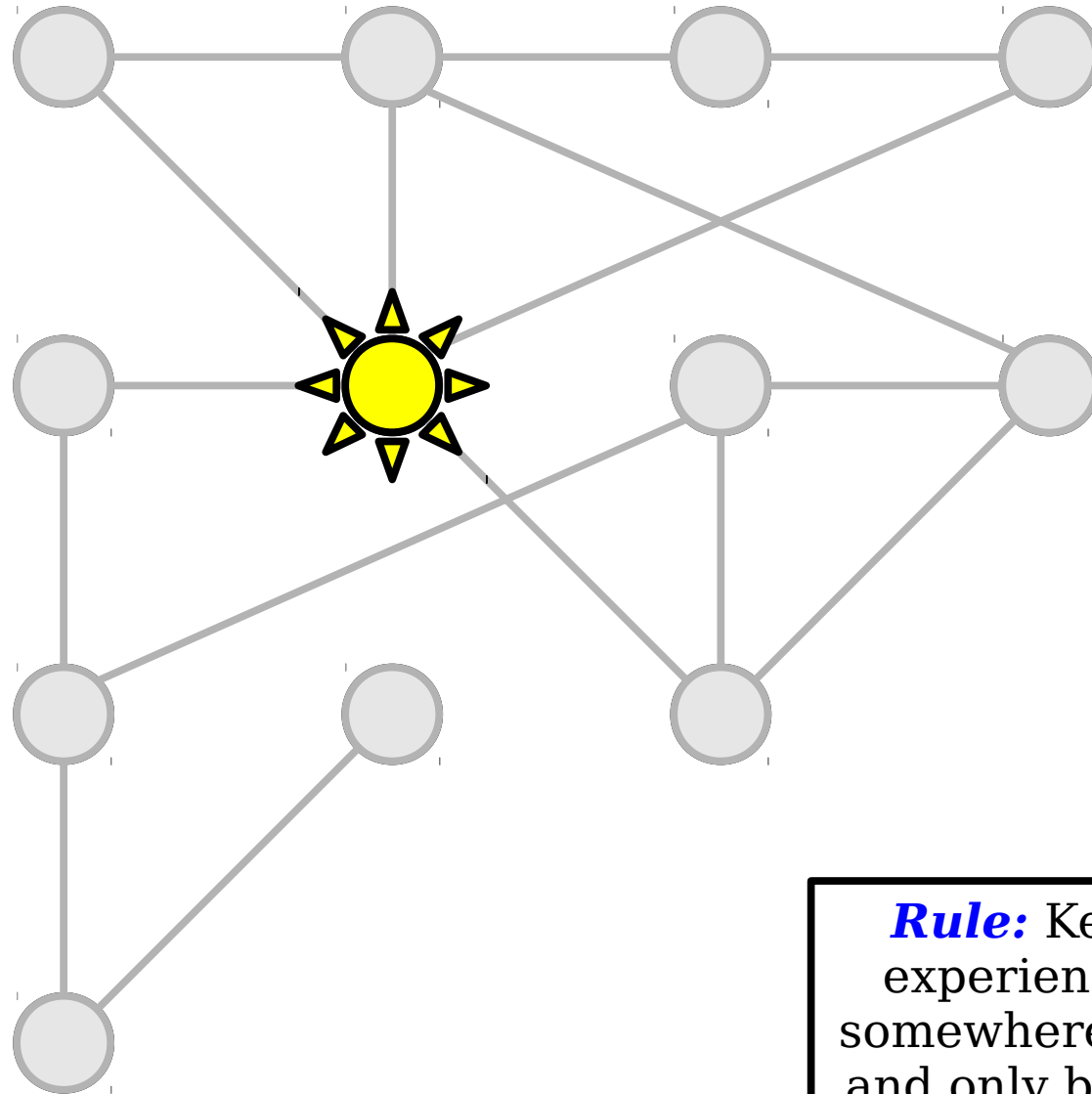
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



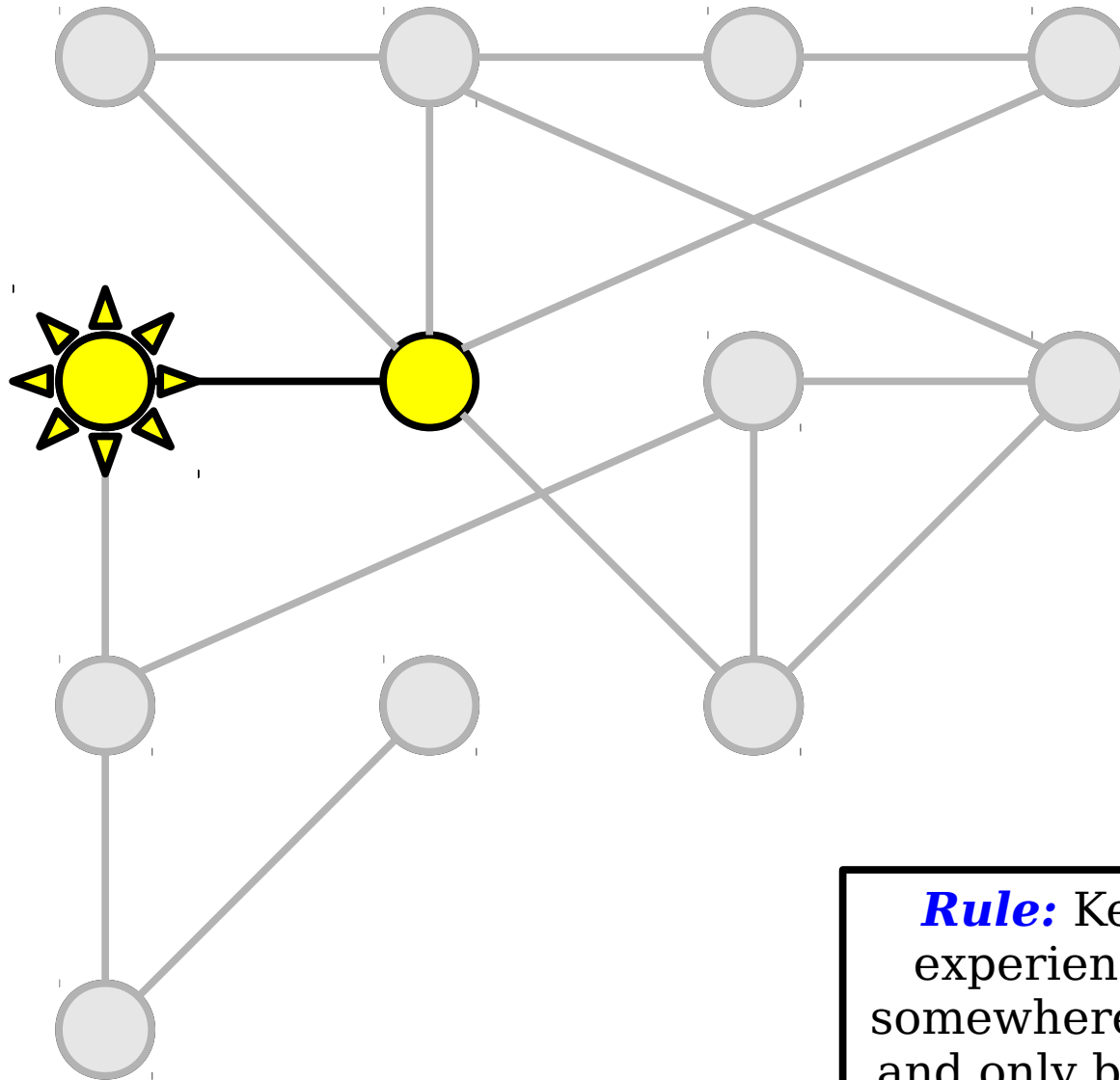
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



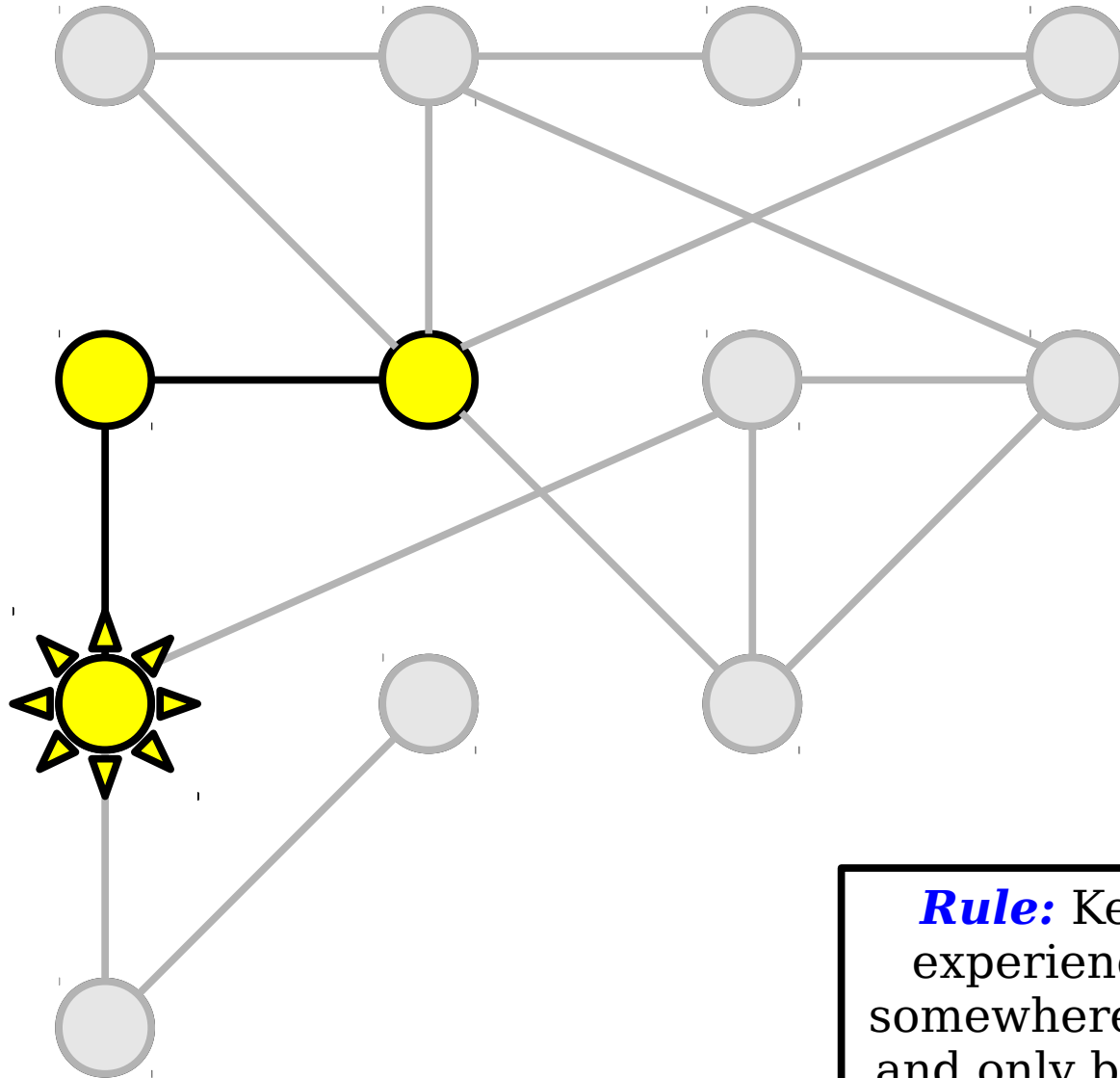
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



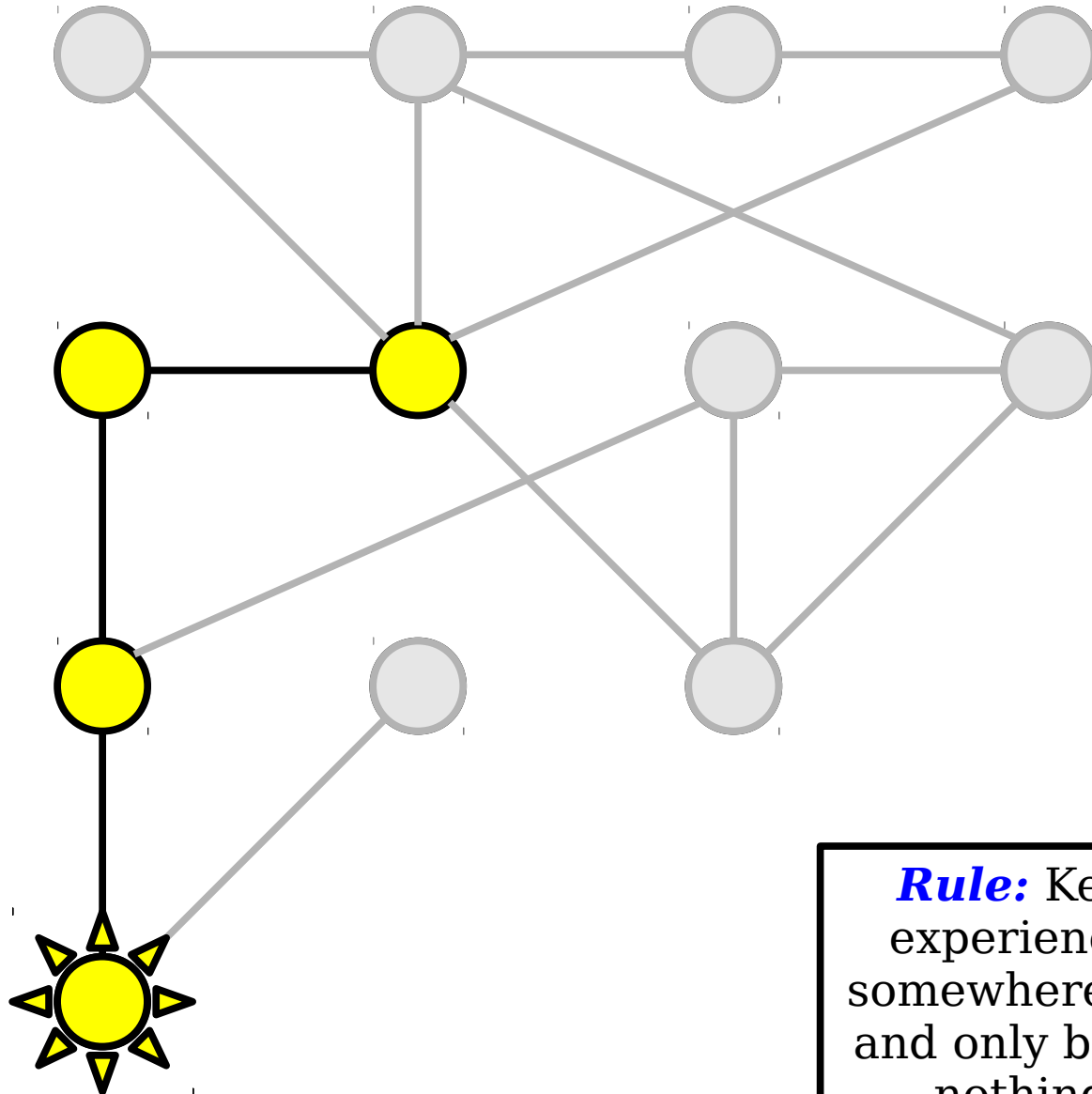
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



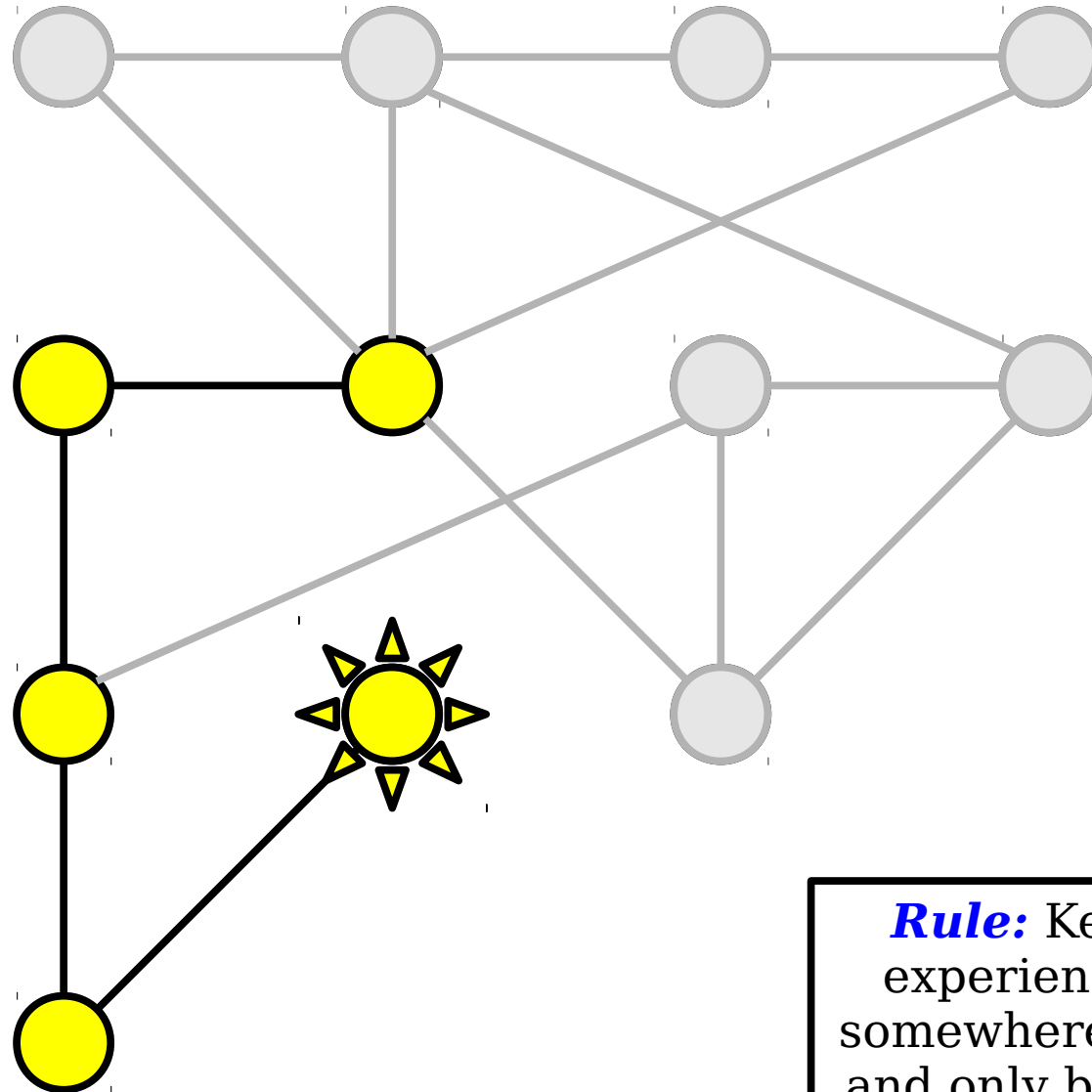
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



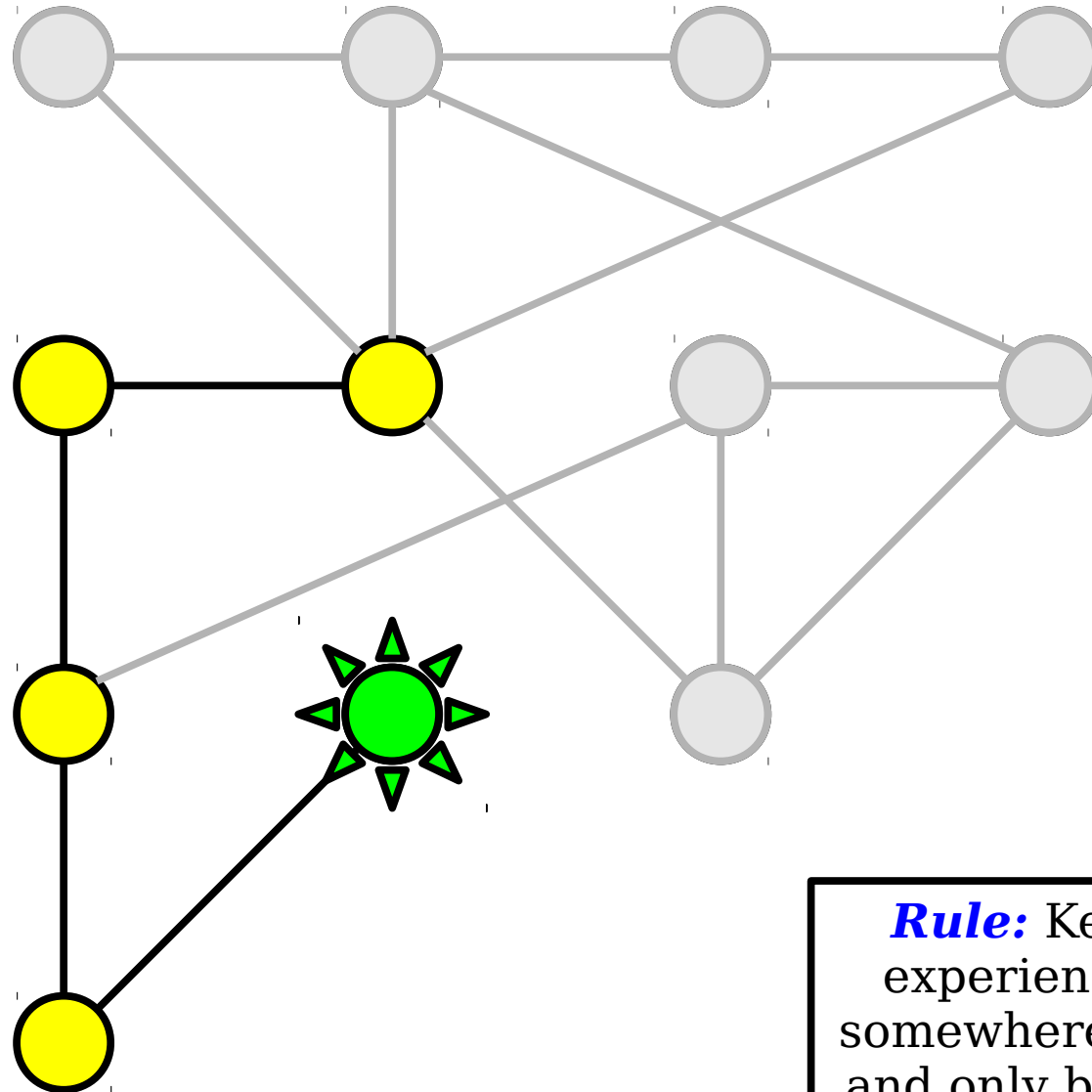
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



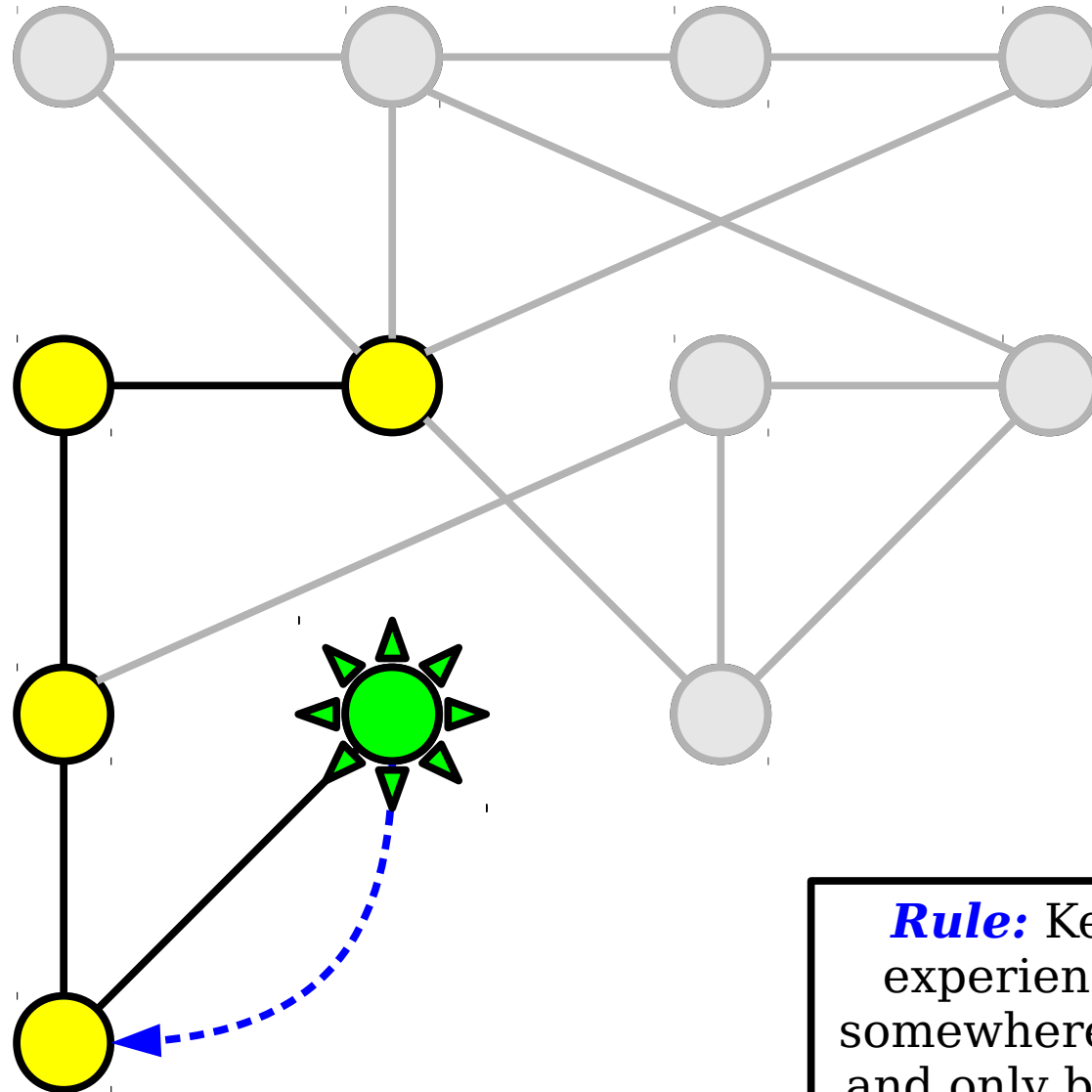
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



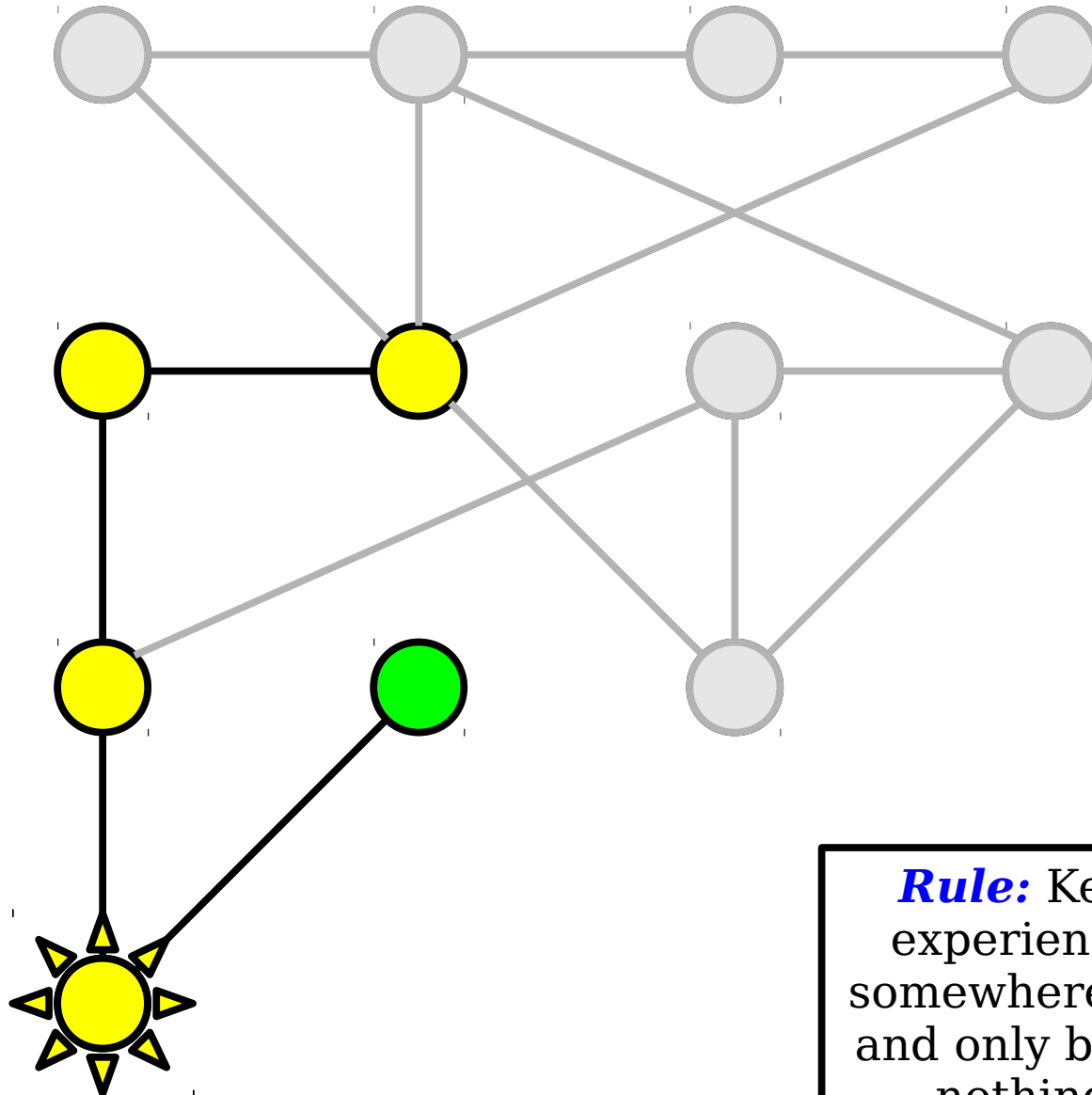
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



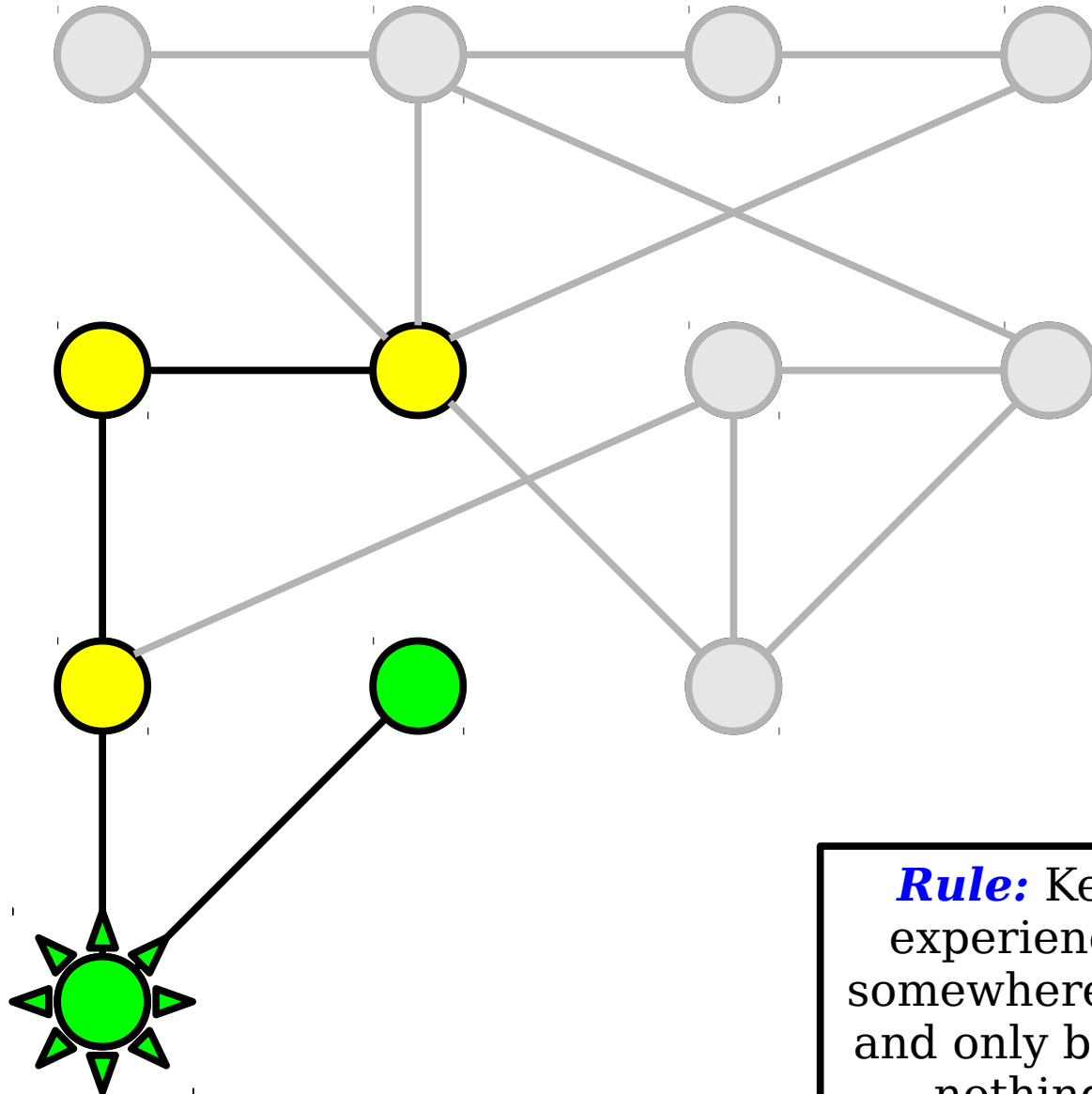
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



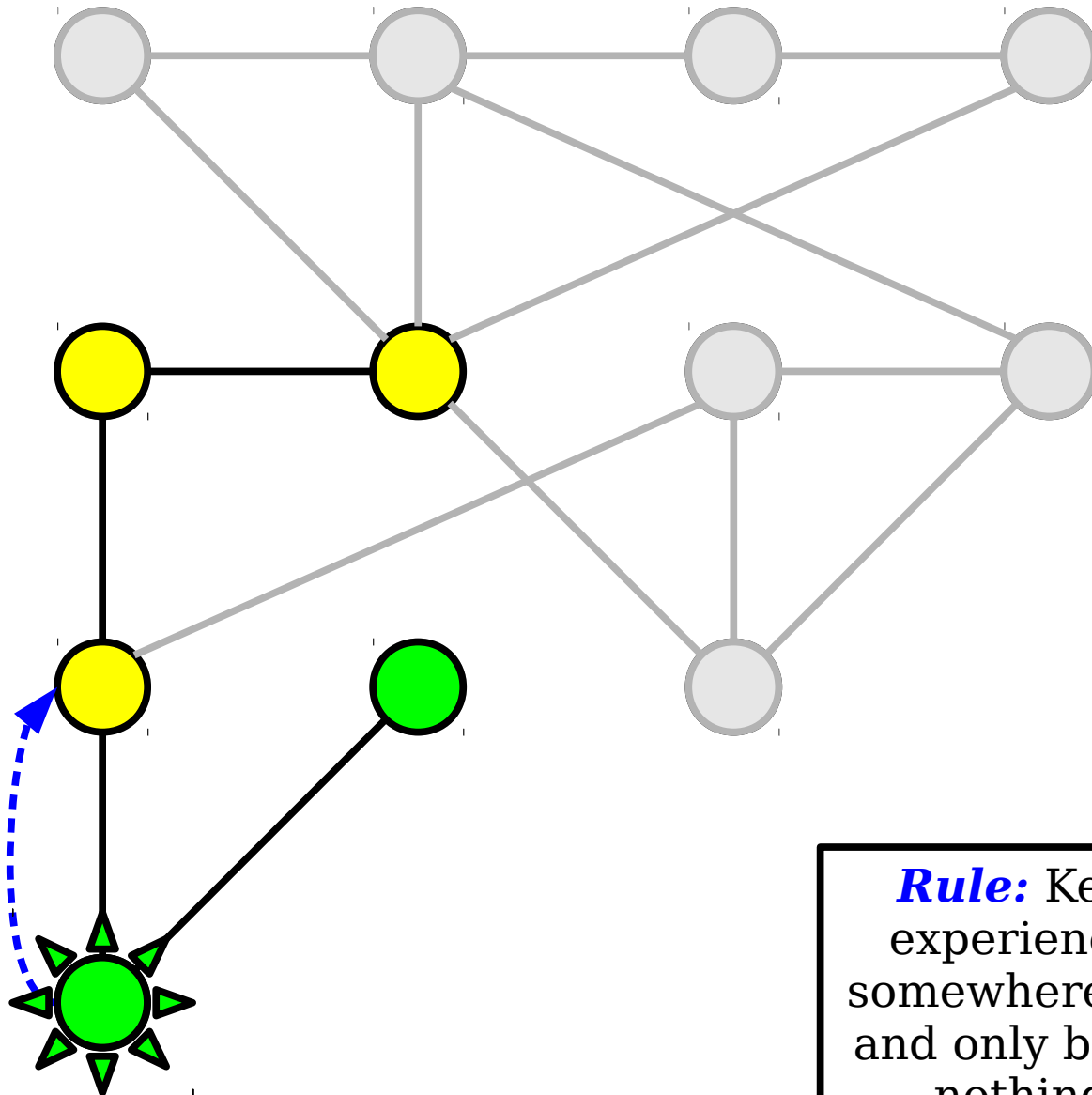
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



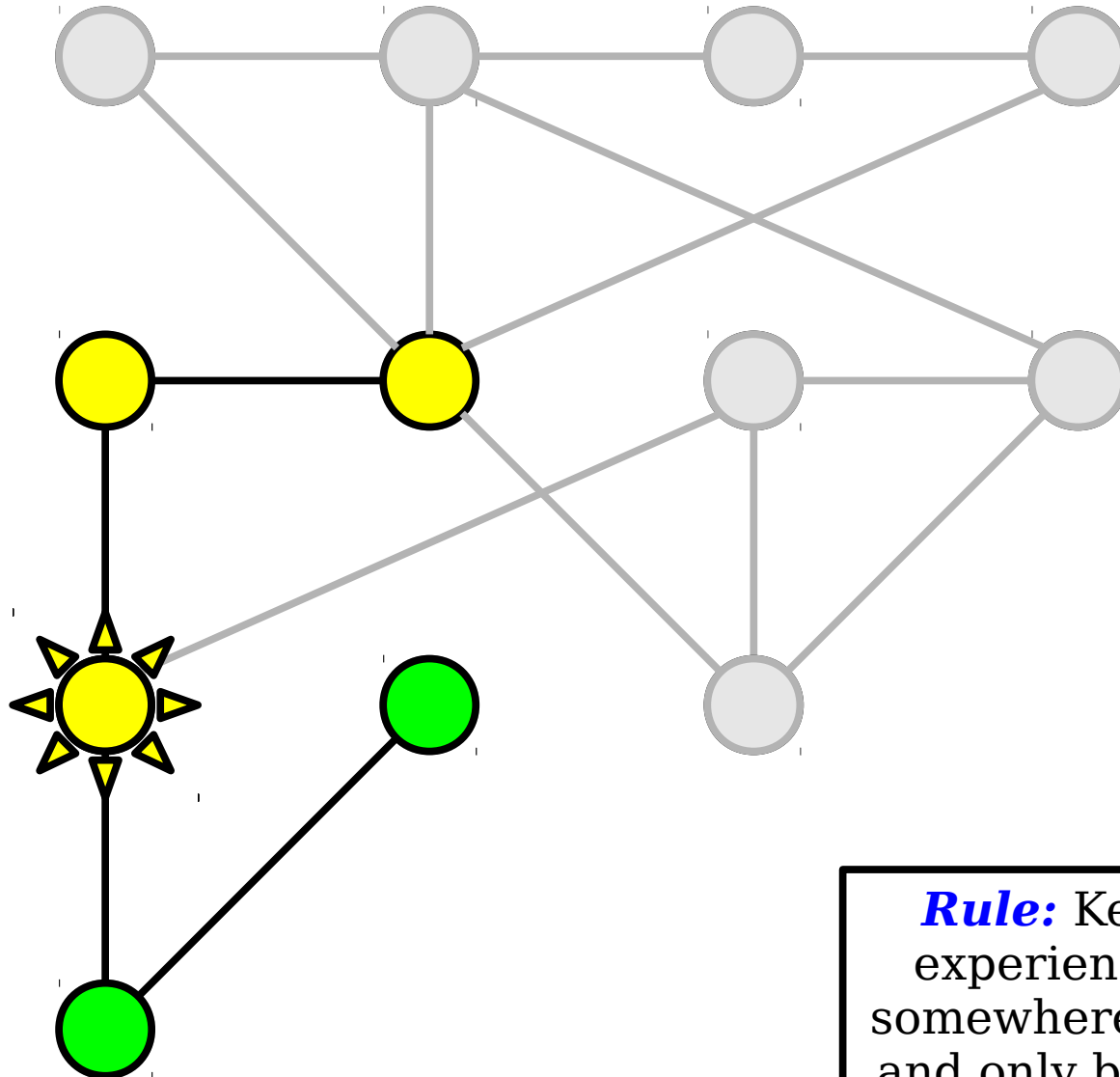
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



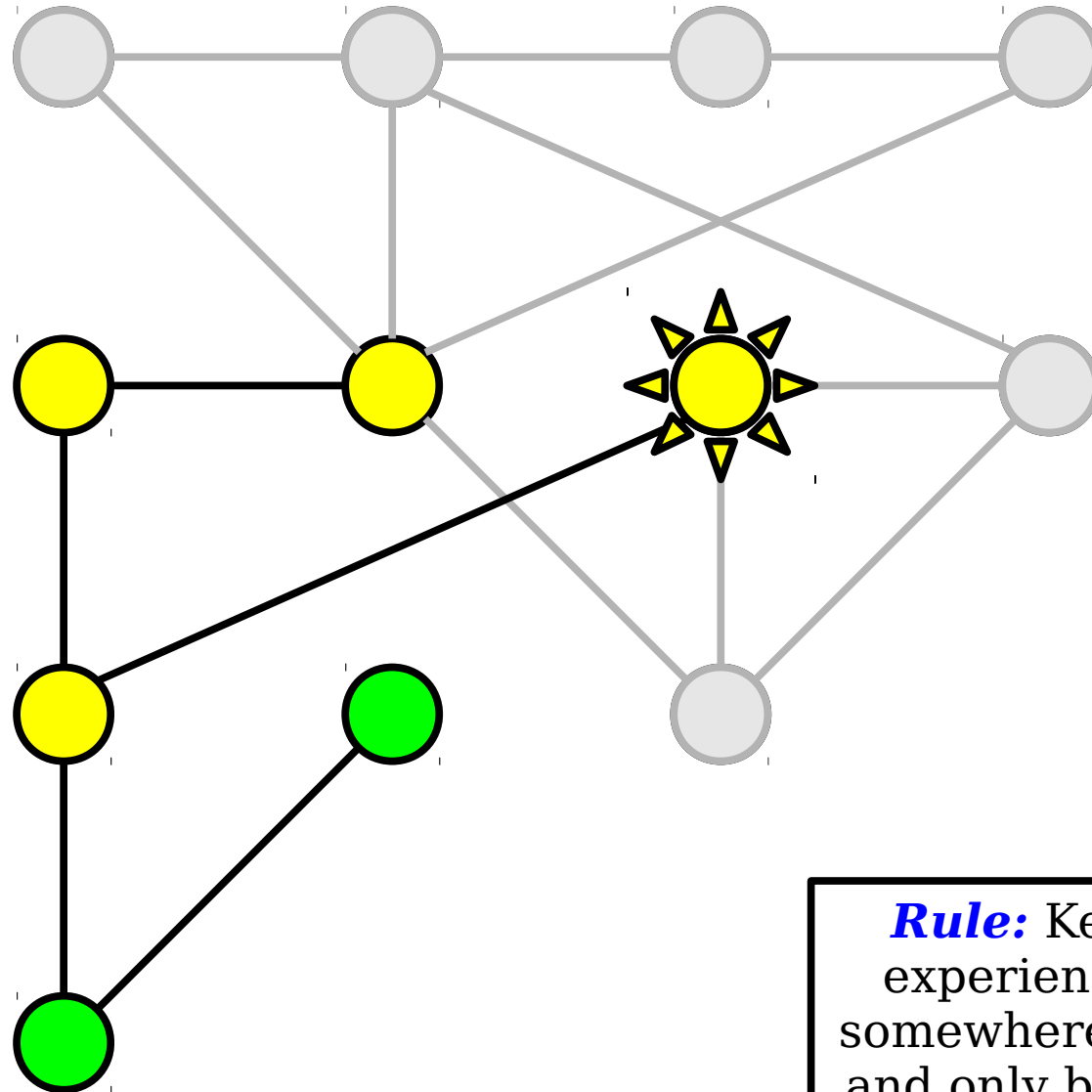
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



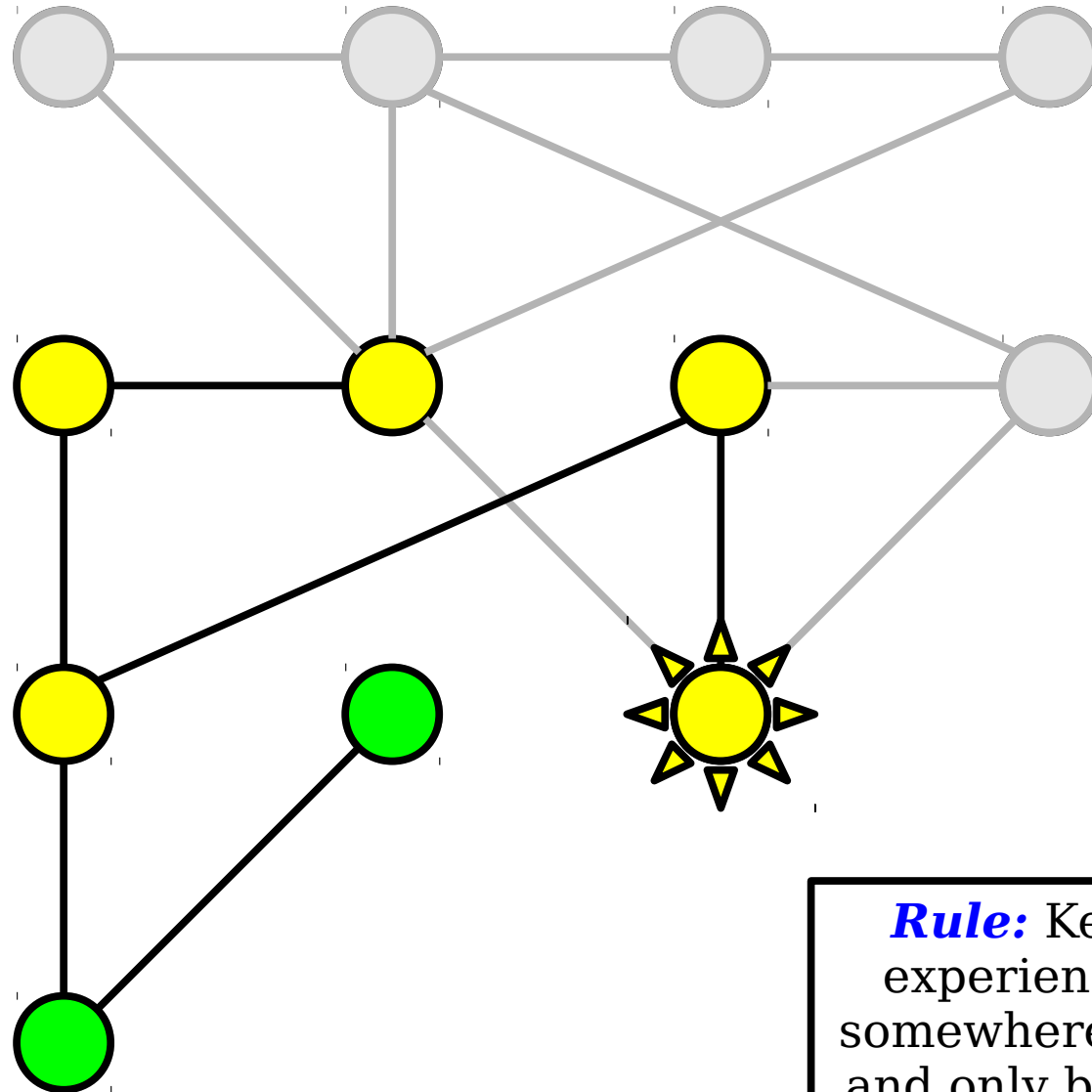
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



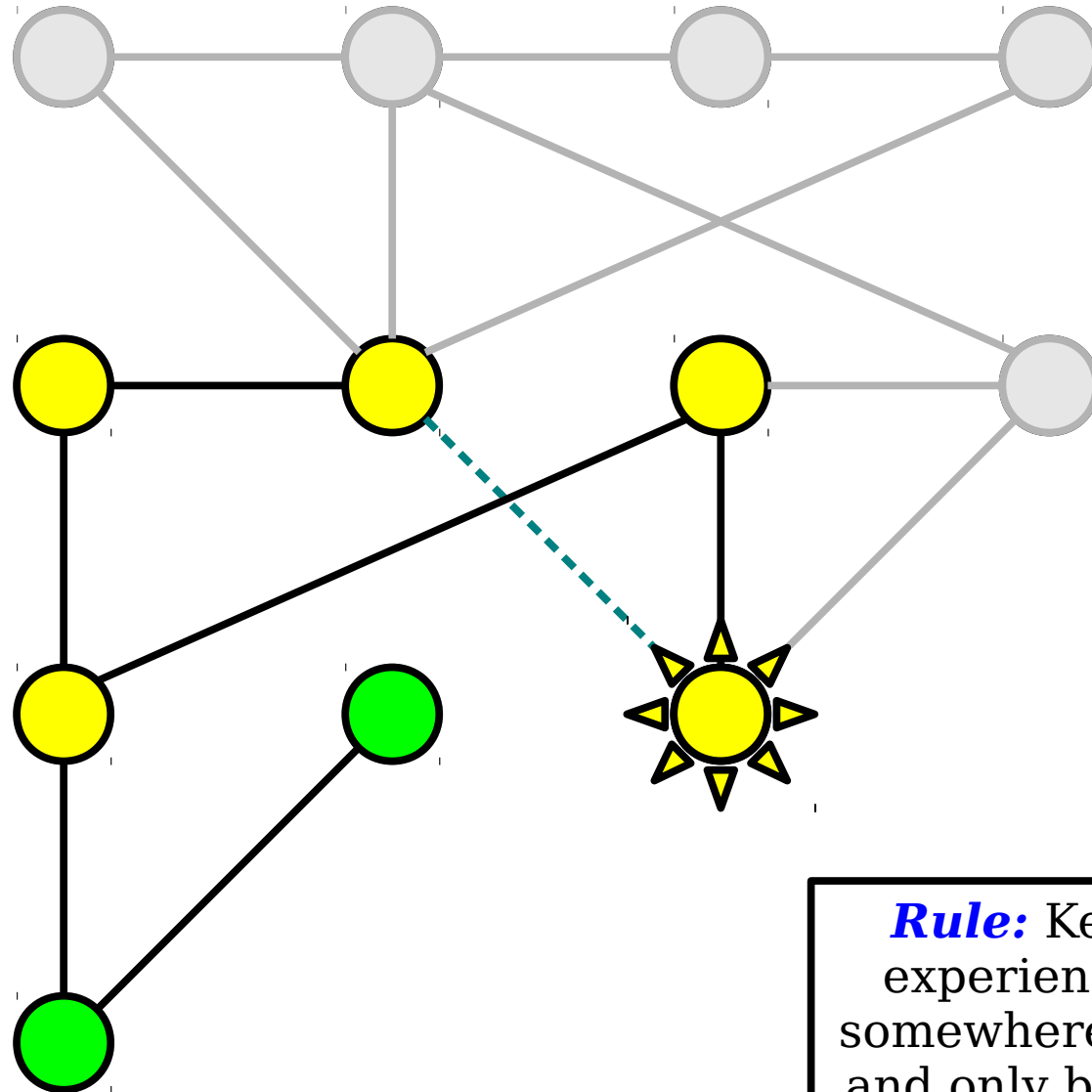
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



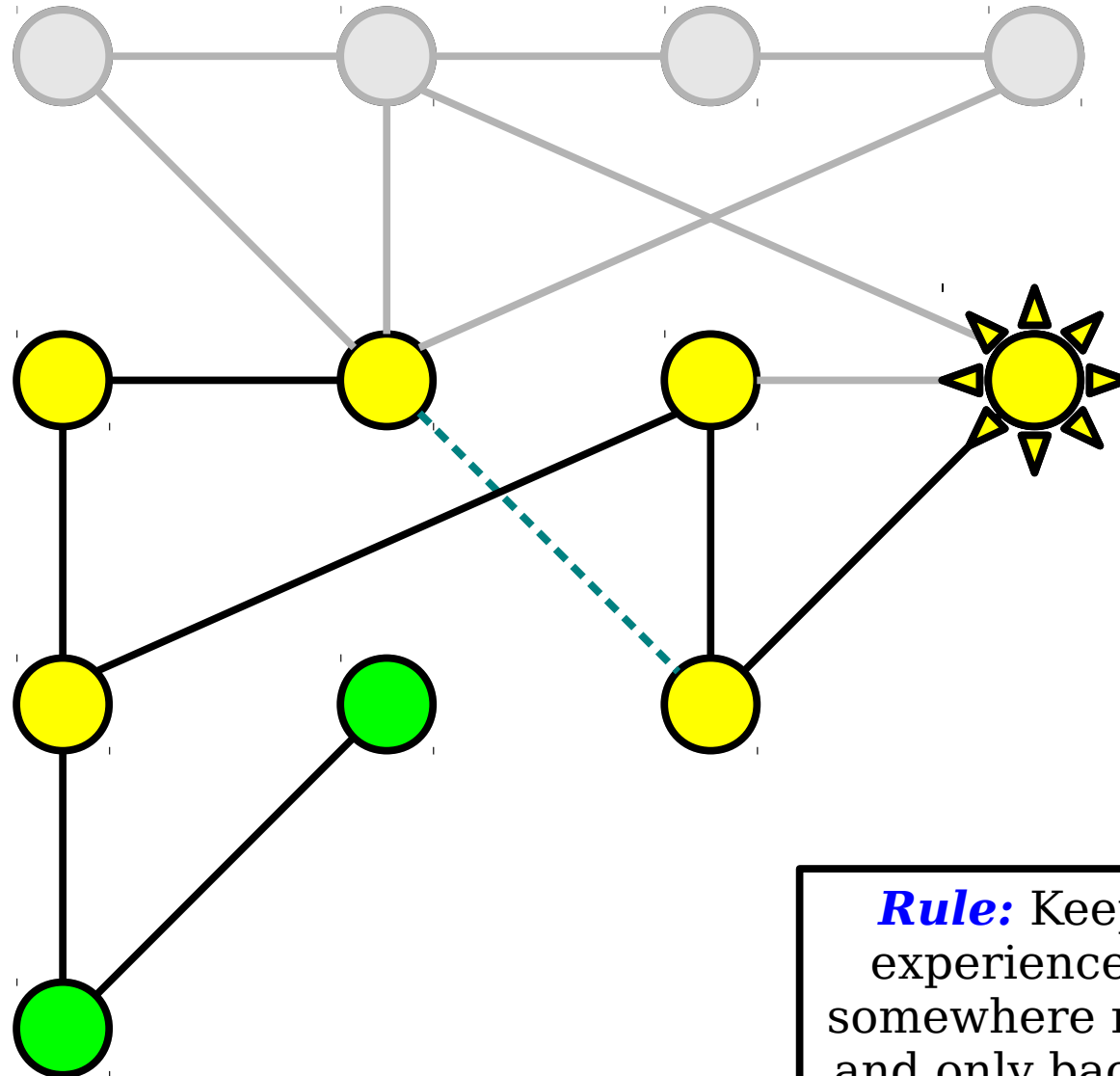
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



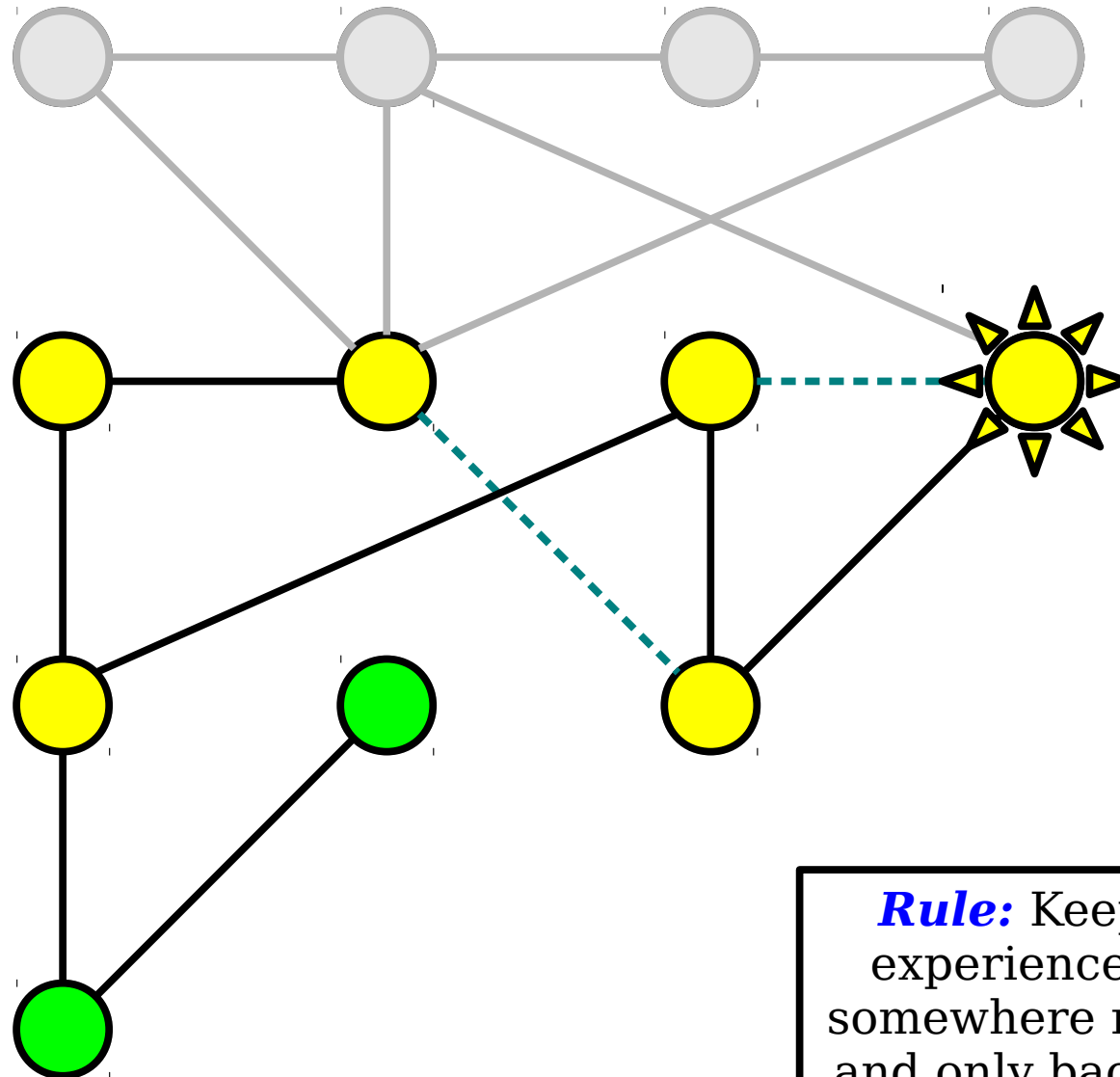
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



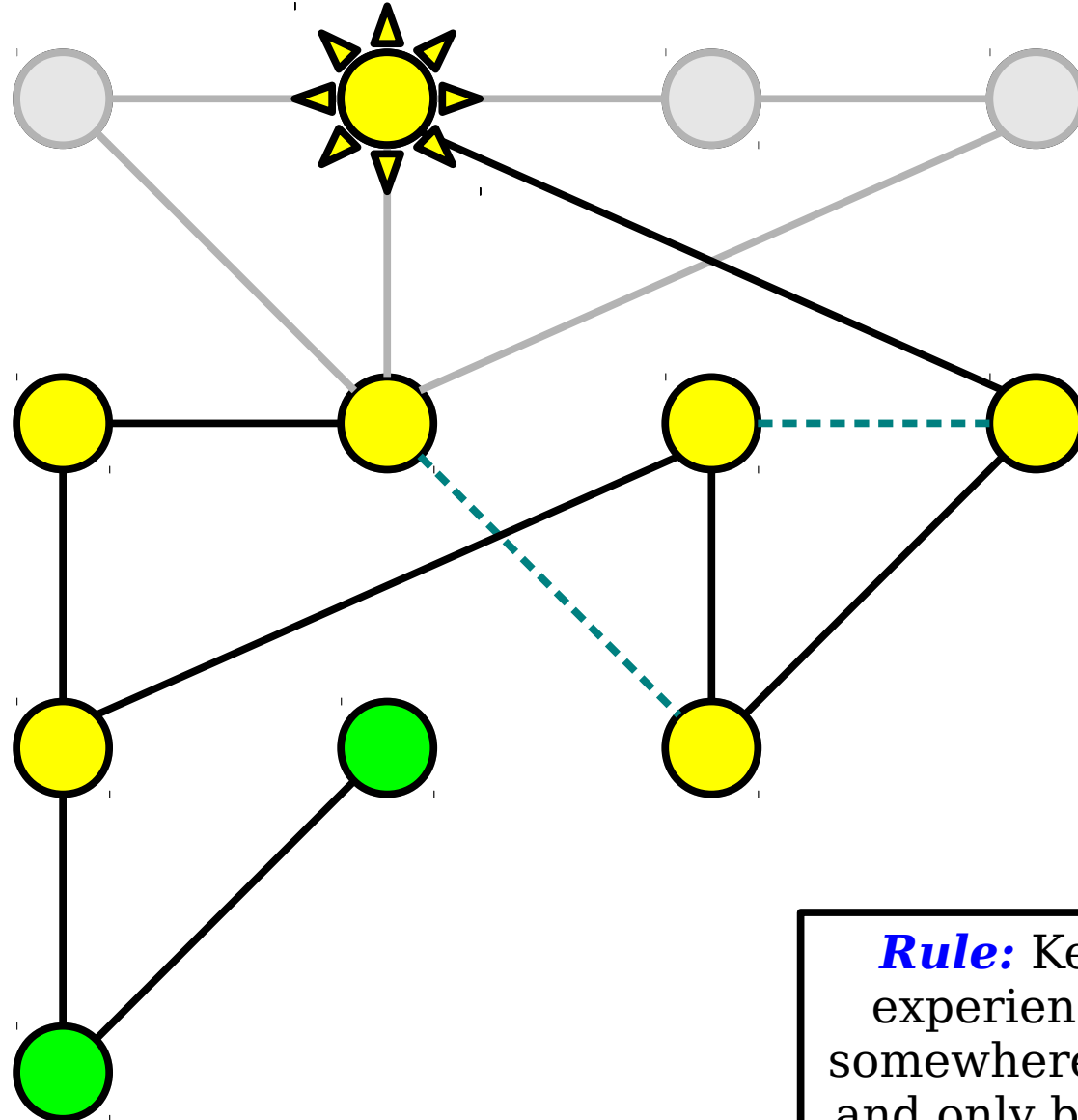
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



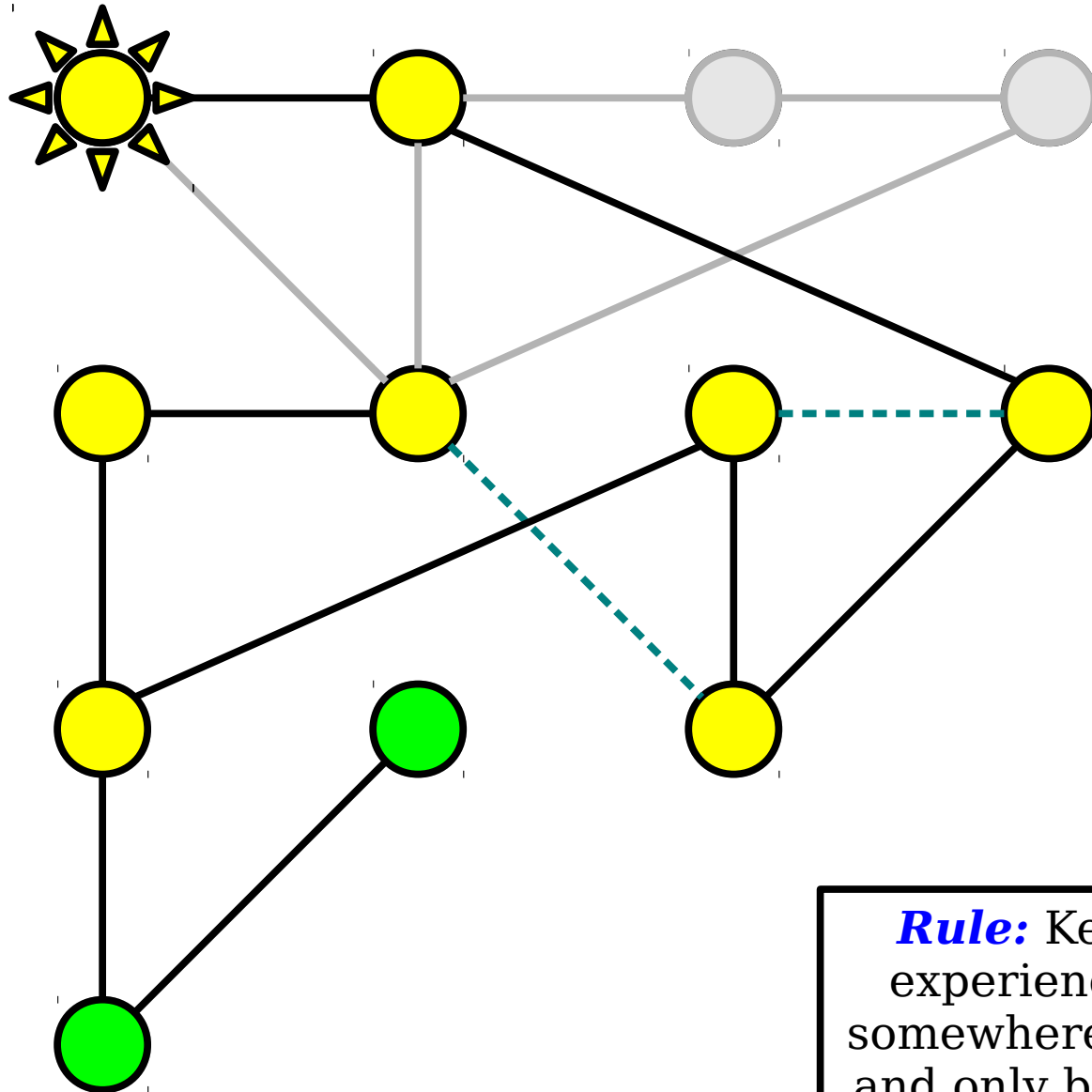
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



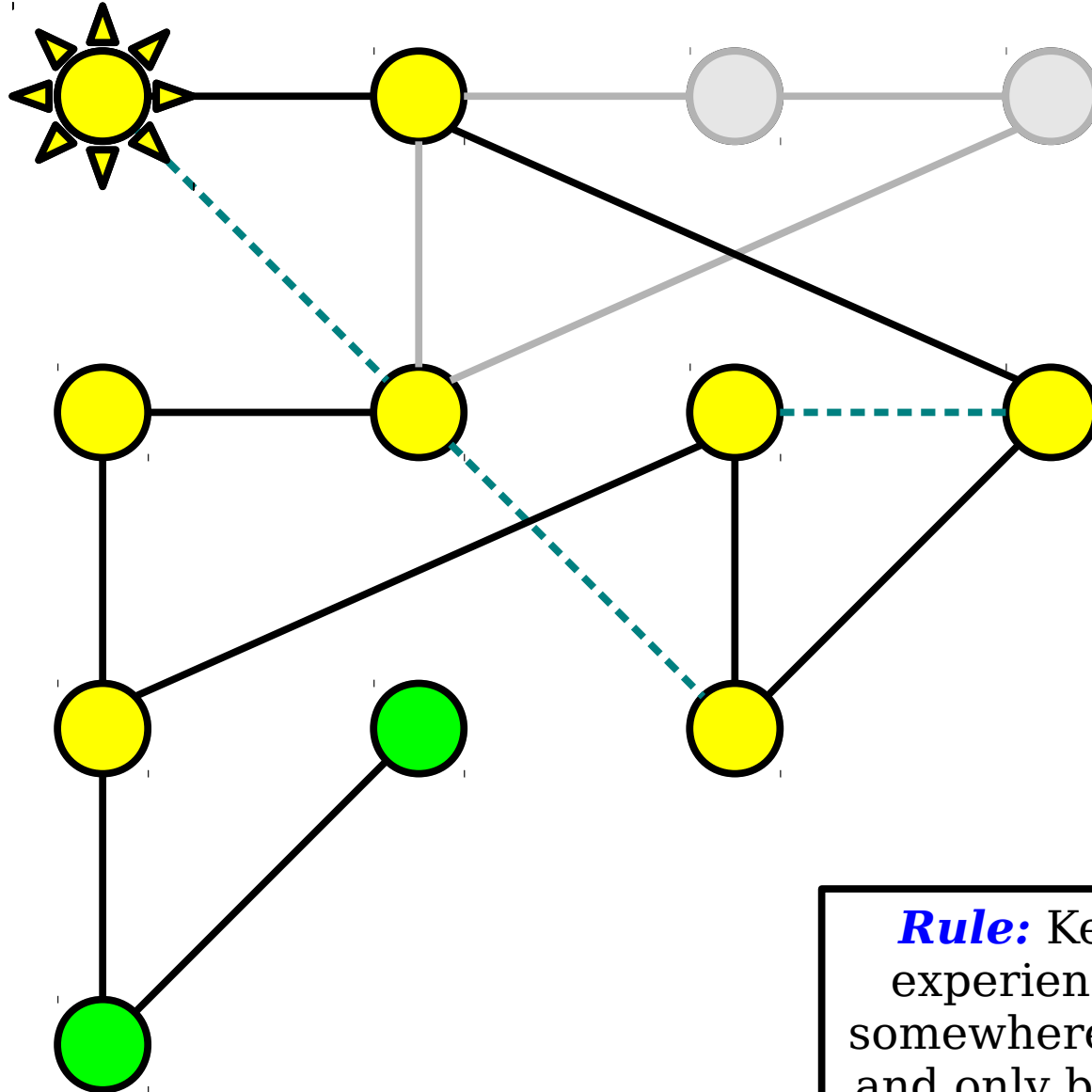
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



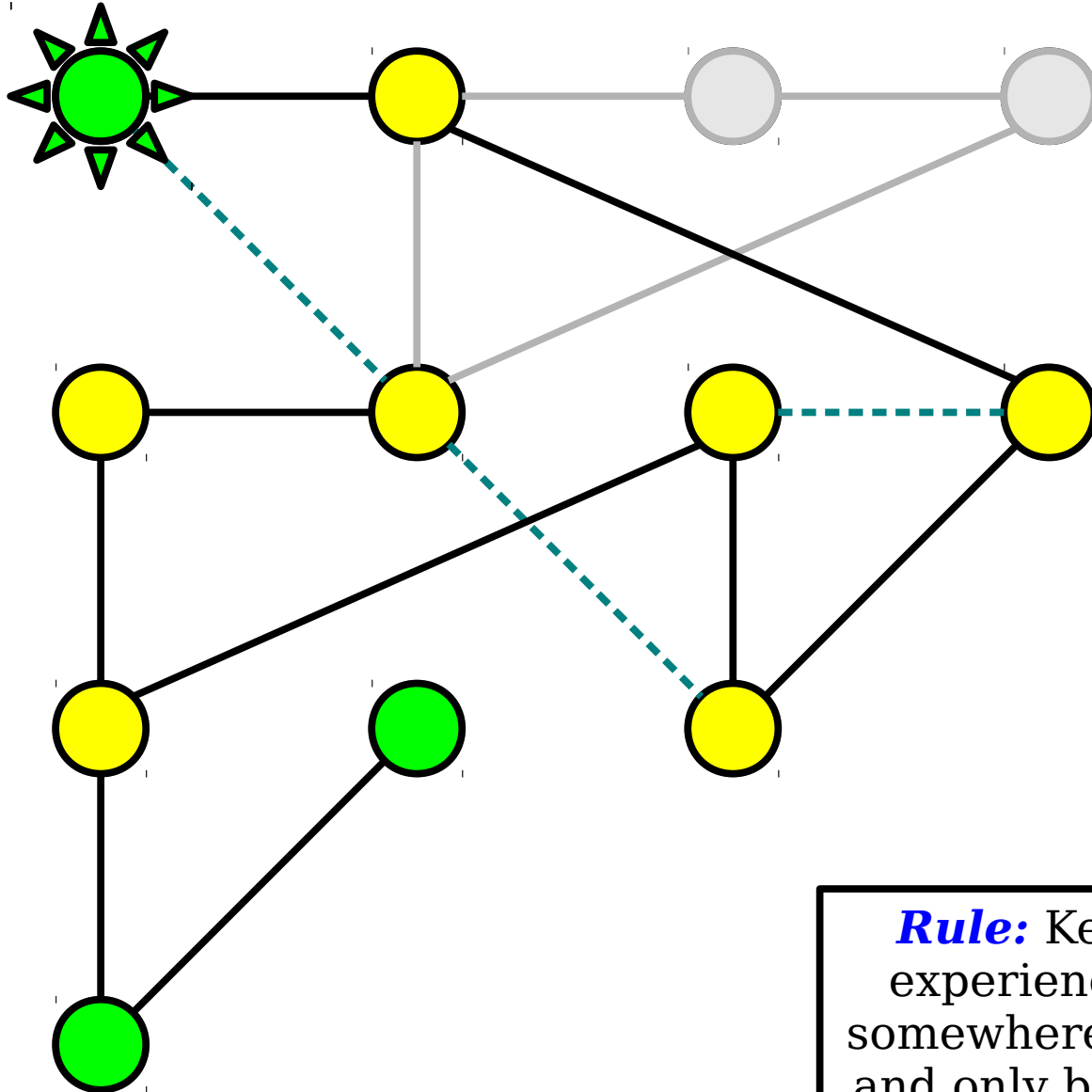
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



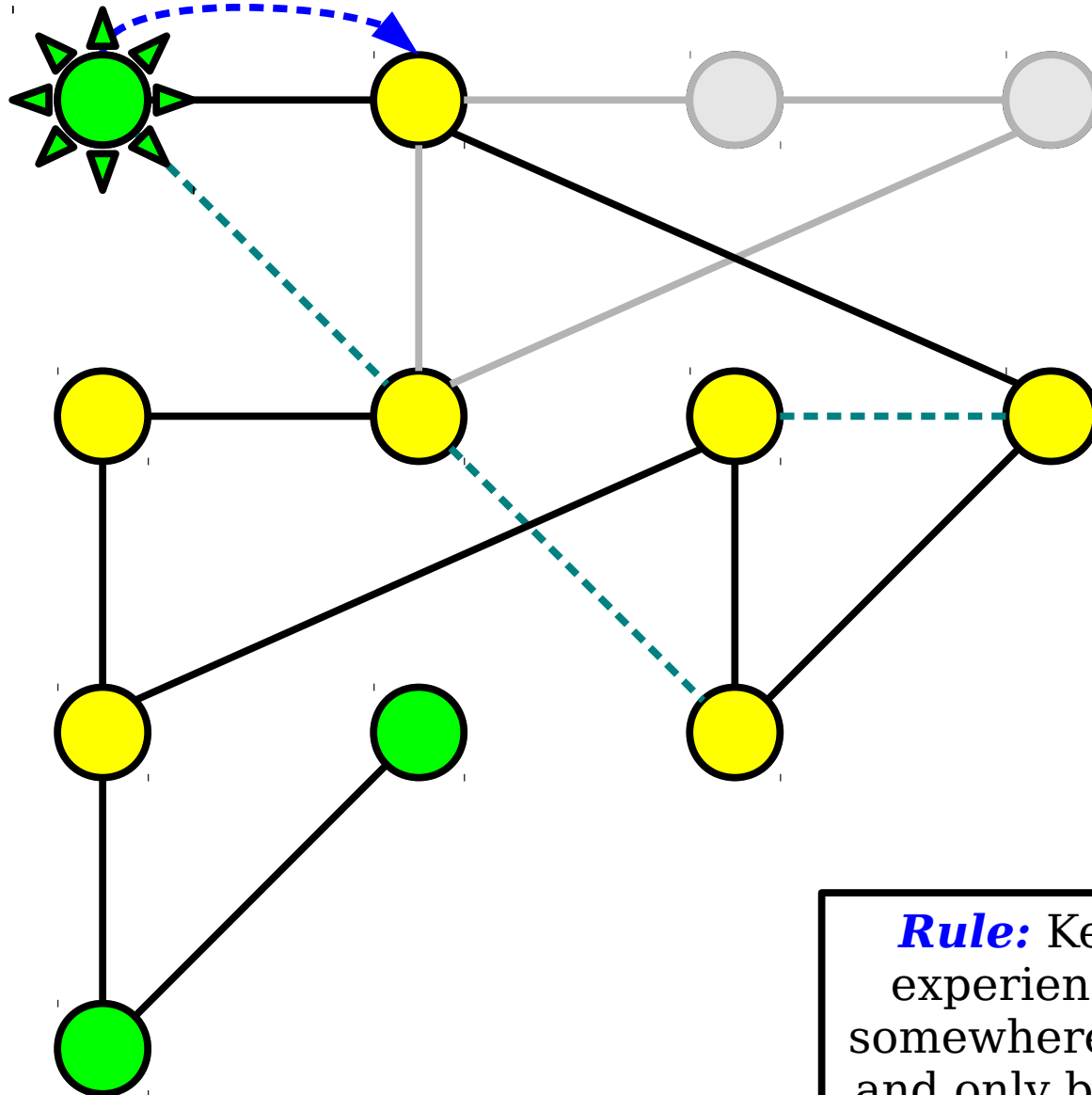
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



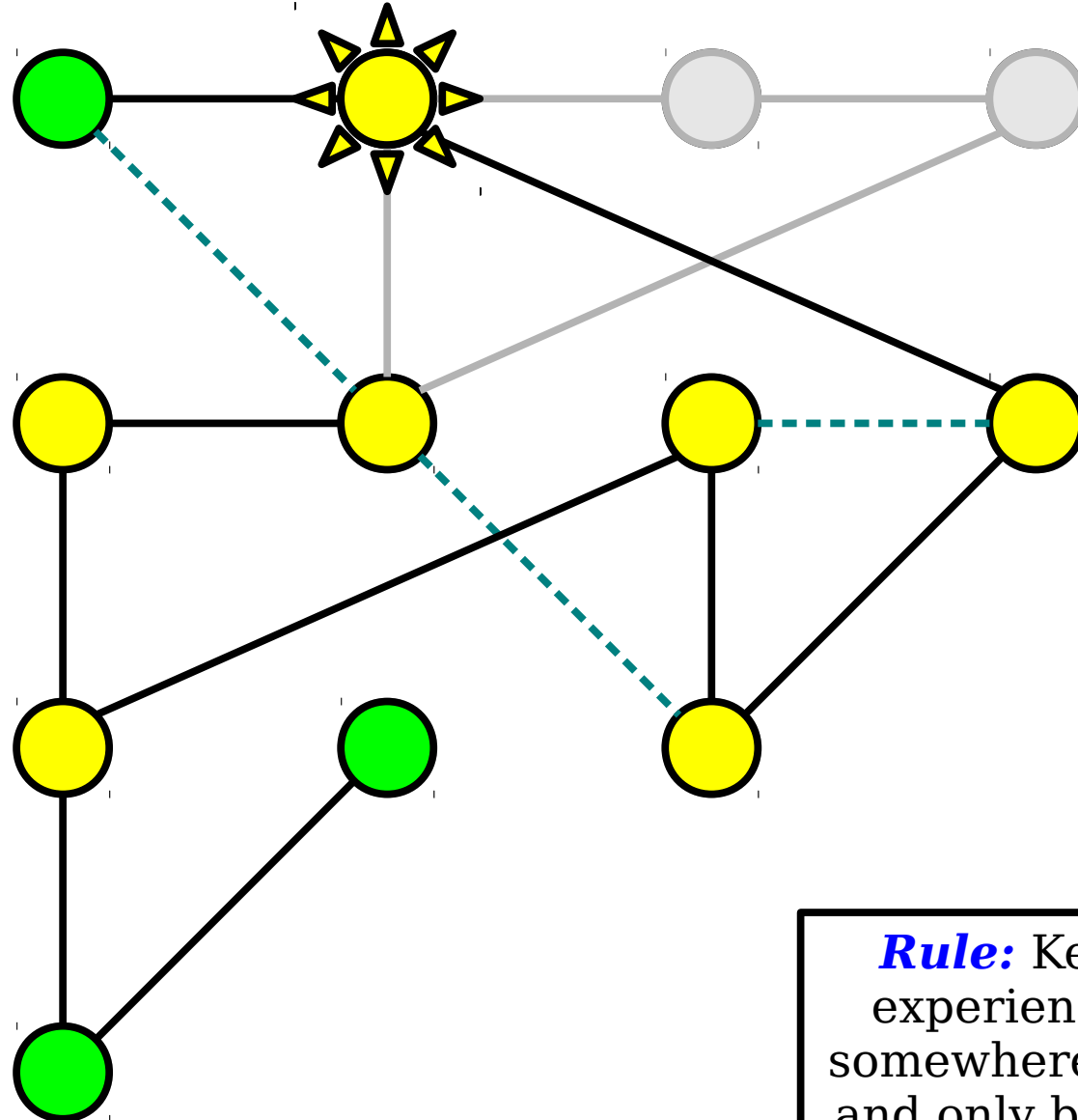
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



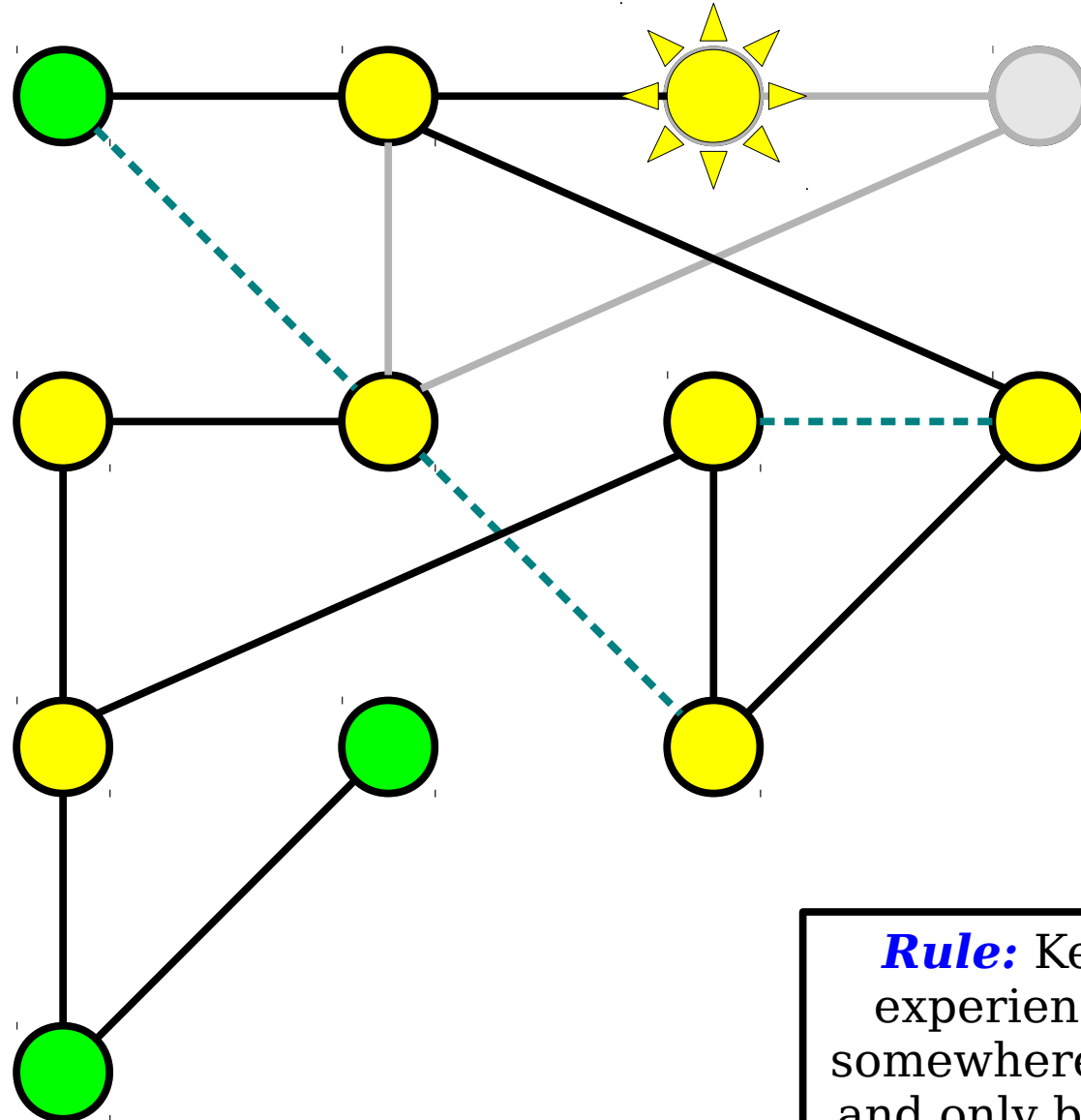
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



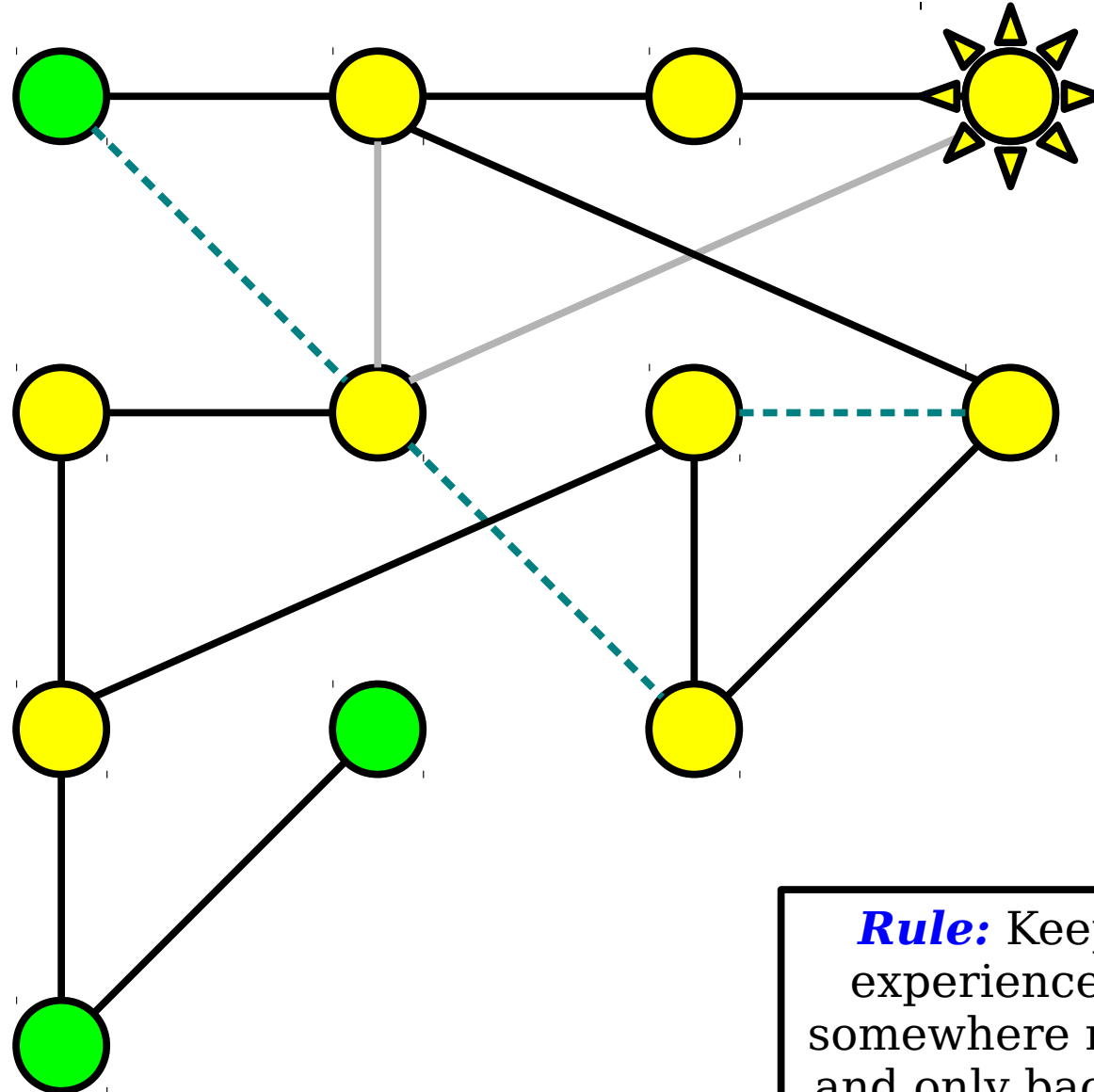
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



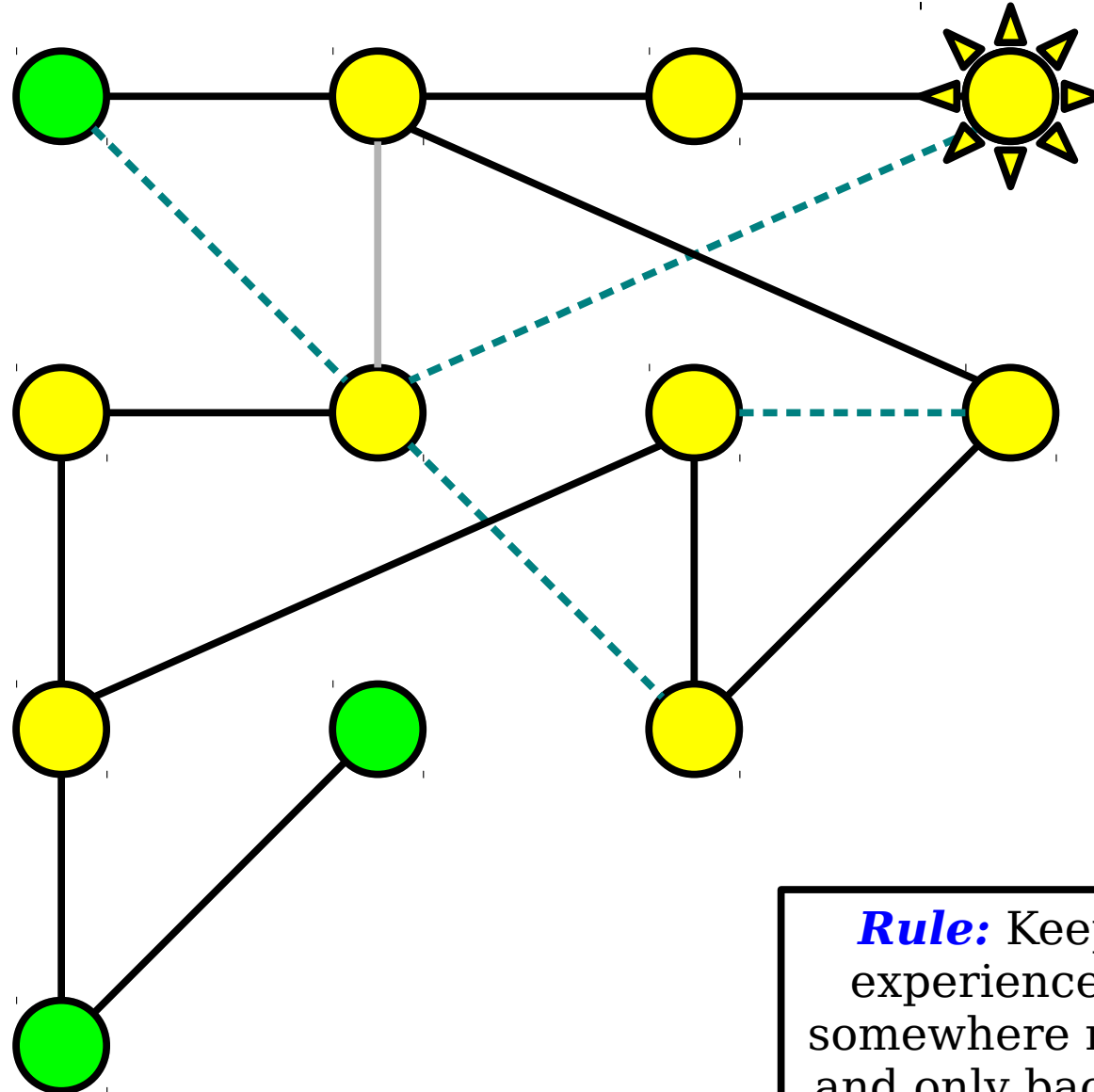
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



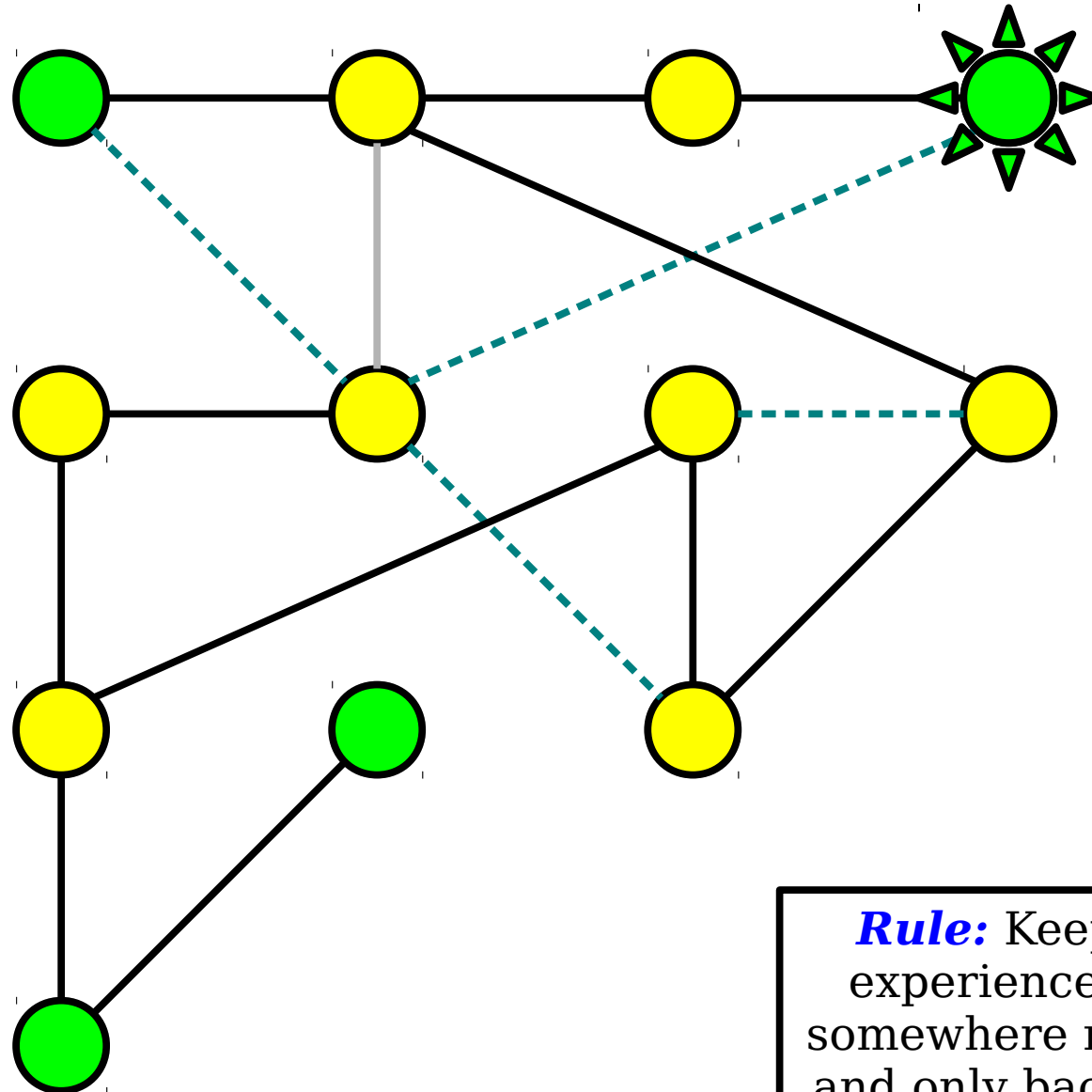
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



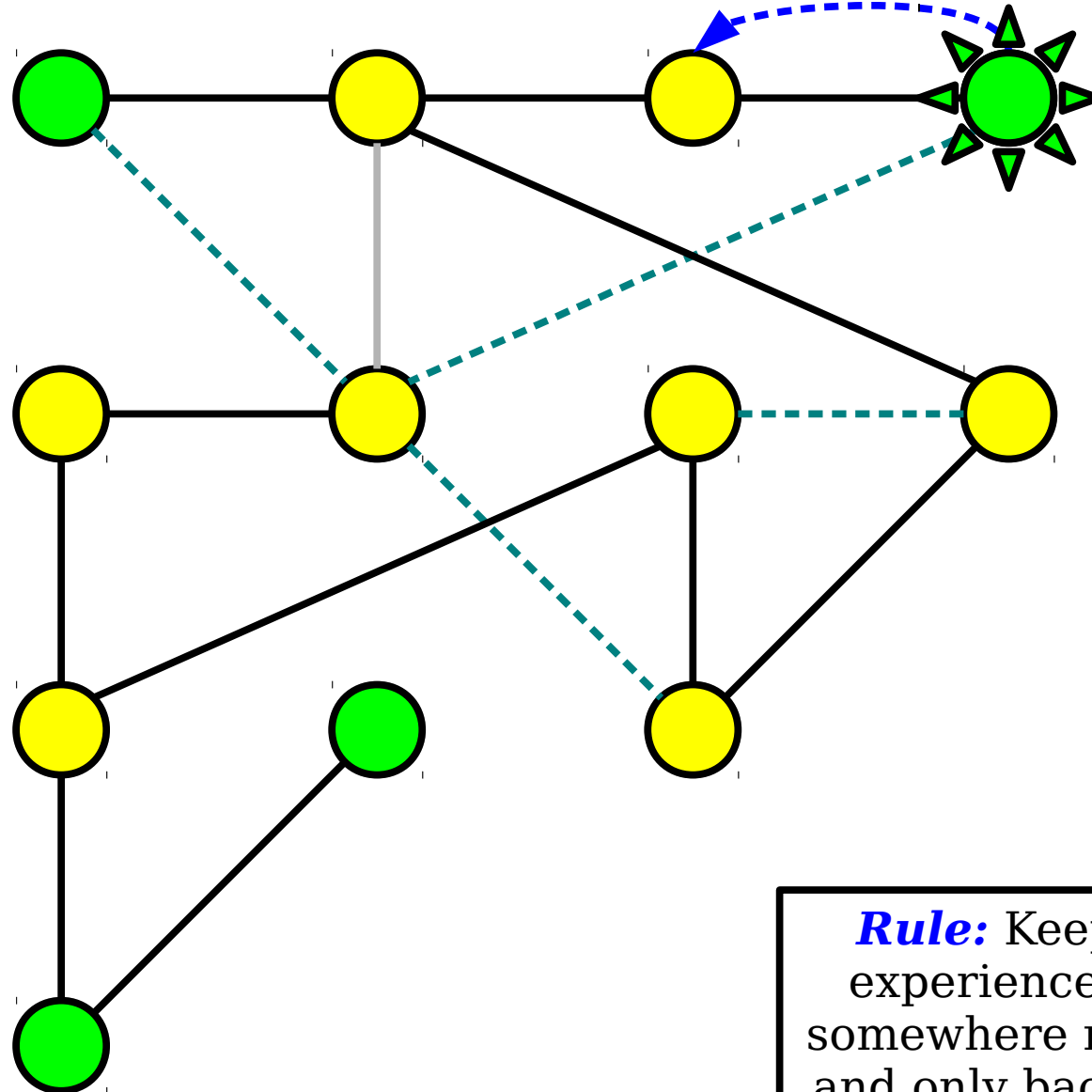
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



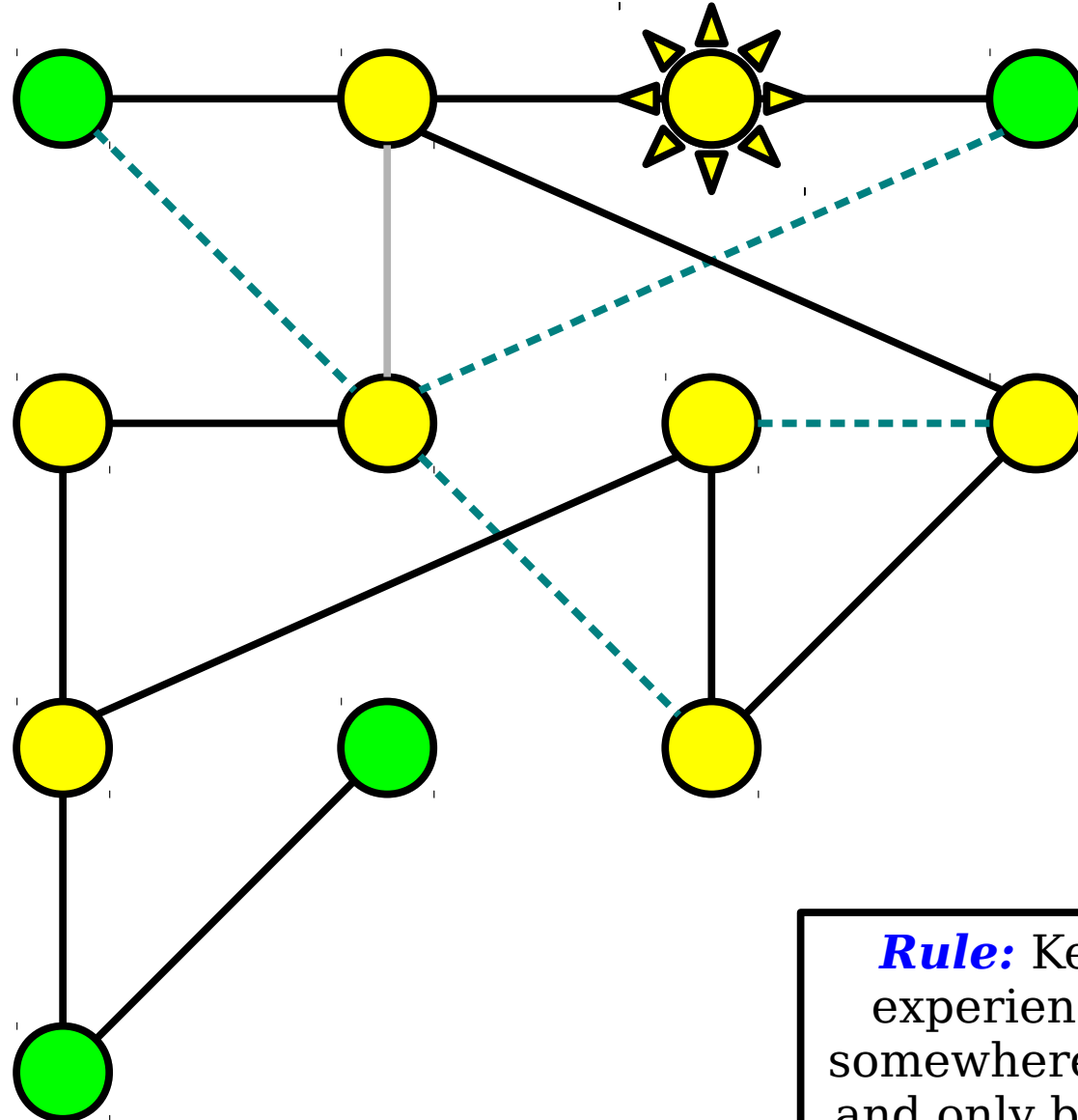
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



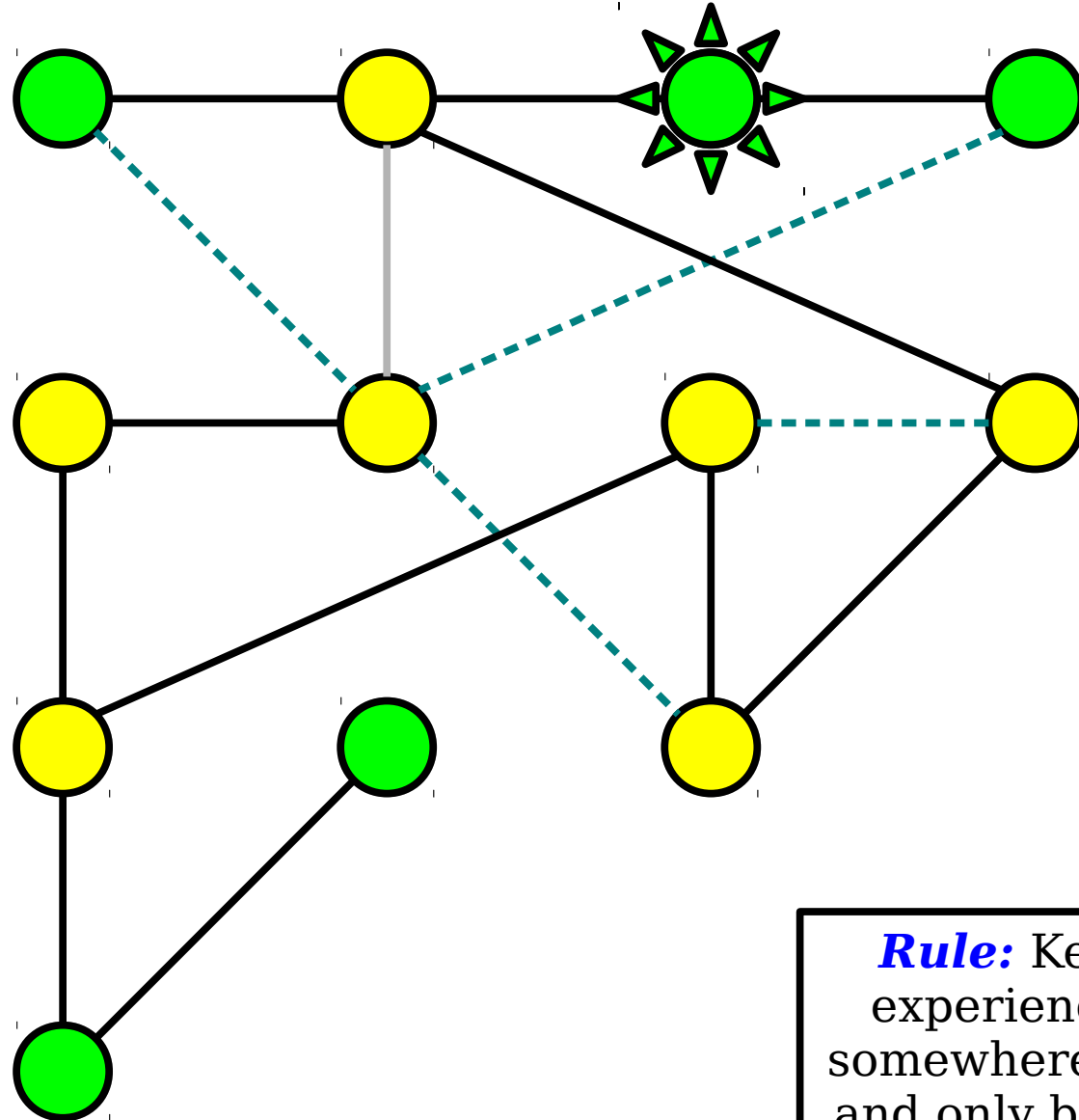
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



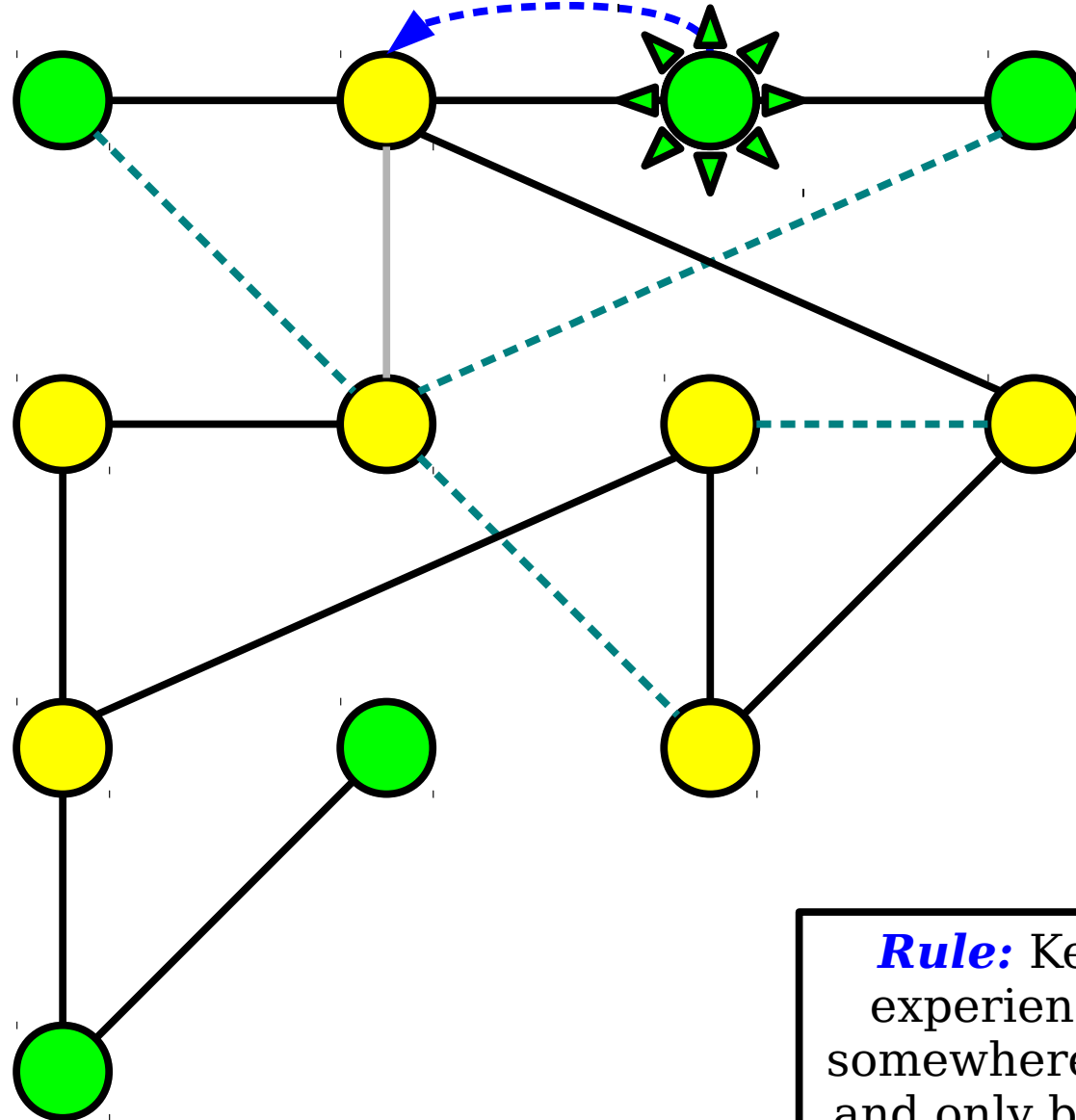
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



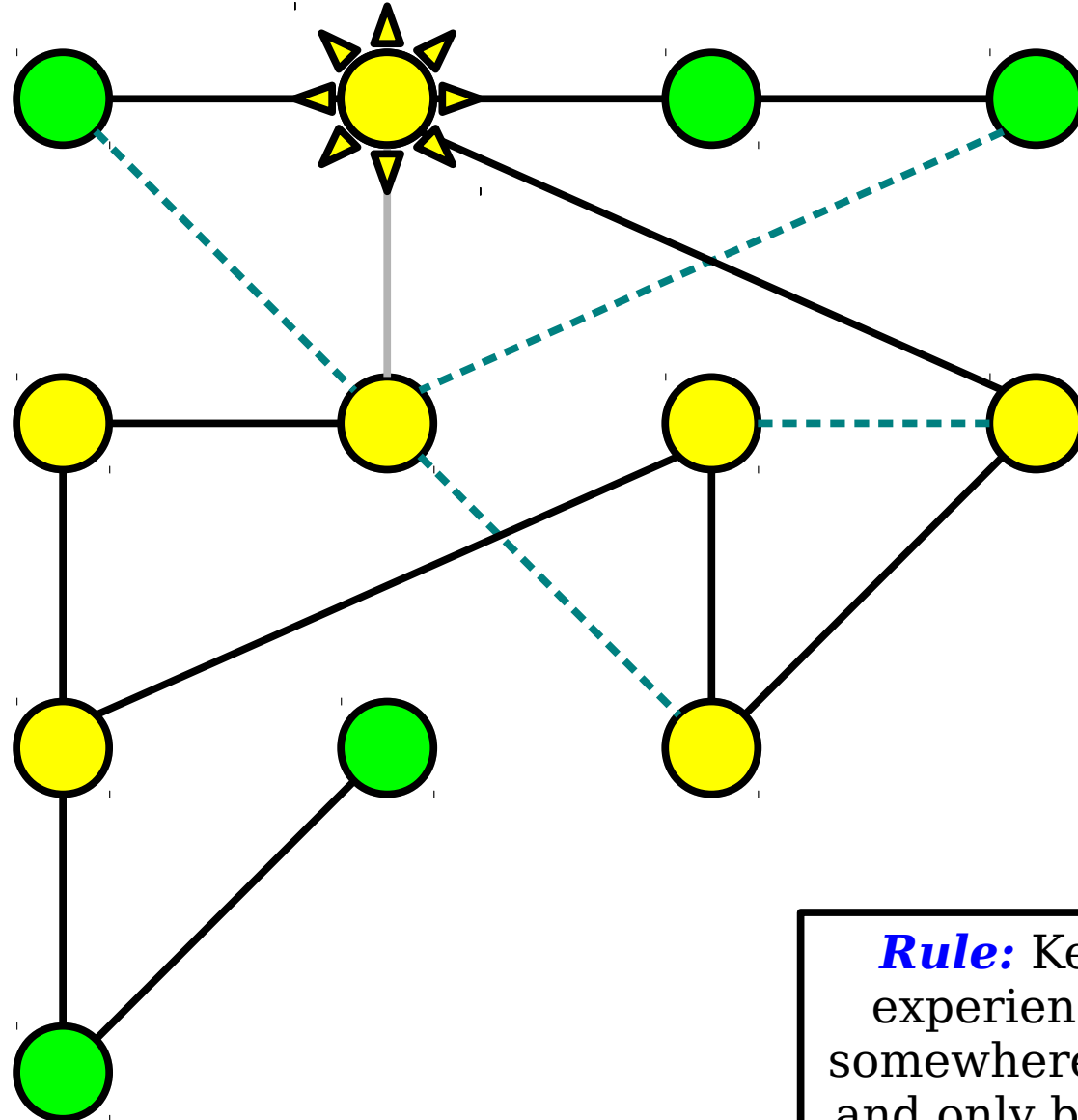
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



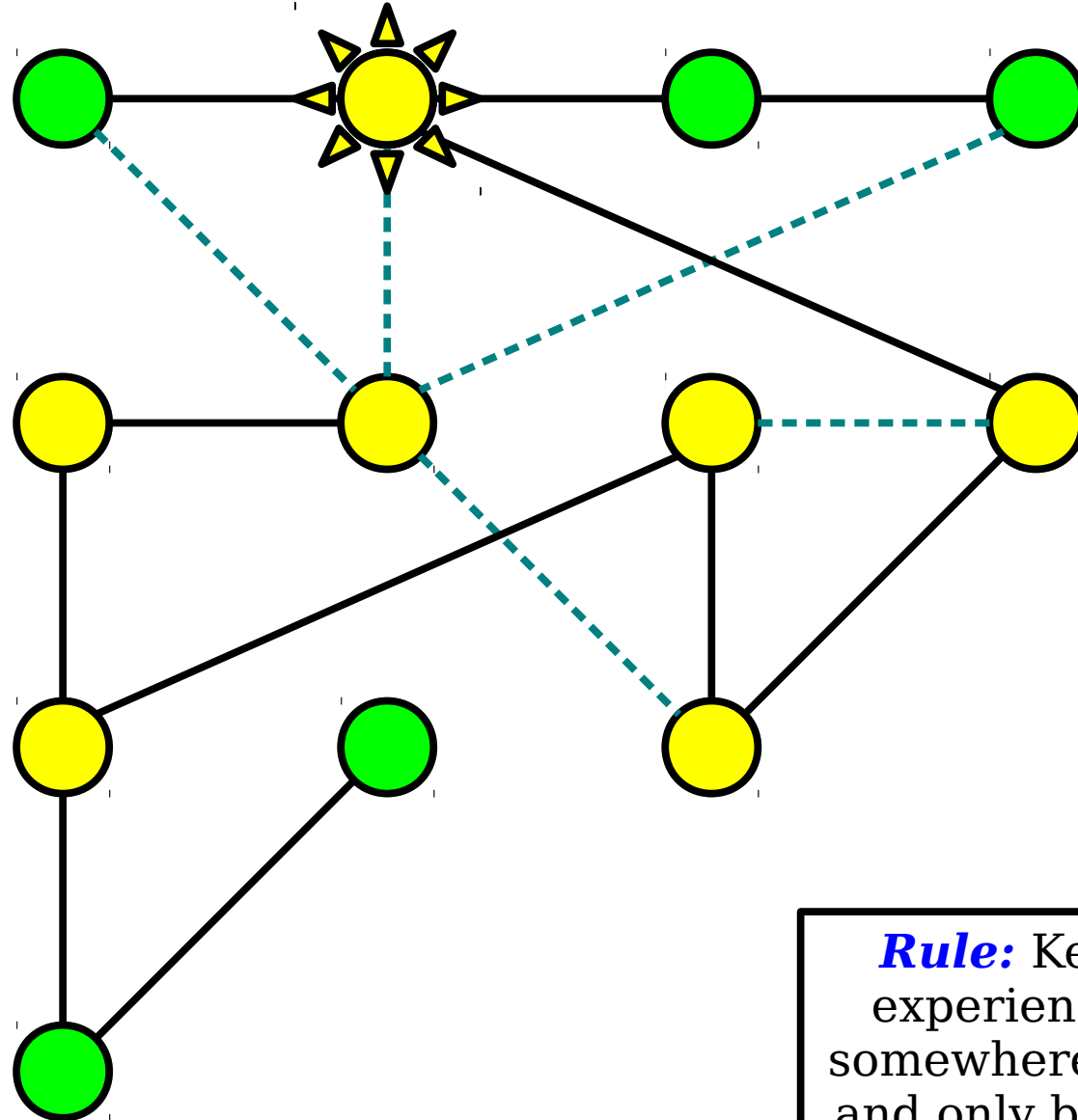
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



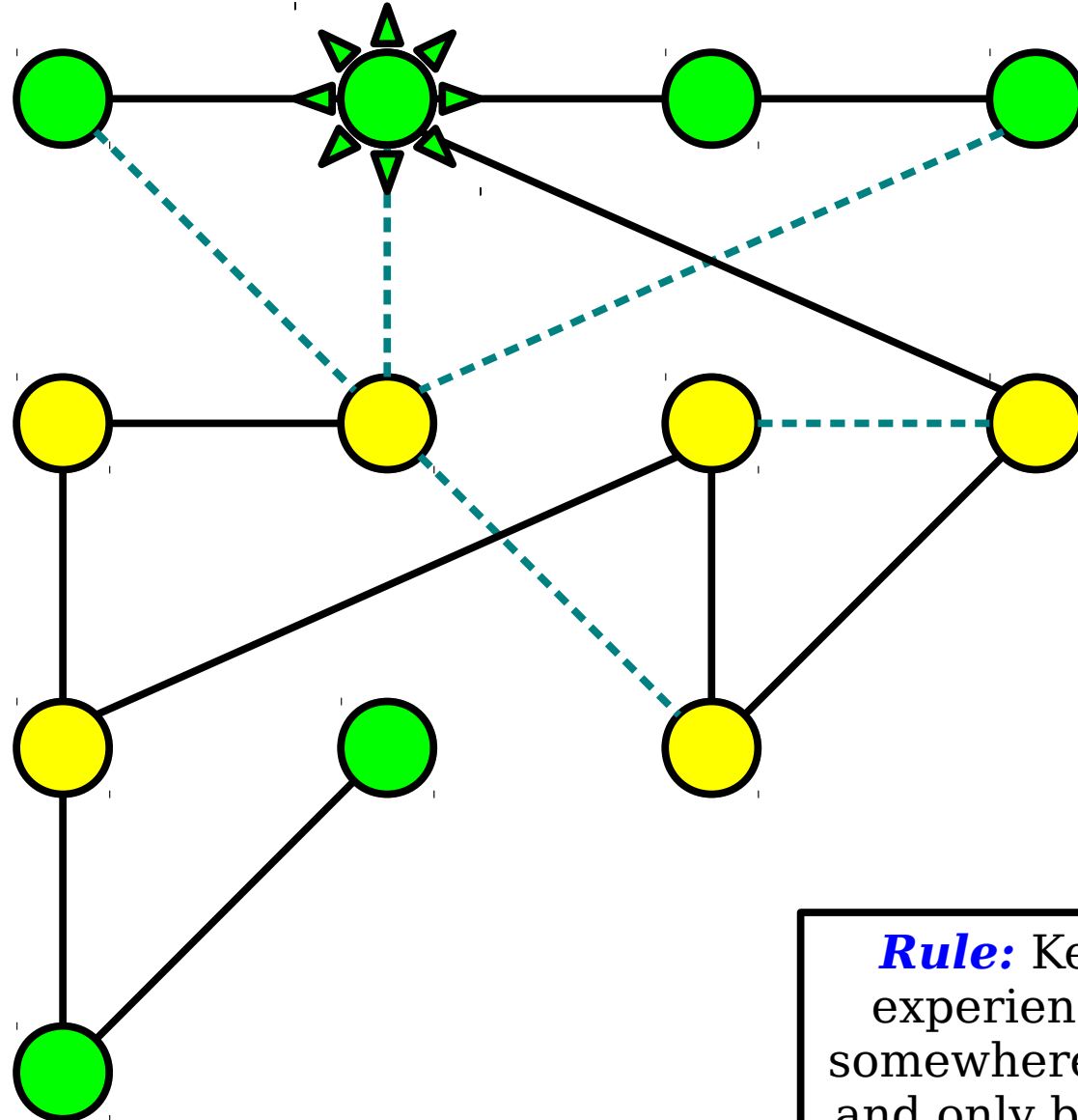
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



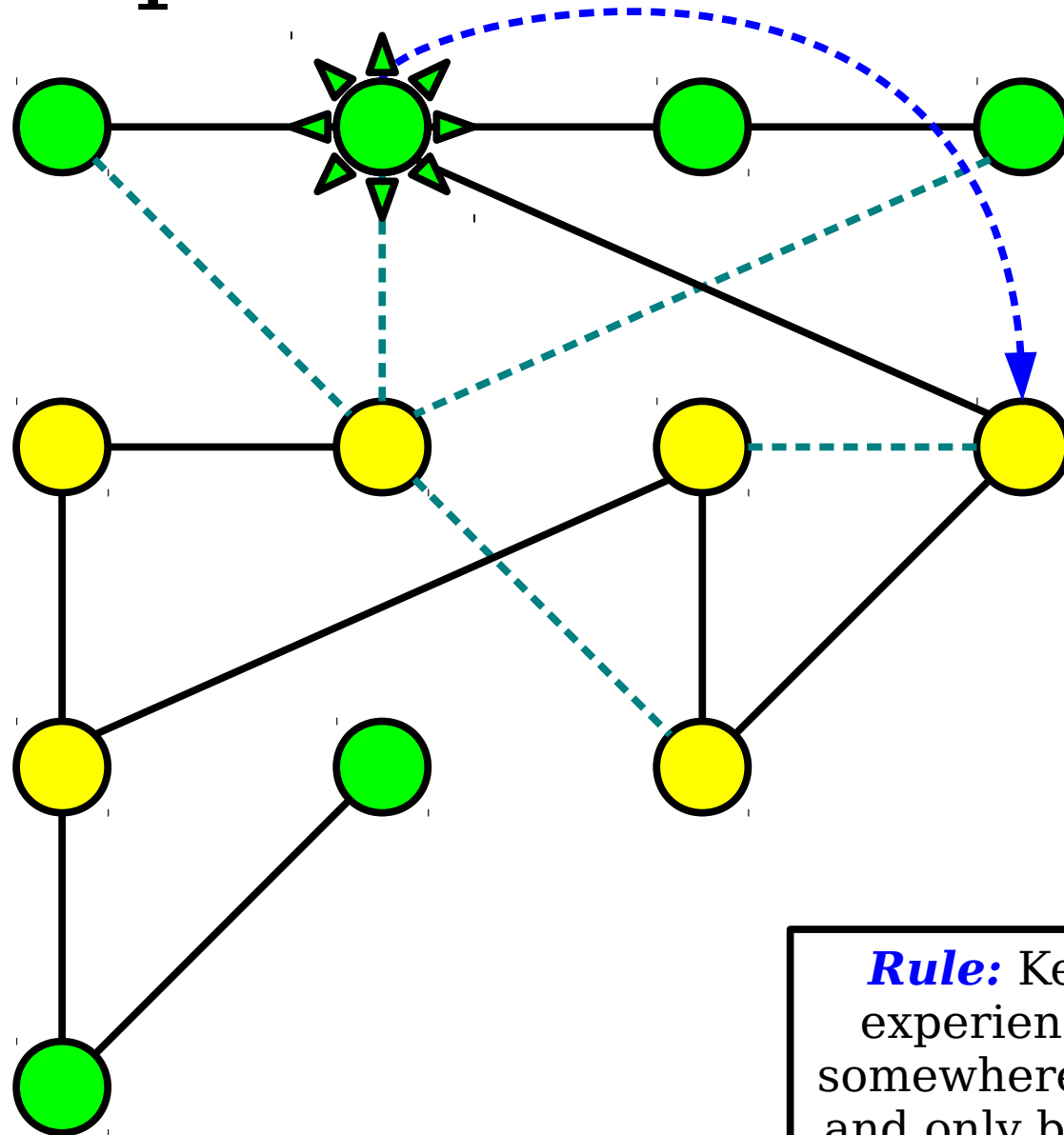
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



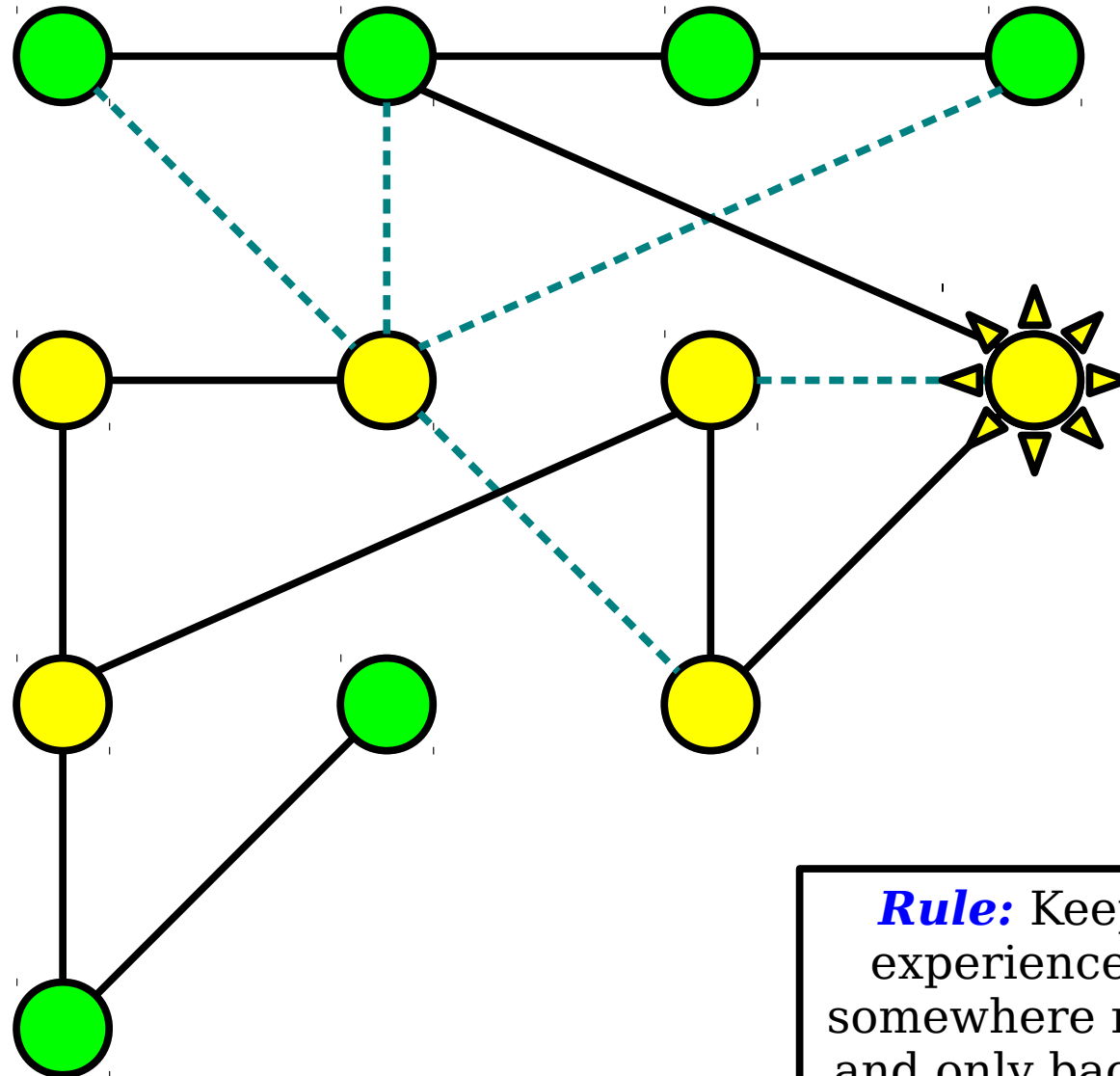
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



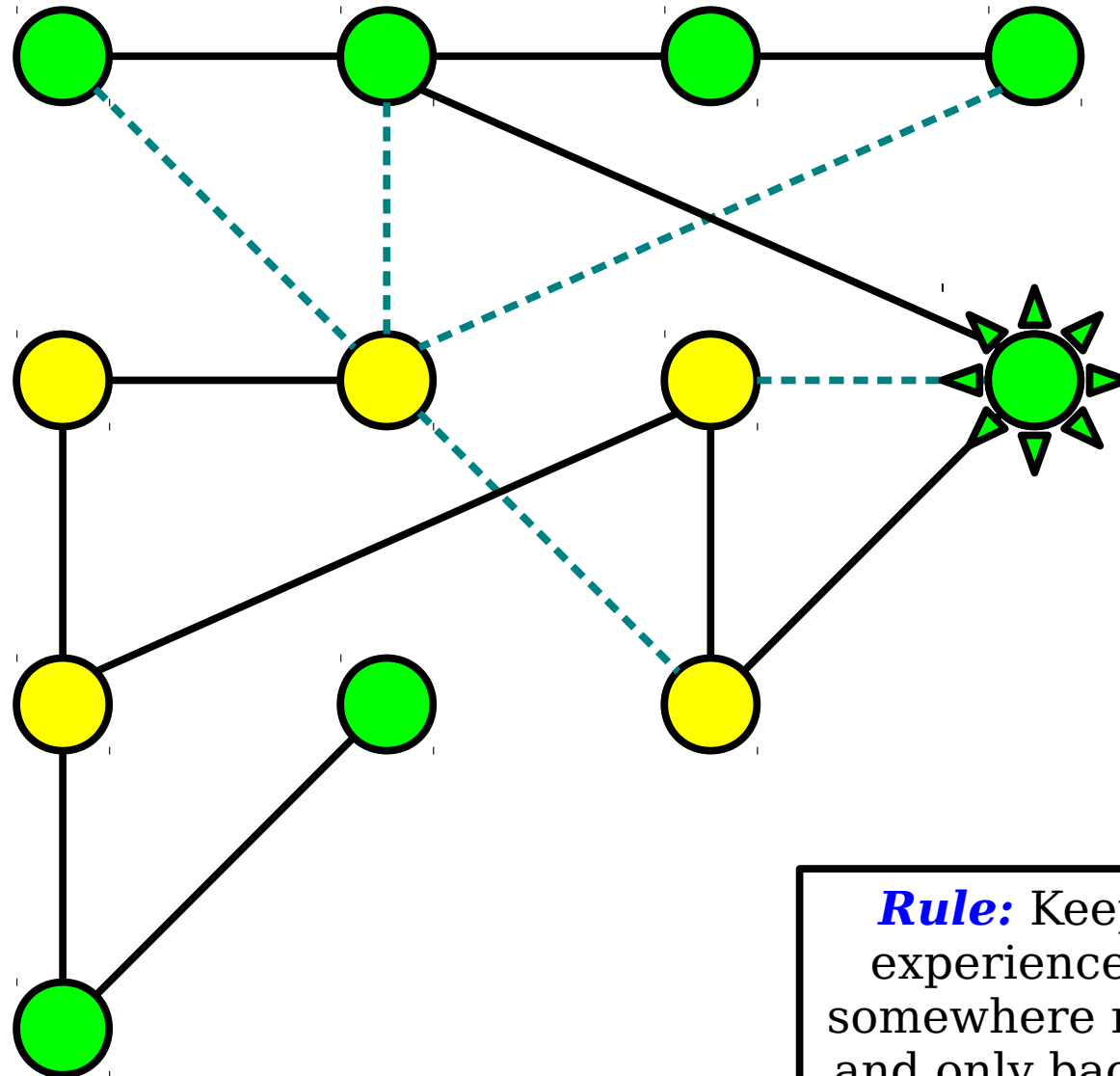
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



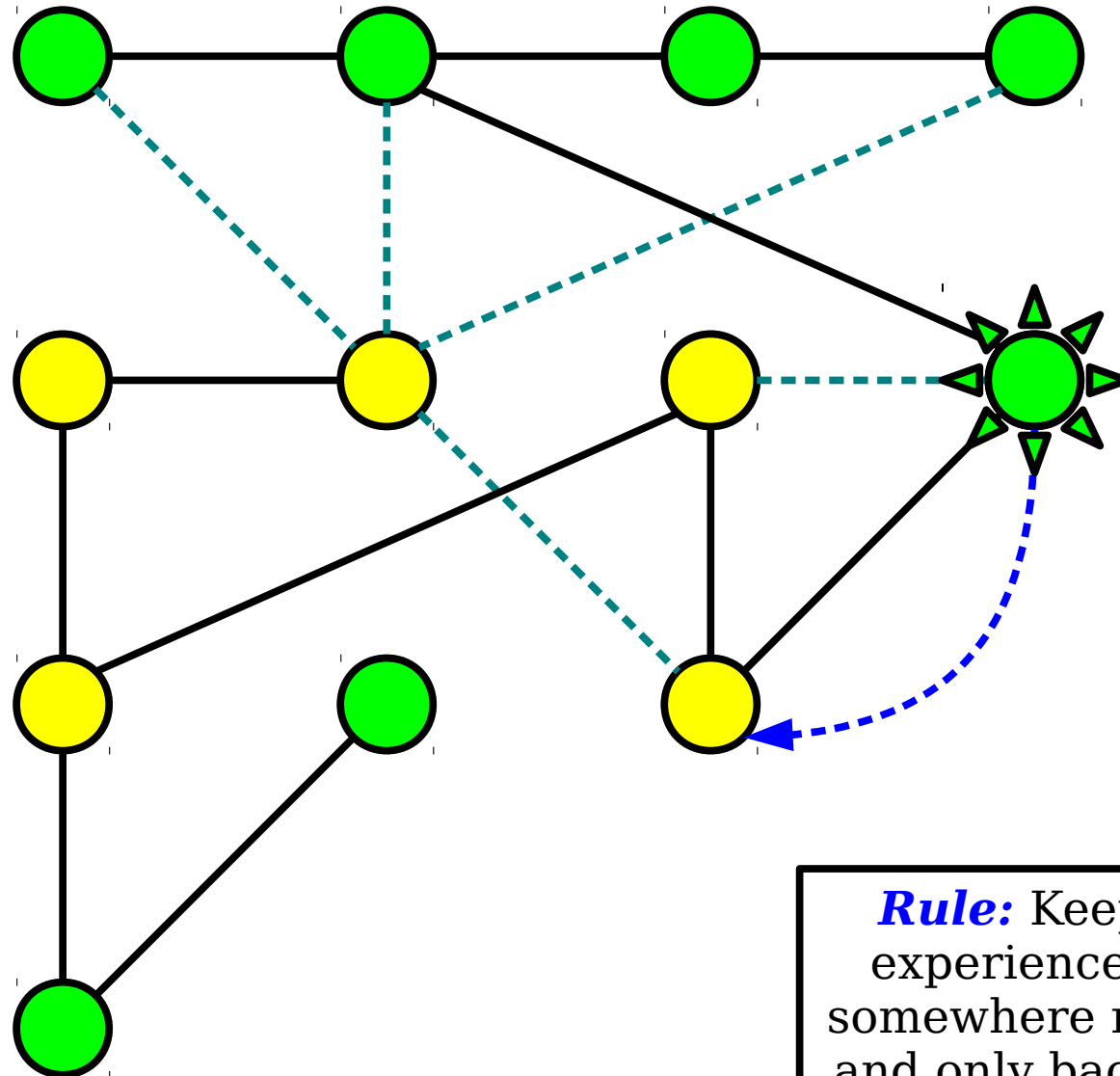
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



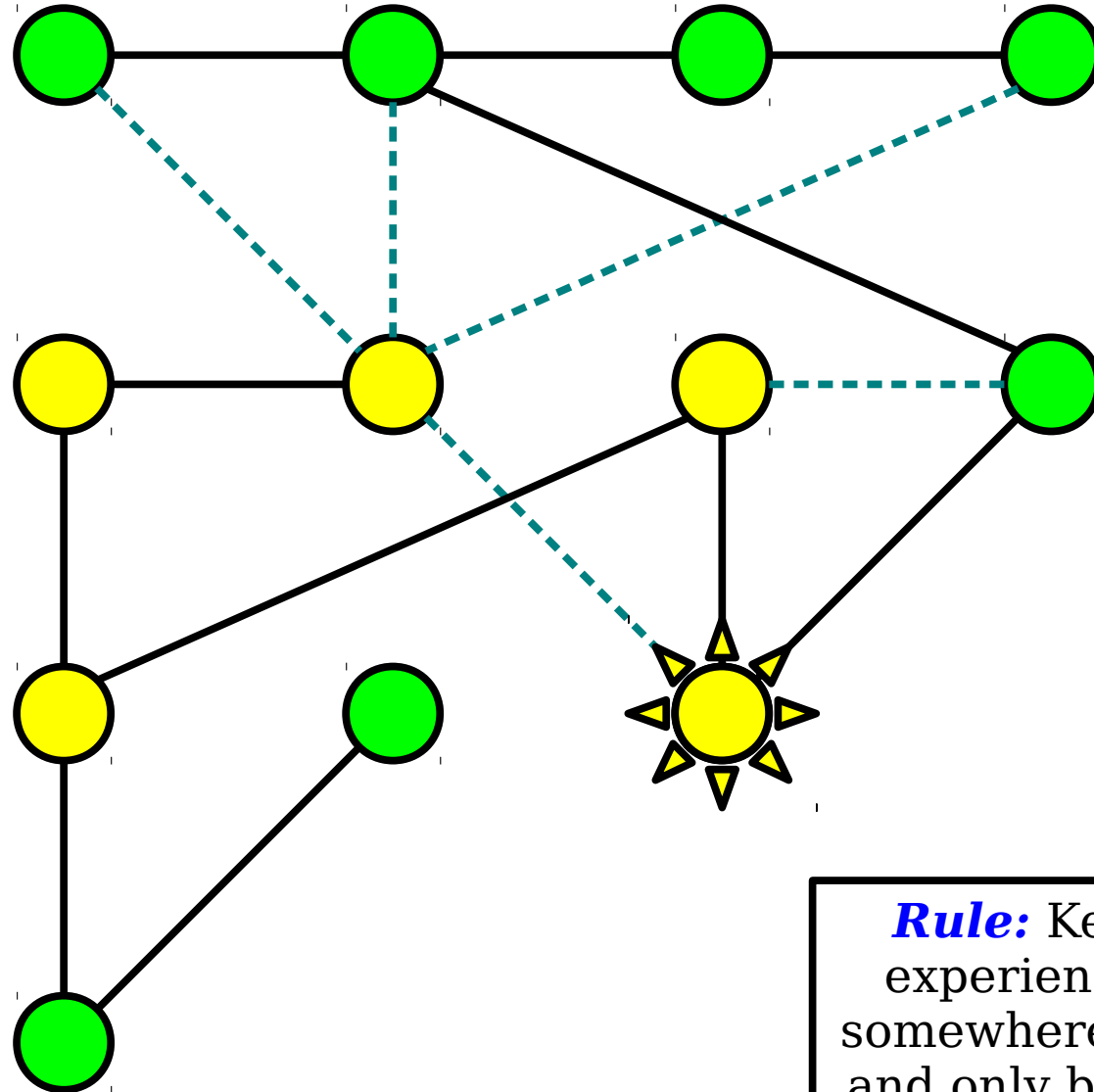
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



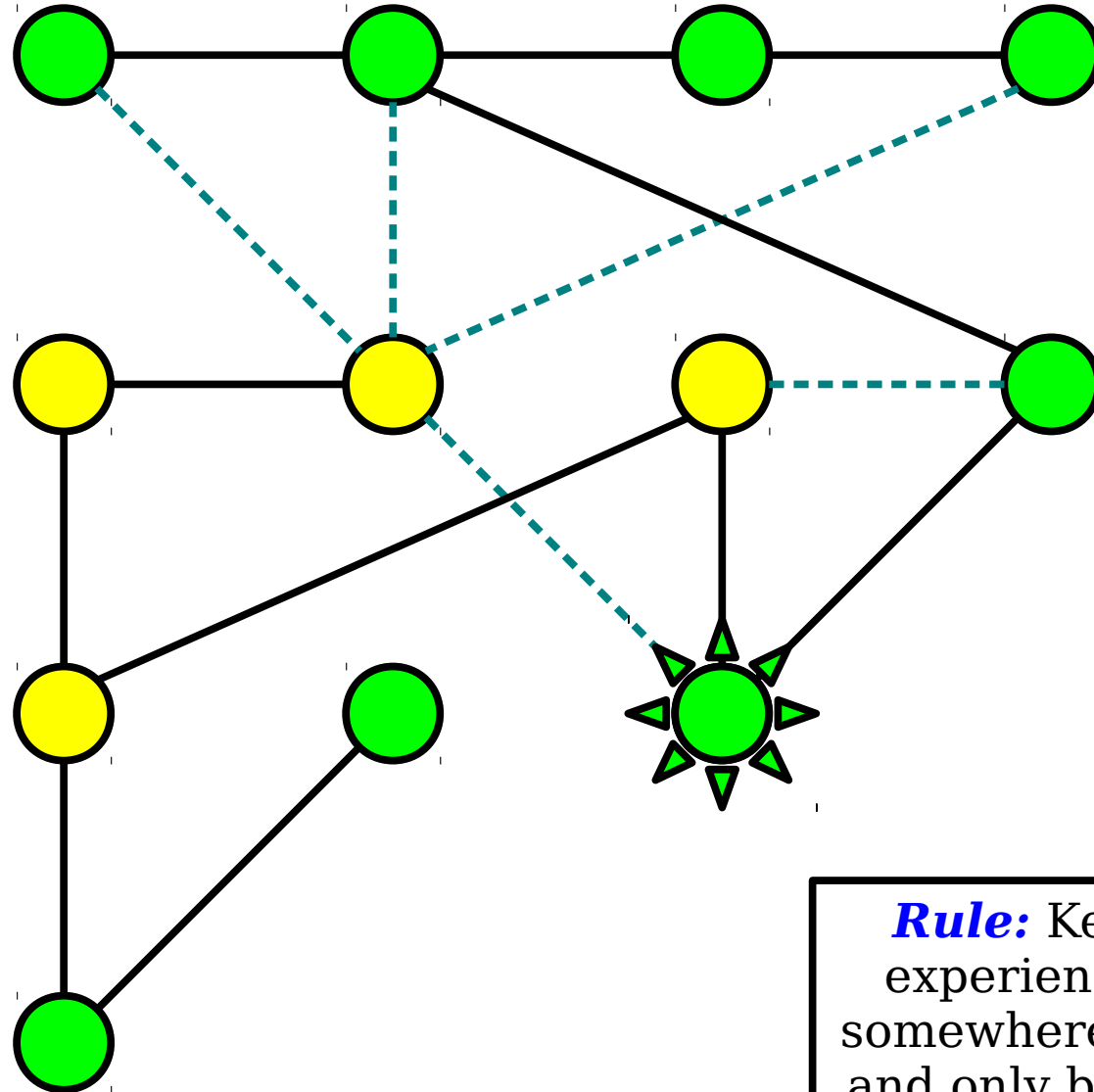
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



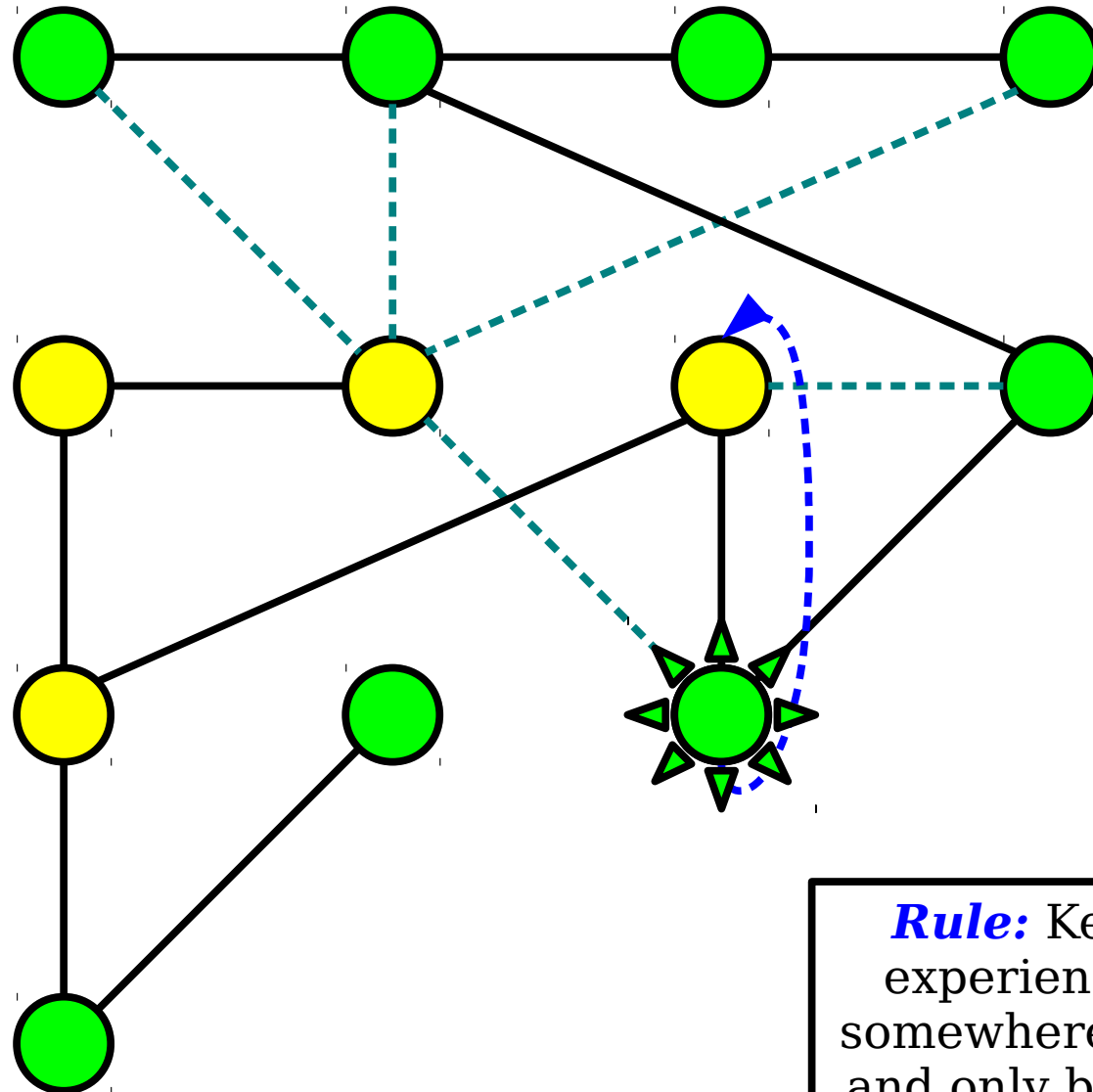
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



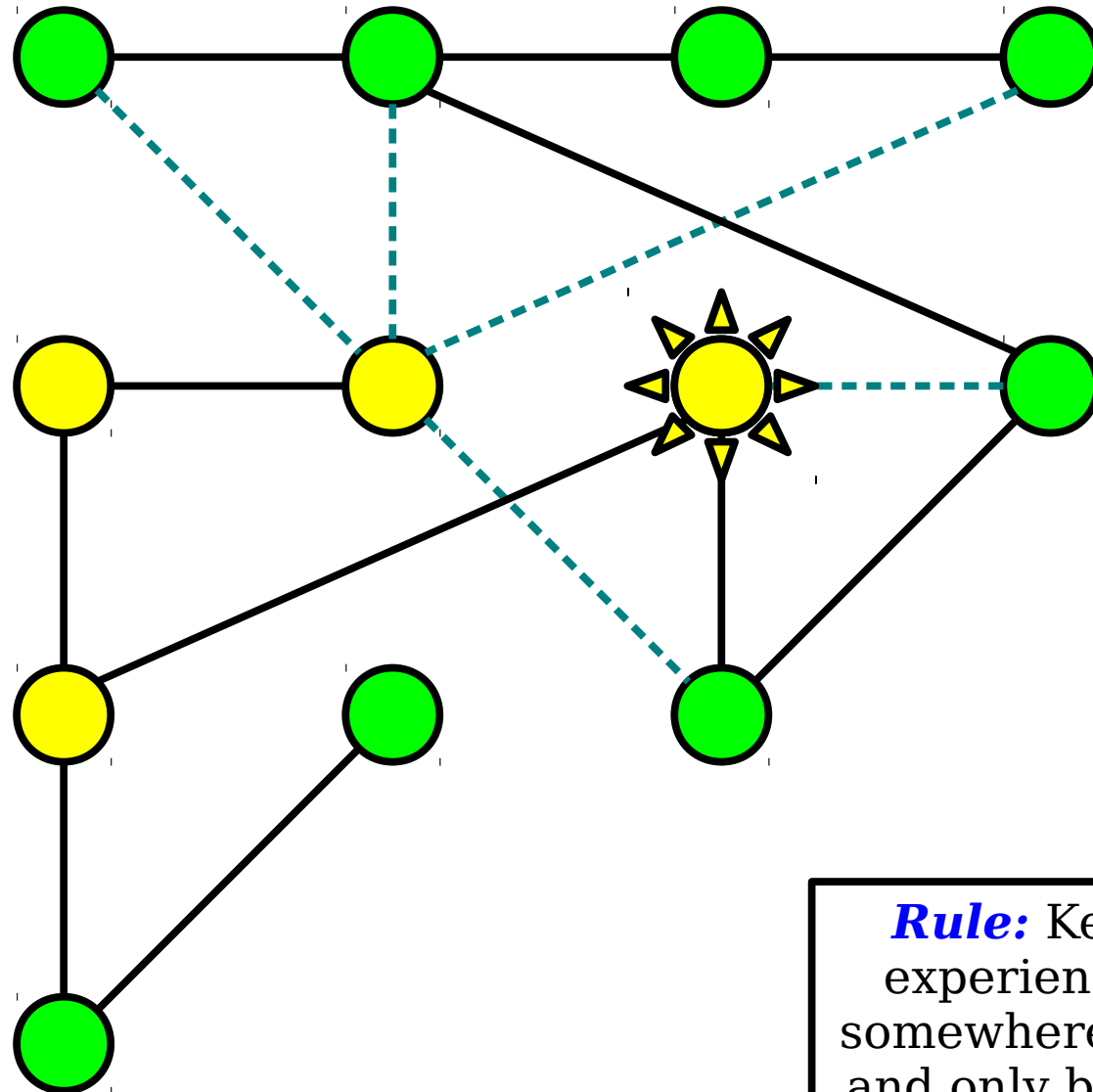
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



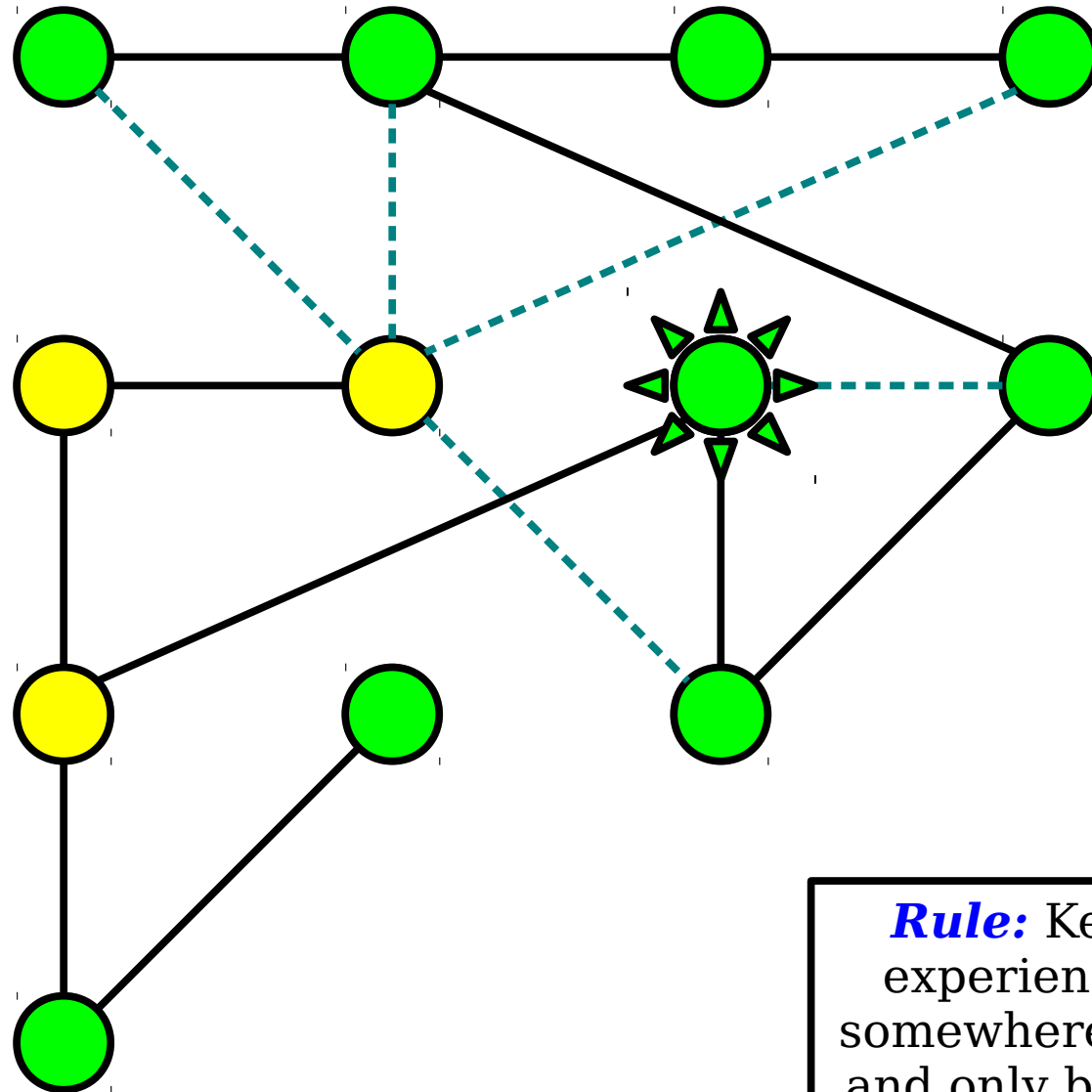
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



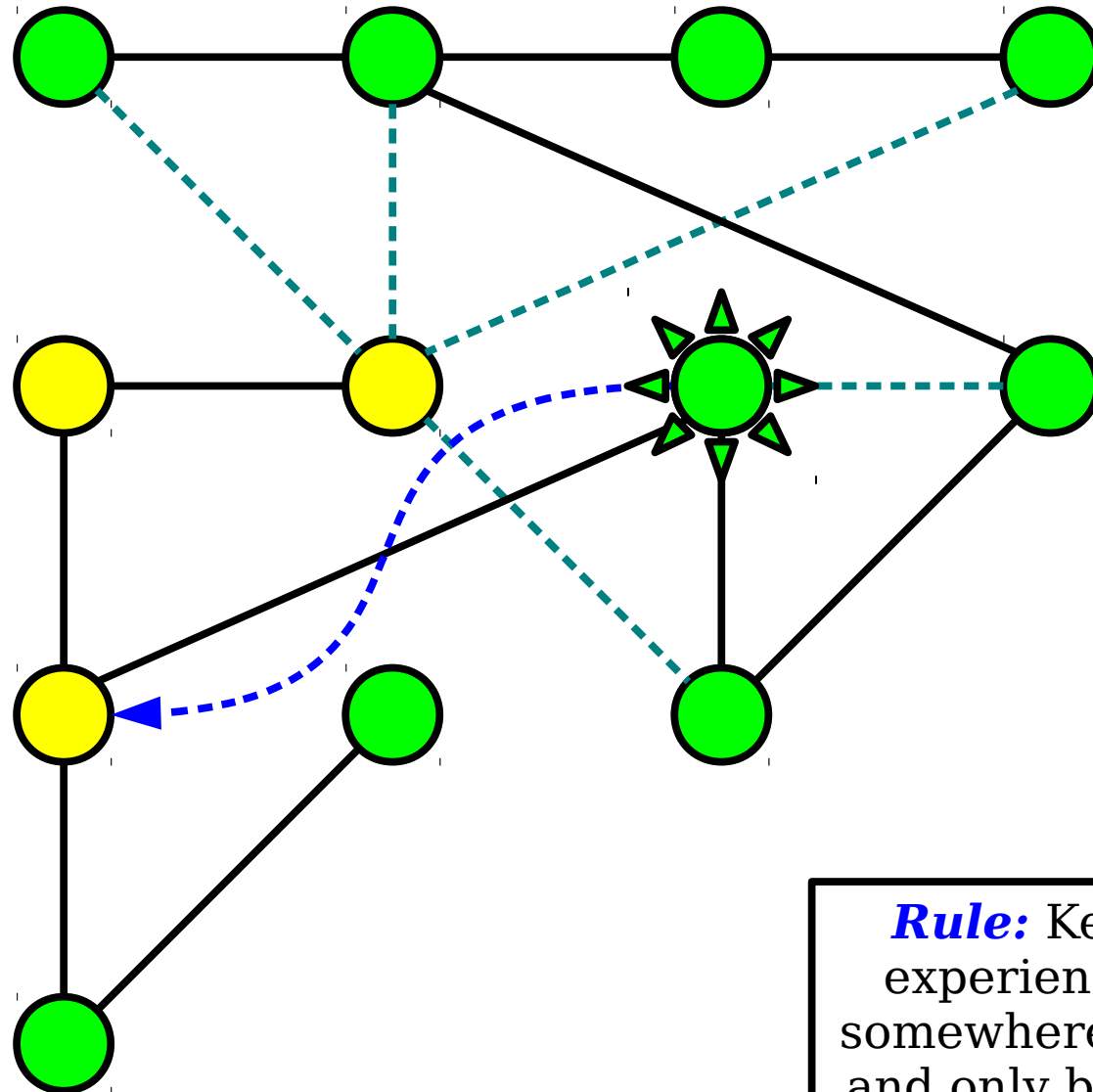
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



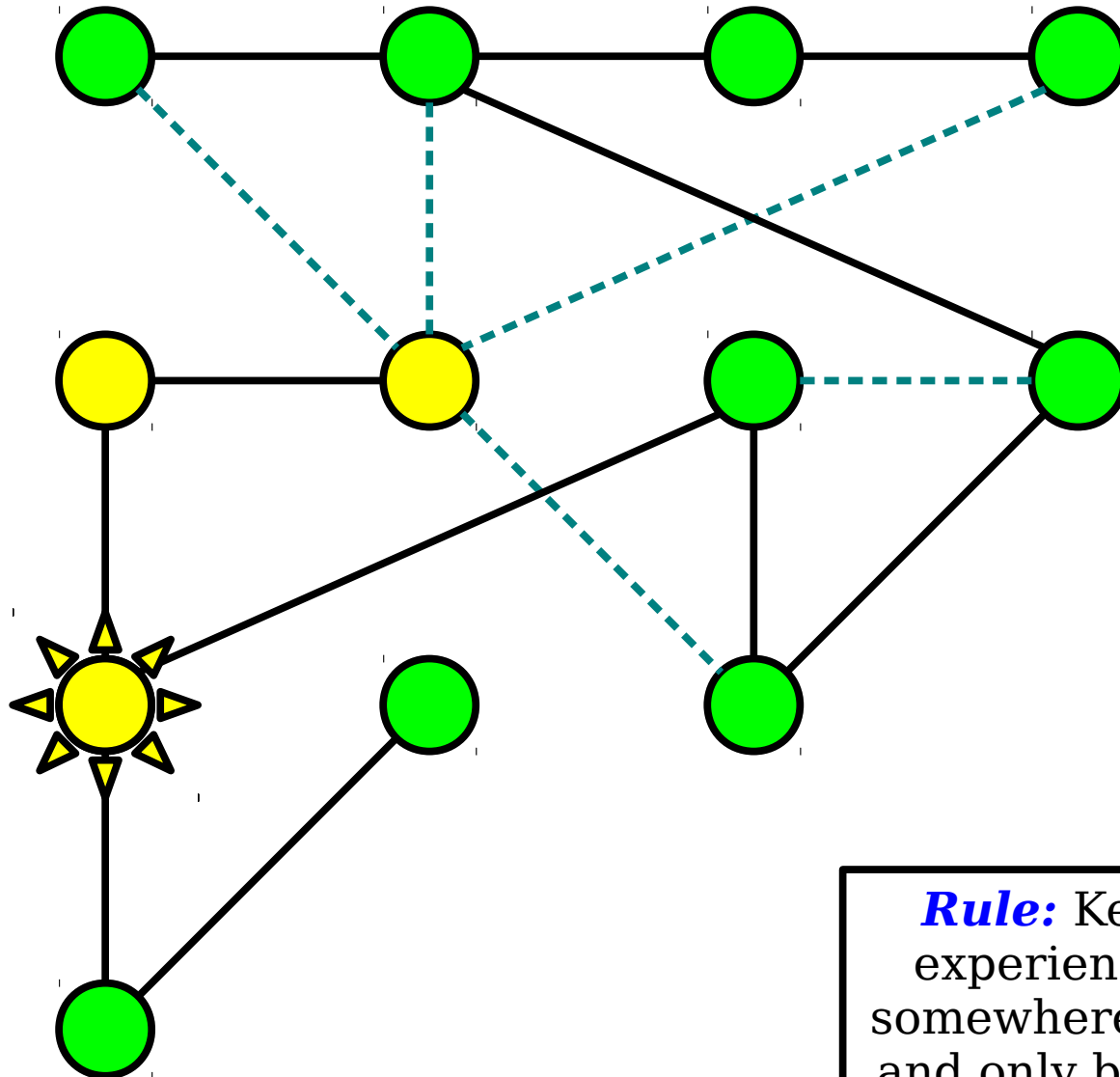
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



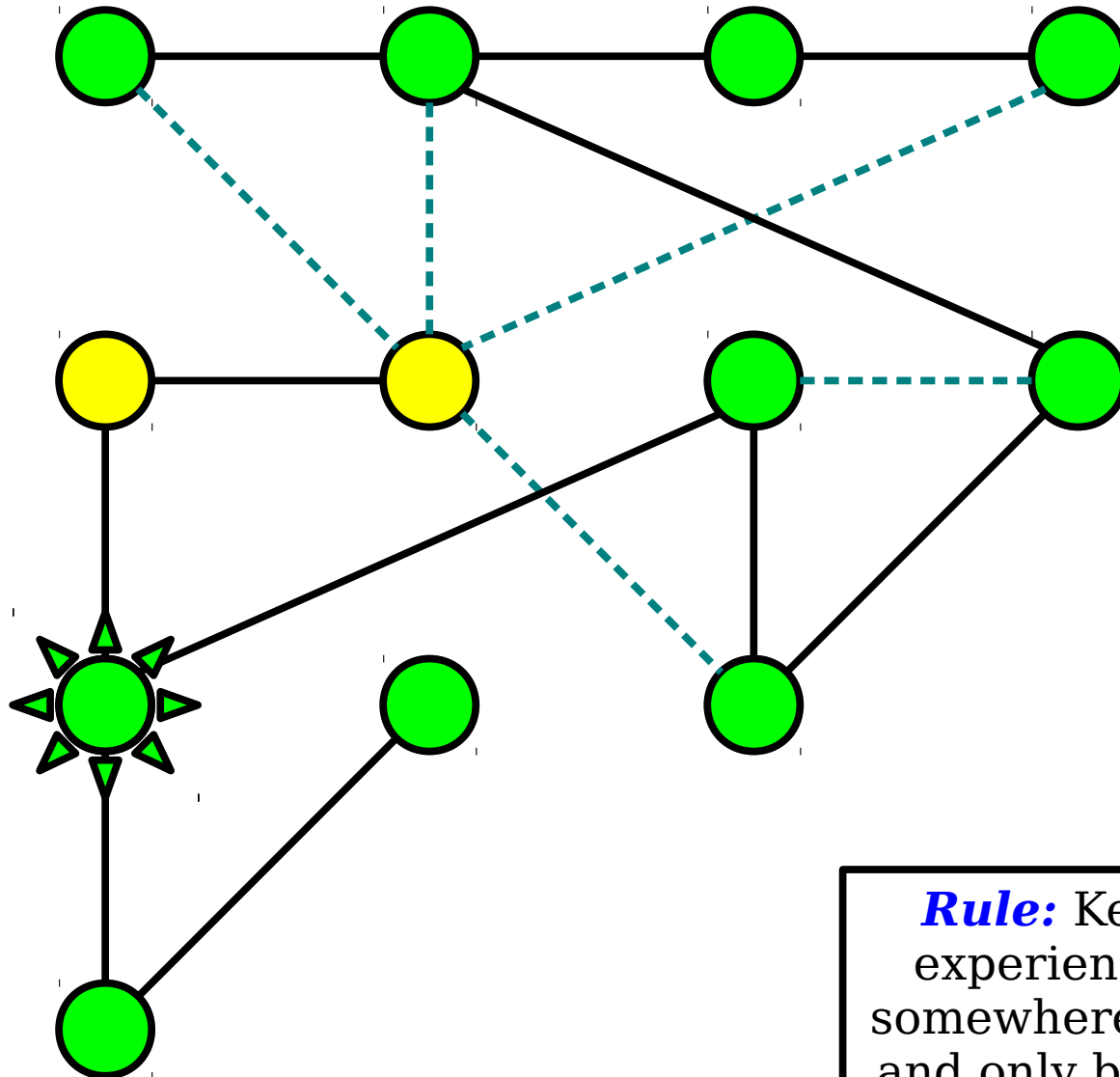
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



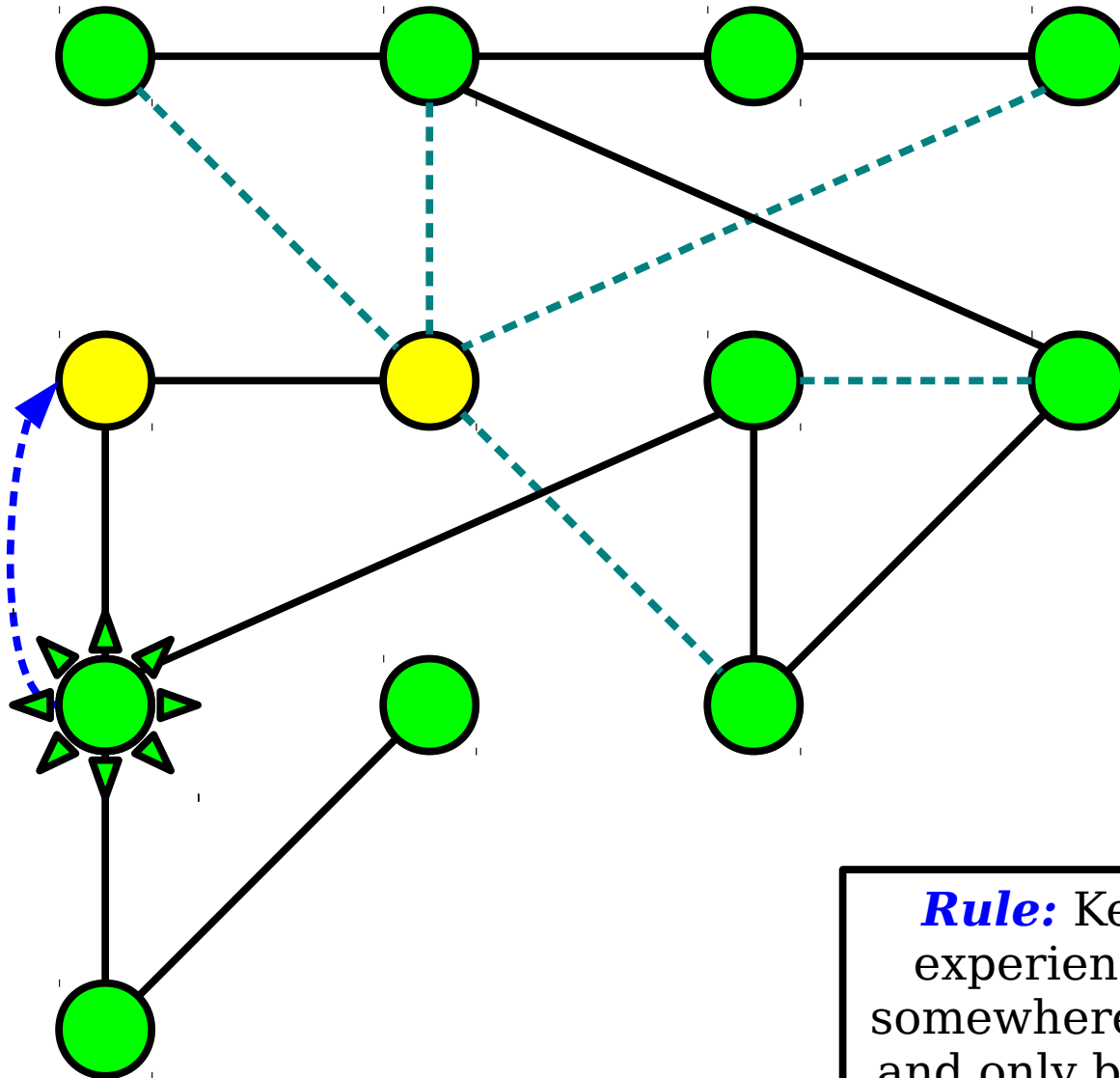
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



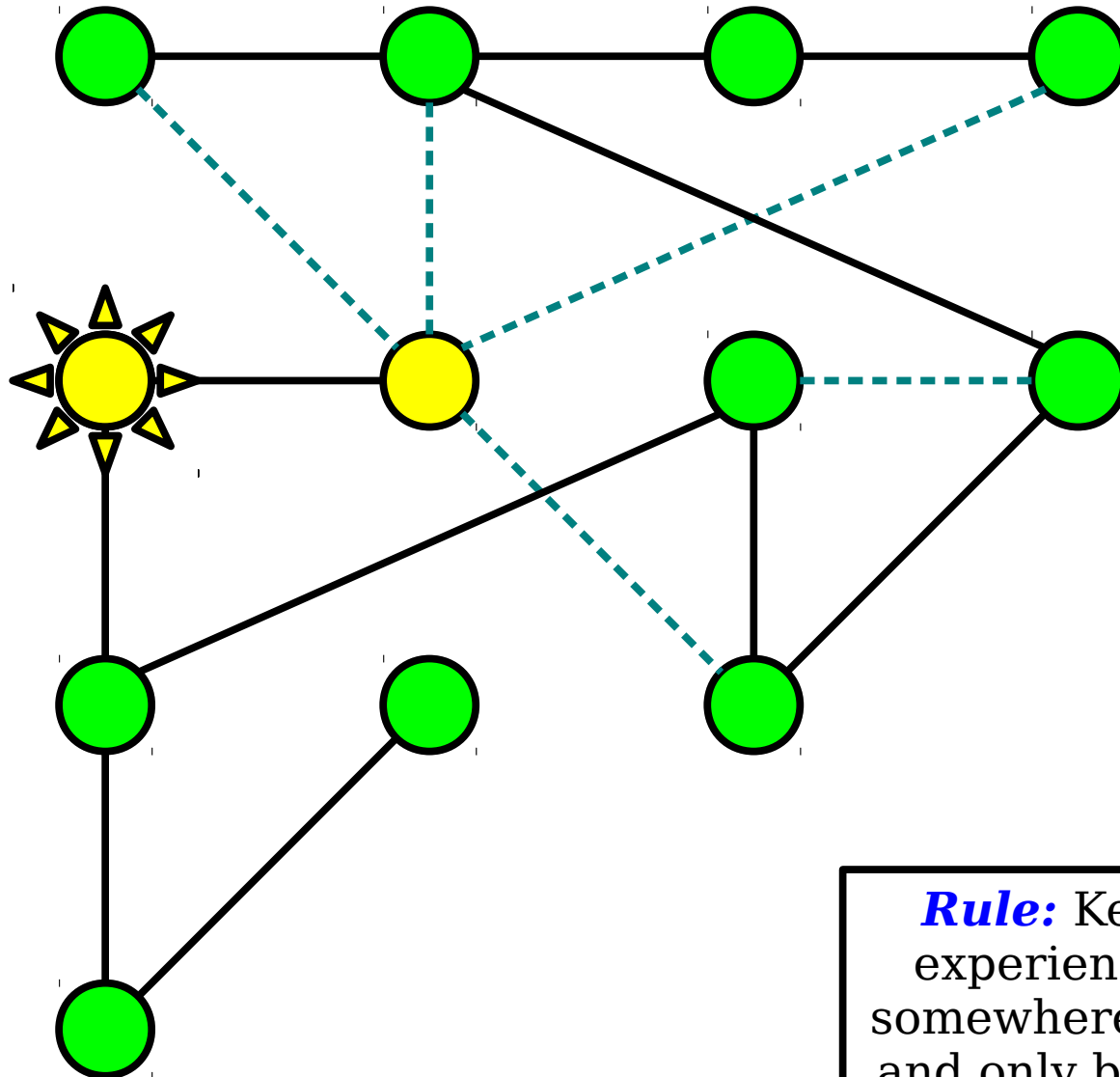
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



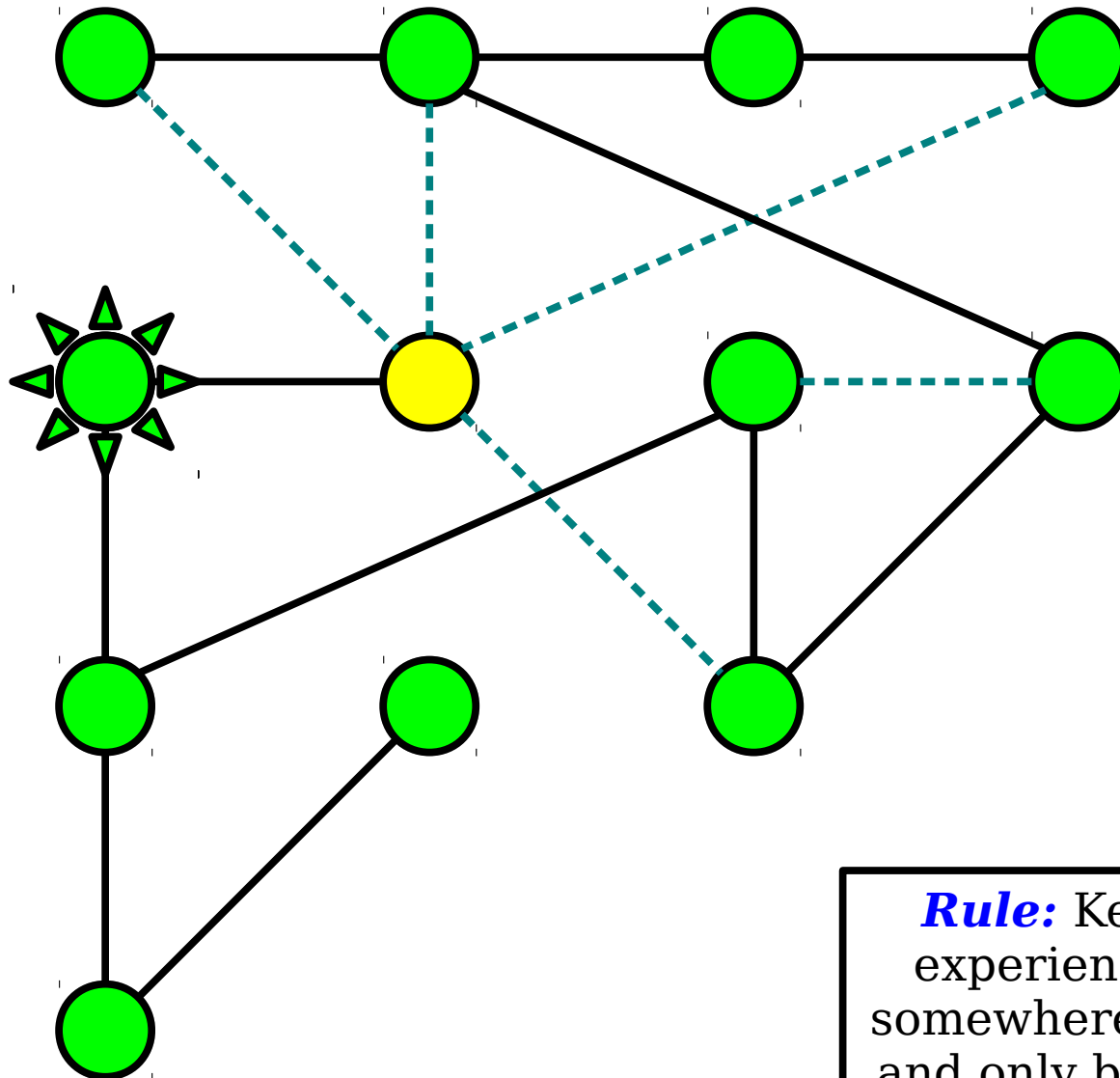
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



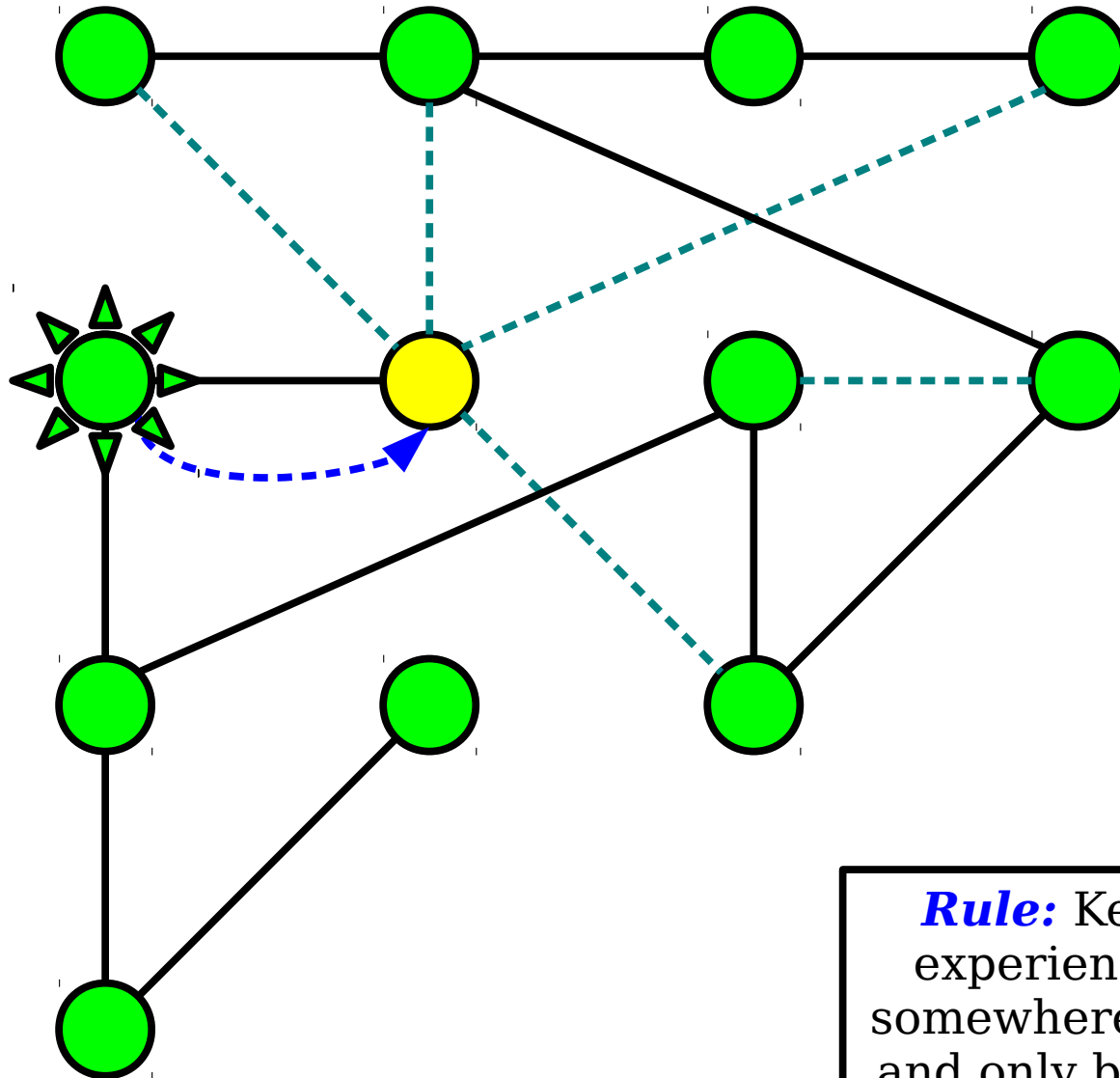
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



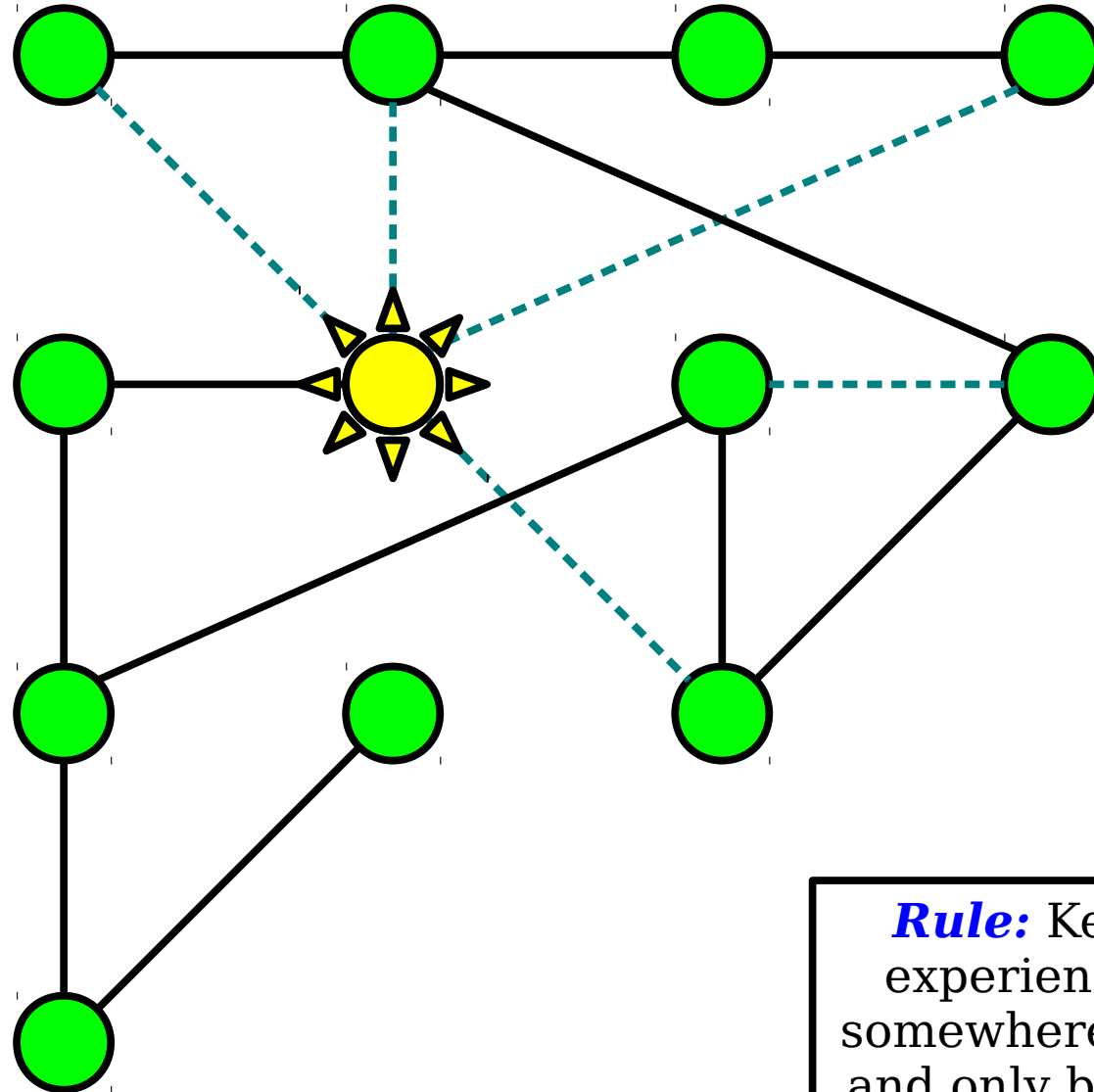
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



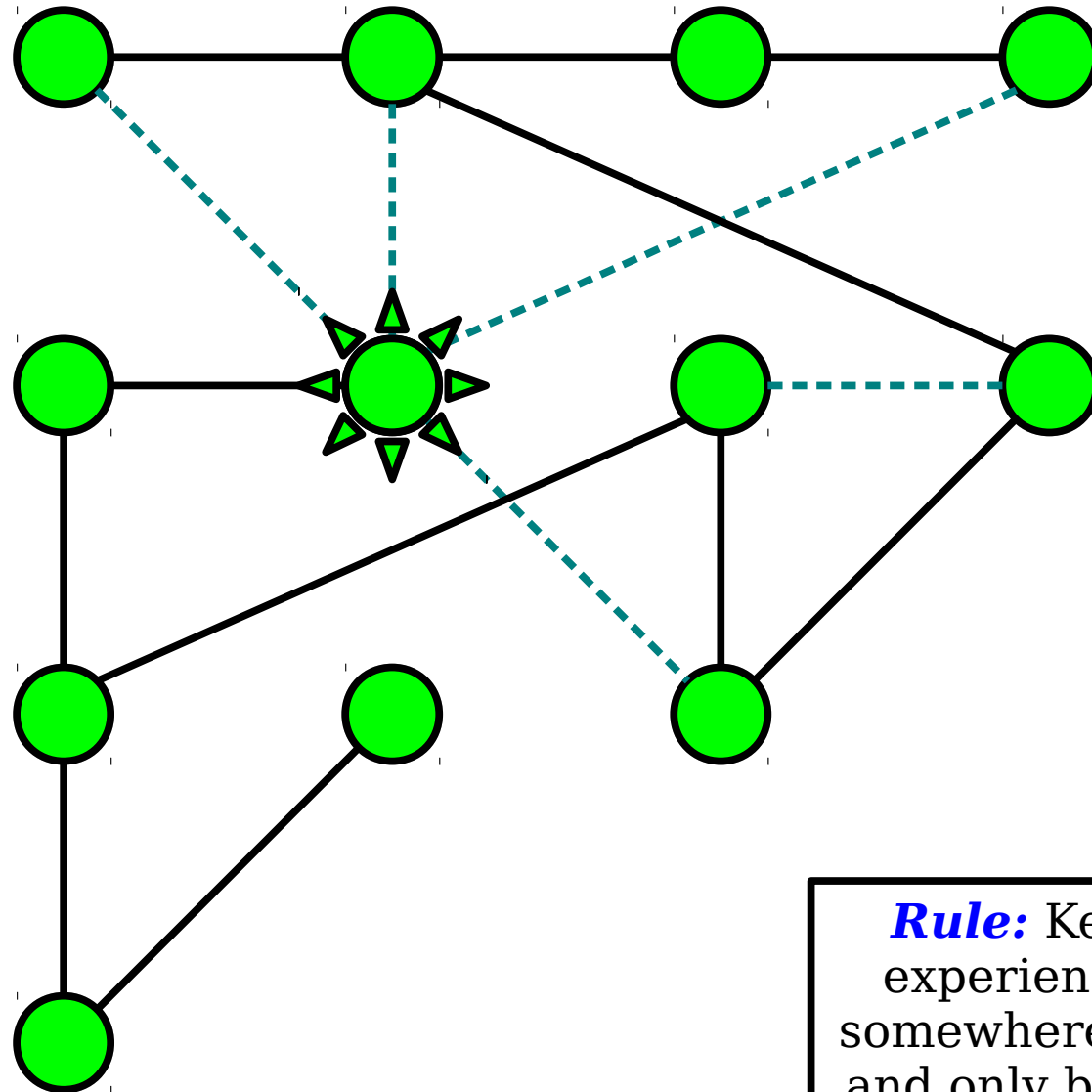
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



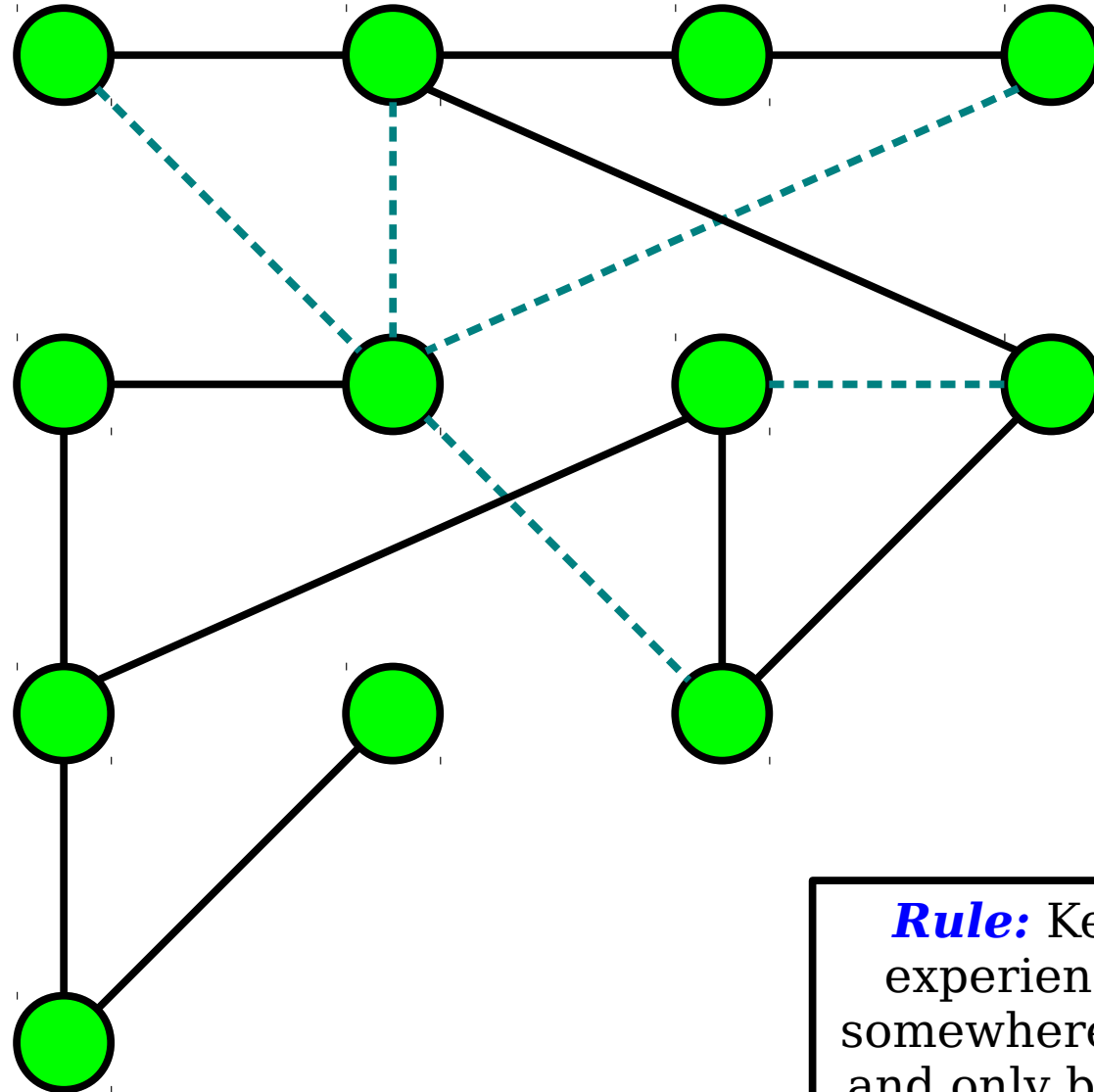
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search



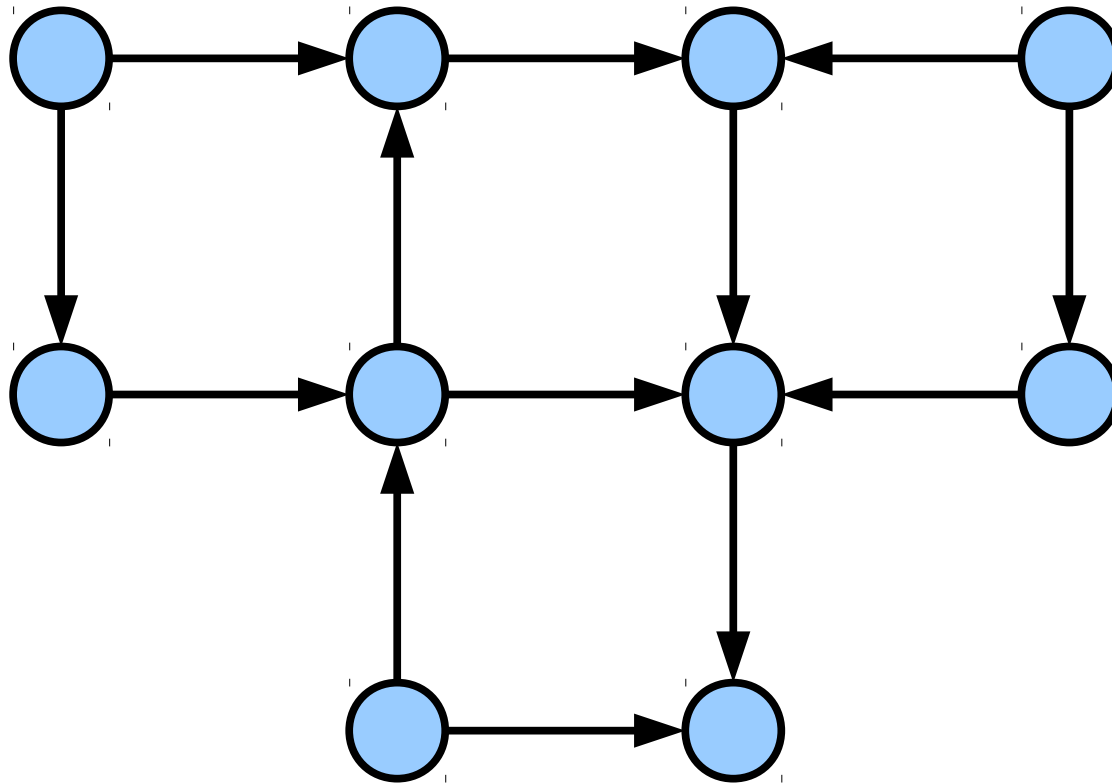
Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

Depth-First Search

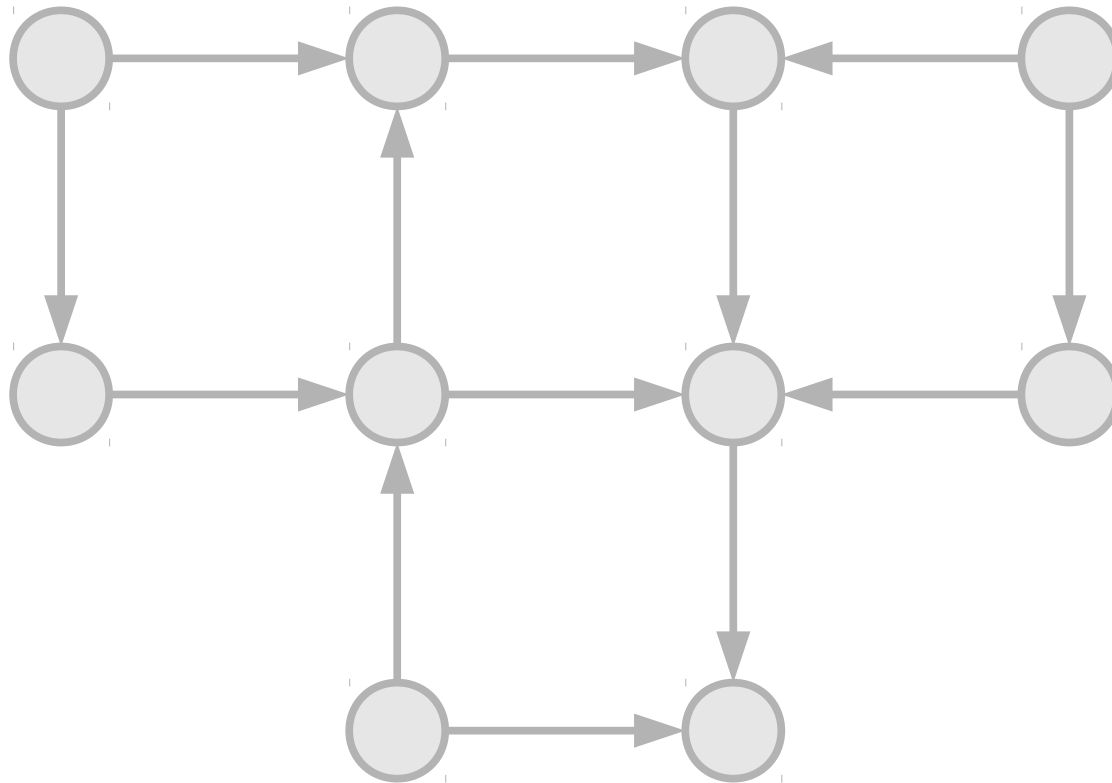


Rule: Keep trying new experiences! Always go somewhere new if you can, and only back up if there's nothing new to see.

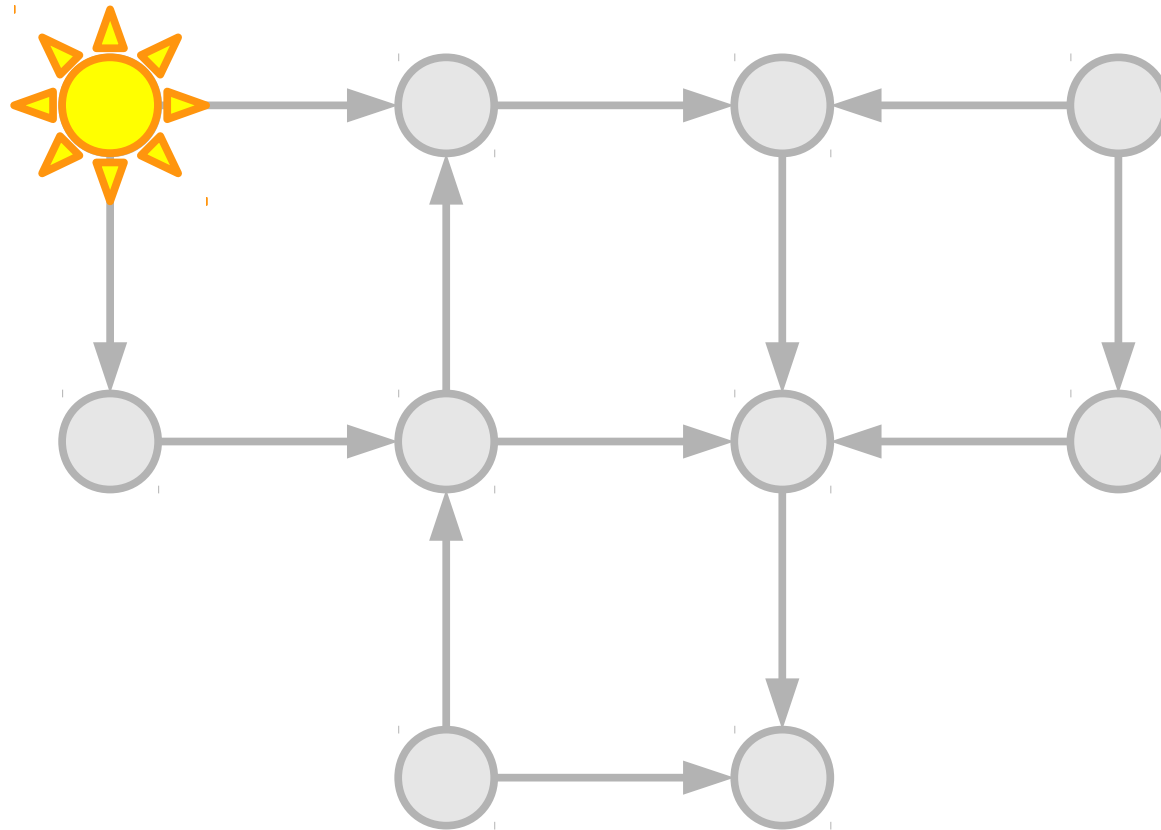
Depth-First Search, Again



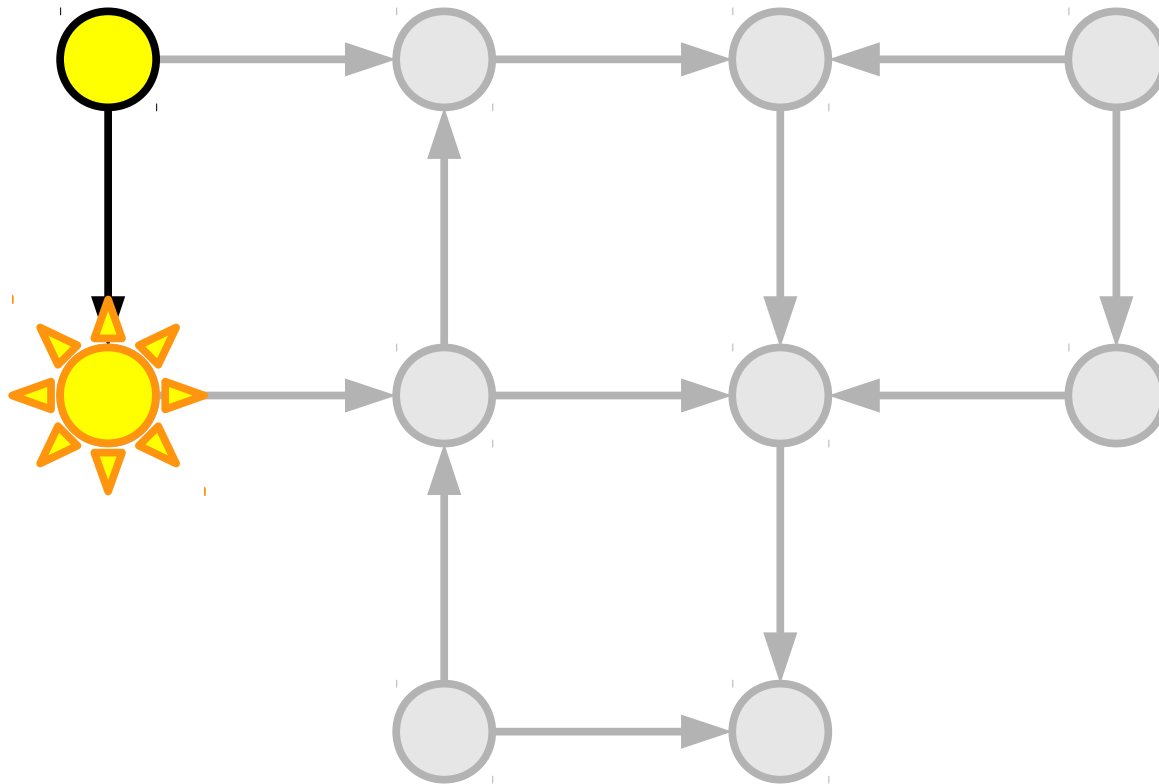
Depth-First Search, Again



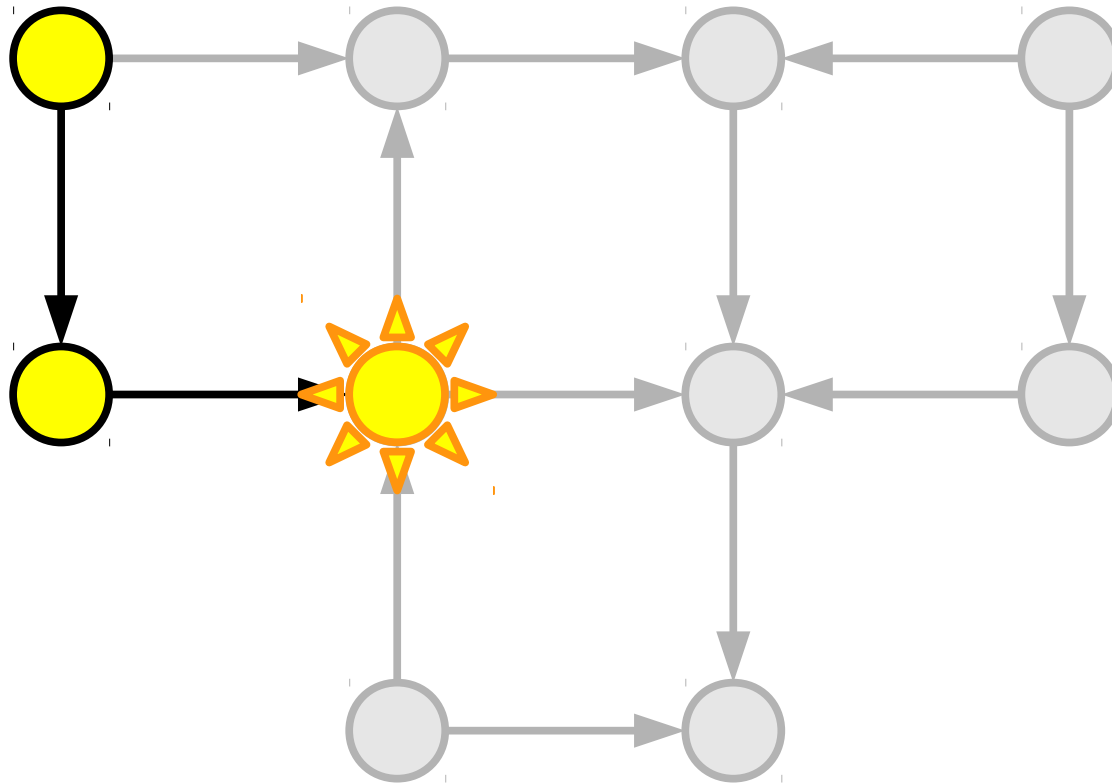
Depth-First Search, Again



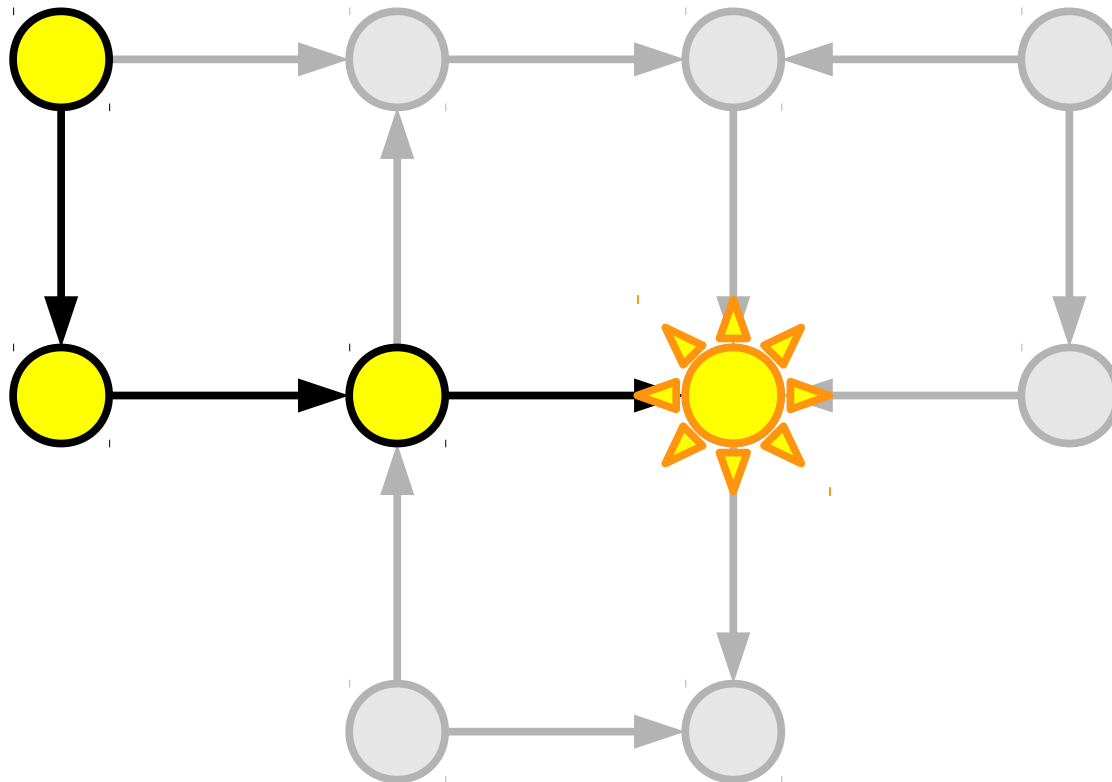
Depth-First Search, Again



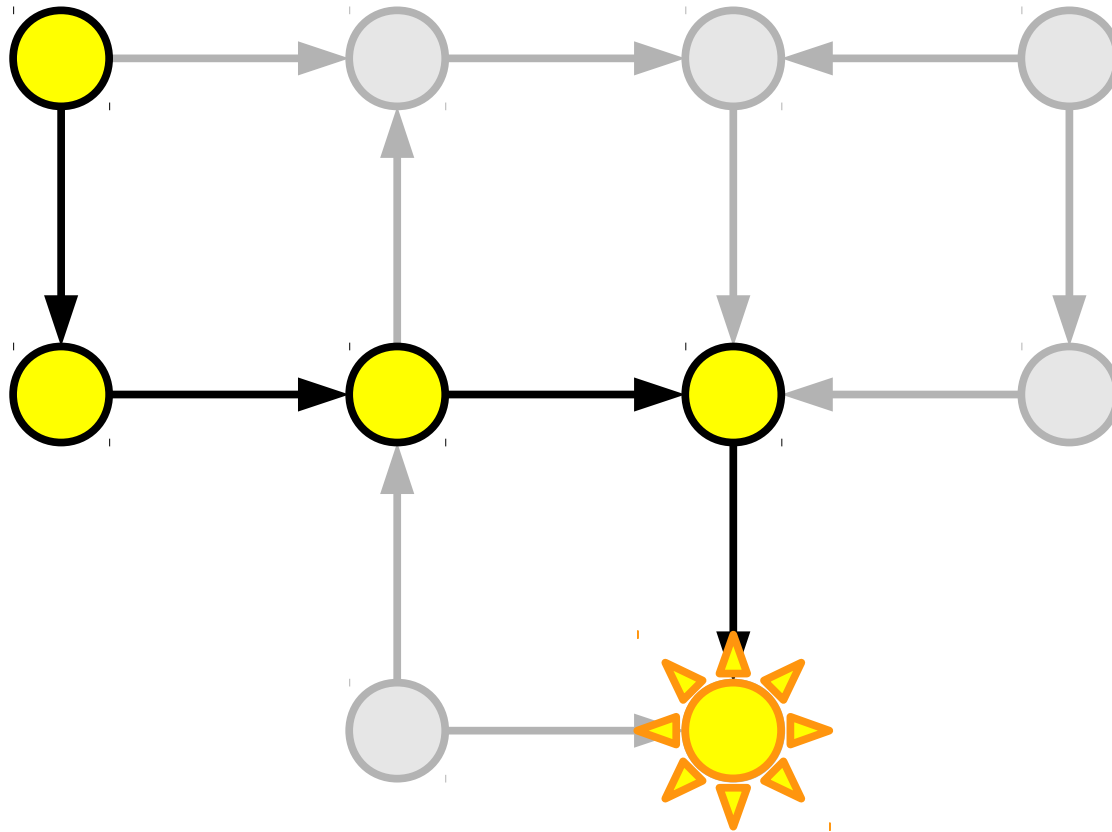
Depth-First Search, Again



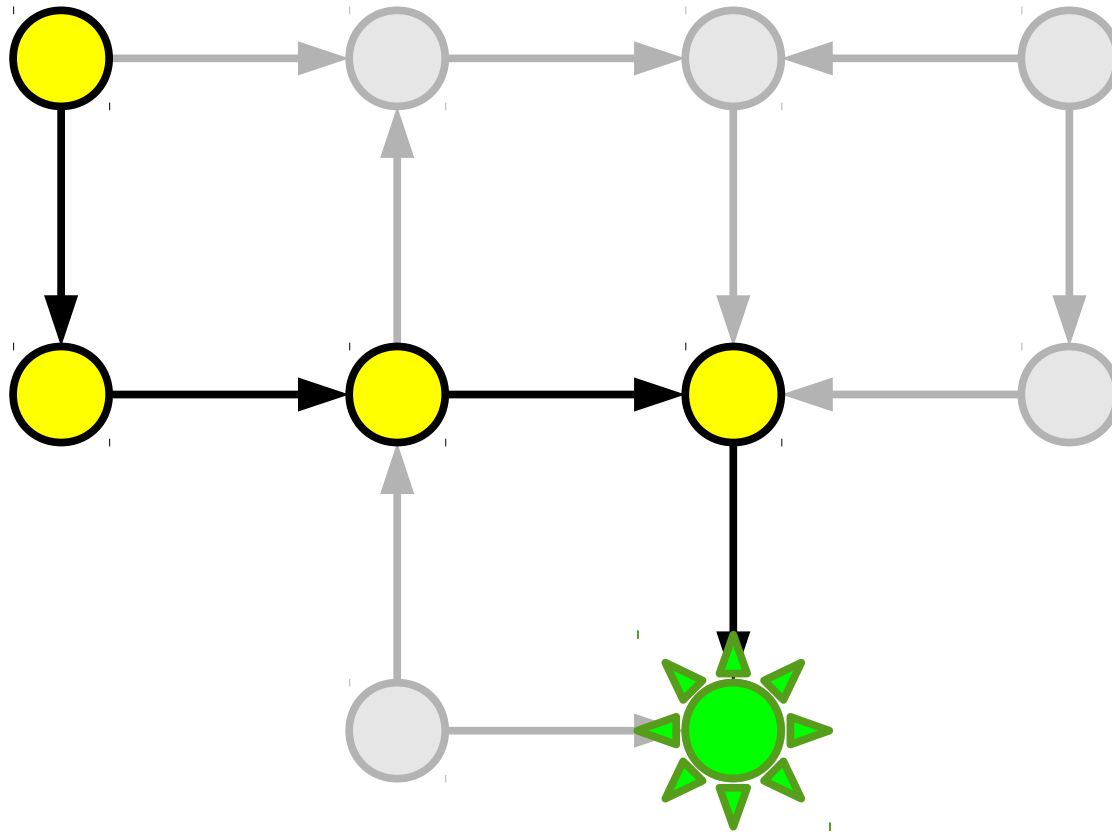
Depth-First Search, Again



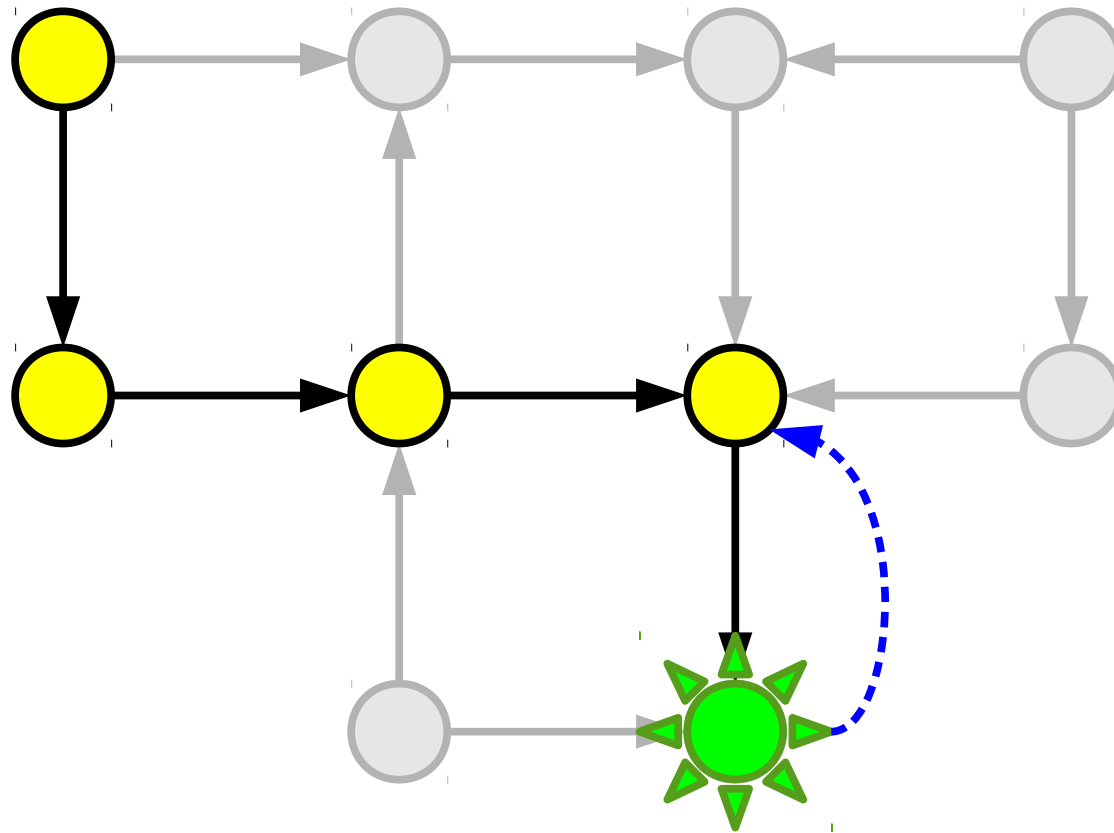
Depth-First Search, Again



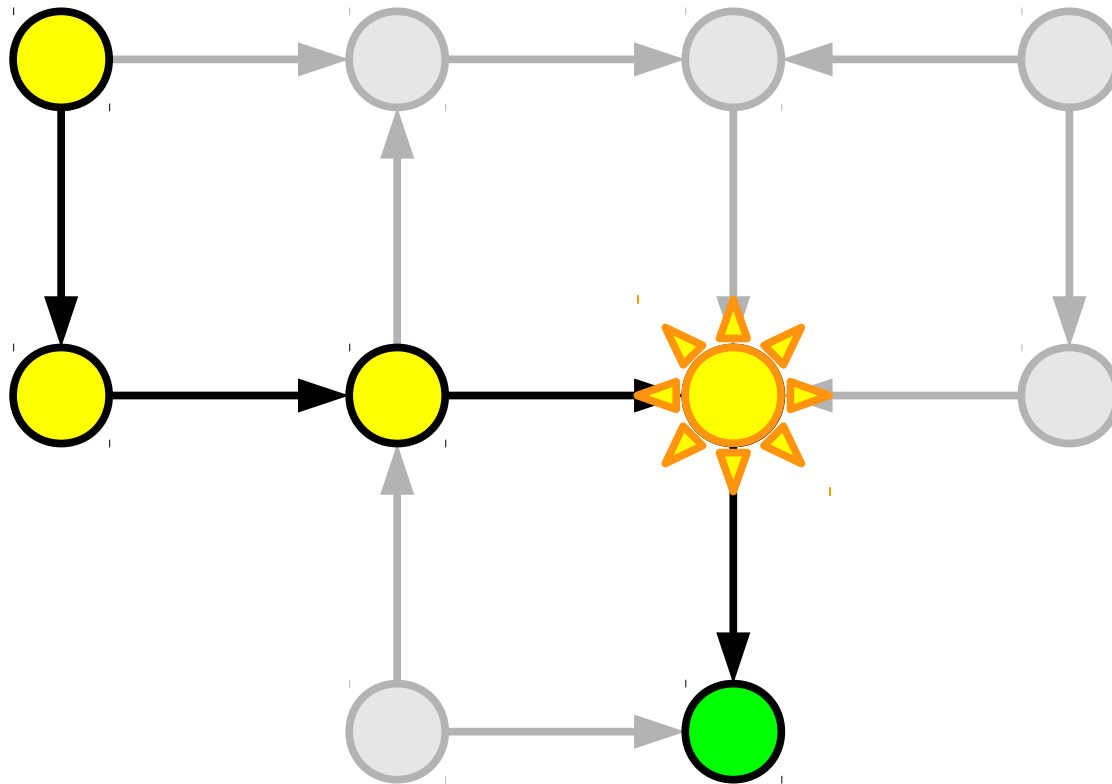
Depth-First Search, Again



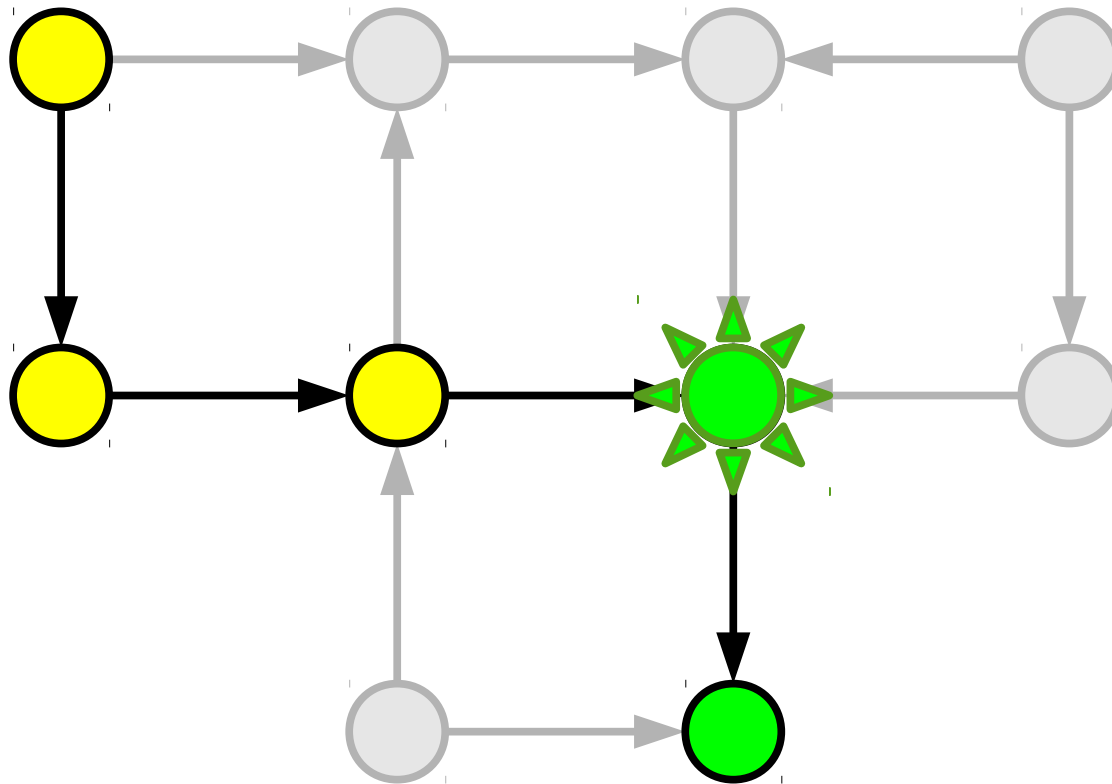
Depth-First Search, Again



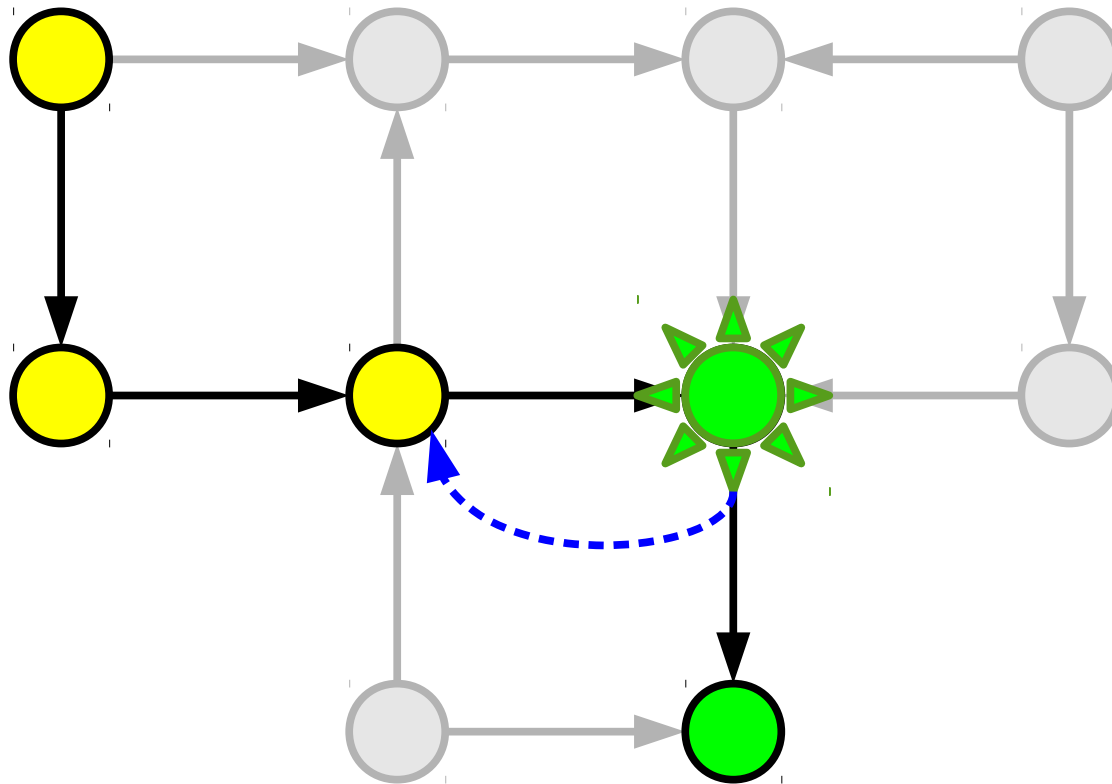
Depth-First Search, Again



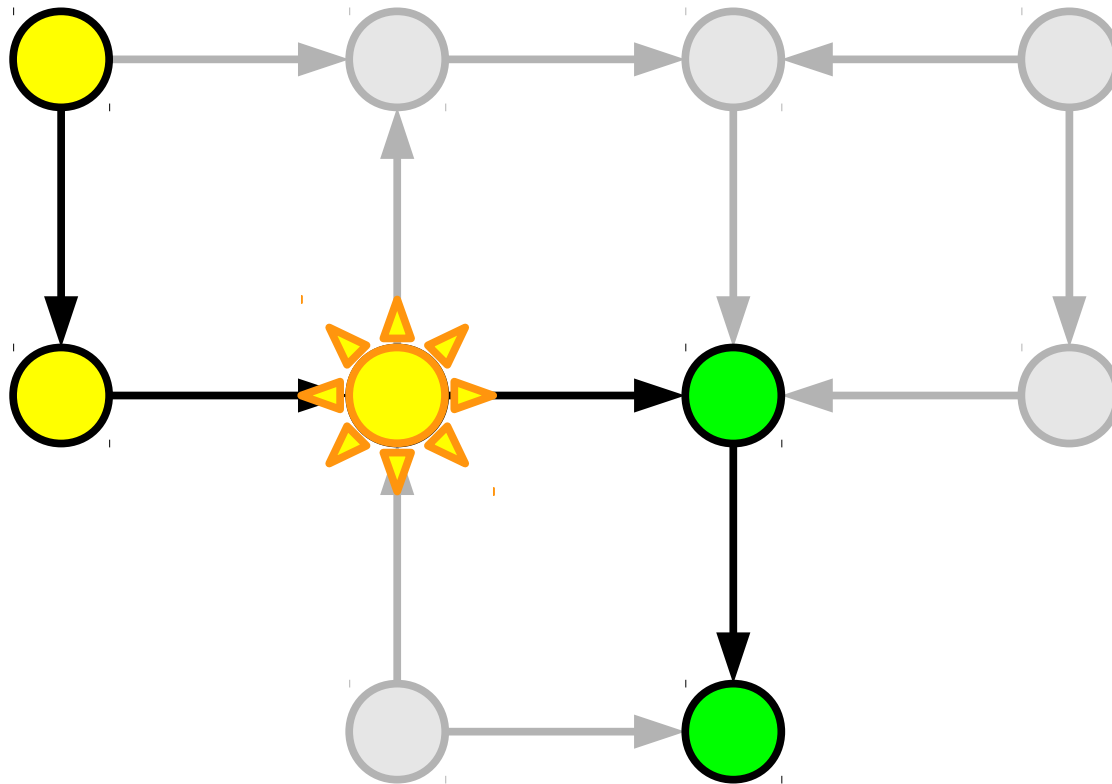
Depth-First Search, Again



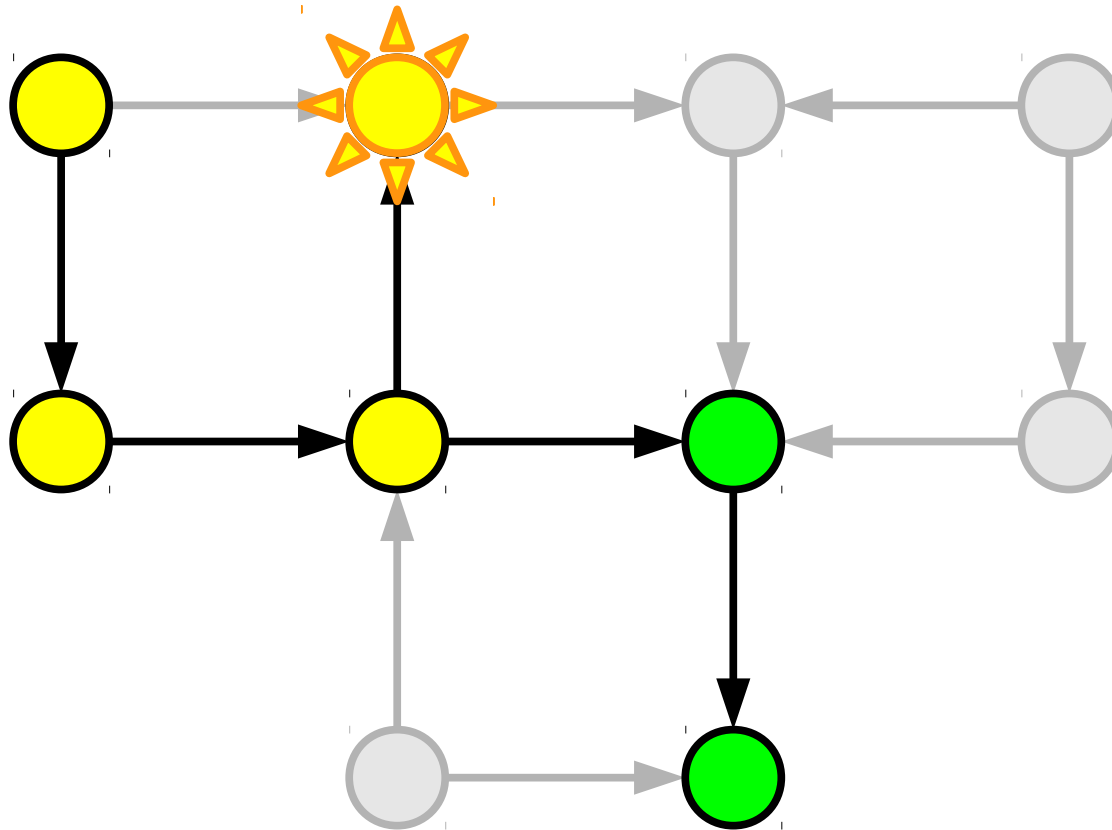
Depth-First Search, Again



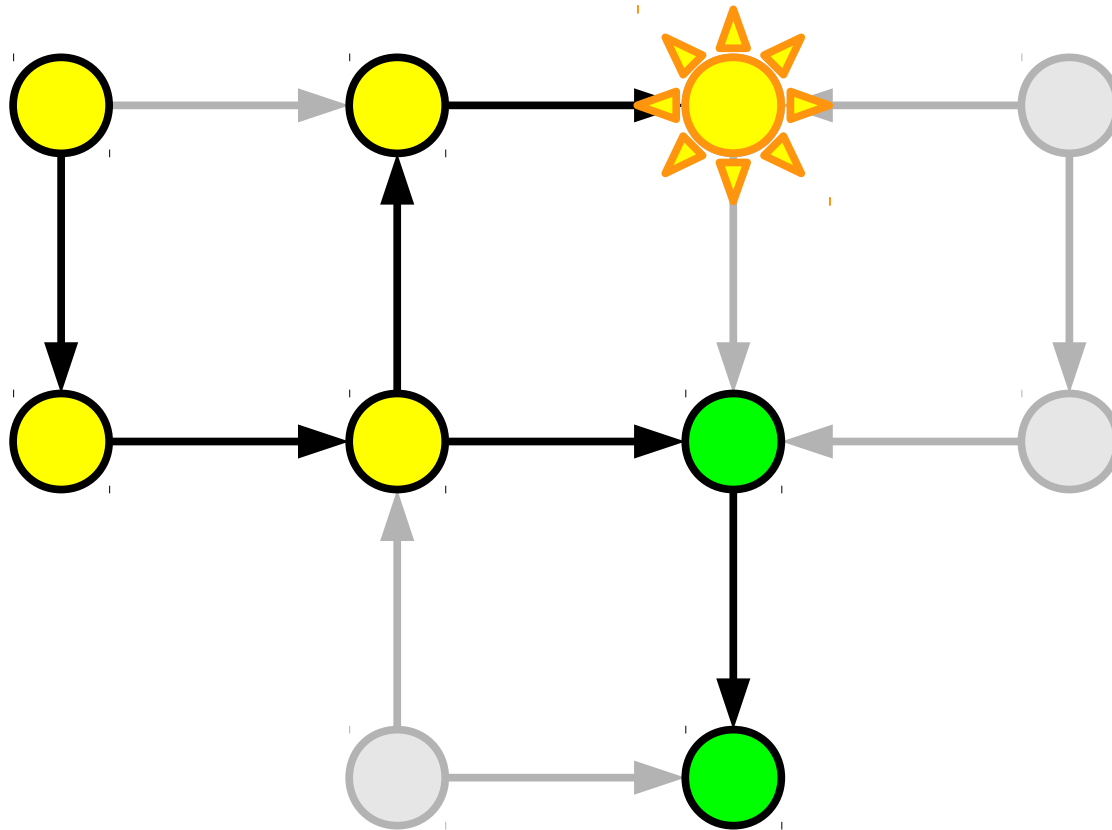
Depth-First Search, Again



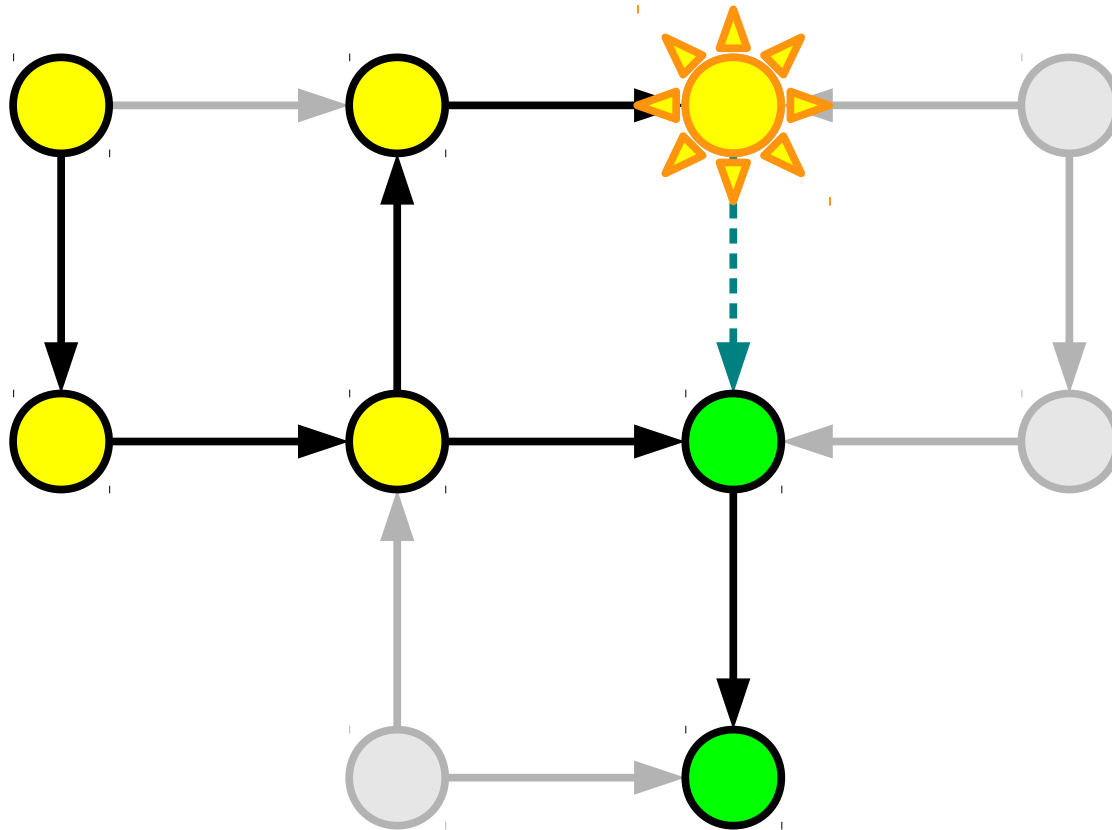
Depth-First Search, Again



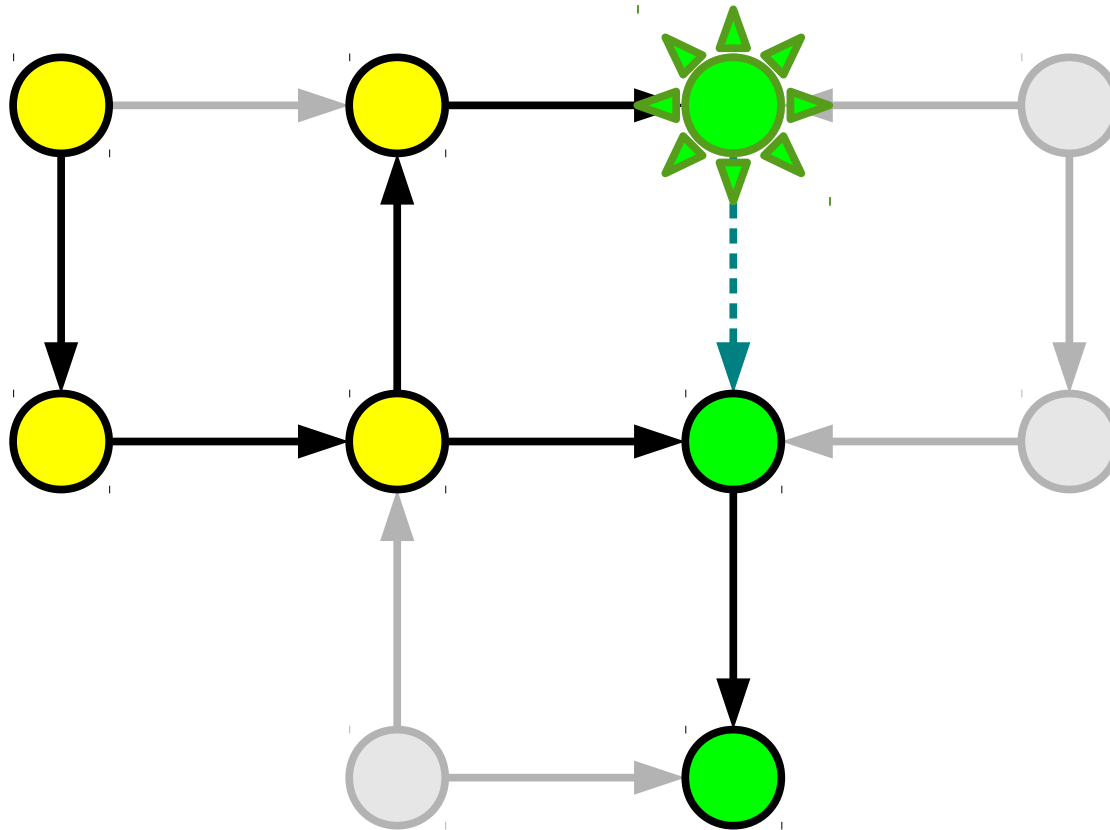
Depth-First Search, Again



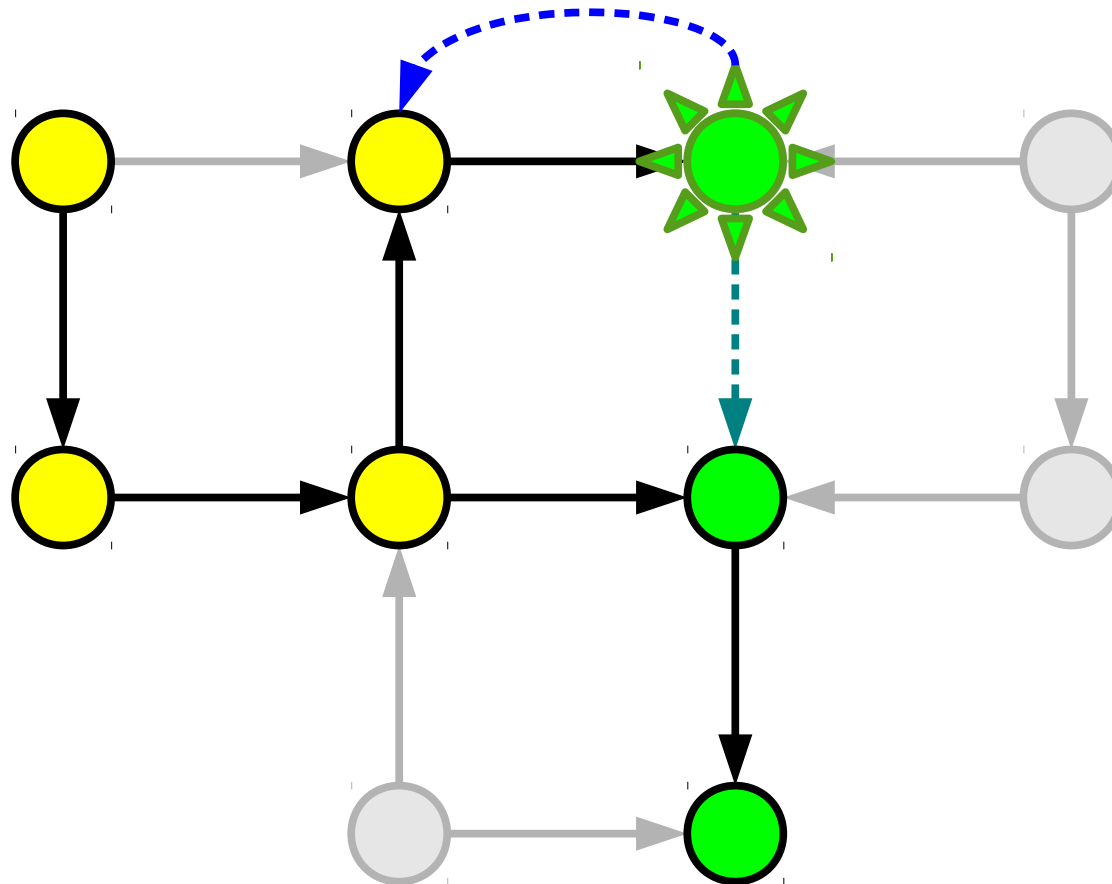
Depth-First Search, Again



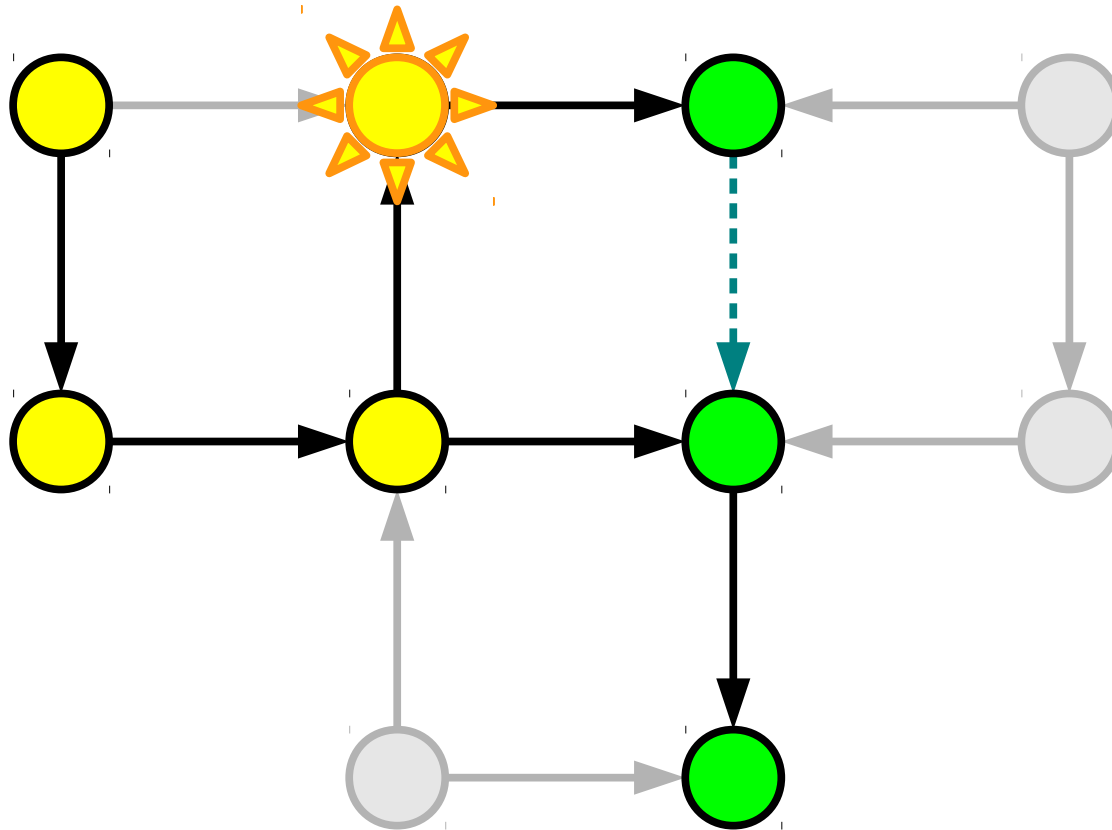
Depth-First Search, Again



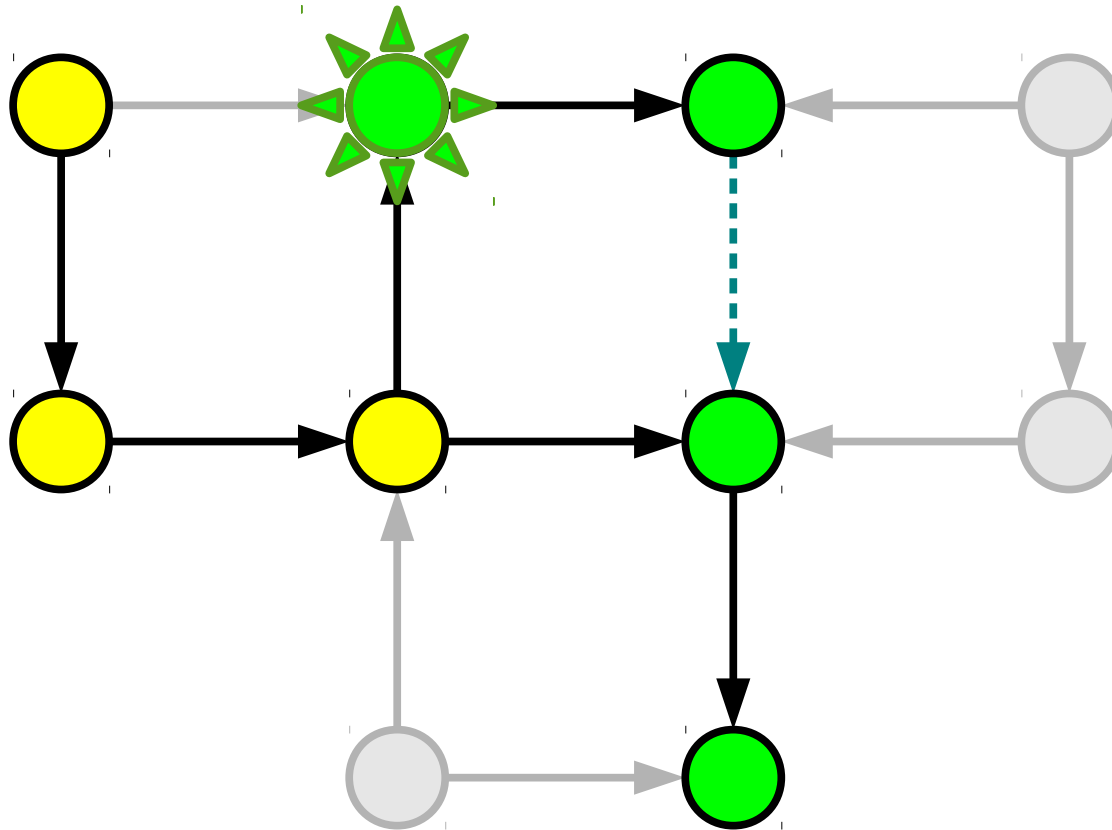
Depth-First Search, Again



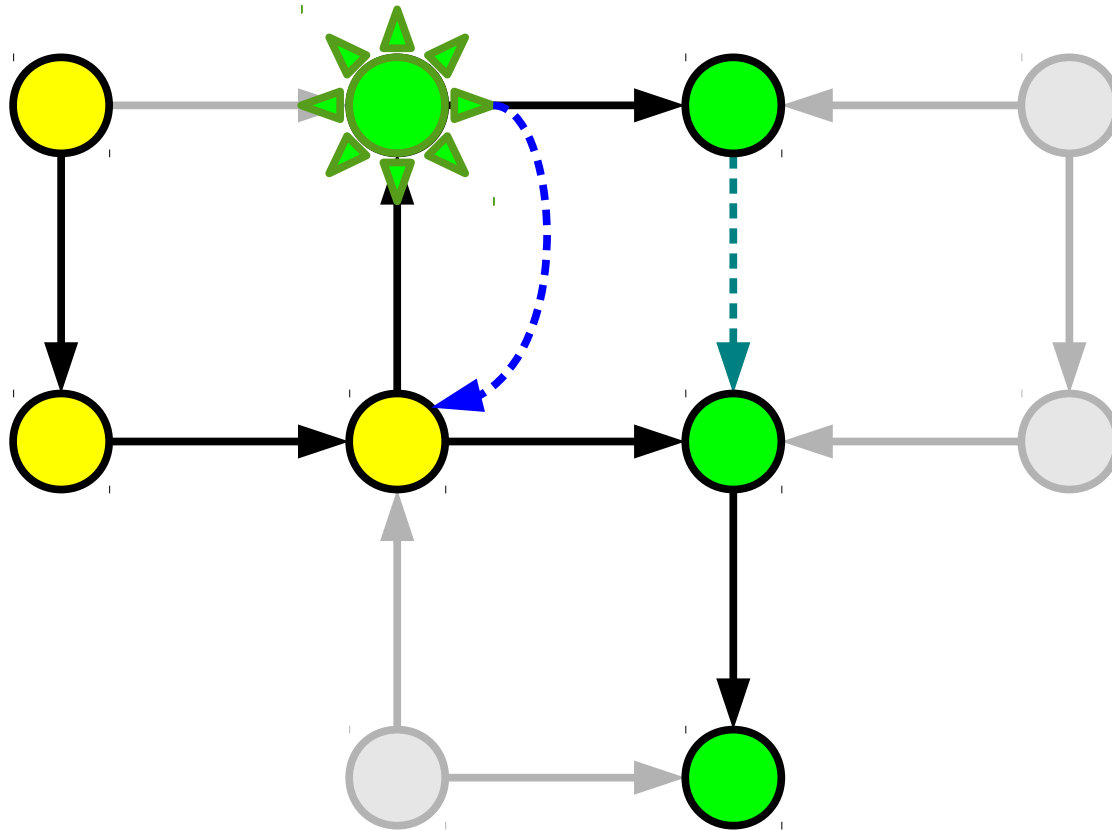
Depth-First Search, Again



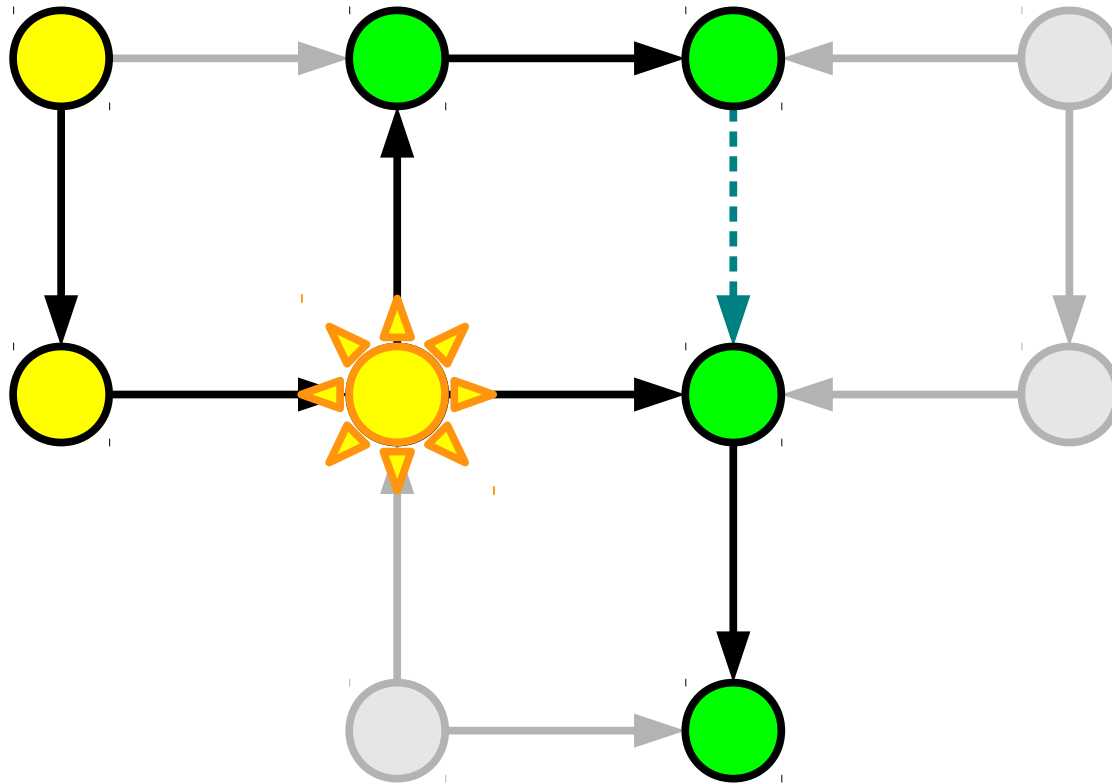
Depth-First Search, Again



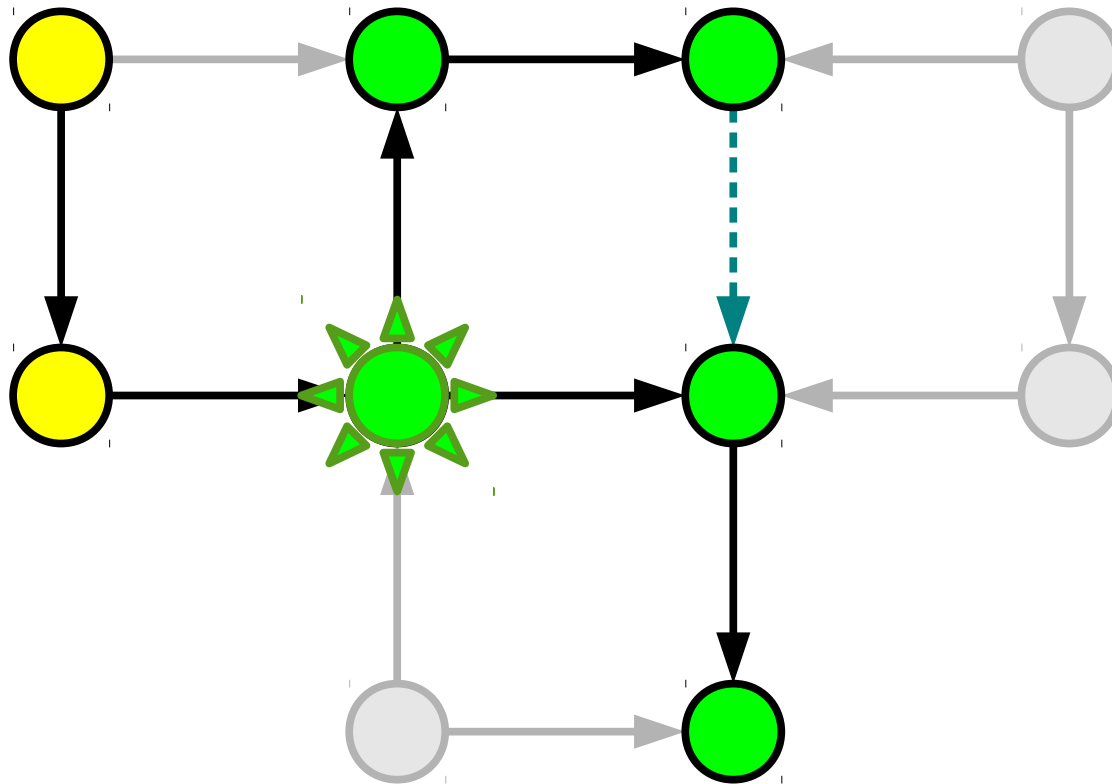
Depth-First Search, Again



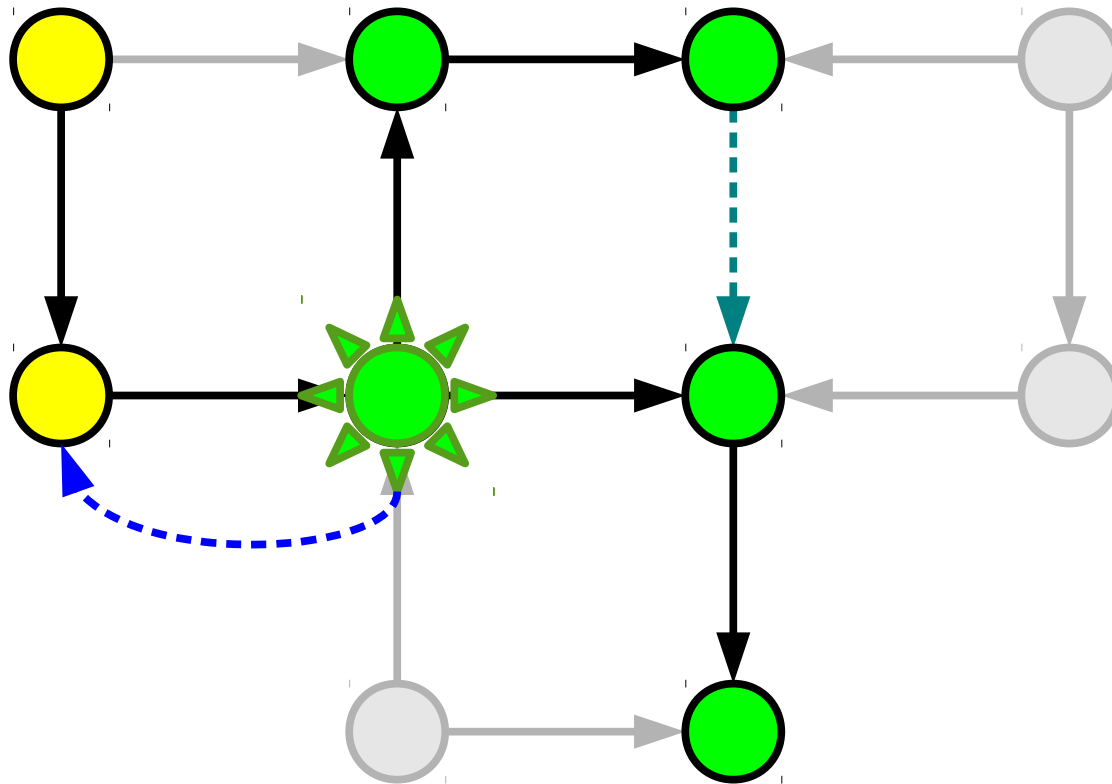
Depth-First Search, Again



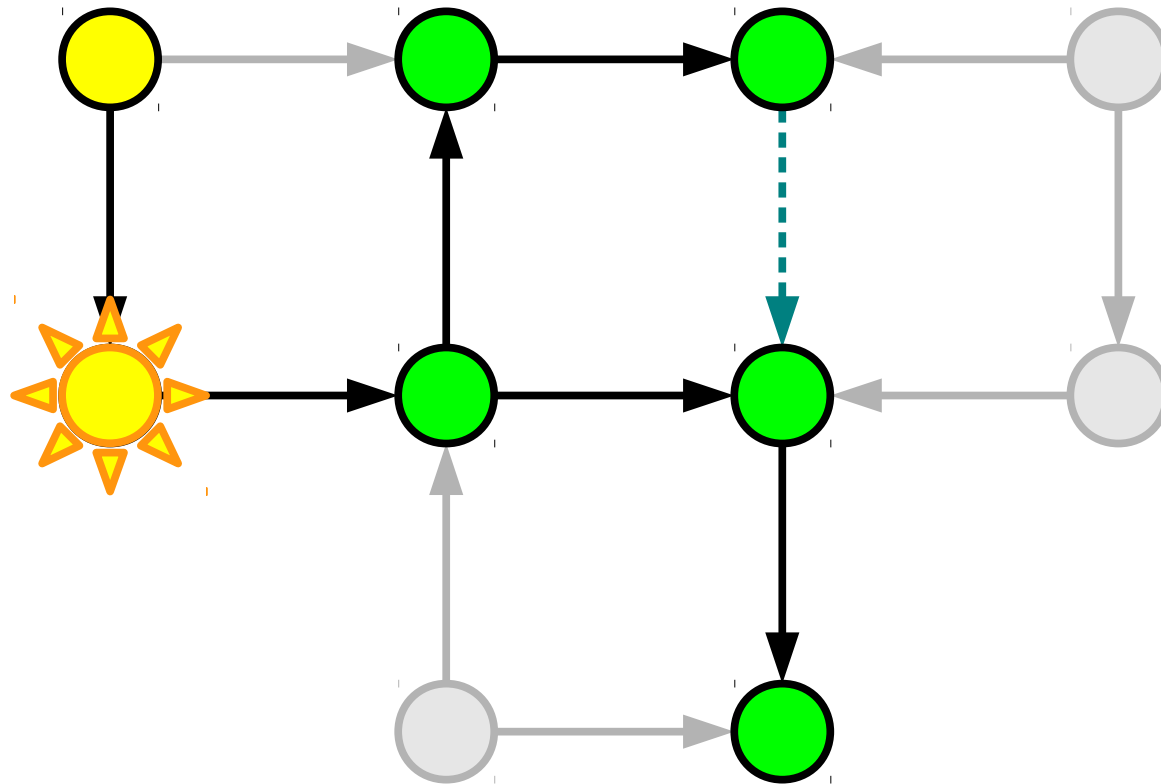
Depth-First Search, Again



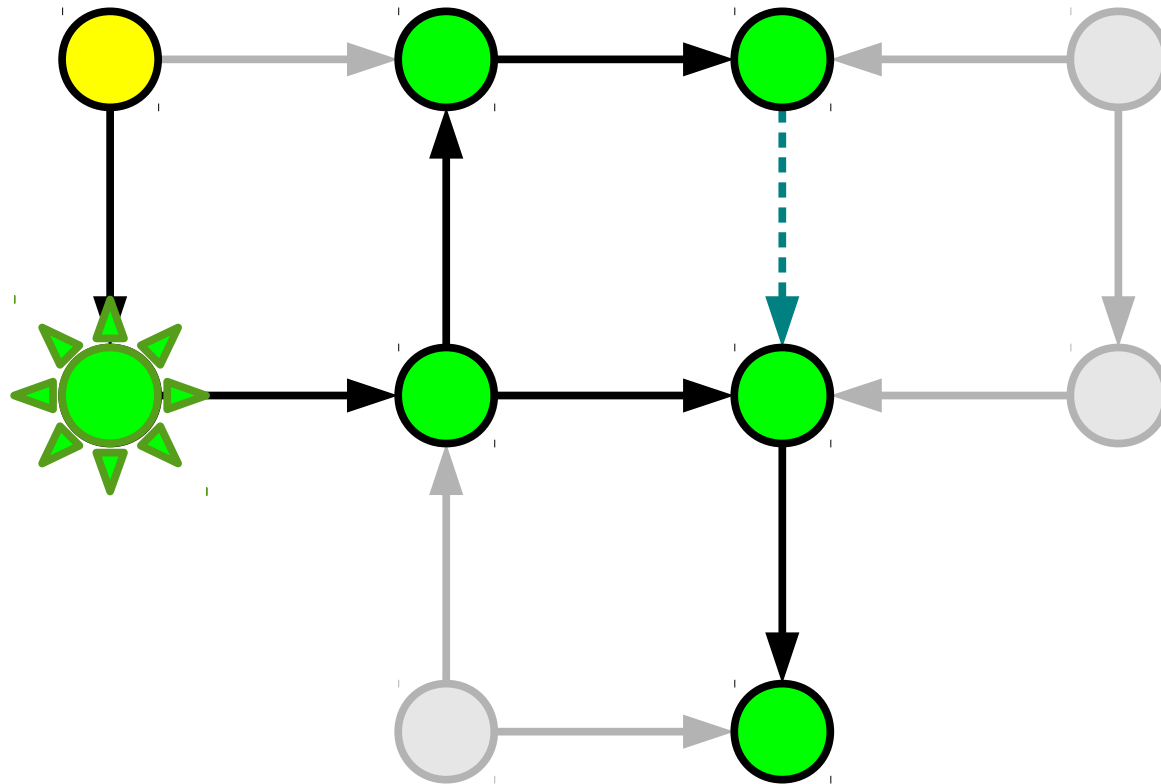
Depth-First Search, Again



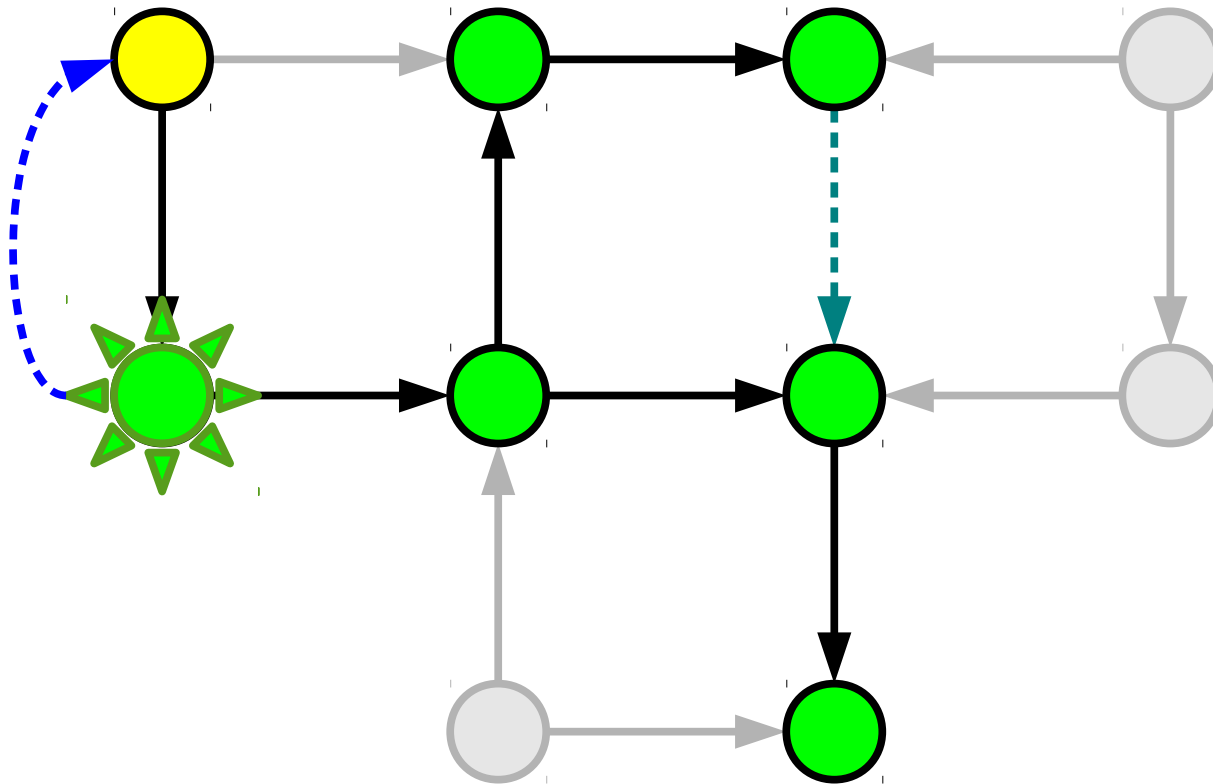
Depth-First Search, Again



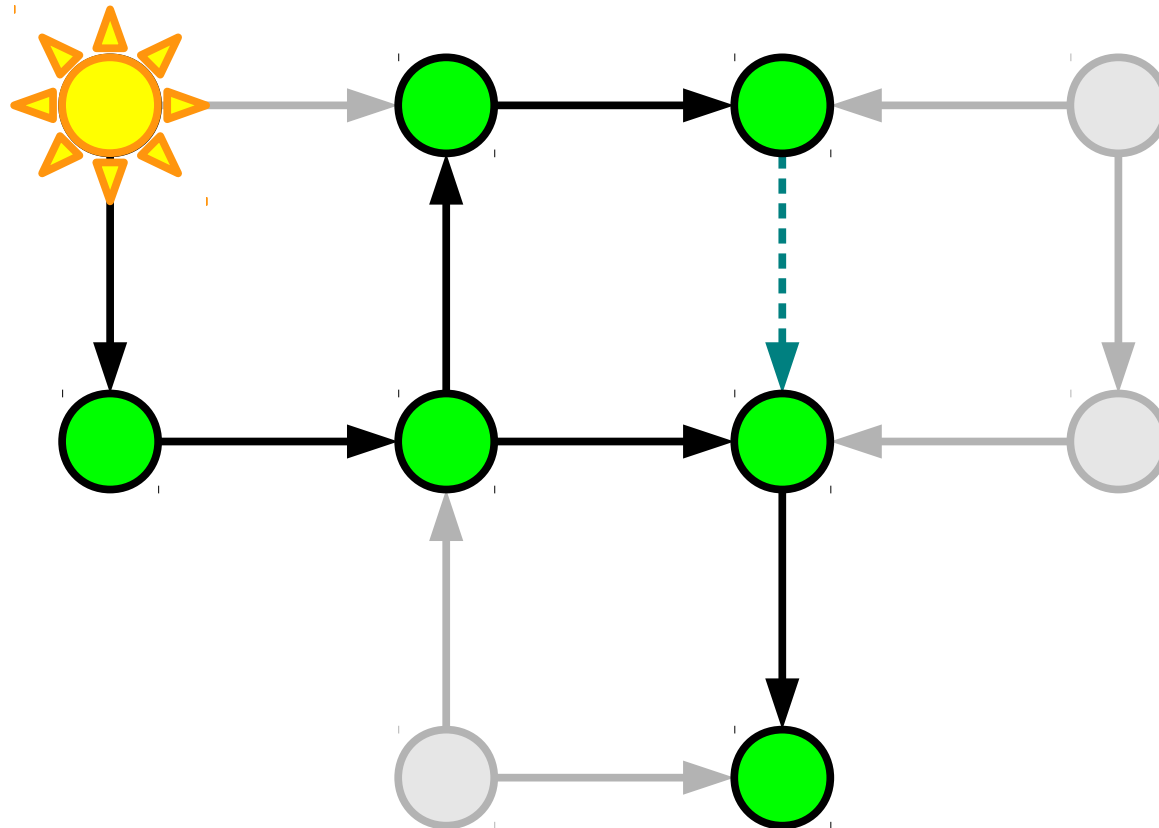
Depth-First Search, Again



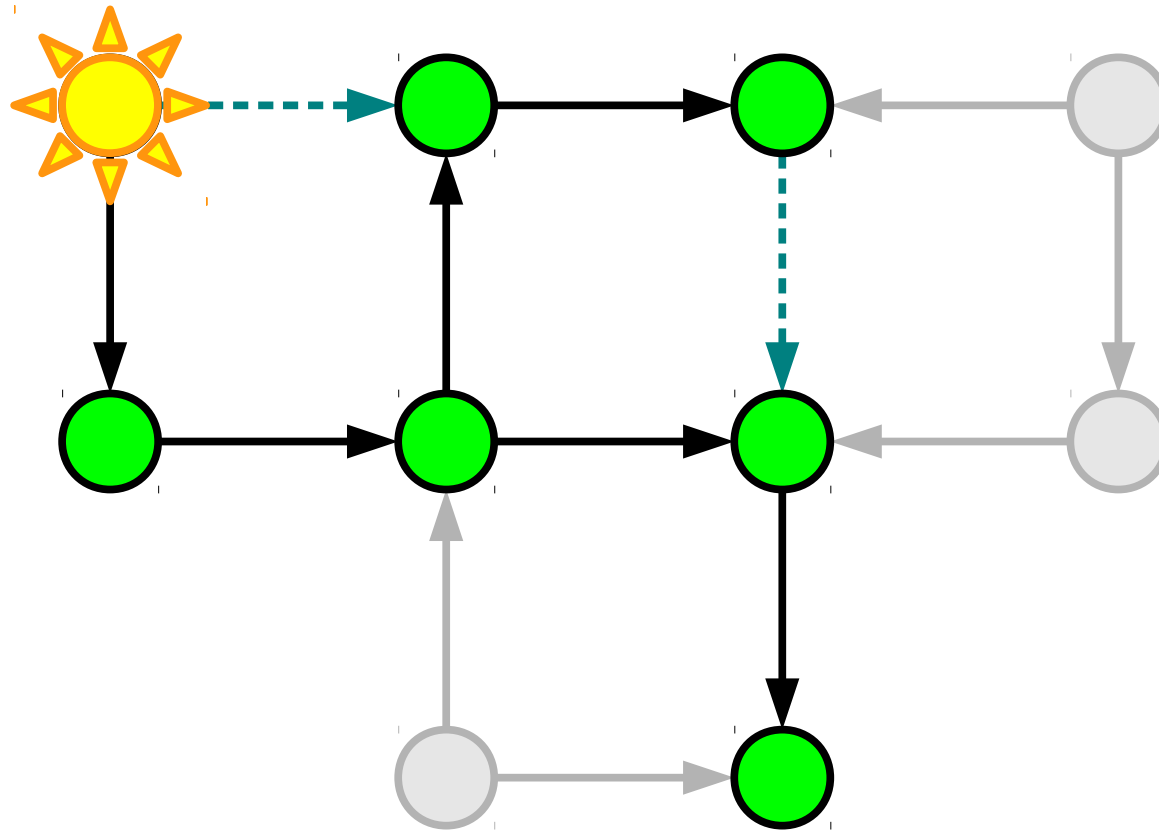
Depth-First Search, Again



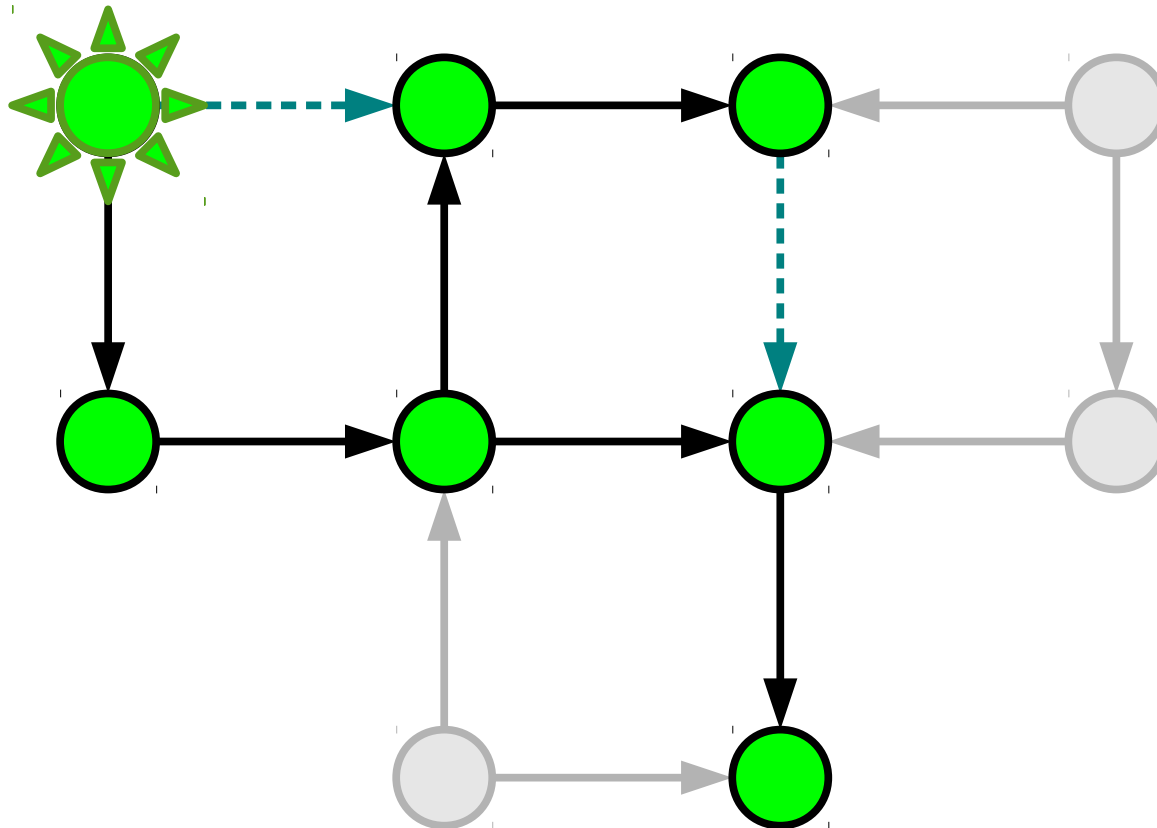
Depth-First Search, Again



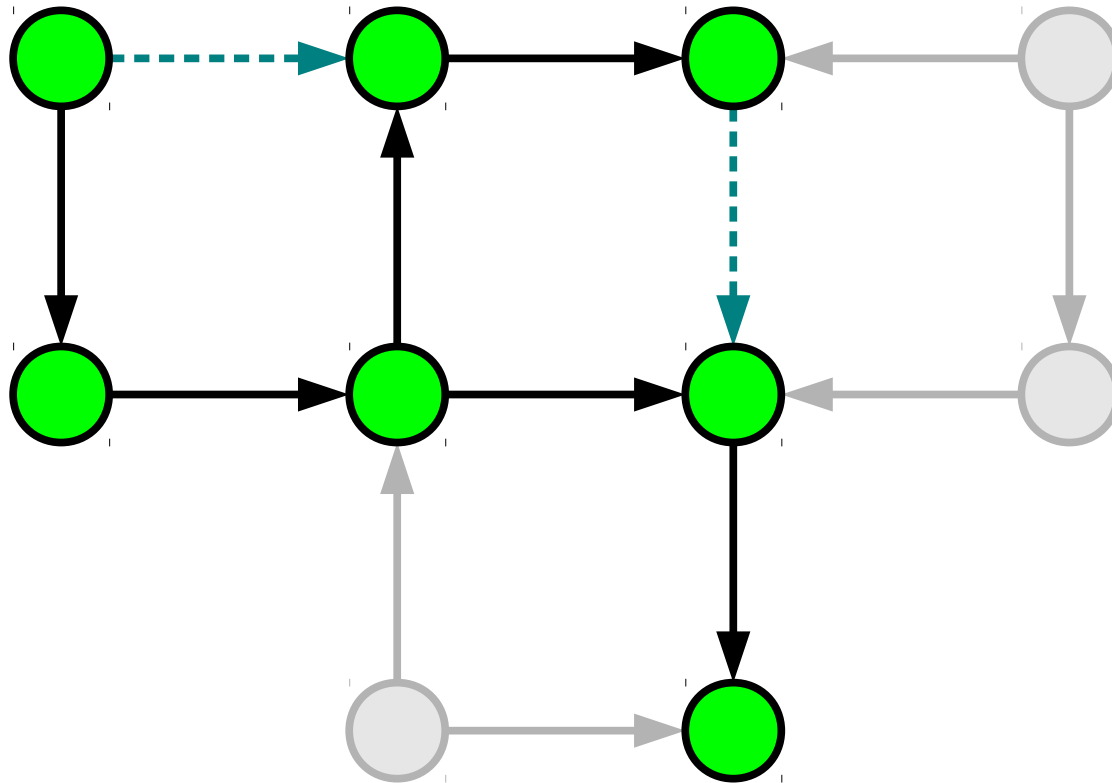
Depth-First Search, Again



Depth-First Search, Again



Depth-First Search, Again



How can we implement this?

Breadth-First Search

Queue: **X** **Q** **V** **A** **L**

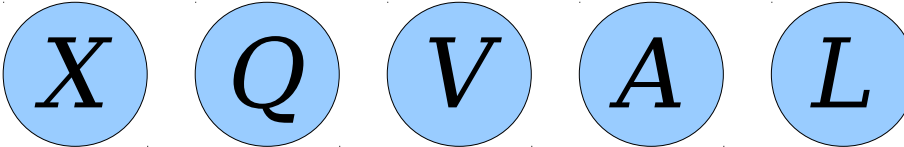
I've just been discovered! Pay attention to me!

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

C

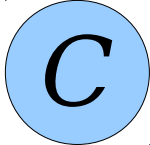
```
bfs-from(node v) {  
  make a queue of nodes, initially seeded with v.  
  while the queue isn't empty:  
    dequeue a node curr.  
    process the node curr.  
    for each node adjacent to curr:  
      if that node has never been enqueued:  
        enqueue that node.  
}
```

Depth-First Search

Stack: 

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

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.  
}
```


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

Recursive DFS

```
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  
}
```

```
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 Efficiency

- We visit each node with DFS at most once.
- Each time we visit a node, we do
 - some fixed work processing the node for the first time, then
 - some additional work processing its neighbors.

```
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  
}
```

DFS Efficiency

- On average, each node has roughly m/n neighbors.

```
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  
}
```

DFS Efficiency

- On average, each node has roughly m/n neighbors.
- Work done by DFS is

#nodes × average-work-per-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  
}
```

DFS Efficiency

- On average, each node has roughly m/n neighbors.
- Work done by DFS is

#nodes × average-work-per-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  
}
```

DFS Efficiency

- On average, each node has roughly m/n neighbors.
- Work done by DFS is

$$\begin{aligned} & \#nodes \times \text{average-work-per-node} \\ = & \quad n \quad \times \quad (\mathbf{O}(m/n) + \mathbf{O}(1)) \end{aligned}$$

```
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  
}
```

DFS Efficiency

- On average, each node has roughly m/n neighbors.
- Work done by DFS is

$$\begin{aligned} & \#nodes \times average\text{-}work\text{-}per\text{-}node \\ = & n \times (O(m/n) + O(1)) \\ = & n \cdot O(m/n) + n \cdot O(1) \end{aligned}$$

```
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  
}
```


DFS Efficiency

- On average, each node has roughly m/n neighbors.
- Work done by DFS is

$$\begin{aligned} & \#nodes \times average-work-per-node \\ = & n \times (O(m/n) + O(1)) \\ = & n \cdot O(m/n) + n \cdot O(1) \\ = & O(m + n). \end{aligned}$$

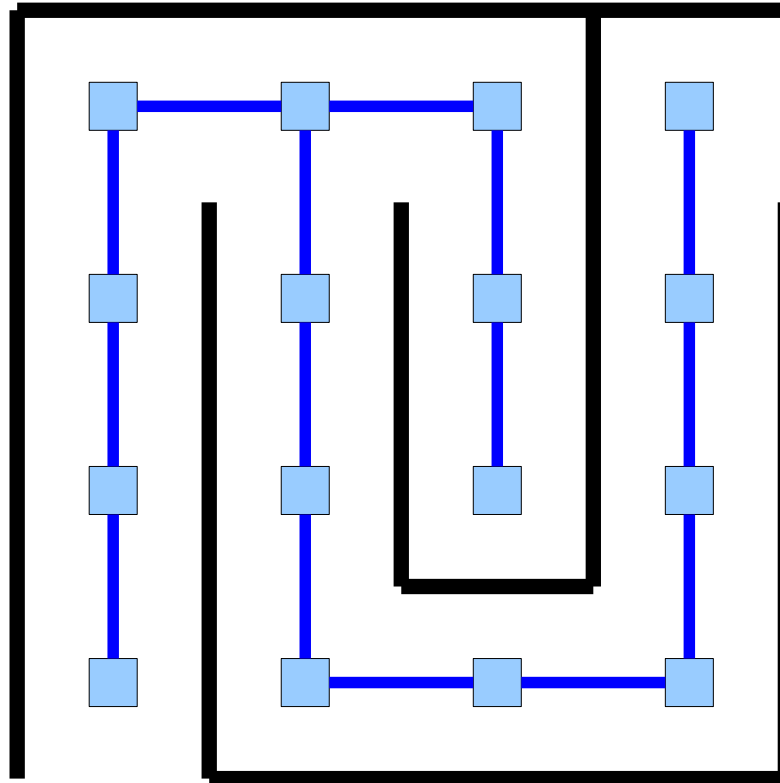
```
dfs-from(node v) {  
    if this is first time we've called dfs-from(v):  
        process node v  
        for each node adjacent to v:  
            call dfs-from on that node  
}
```

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.
 - More on that later on...

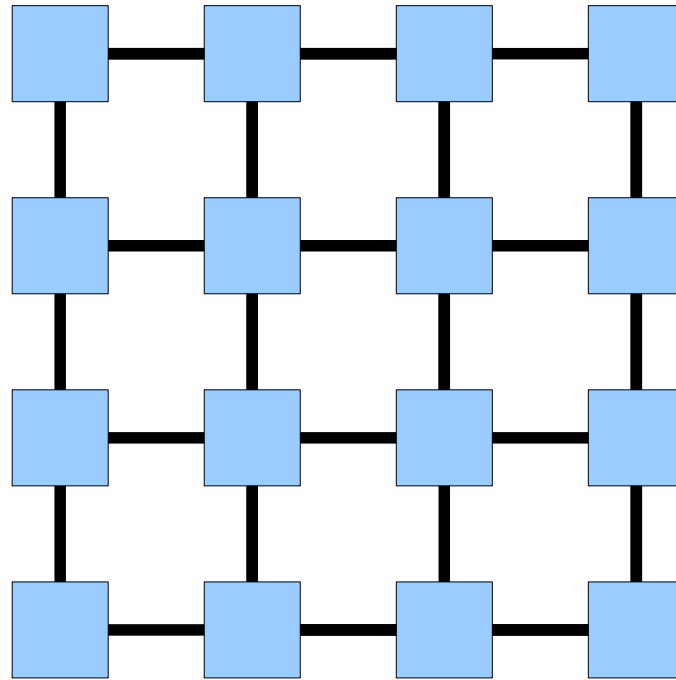
A Whimsical Application

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.

Time Out for Announcements!

Apply to take this spring:

CS 21SI

AI For Social Good

[CS21SI: AI for Social Good](#) is a 2-unit student-taught class in which students learn about and apply cutting-edge artificial intelligence techniques to real-world social good spaces, such as healthcare, government, education, and the environment.

The course alternates between lectures on machine learning theory and discussions with invited speakers, who will challenge students to apply techniques in their social good domains. You can learn more about the class [here](#).

Apply by 11:59pm on Sunday, March 17 at bit.ly/AIForGoodApp.

Apply to the Blueprint Datathon, SHIFT's annual health datathon, to apply big data analytics to challenges in healthcare and compete for large cash prizes. Our 2019 theme is Infectious Diseases.

Apply @ blueprint.stanford.edu - spots are limited, and applicants will be accepted on a rolling basis. Applications are due March 22 for non-Stanford affiliates and April 1 for Stanford students.

Blueprint will take place from April 12 - 14 at Hewlett-Packard Building.

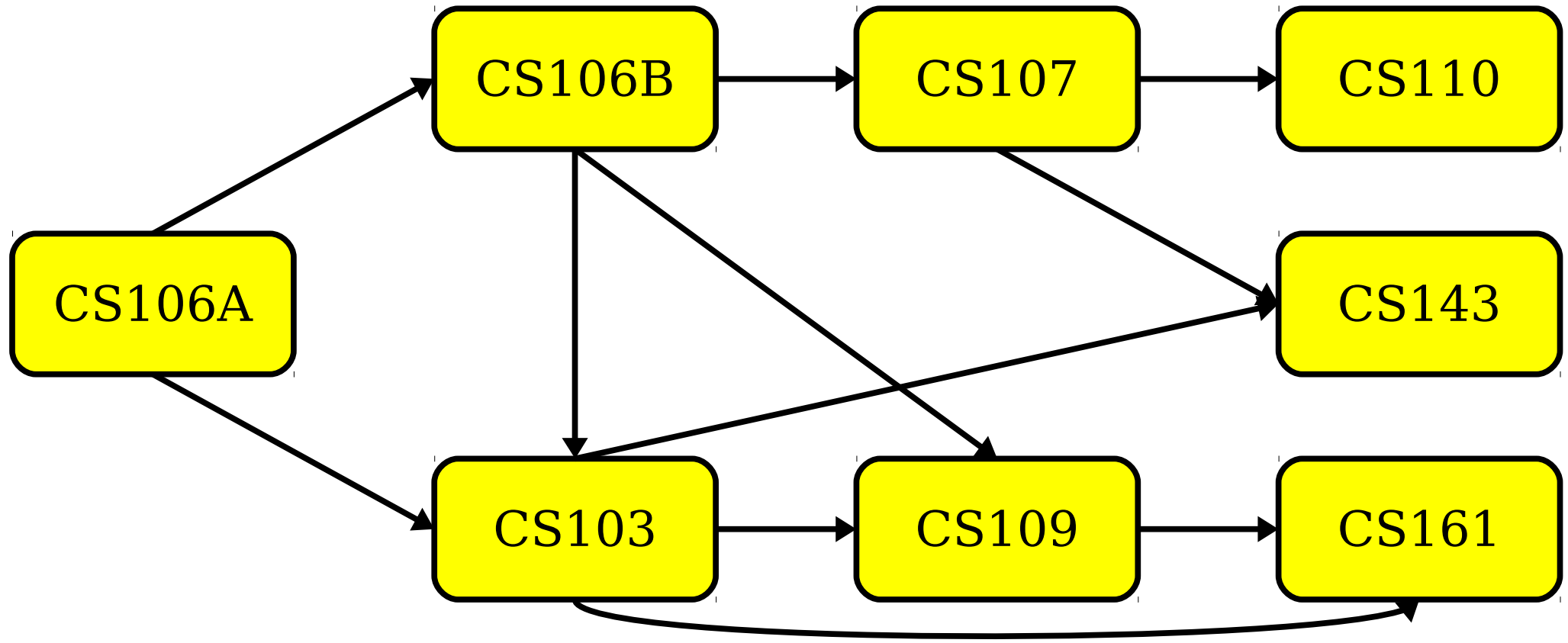
Bring your best ideas and tell your friends!

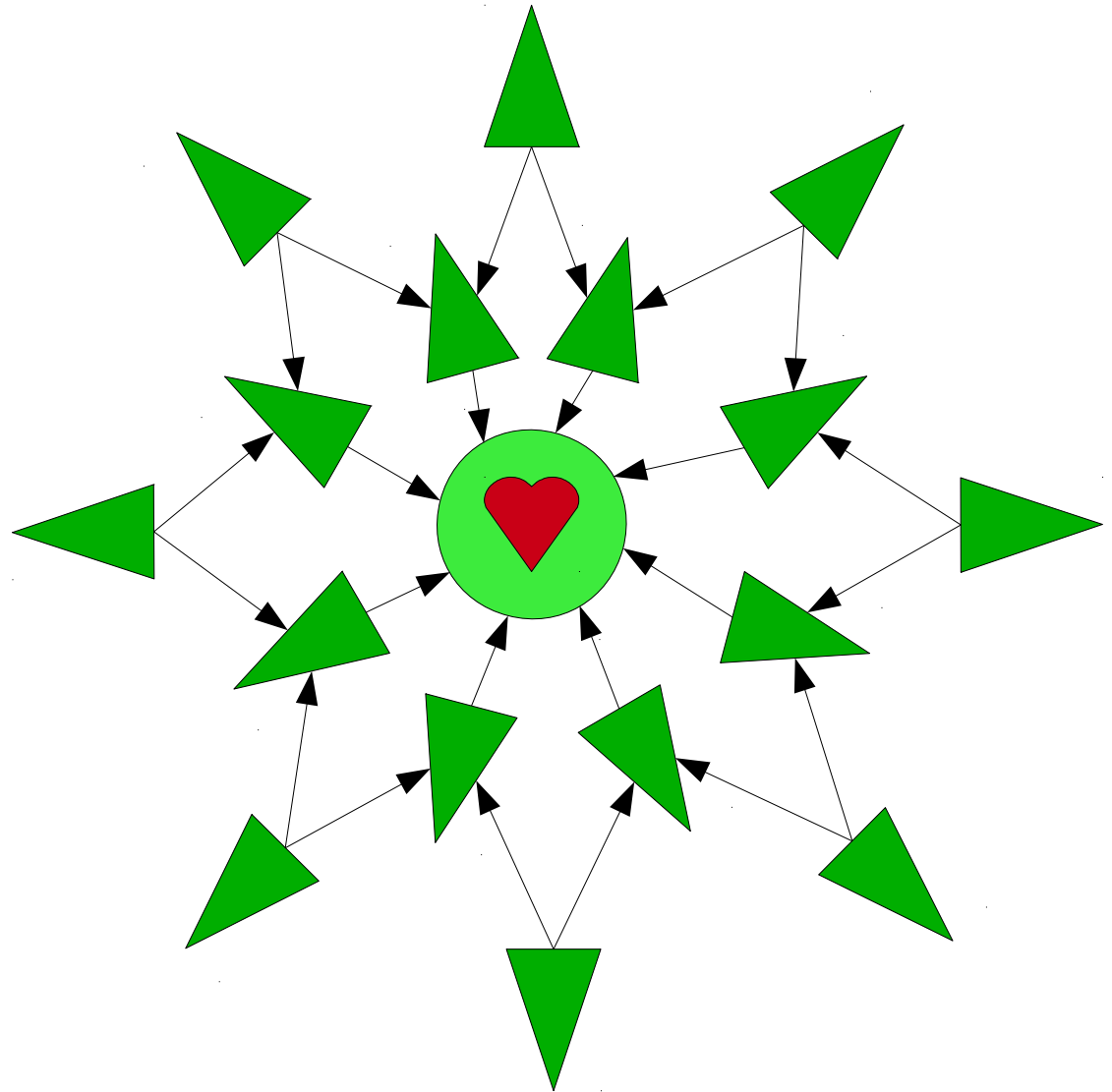
Assignment 6

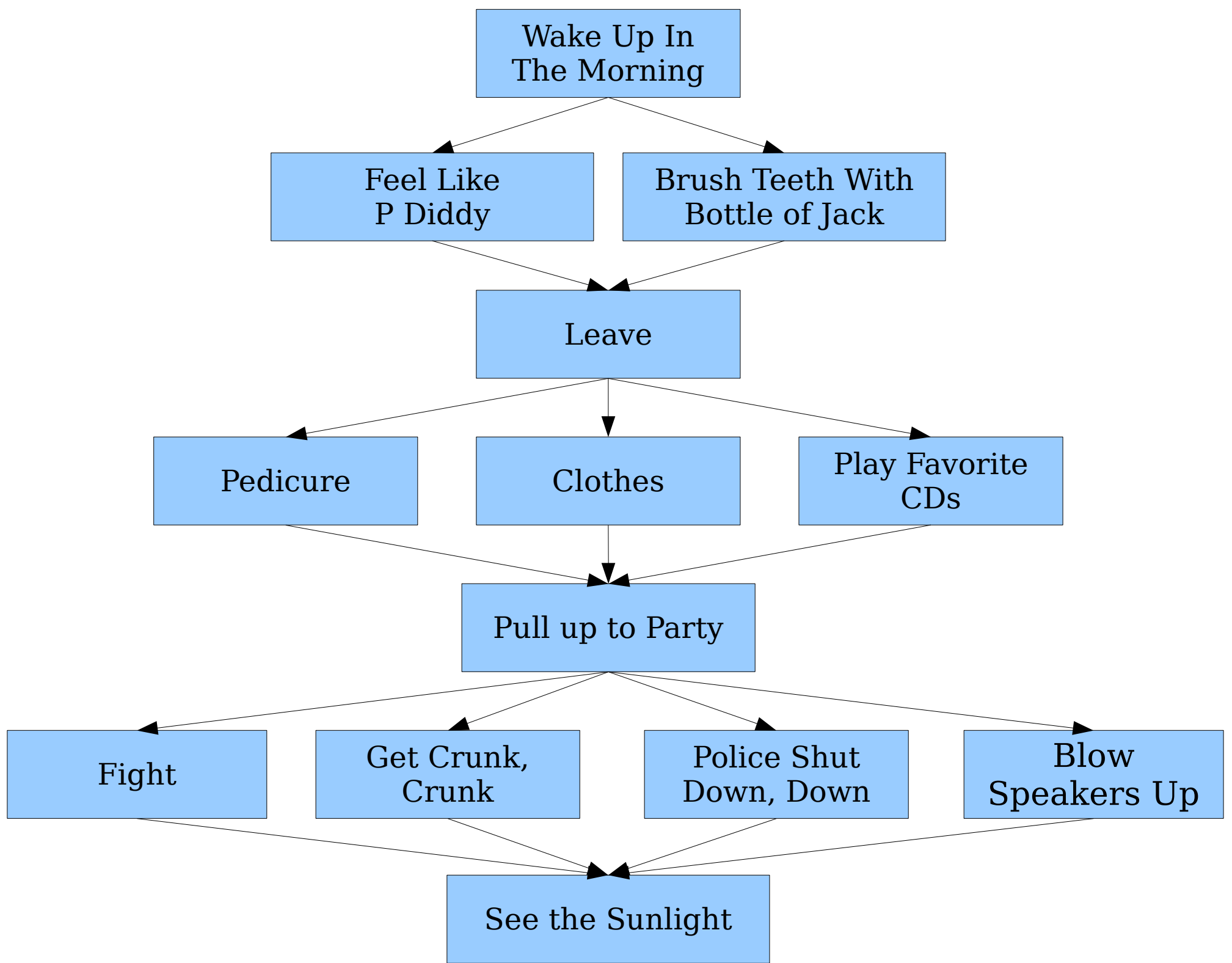
- Assignment 6 (*MiniBrowser*) is due this Friday.
- The following are True Facts:
 - This is the last assignment on which you can use late days. No late submissions will be accepted for Assignment 7. (Sorry, that's university policy.)
 - You should be careful about using late days here, as that will eat into the time for Assignment 7.
- YEAH Hours for Assignment 7 will be held this Friday at 3PM in room 380-380Y. Slides, as usual, will be posted on the website afterward.

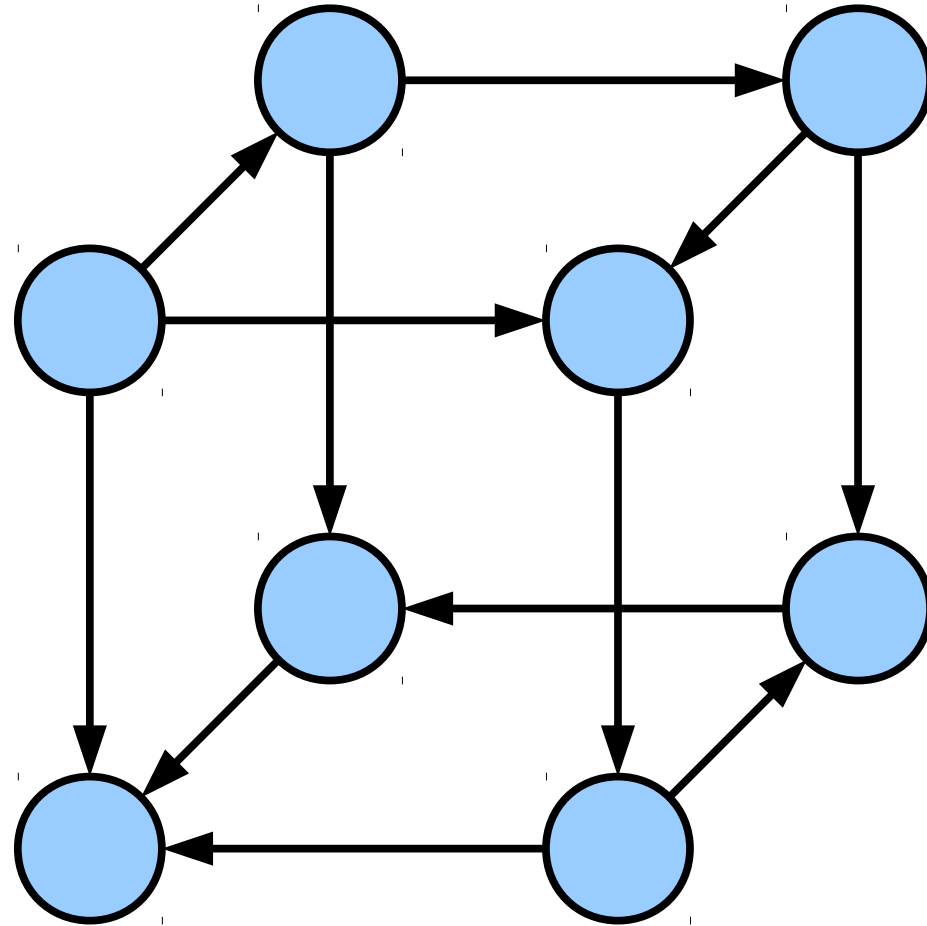
We Have to Go Deeper!

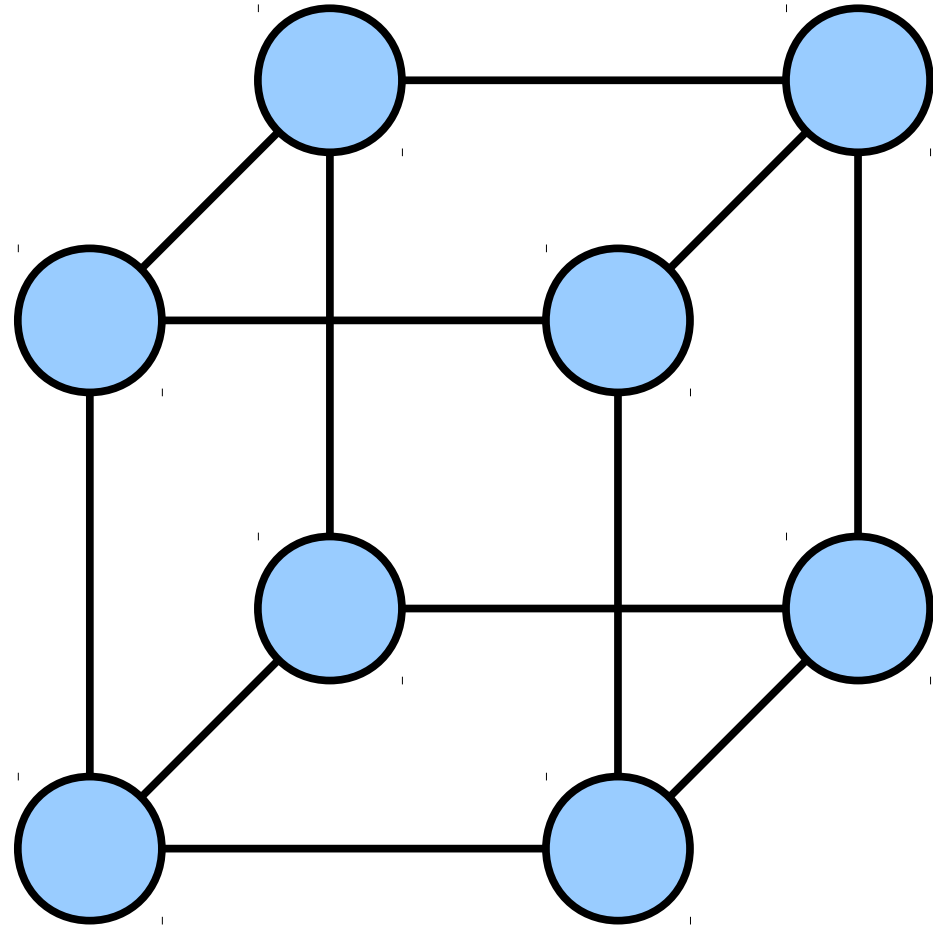
Prerequisite Structures

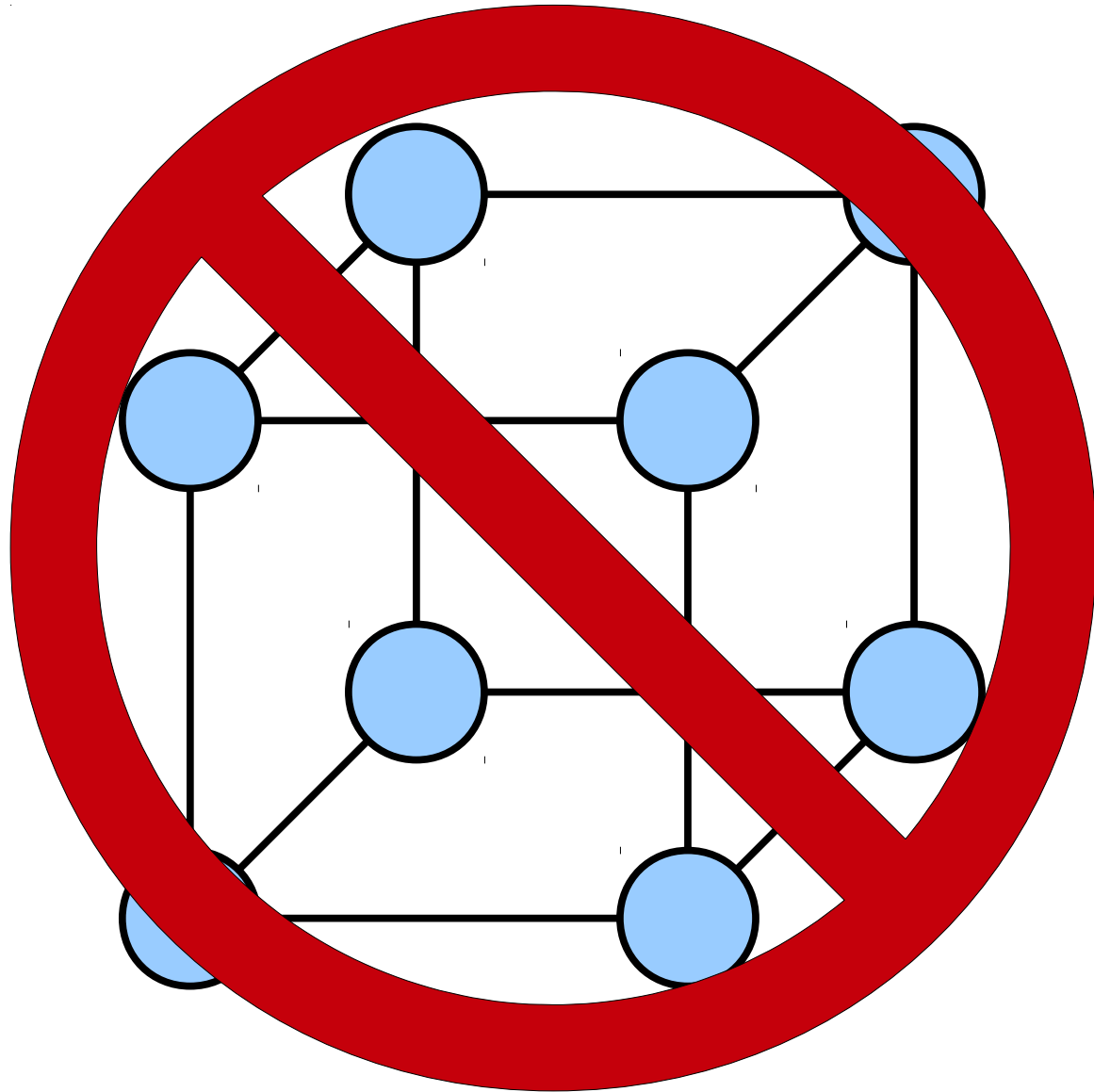


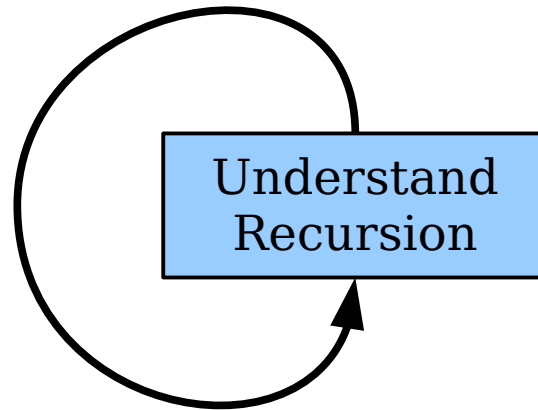


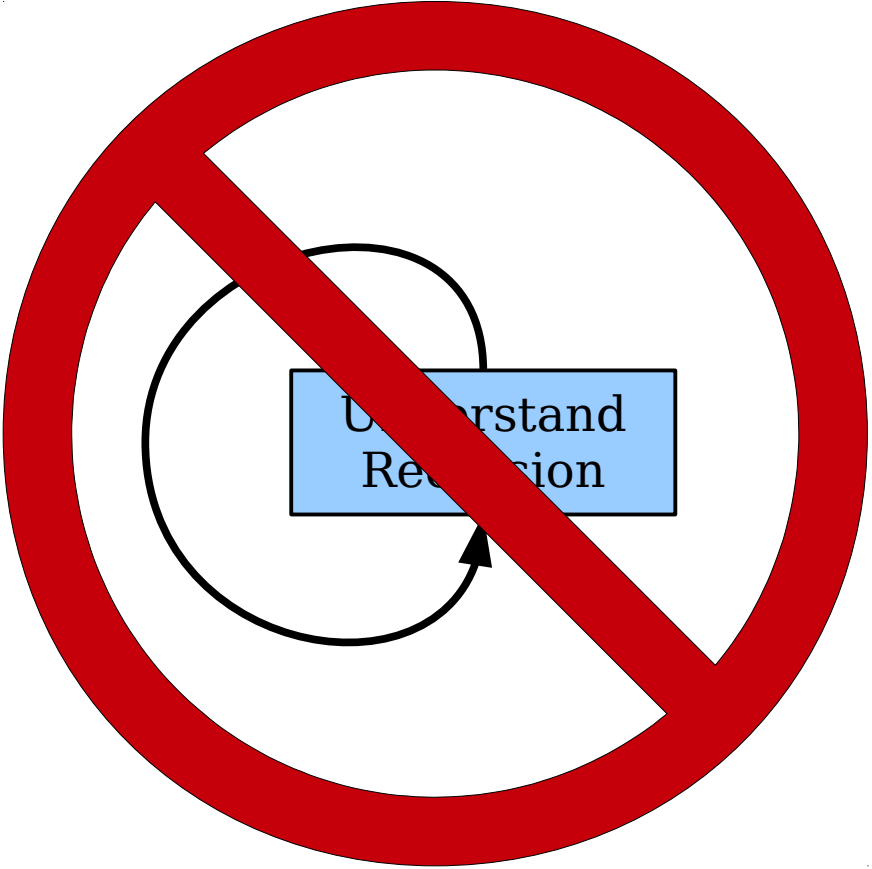










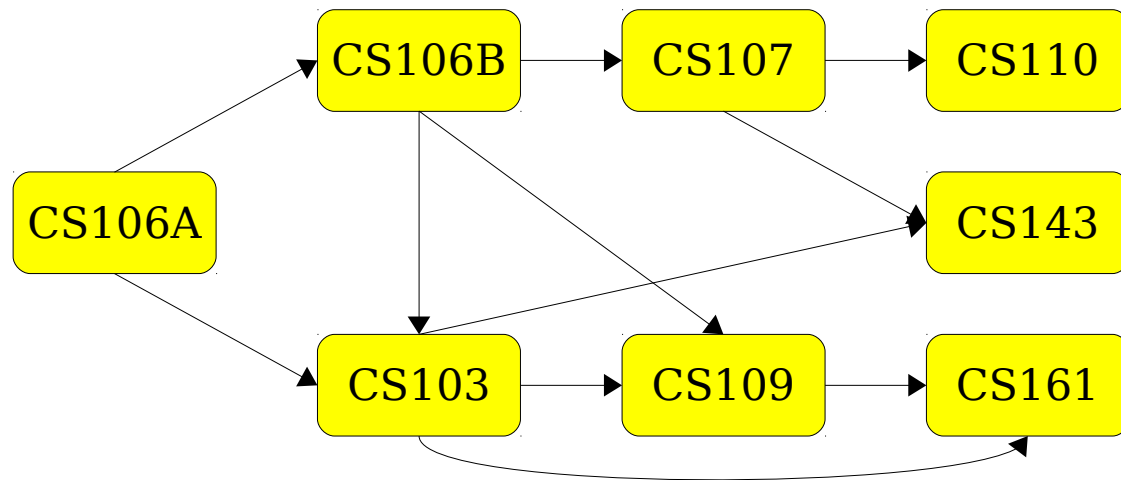


Modeling Prerequisites

- We can model prerequisites as a graph with the following properties:
 - The graph has to be **directed**, since we have to be able to distinguish “A depends on B” from “B depends on A.”
 - The graph has to be **acyclic** (containing no cycles), since otherwise there is no way to accomplish all the tasks.
- A graph with this property is called a **directed acyclic graph**, or **DAG**.

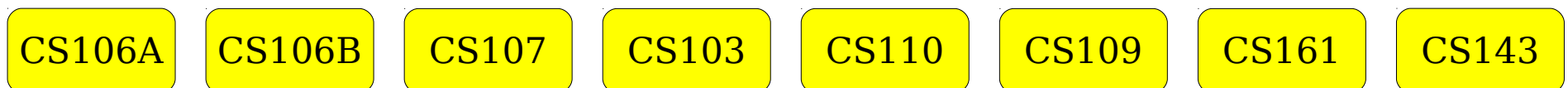
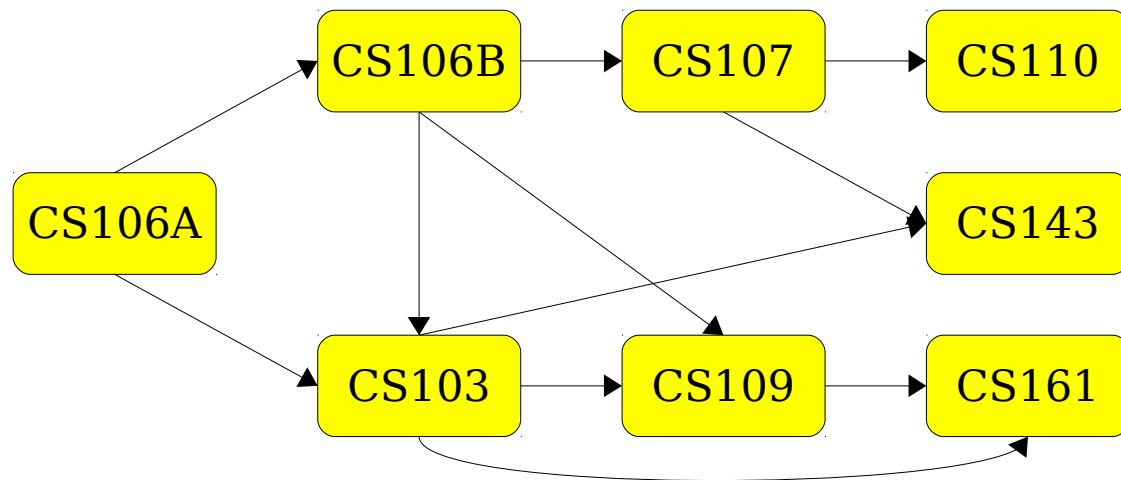
Ordering Prerequisites

- Imagine we have a DAG representing a collection of tasks. We can only start a task once its prerequisite tasks have been completed.
- What order should we do the tasks in?



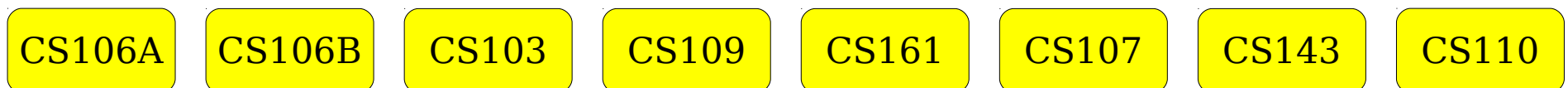
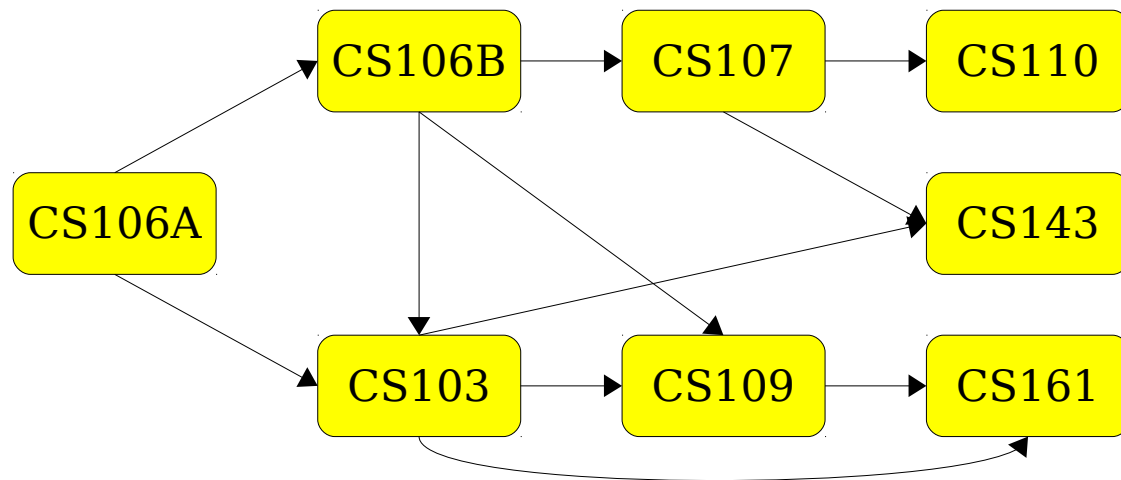
Ordering Prerequisites

- A **topological ordering** of the nodes in a DAG is a list of the nodes so that each node is placed after the nodes that point to it.
- An algorithm for finding a topological ordering is called a **topological sorting algorithm**.



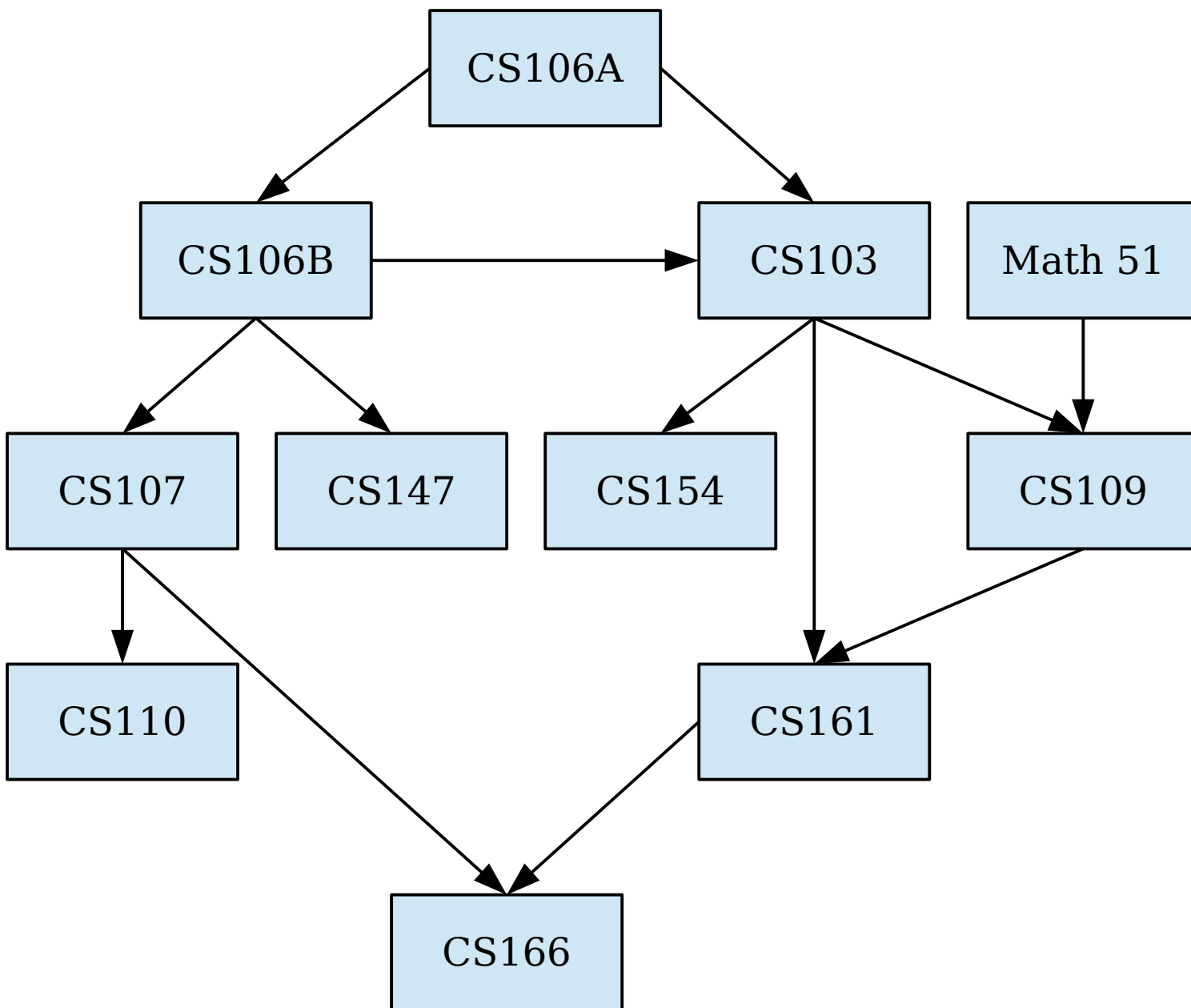
Ordering Prerequisites

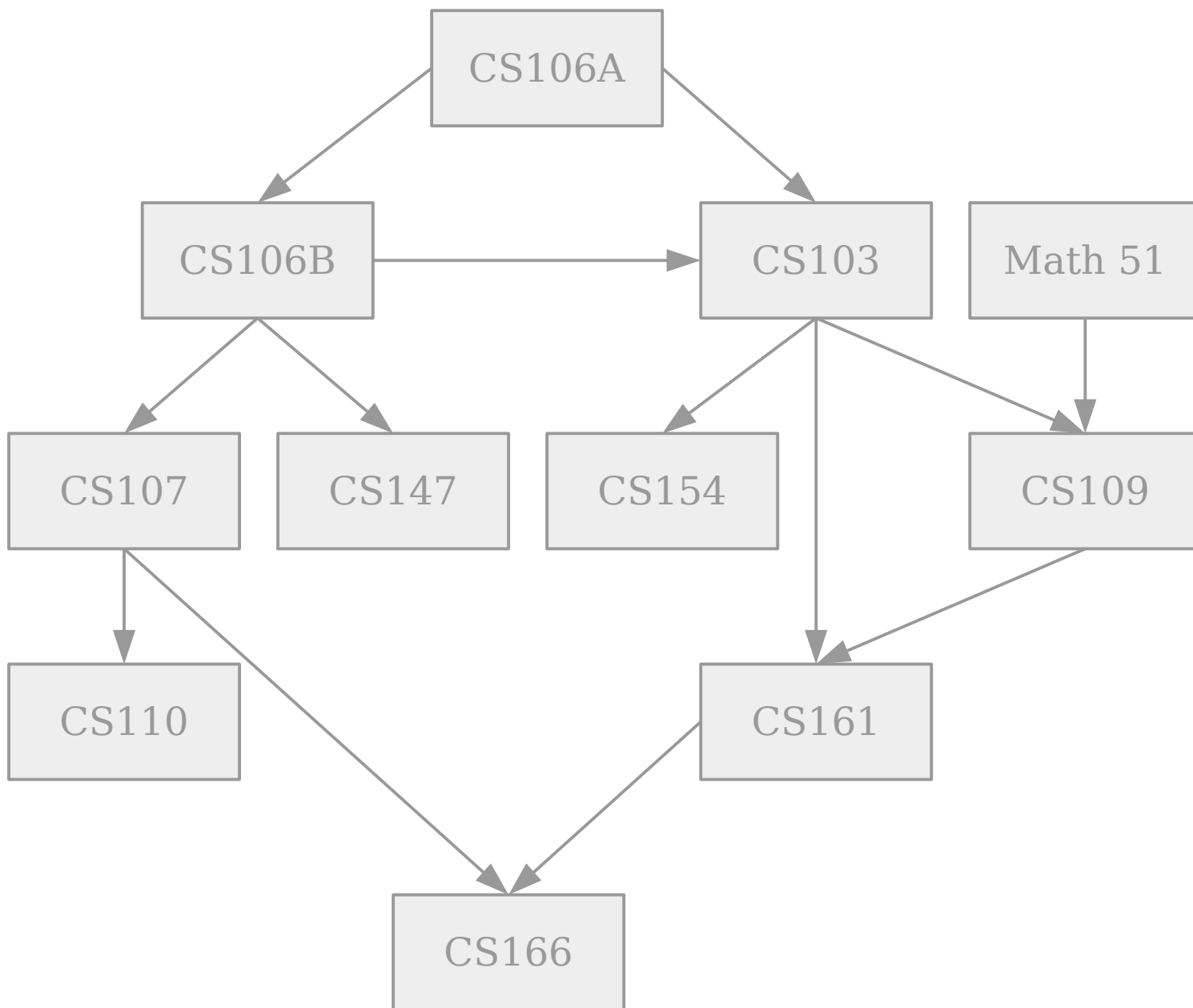
- A **topological ordering** of the nodes in a DAG is a list of the nodes so that each node is placed after the nodes that point to it.
- An algorithm for finding a topological ordering is called a **topological sorting algorithm**.

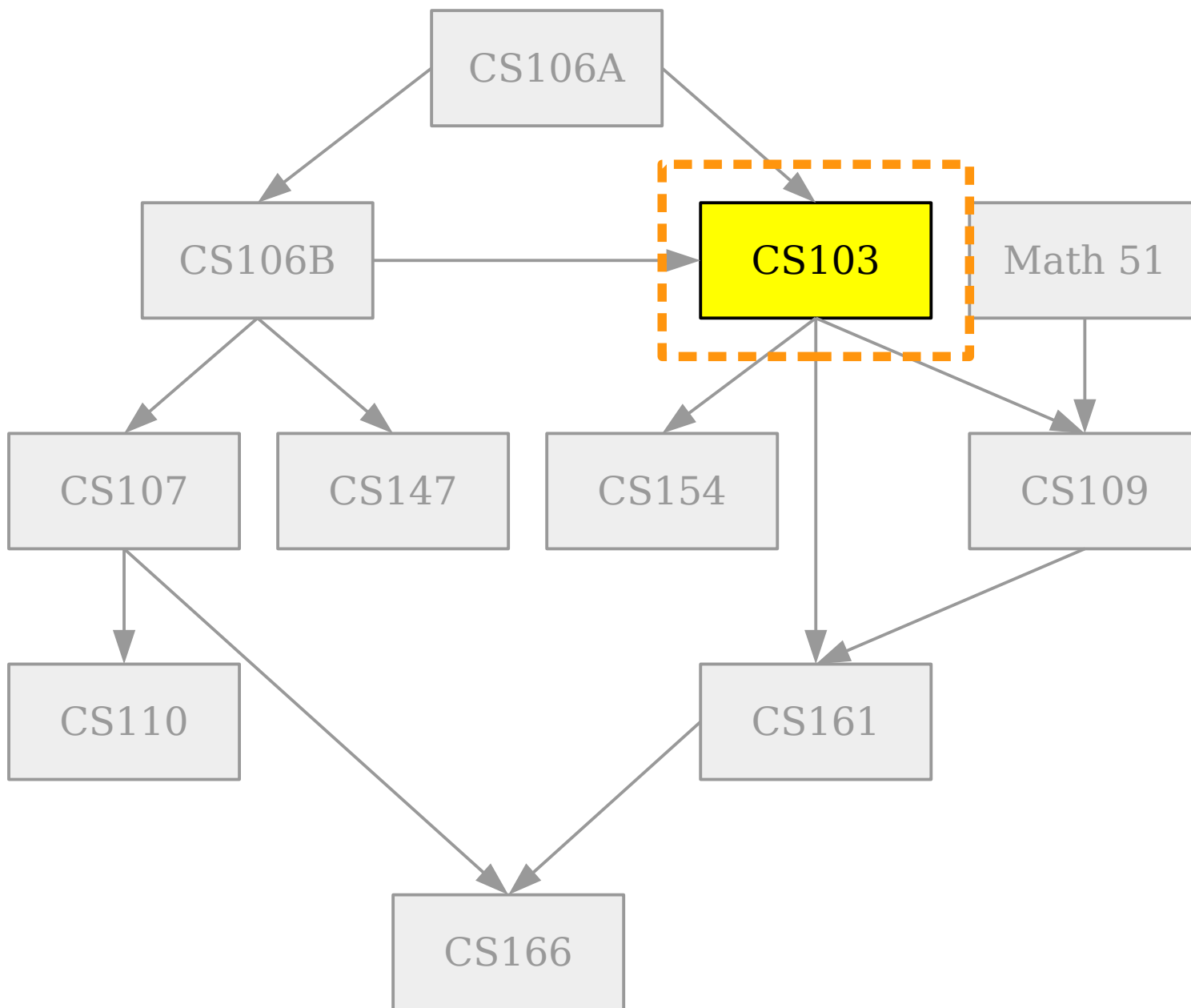


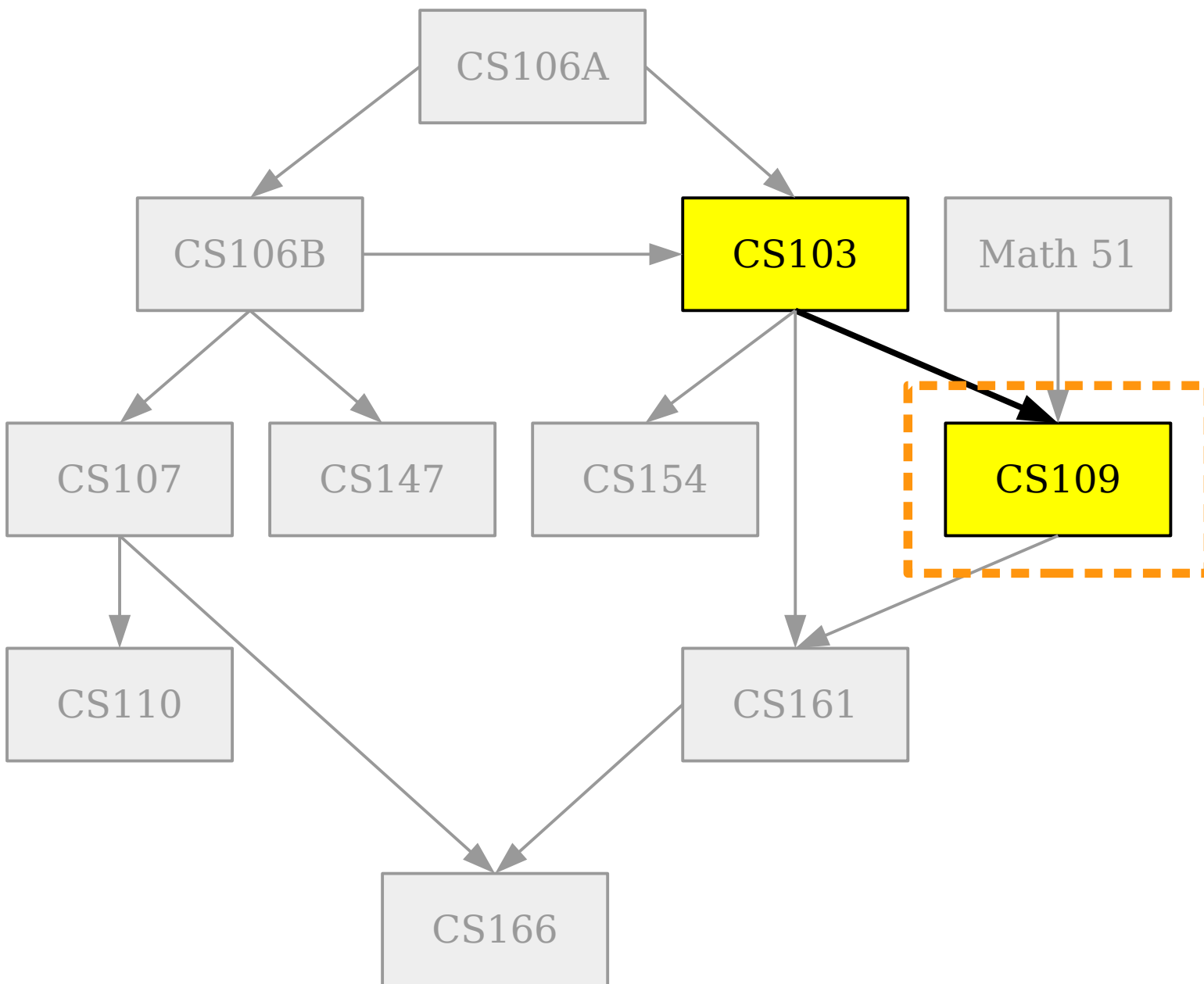
Adventures in Topological Sorting

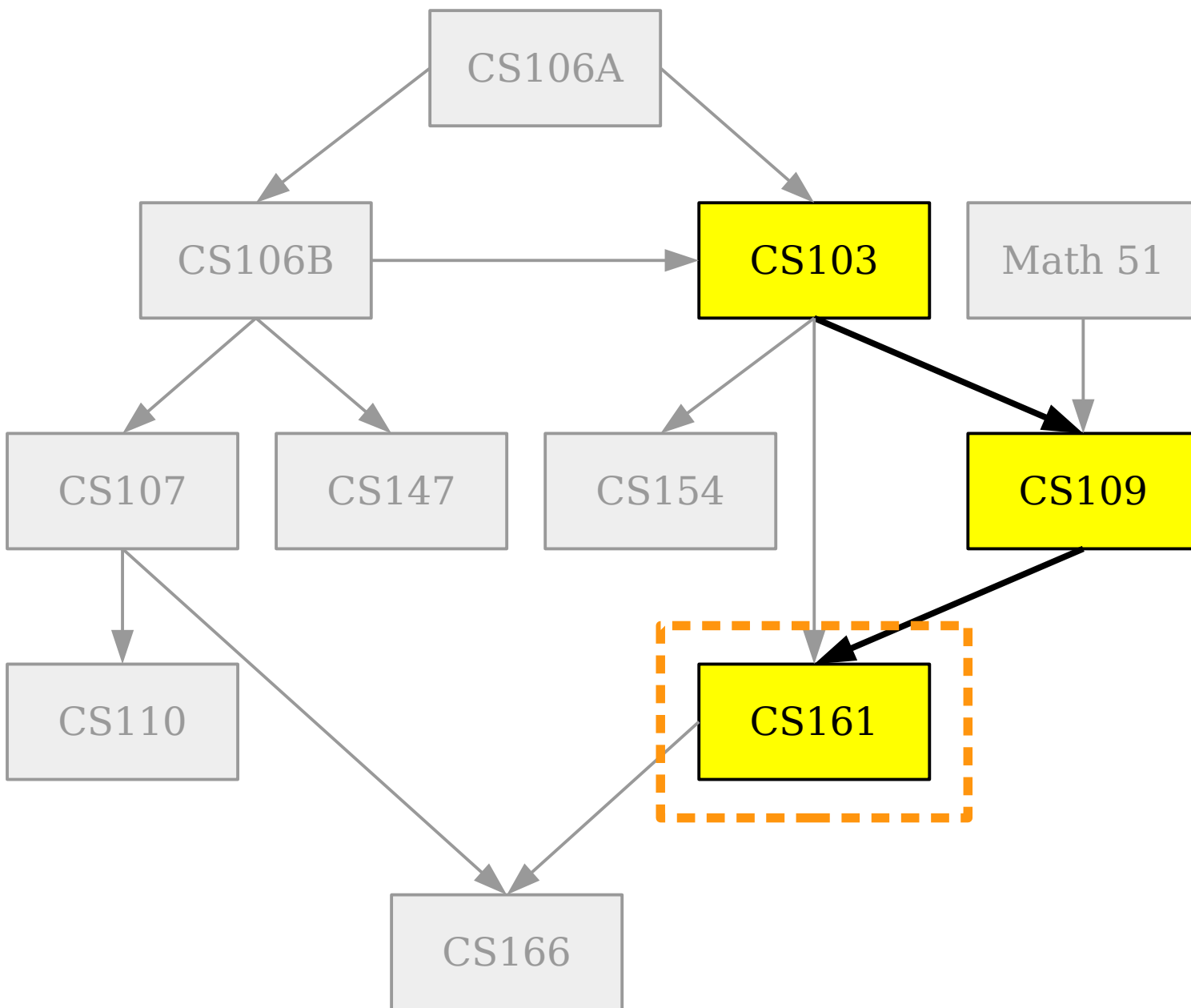
DFS Topological Sorting

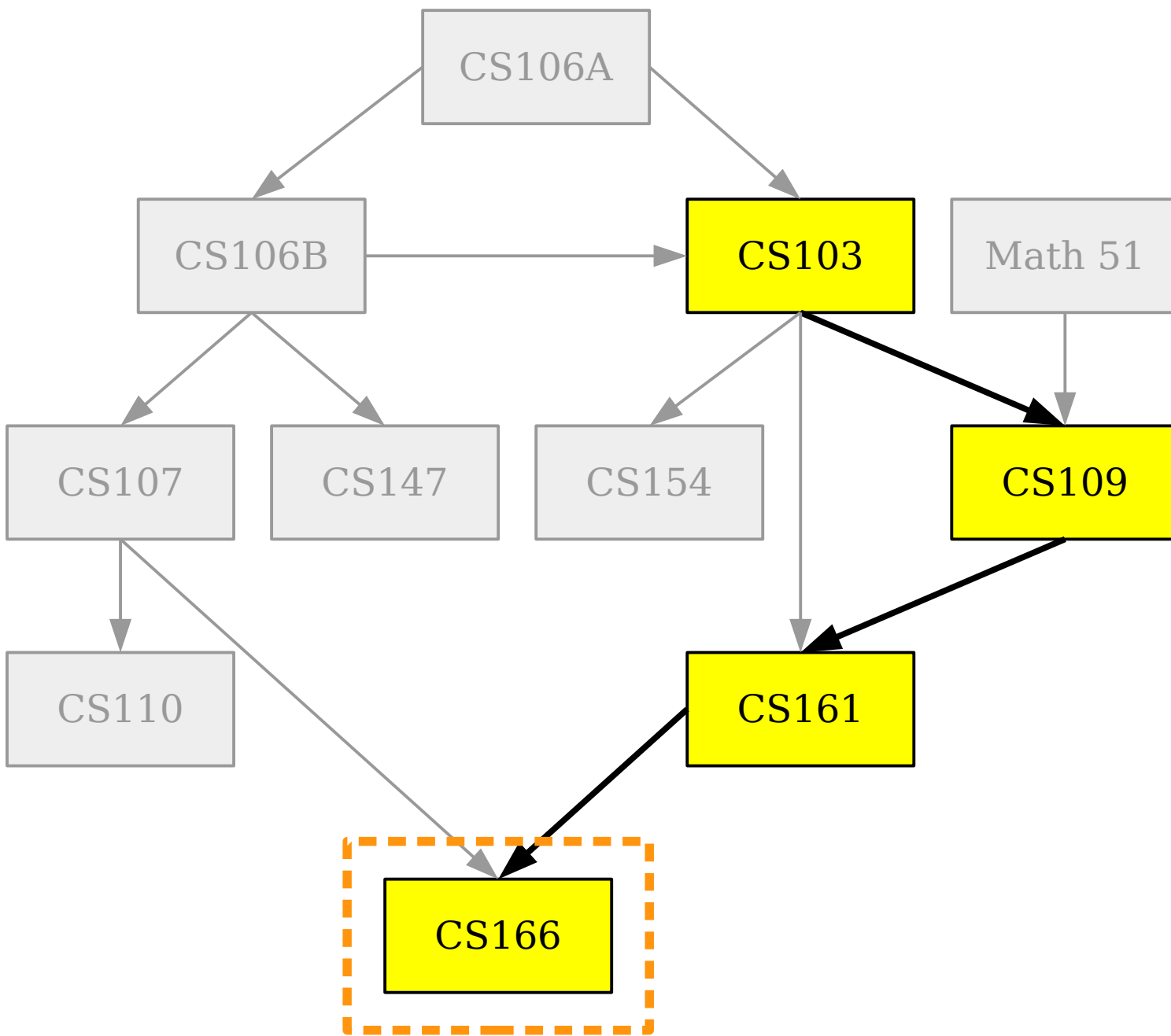


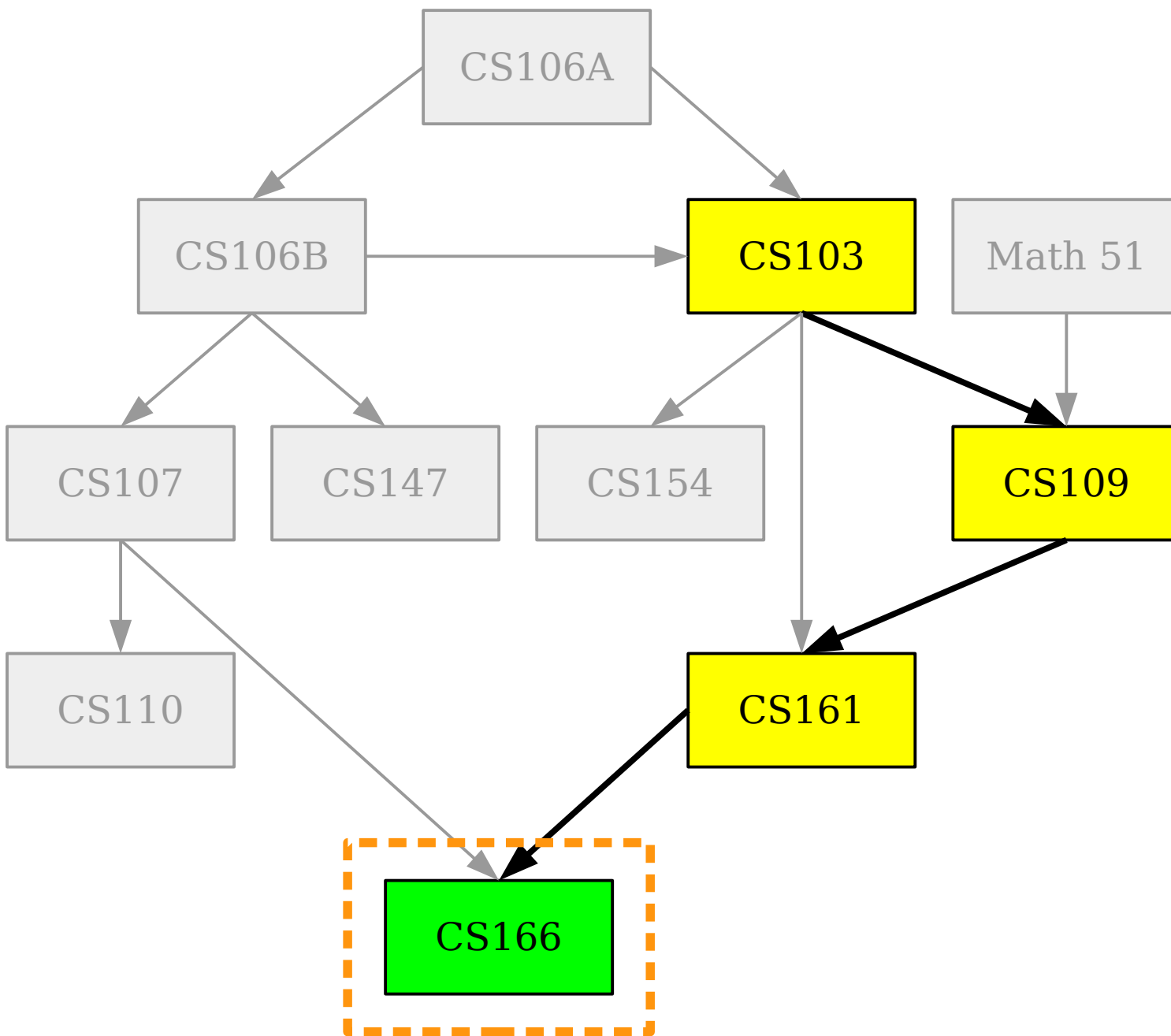


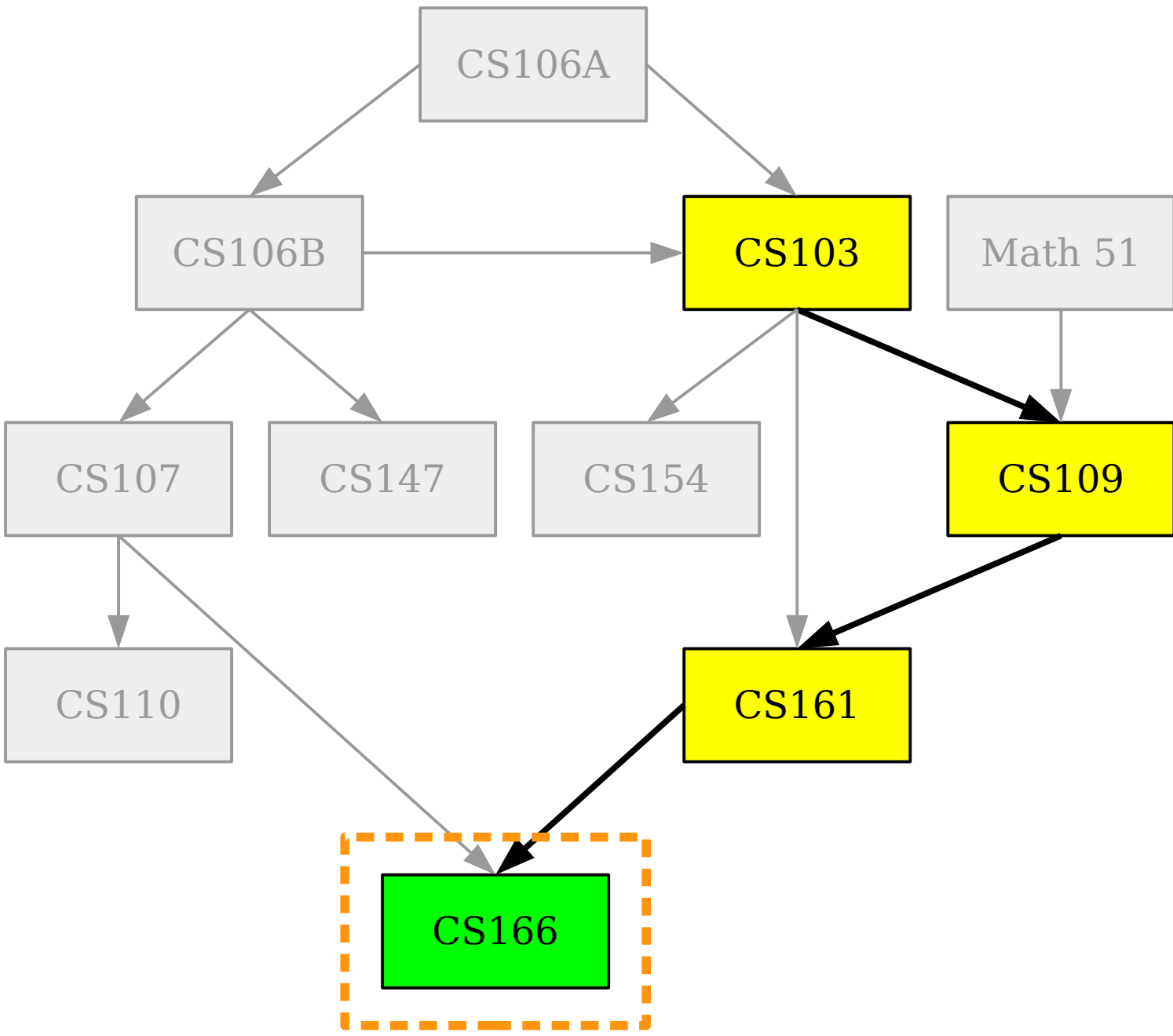


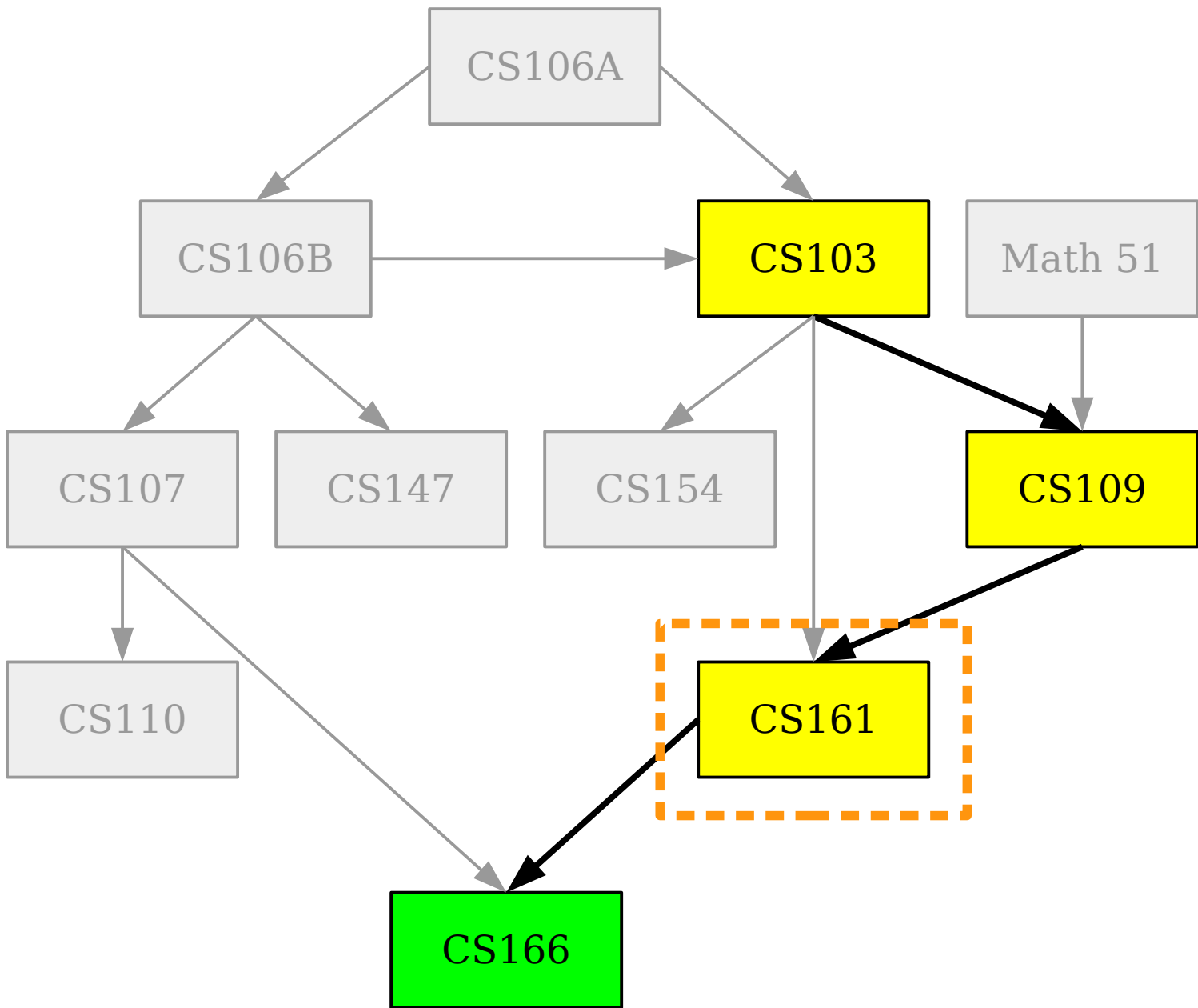


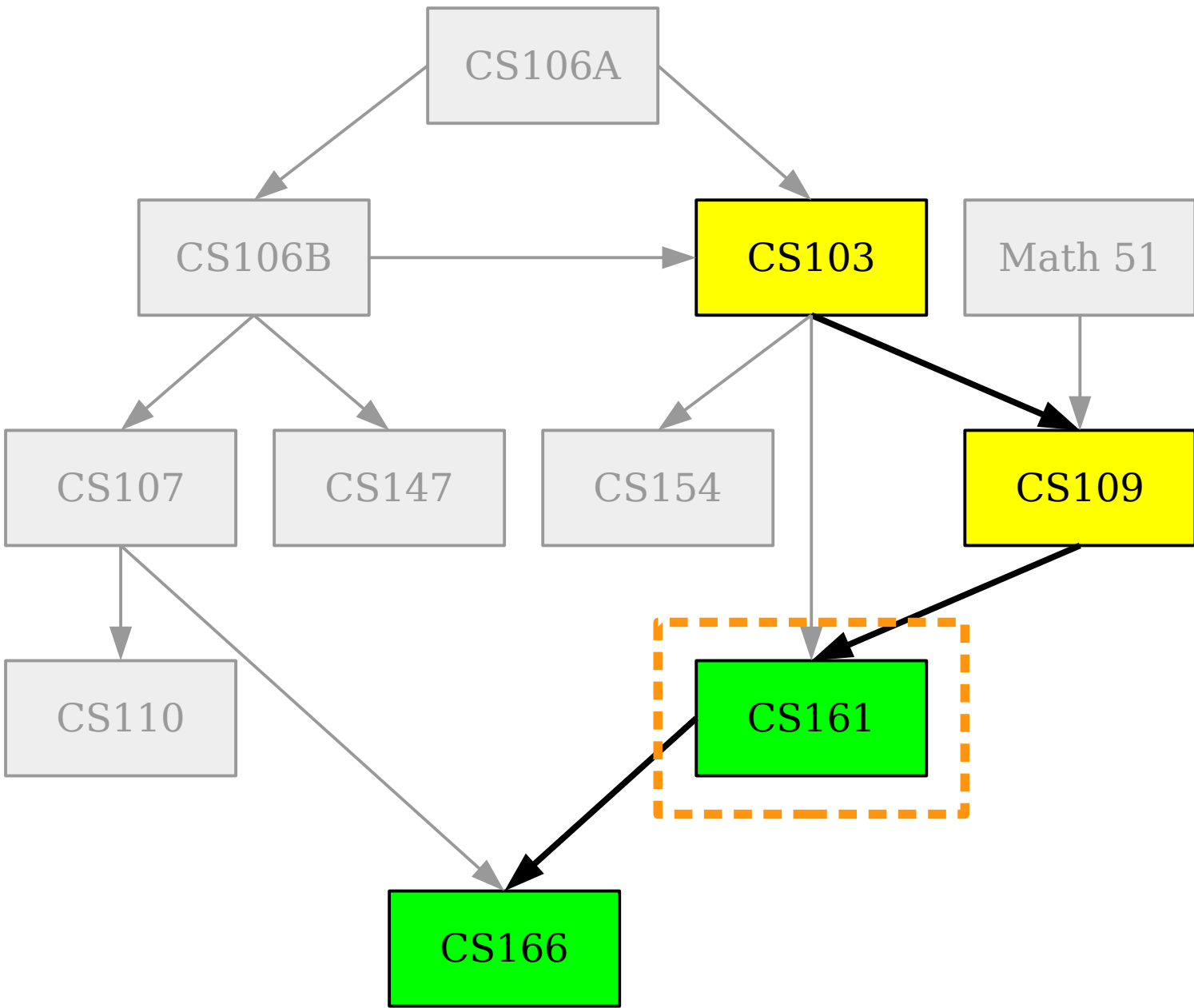


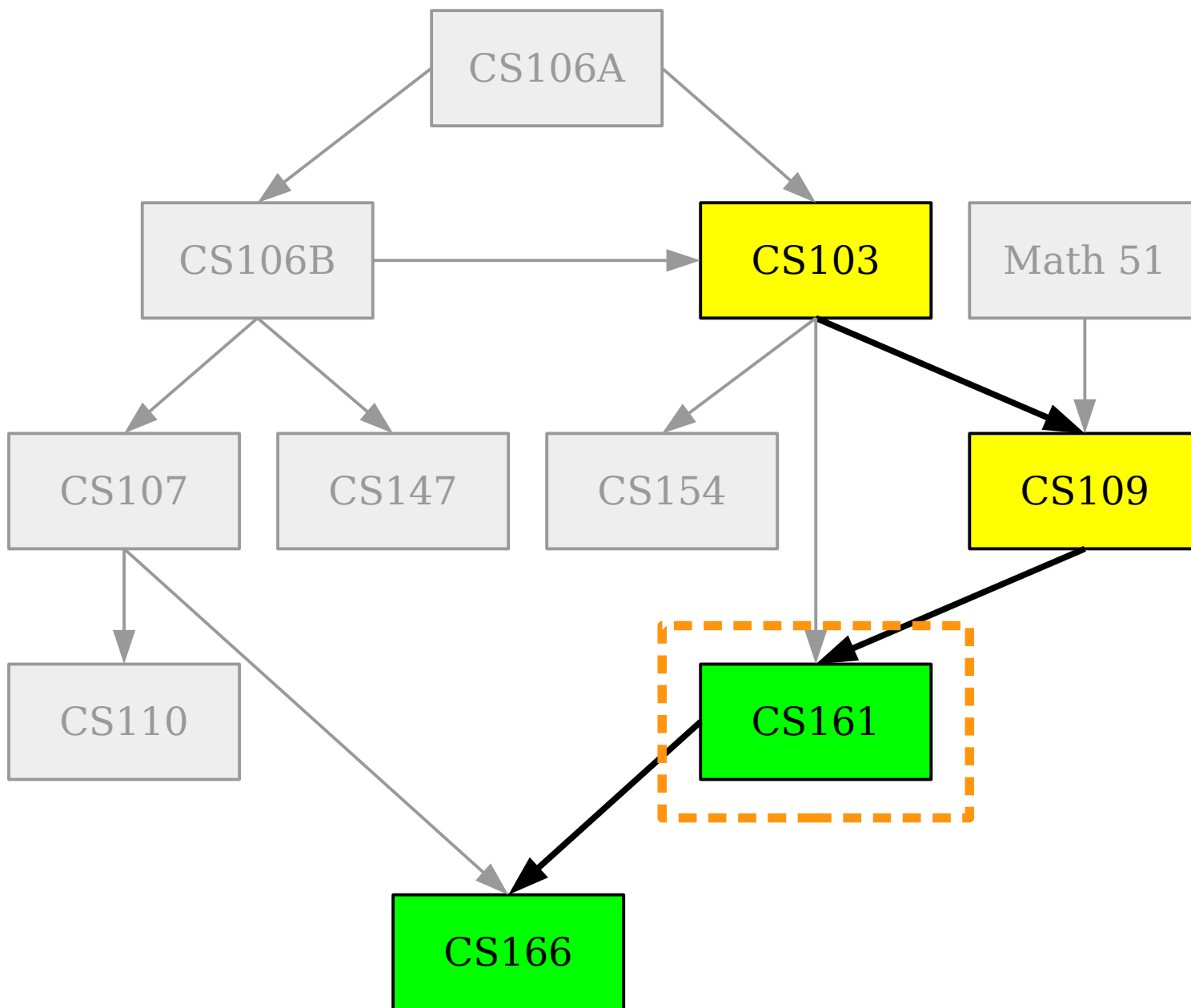




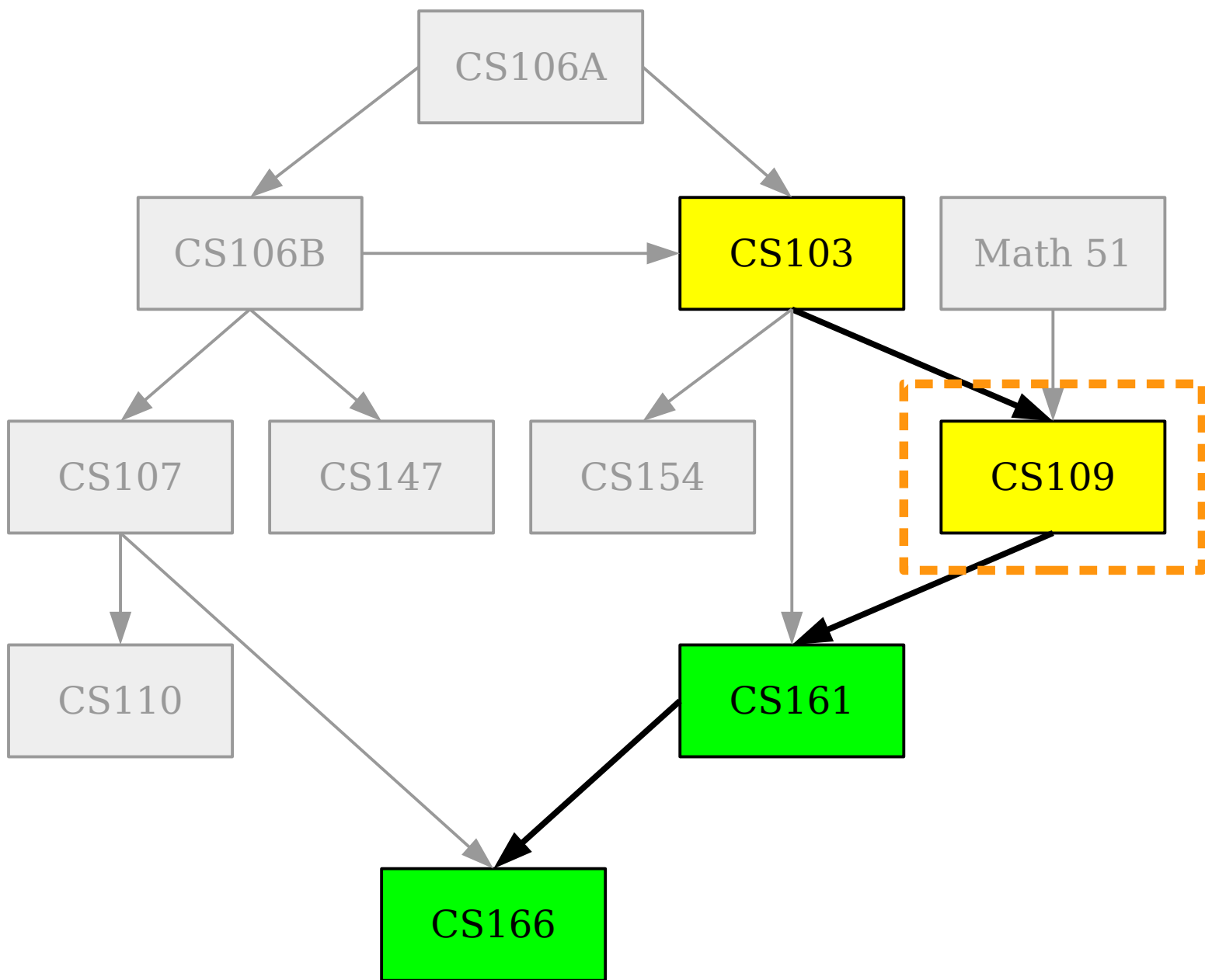




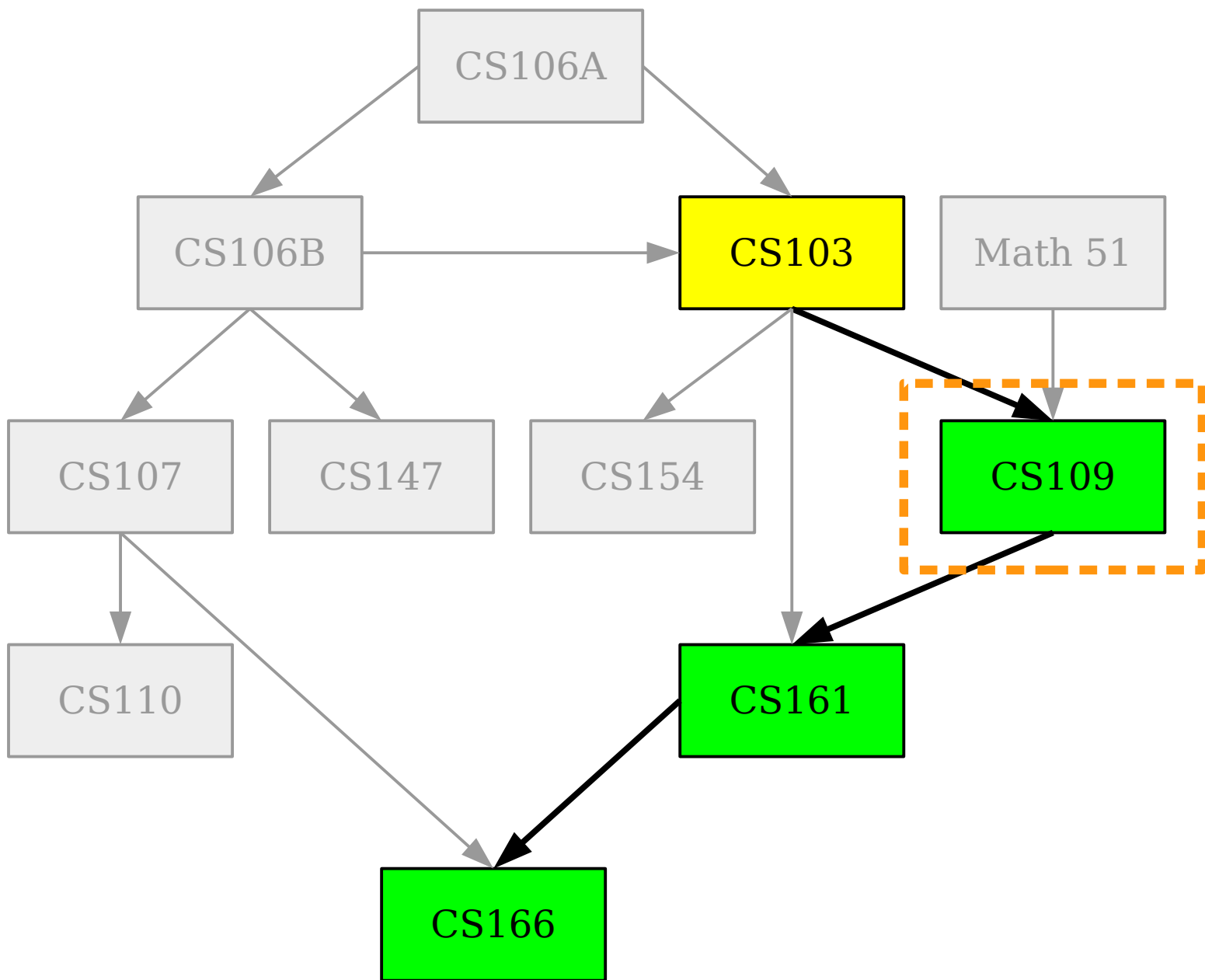




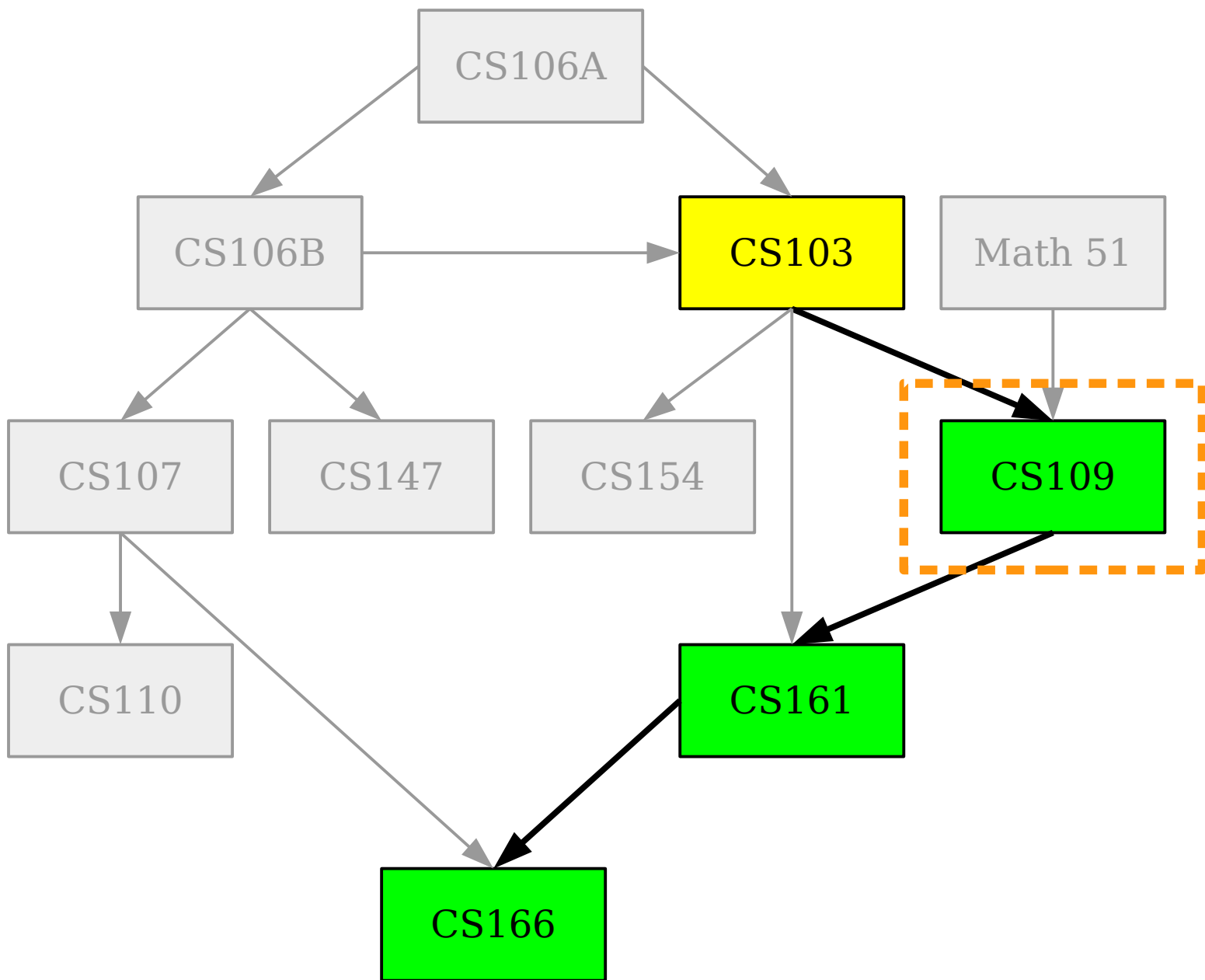
CS166
CS161



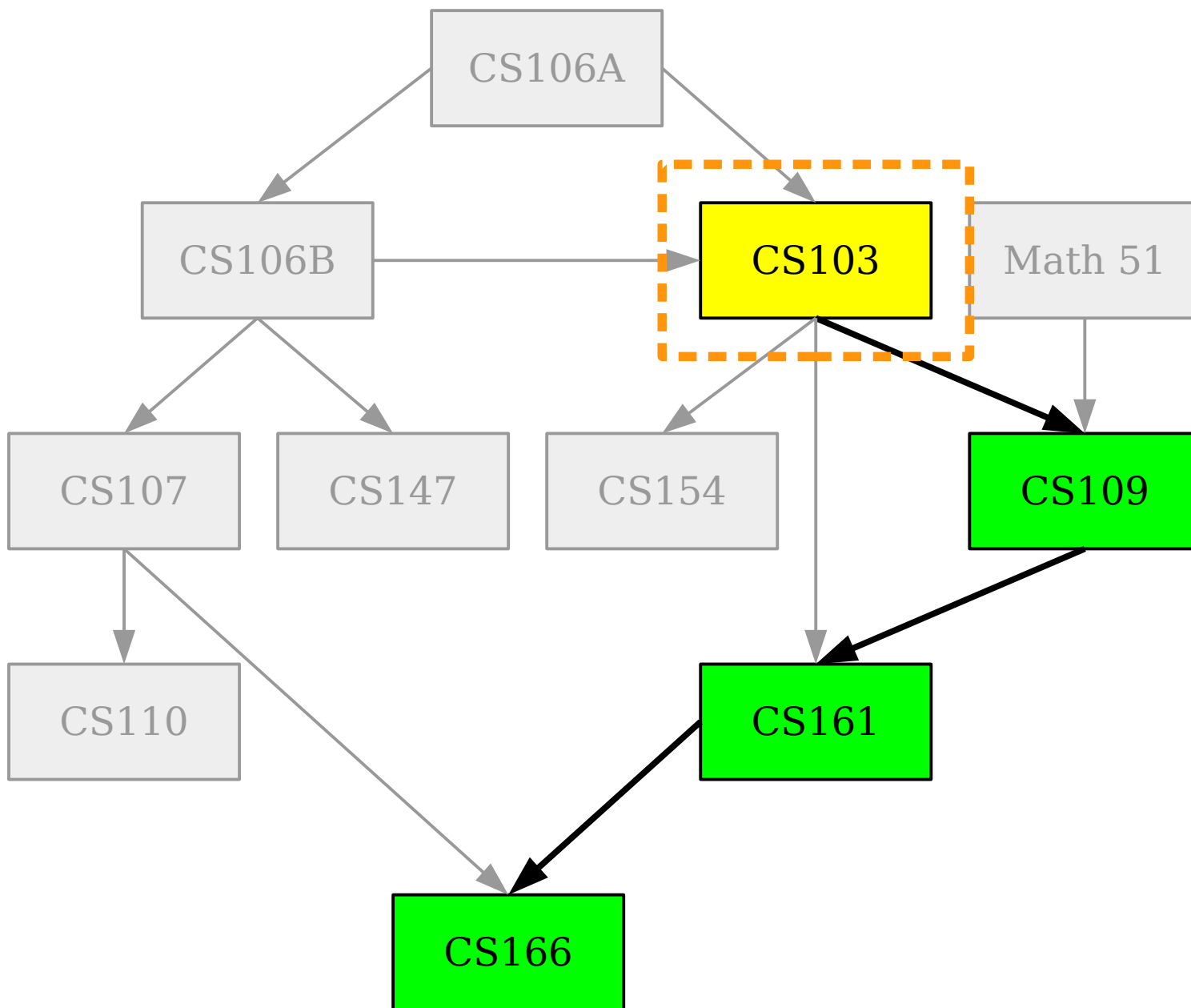
CS166
CS161



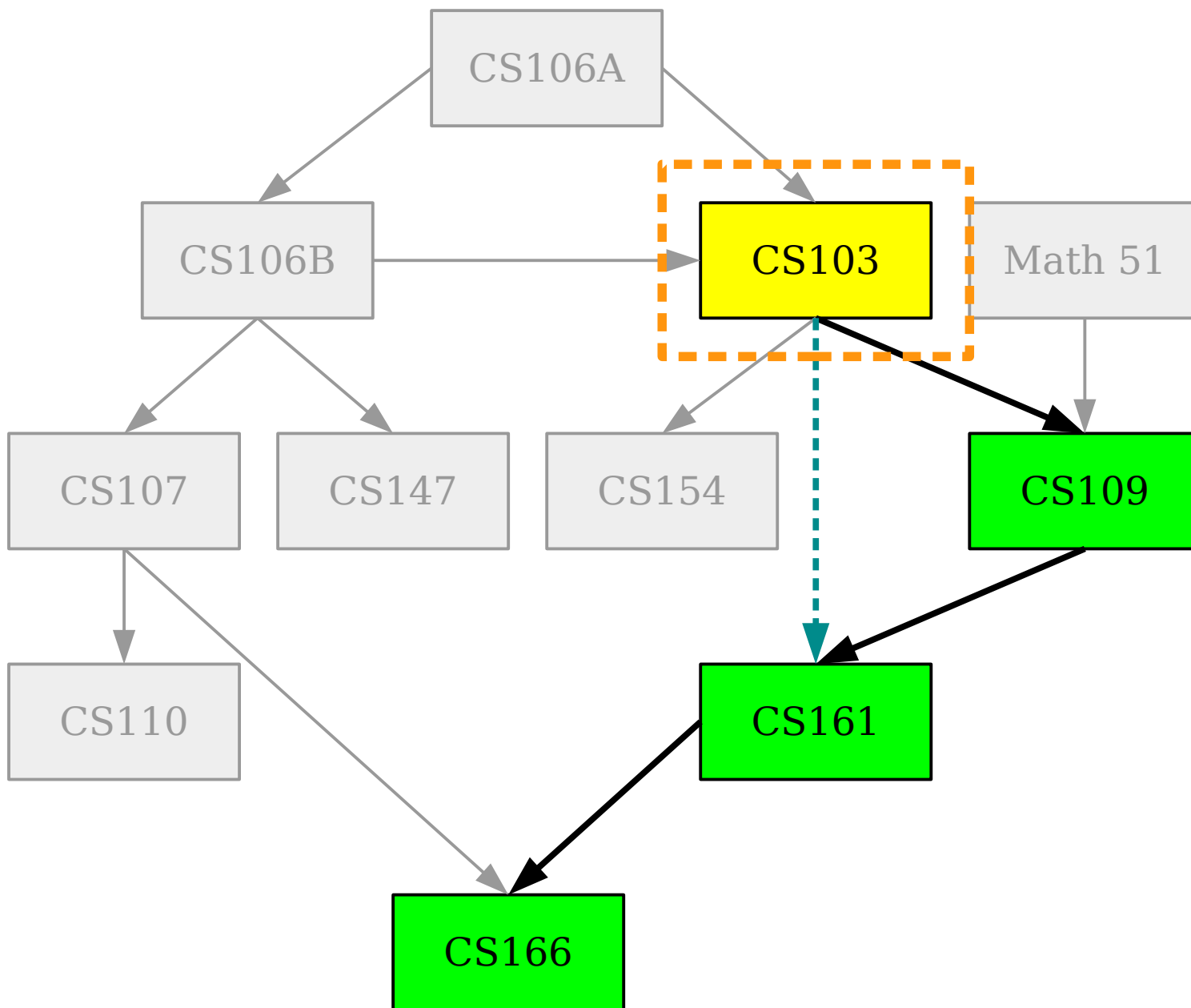
CS166
CS161



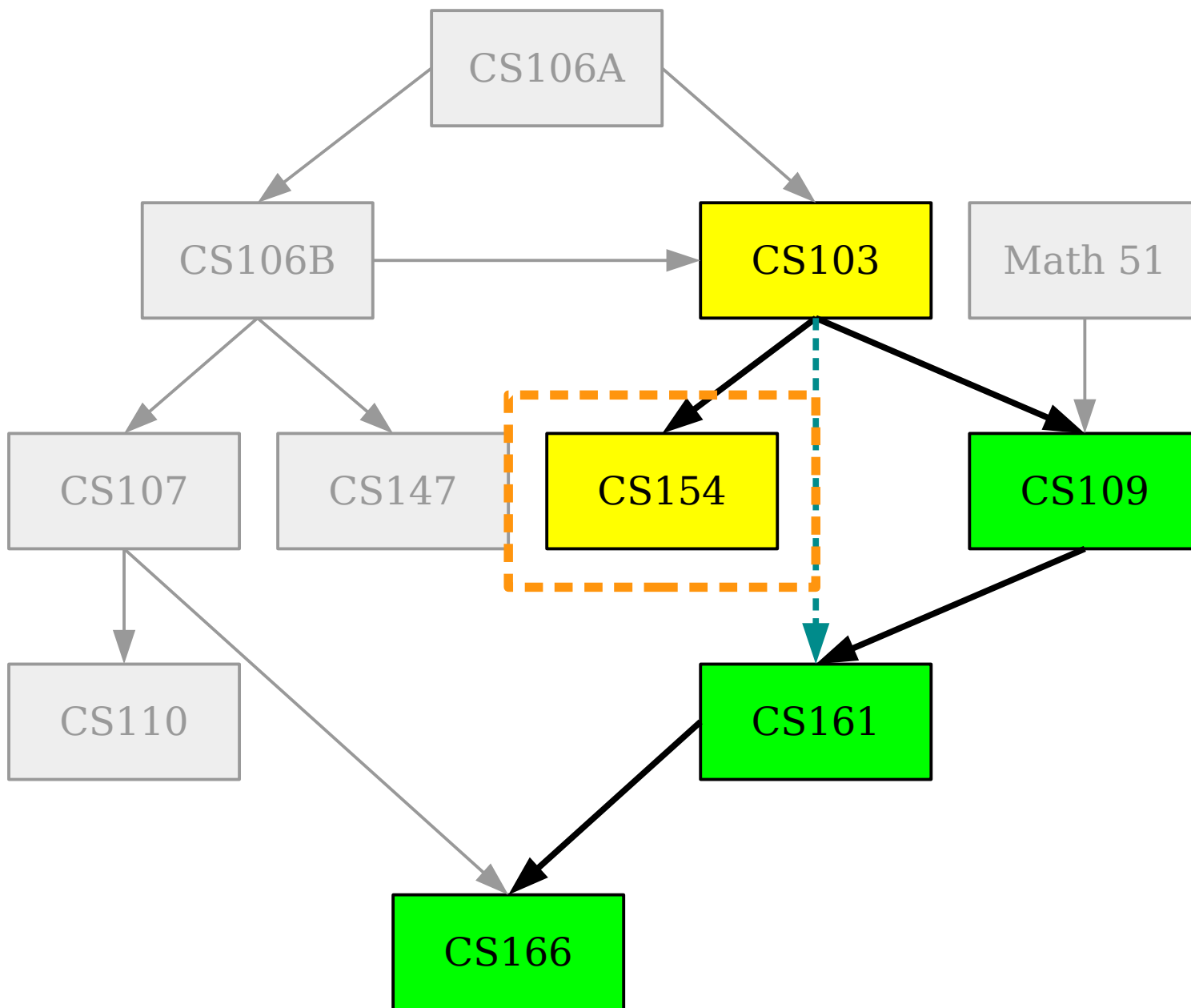
CS166
CS161
CS109



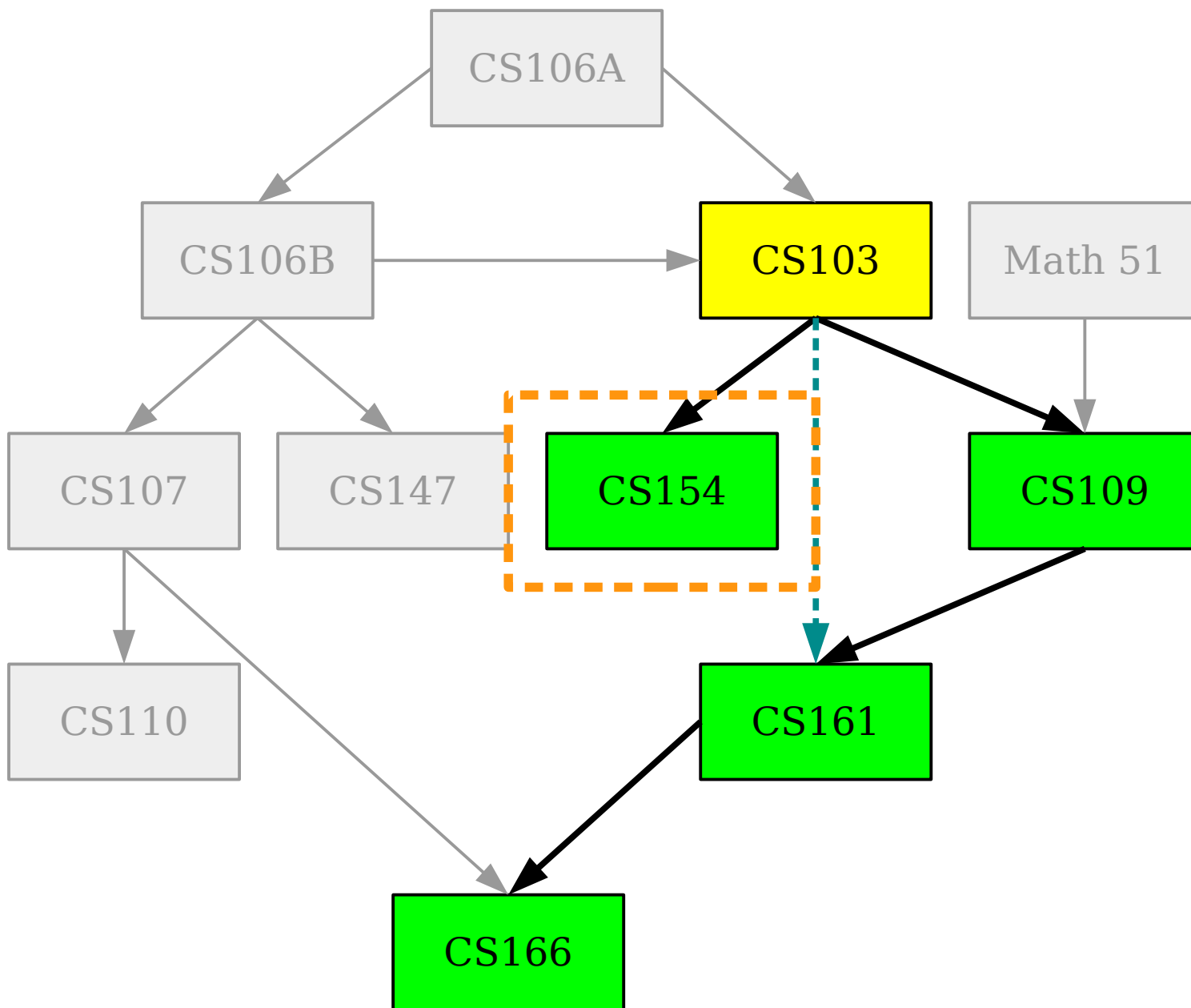
CS166
CS161
CS109



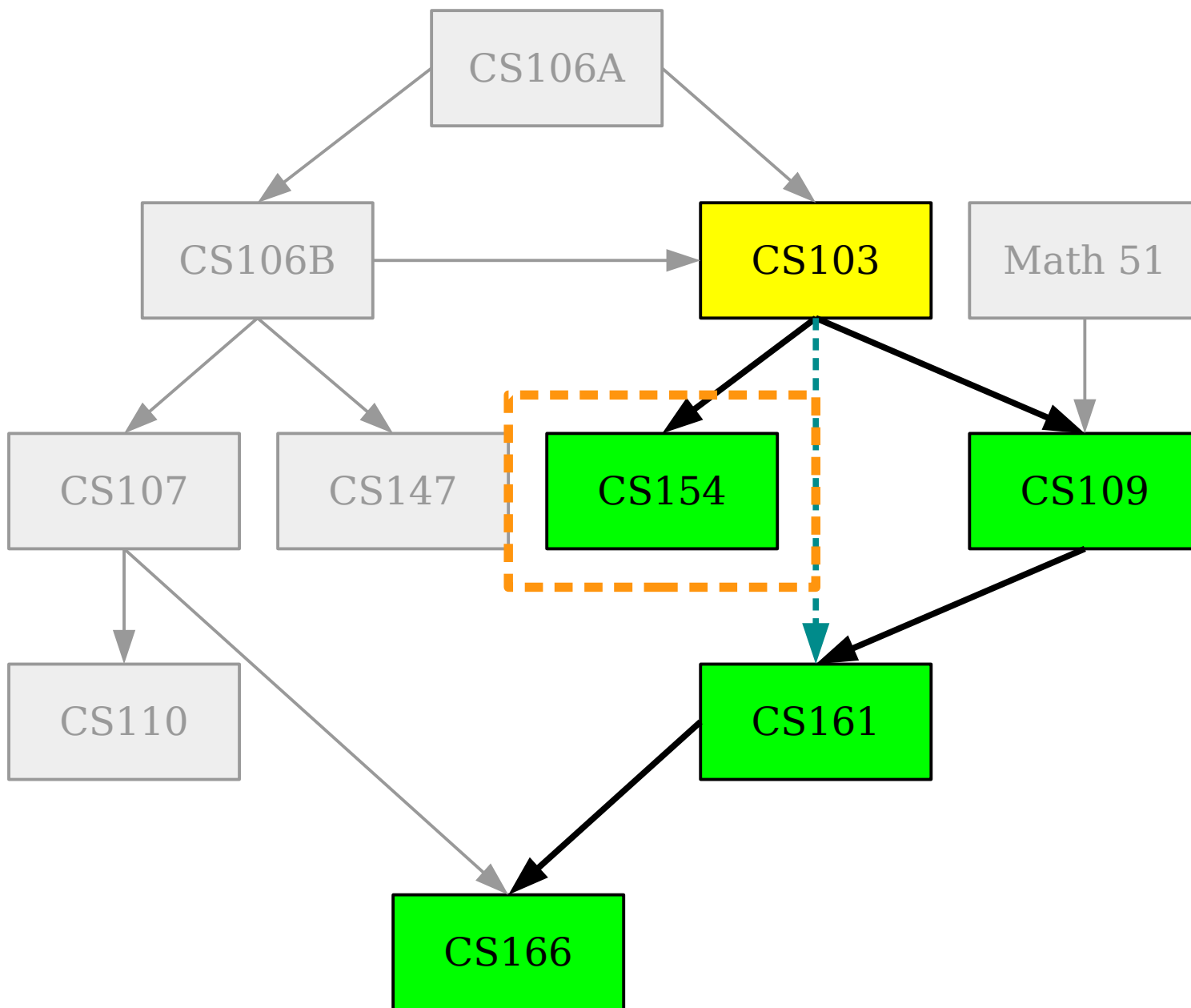
CS166
CS161
CS109



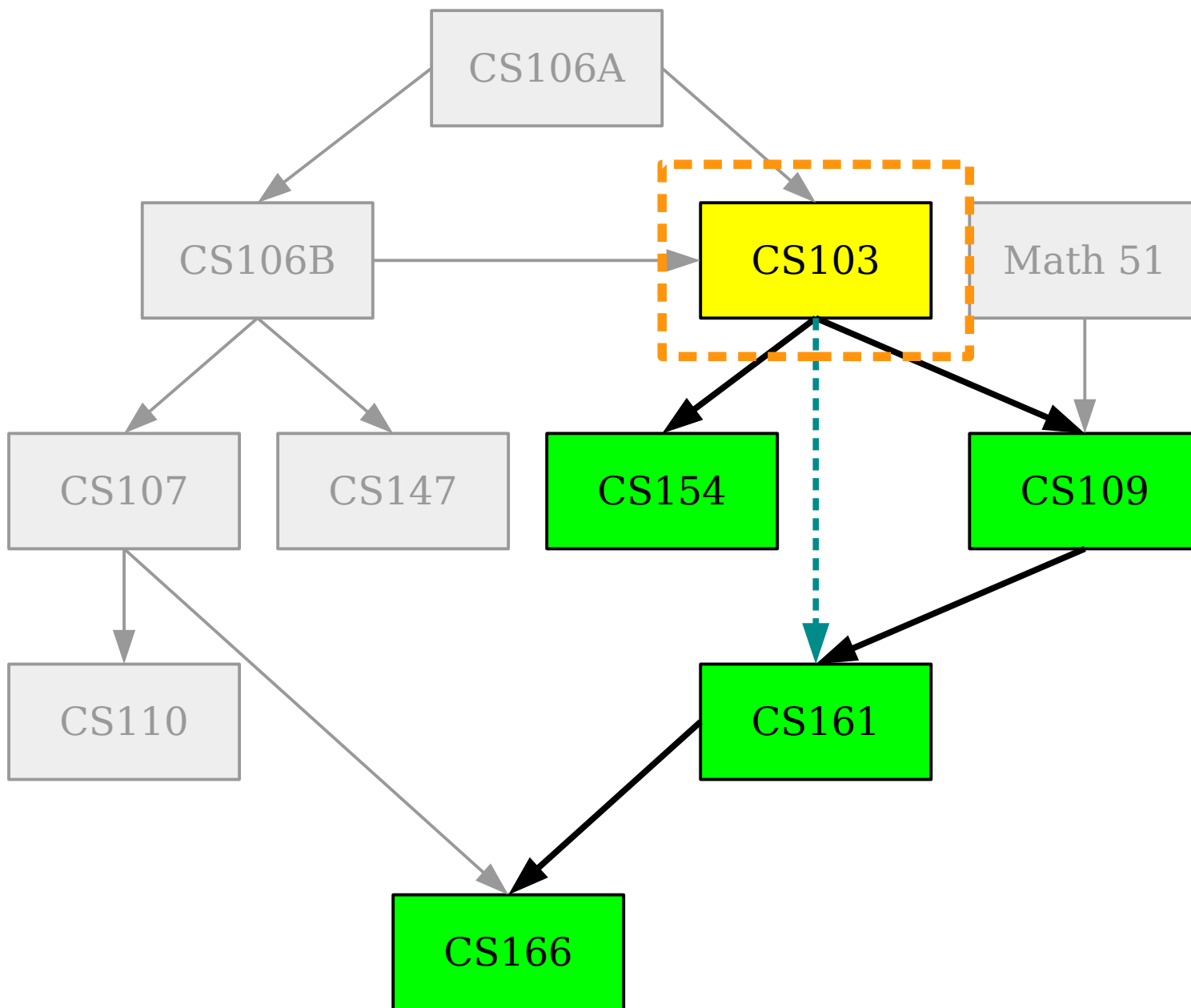
CS166
CS161
CS109



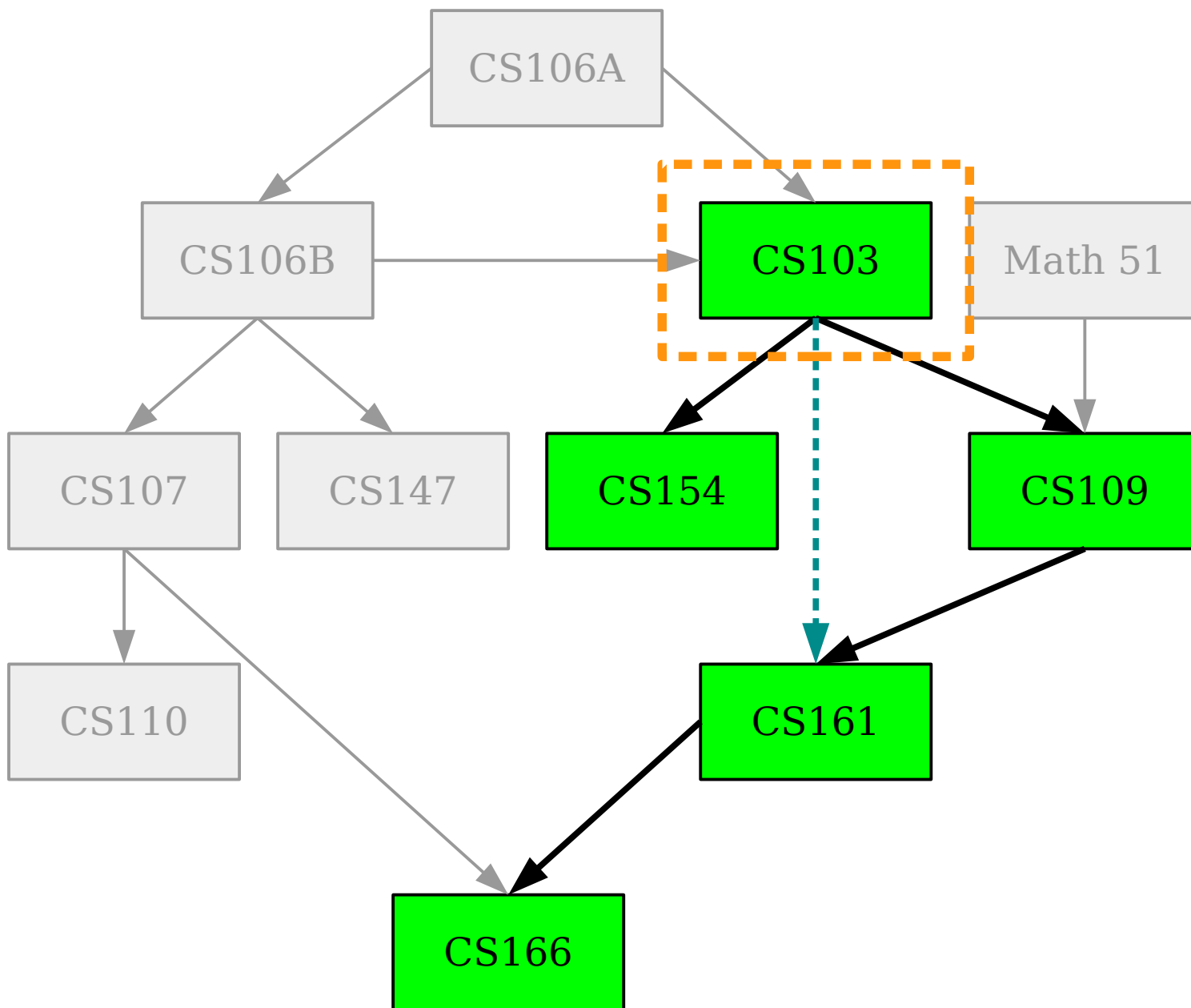
CS166
CS161
CS109



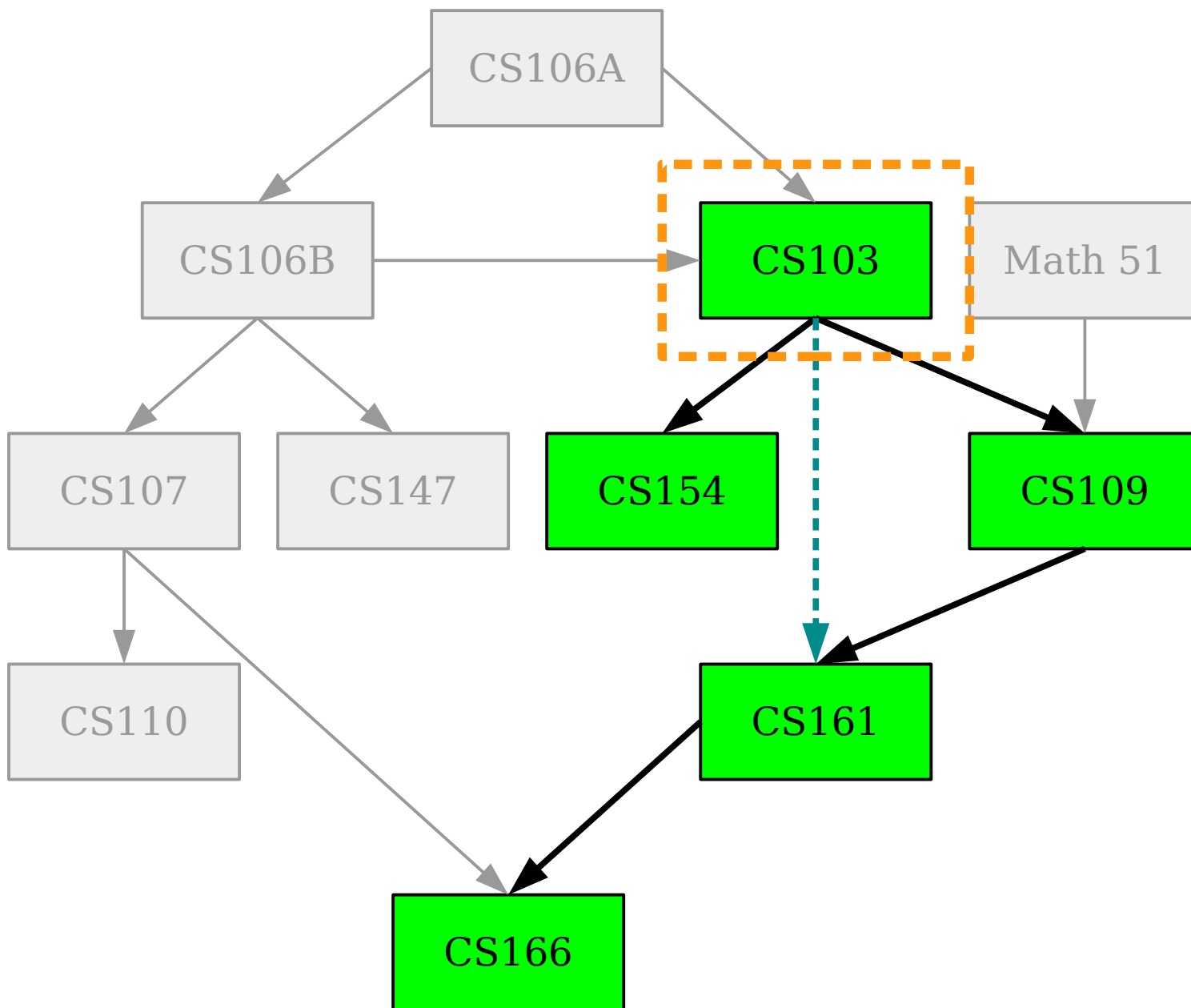
- CS166
- CS161
- CS109
- CS154



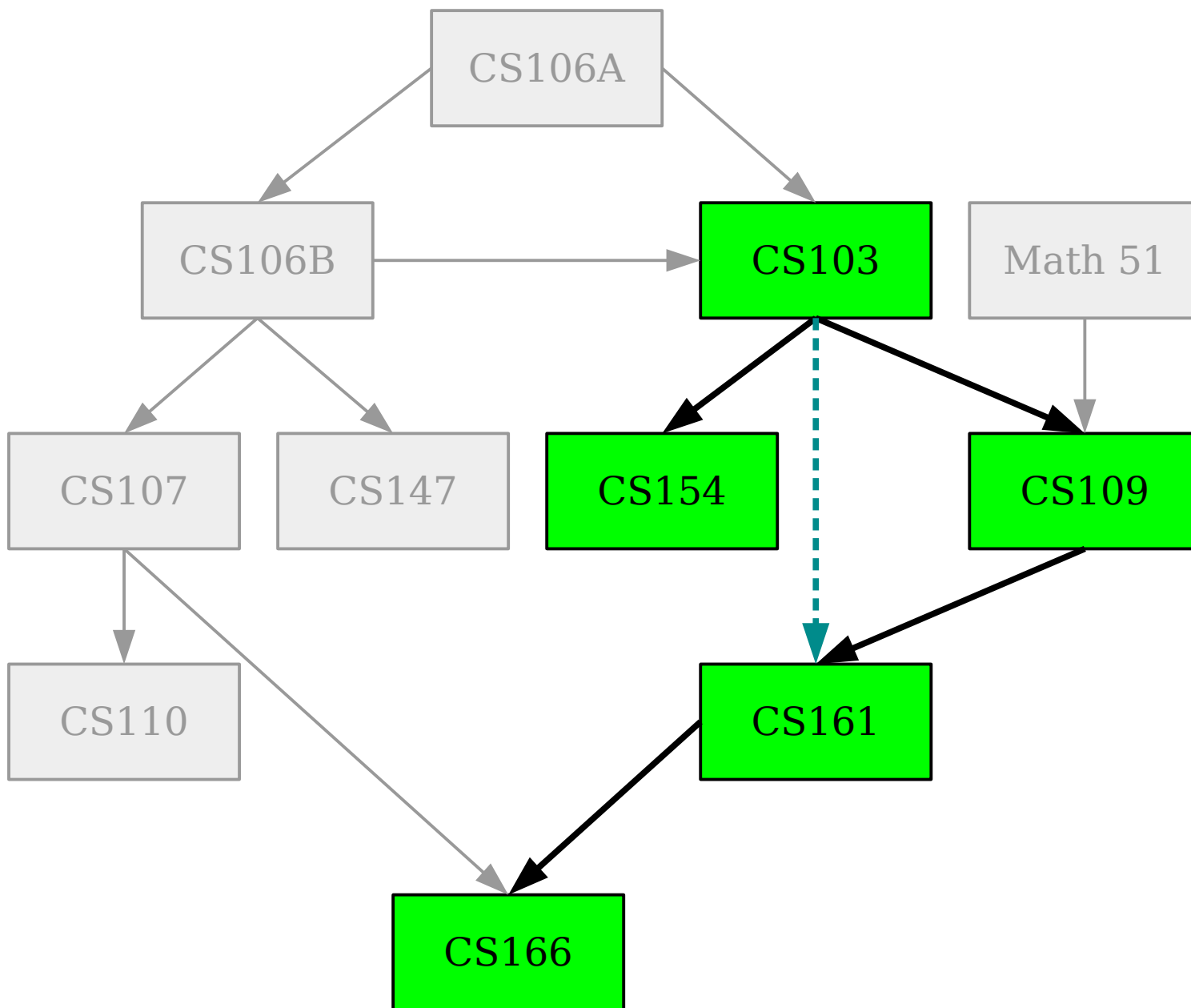
CS166
CS161
CS109
CS154



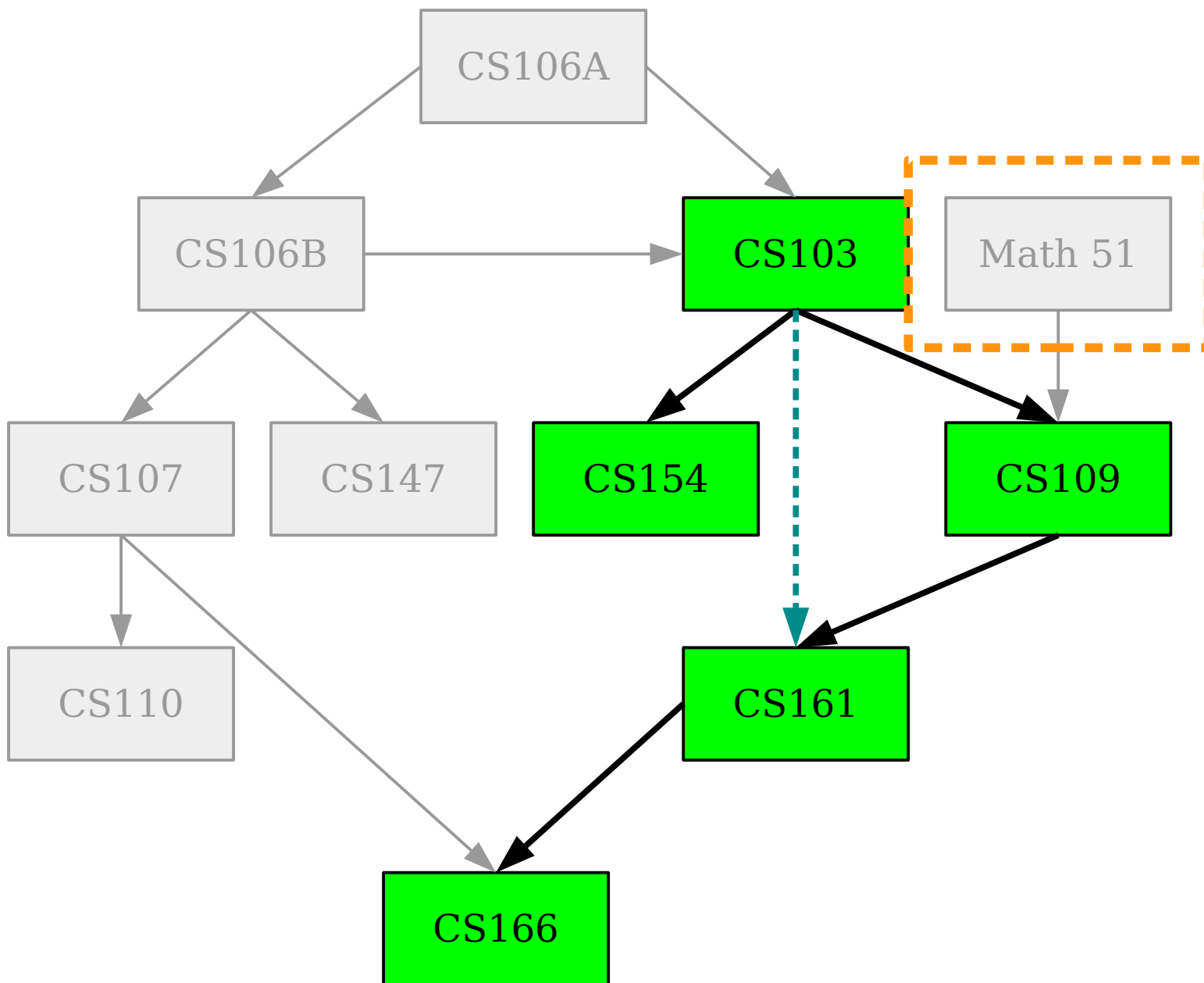
CS166
CS161
CS109
CS154



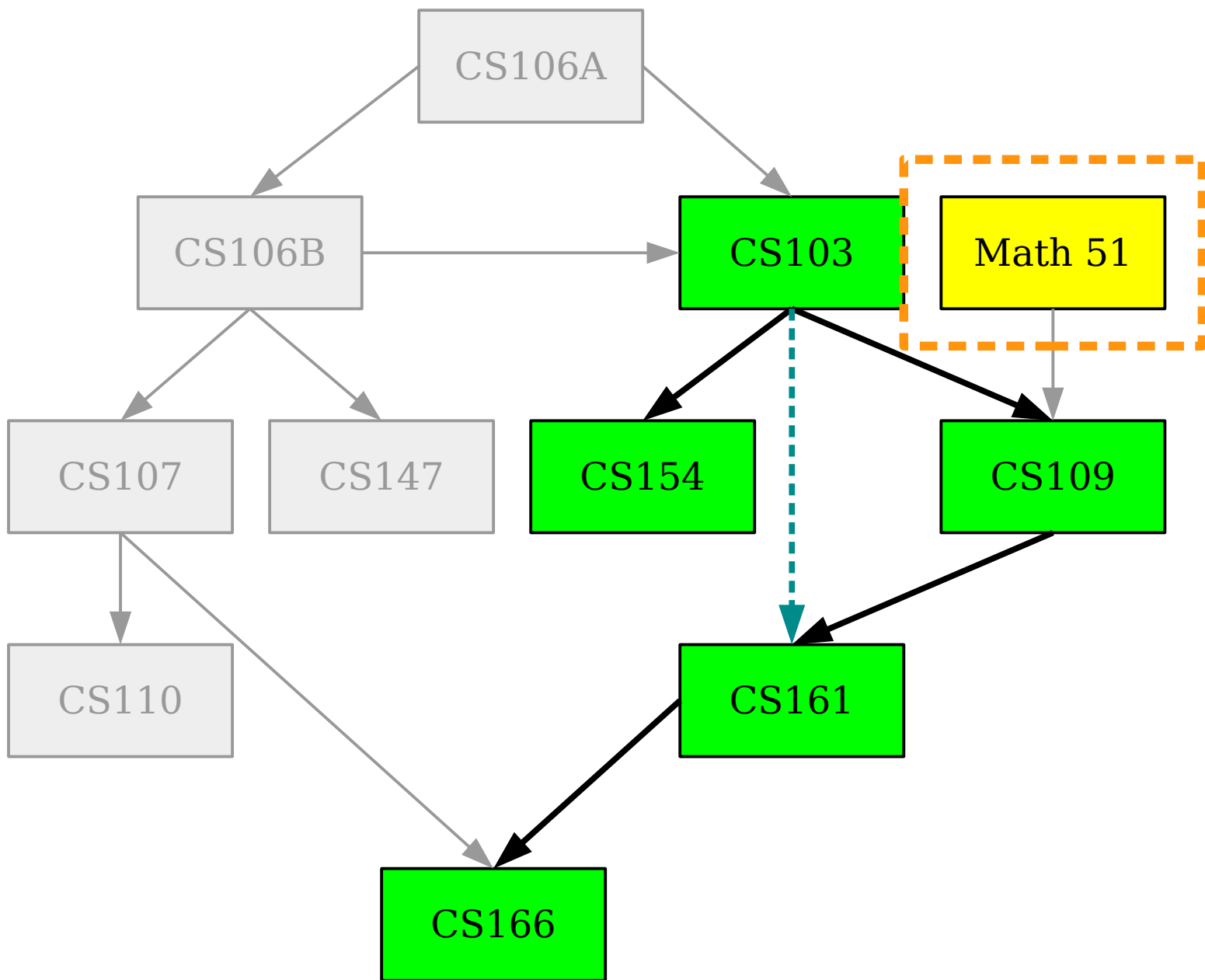
CS166
CS161
CS109
CS154
CS103



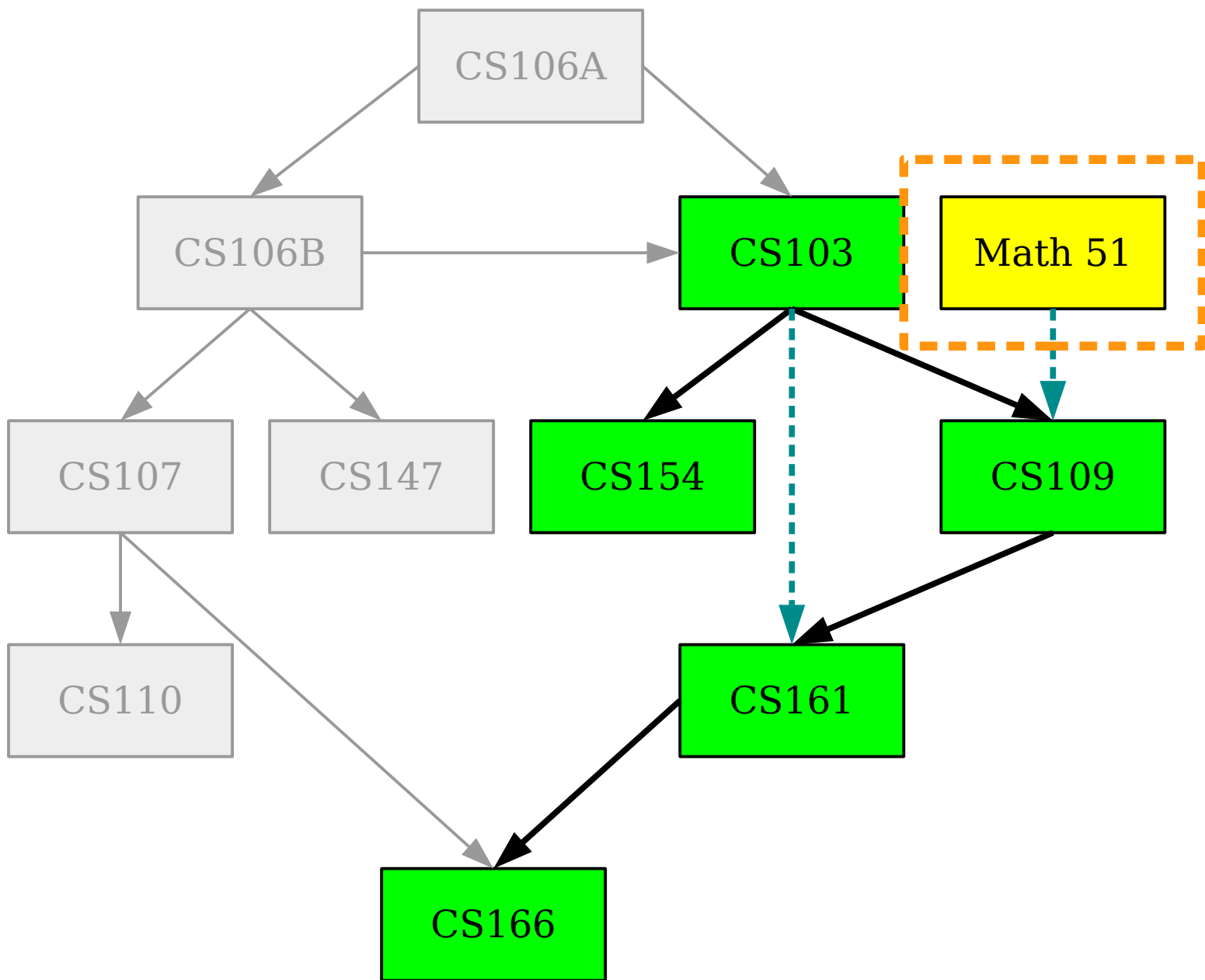
CS166
CS161
CS109
CS154
CS103



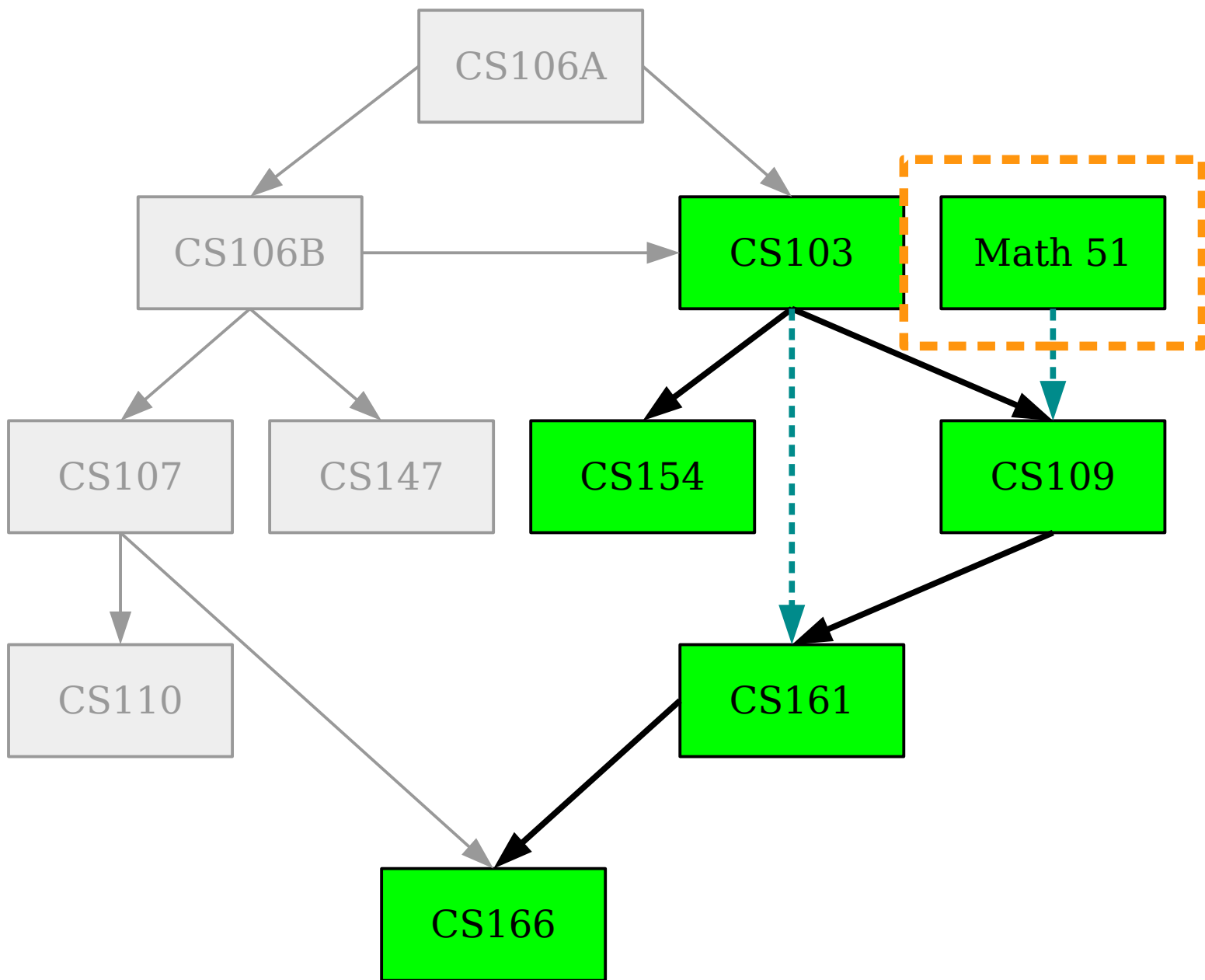
CS166
CS161
CS109
CS154
CS103



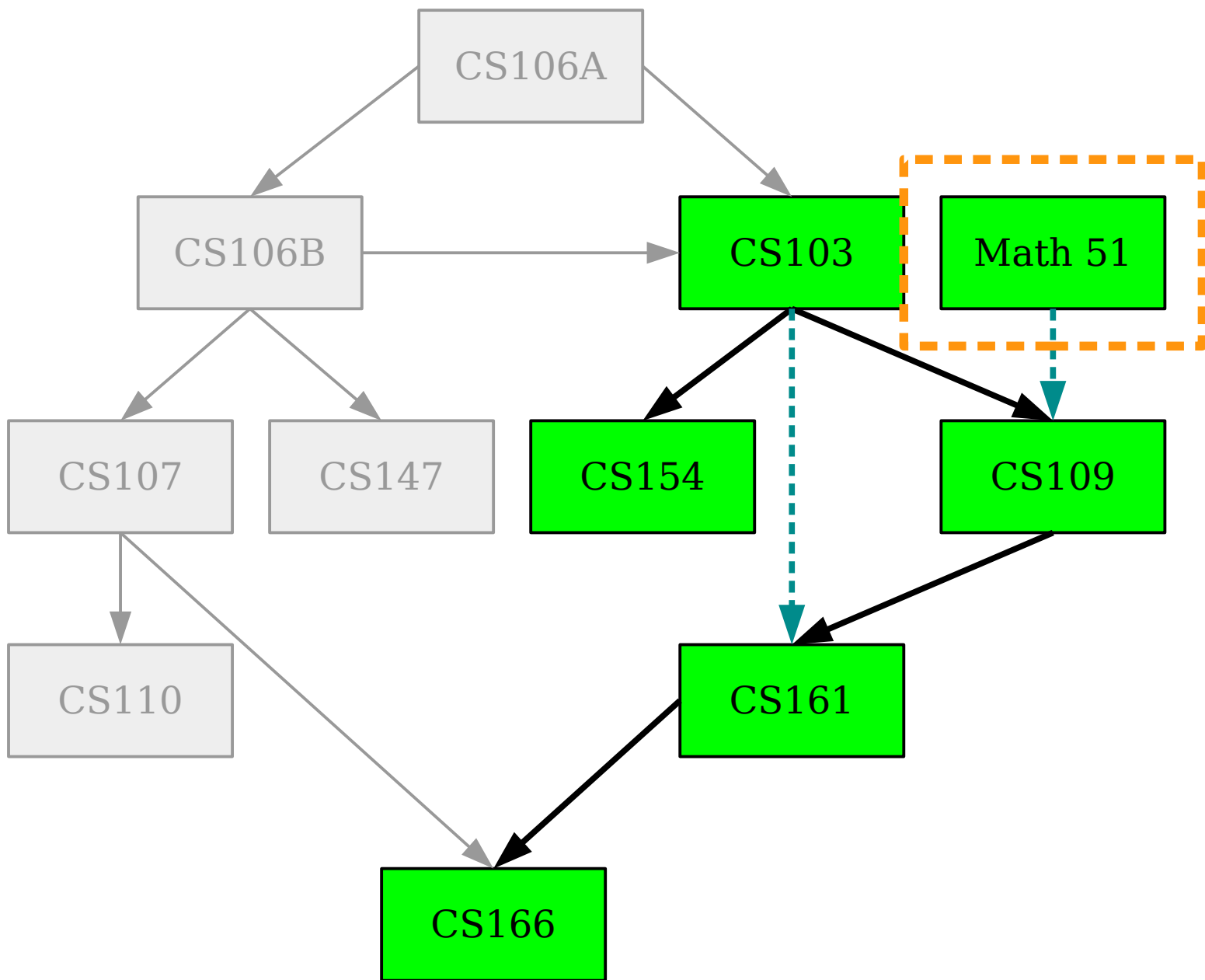
CS166
CS161
CS109
CS154
CS103



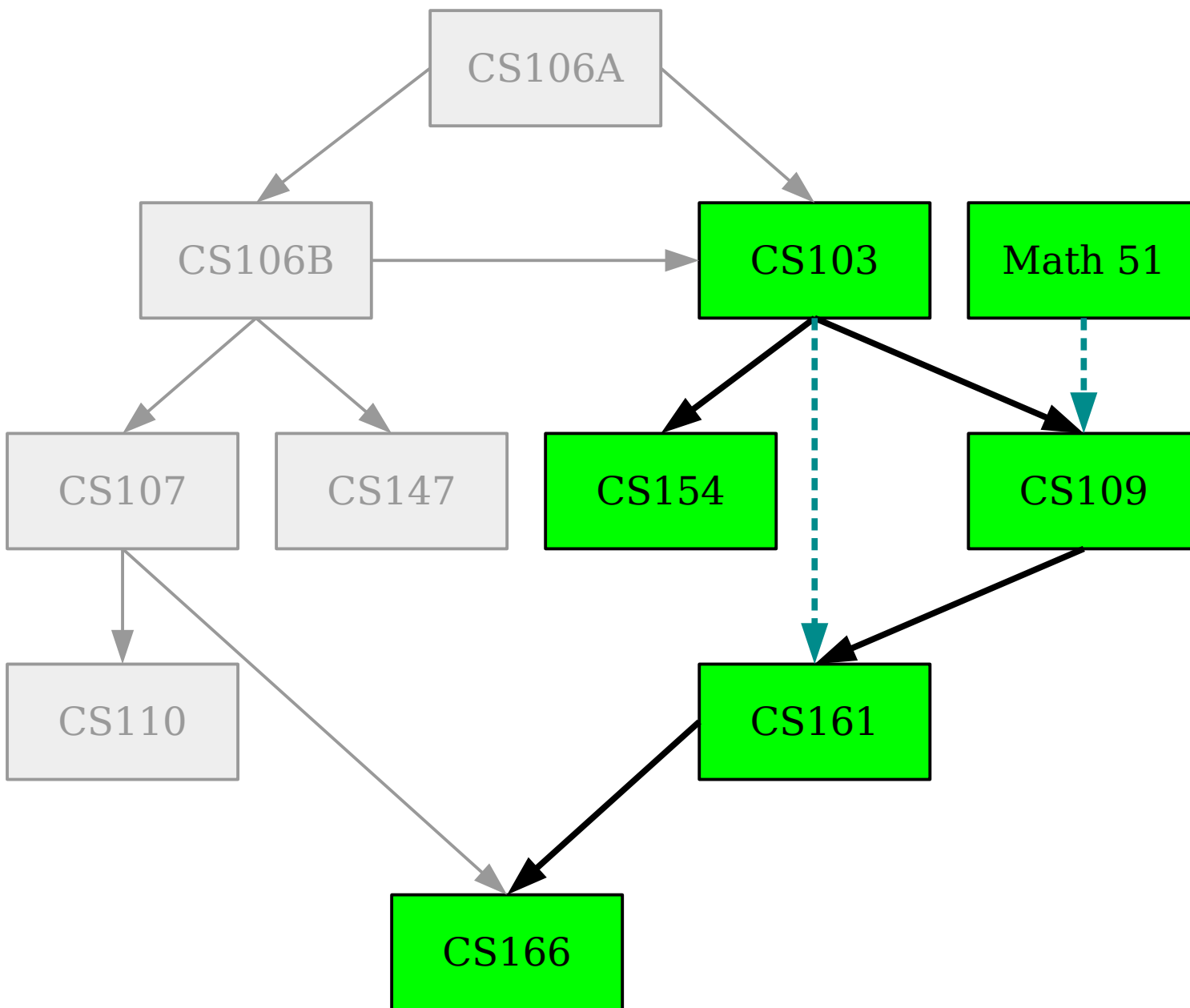
CS166
CS161
CS109
CS154
CS103

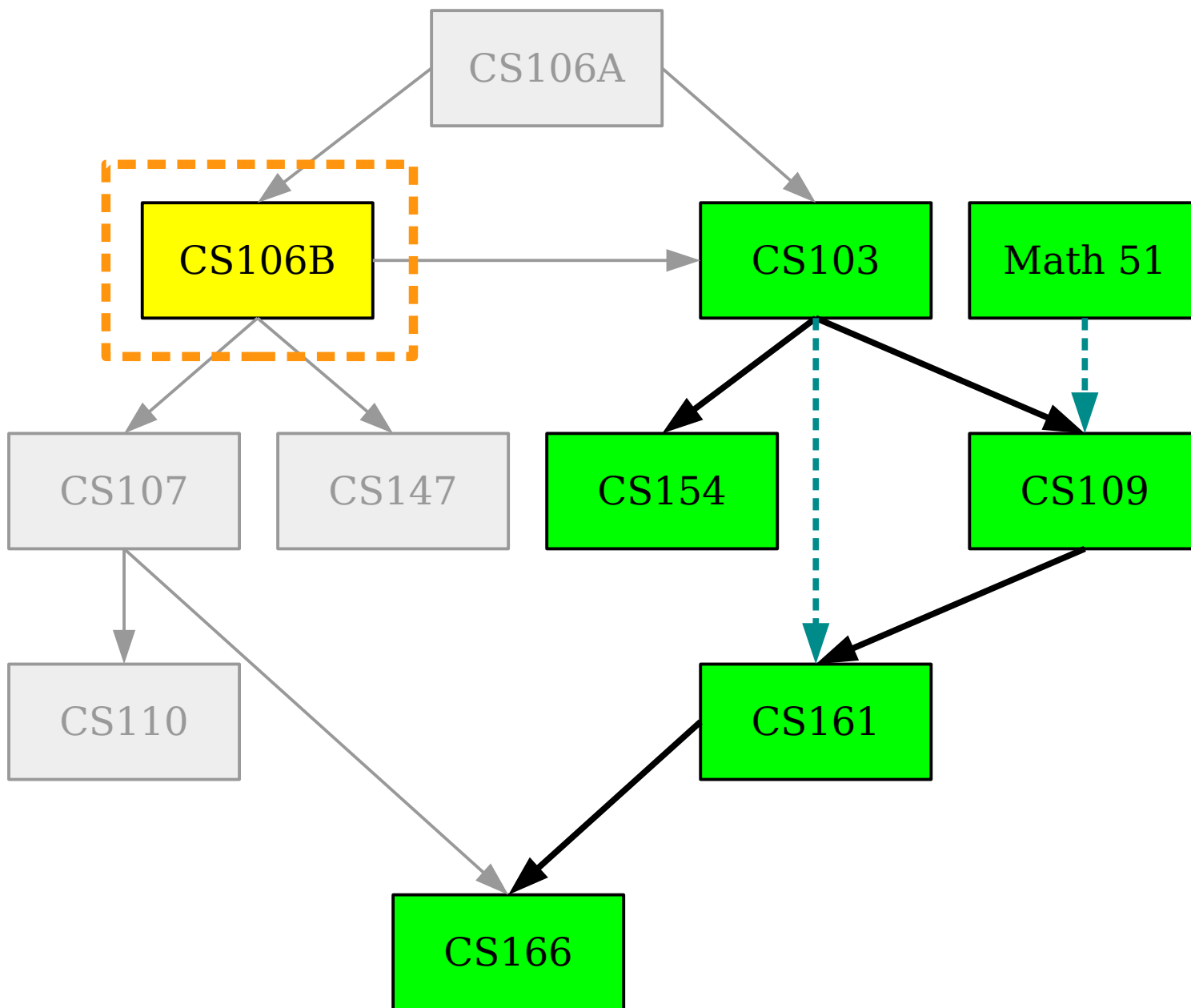


CS166
CS161
CS109
CS154
CS103

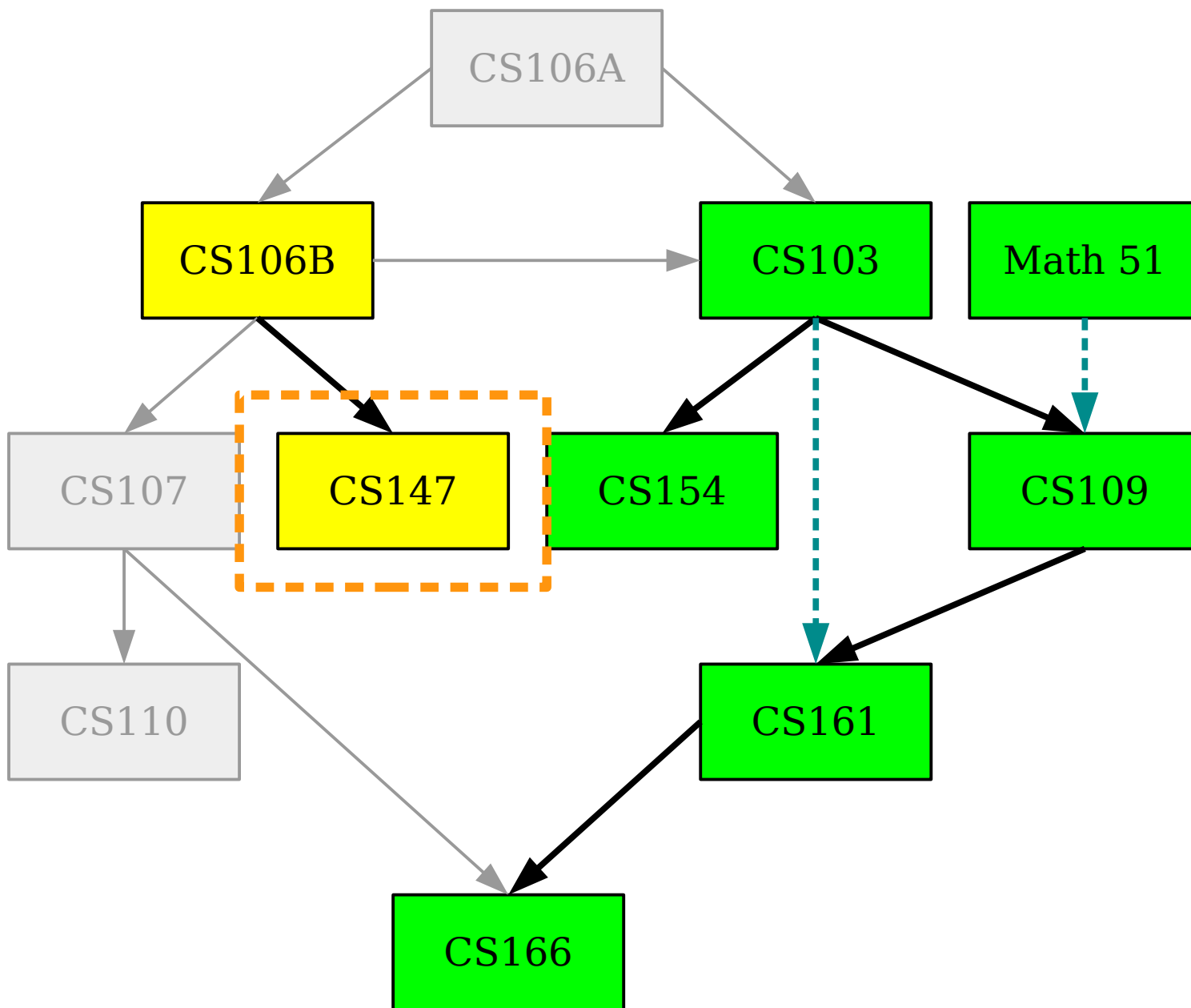


CS166
CS161
CS109
CS154
CS103
Math 51

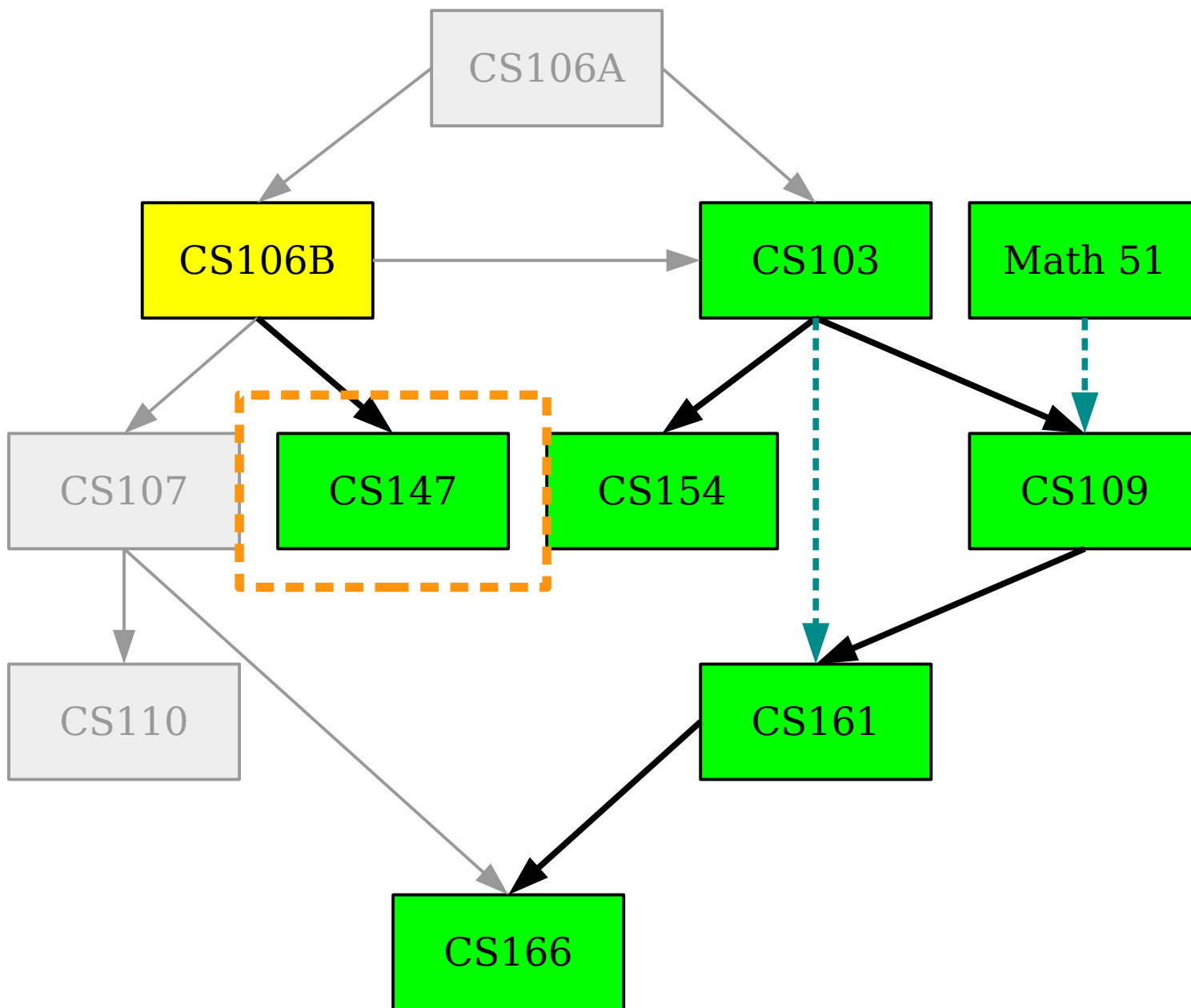




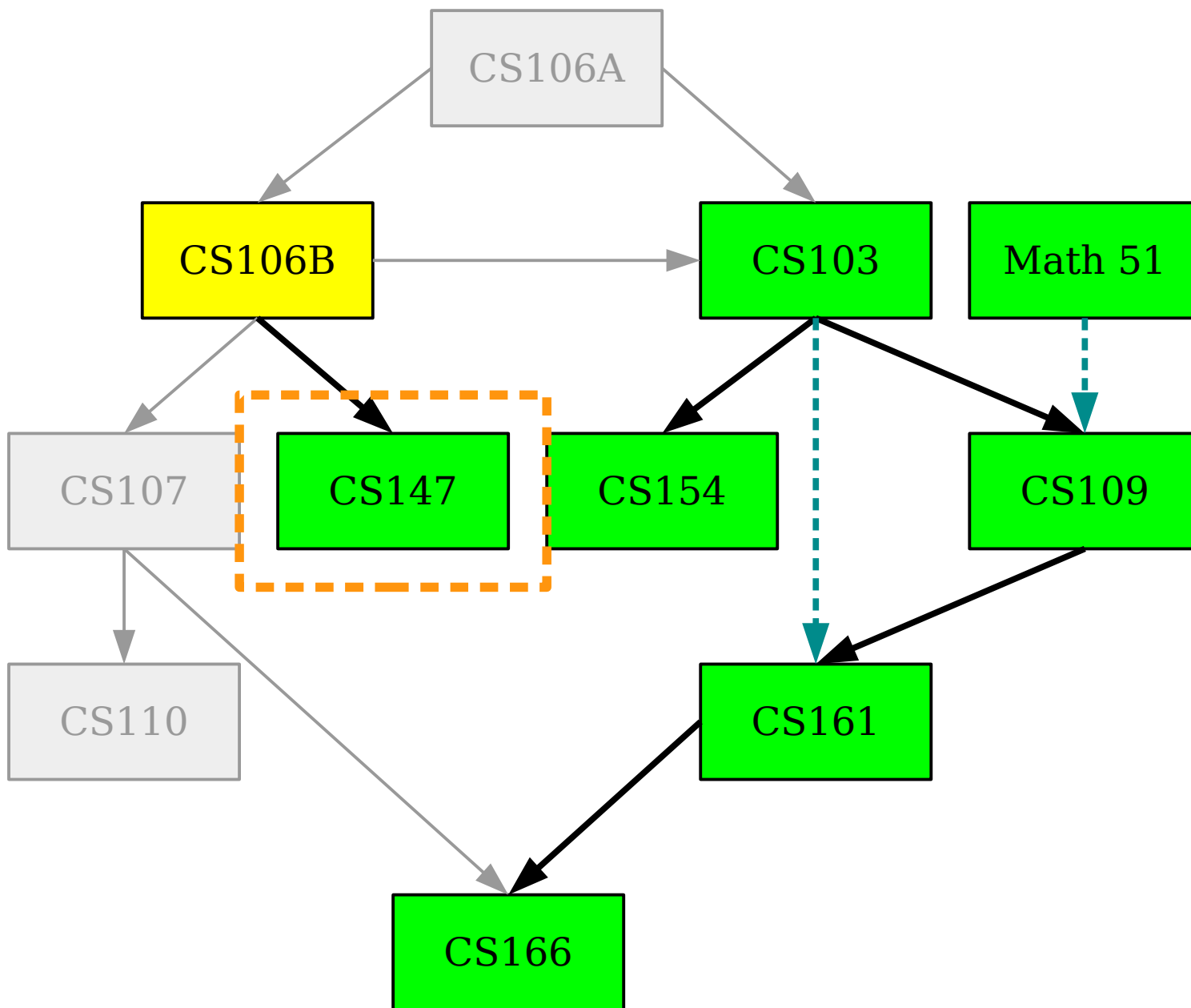
CS166
CS161
CS109
CS154
CS103
Math 51



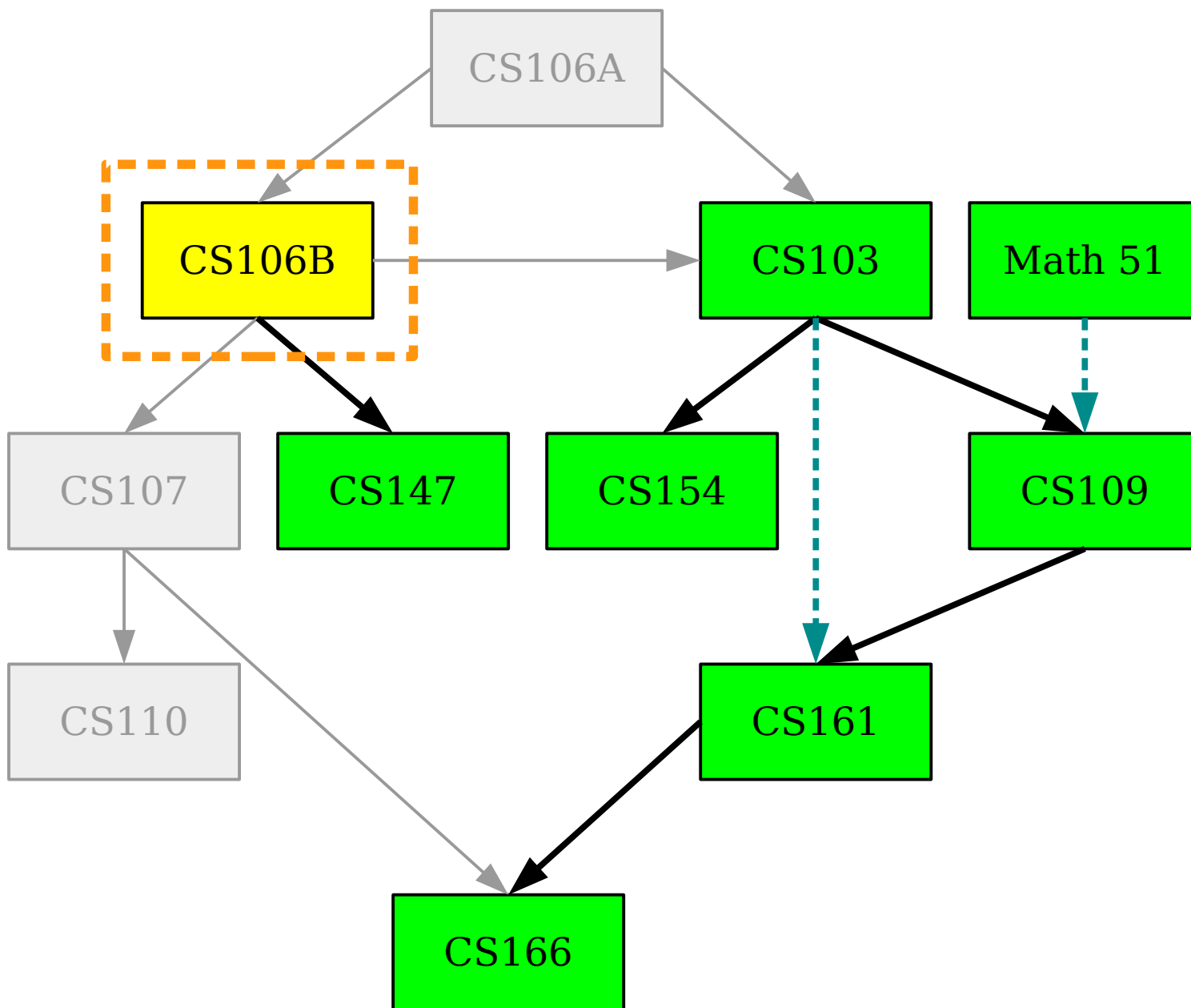
CS166
CS161
CS109
CS154
CS103
Math 51



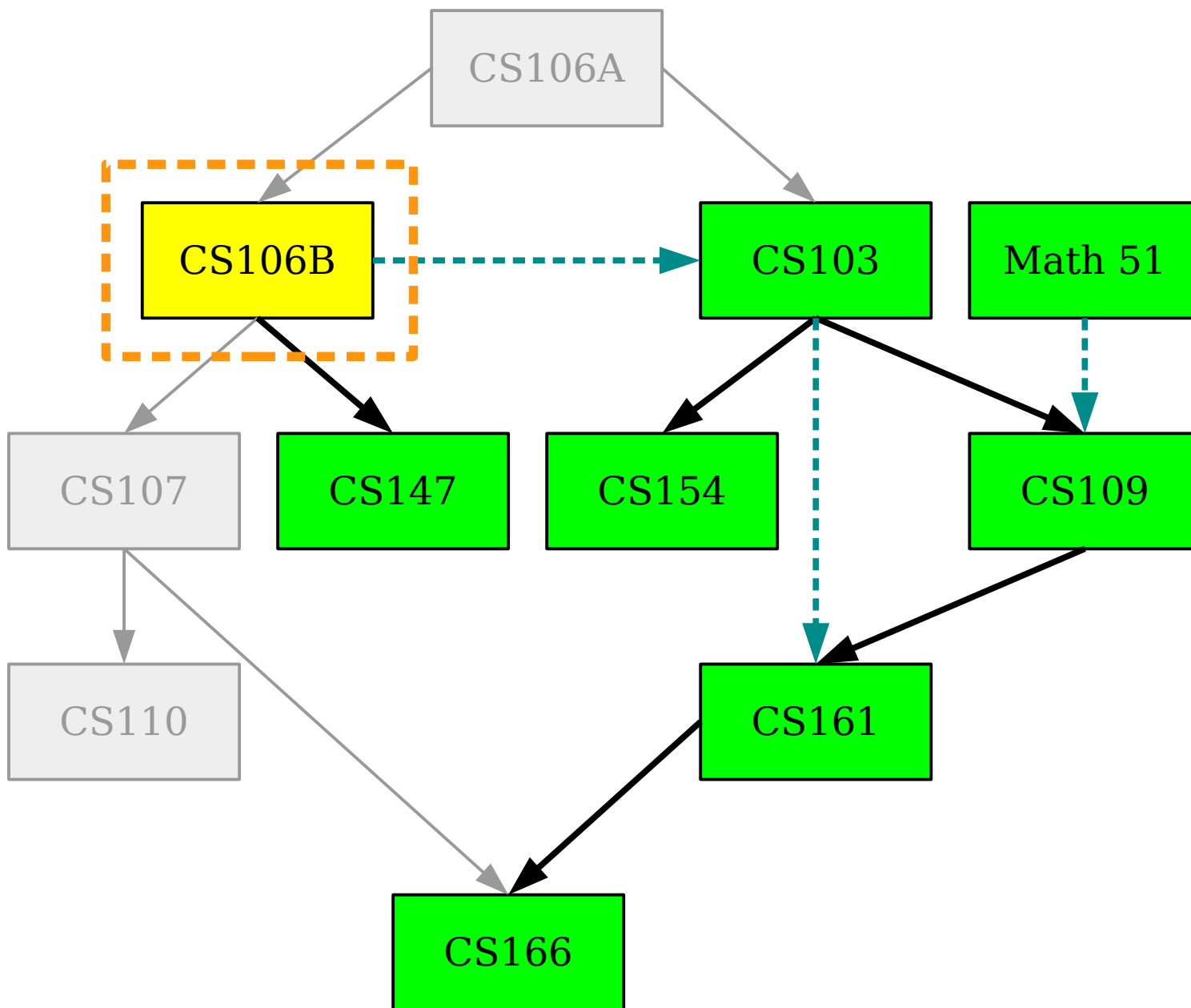
CS166
CS161
CS109
CS154
CS103
Math 51



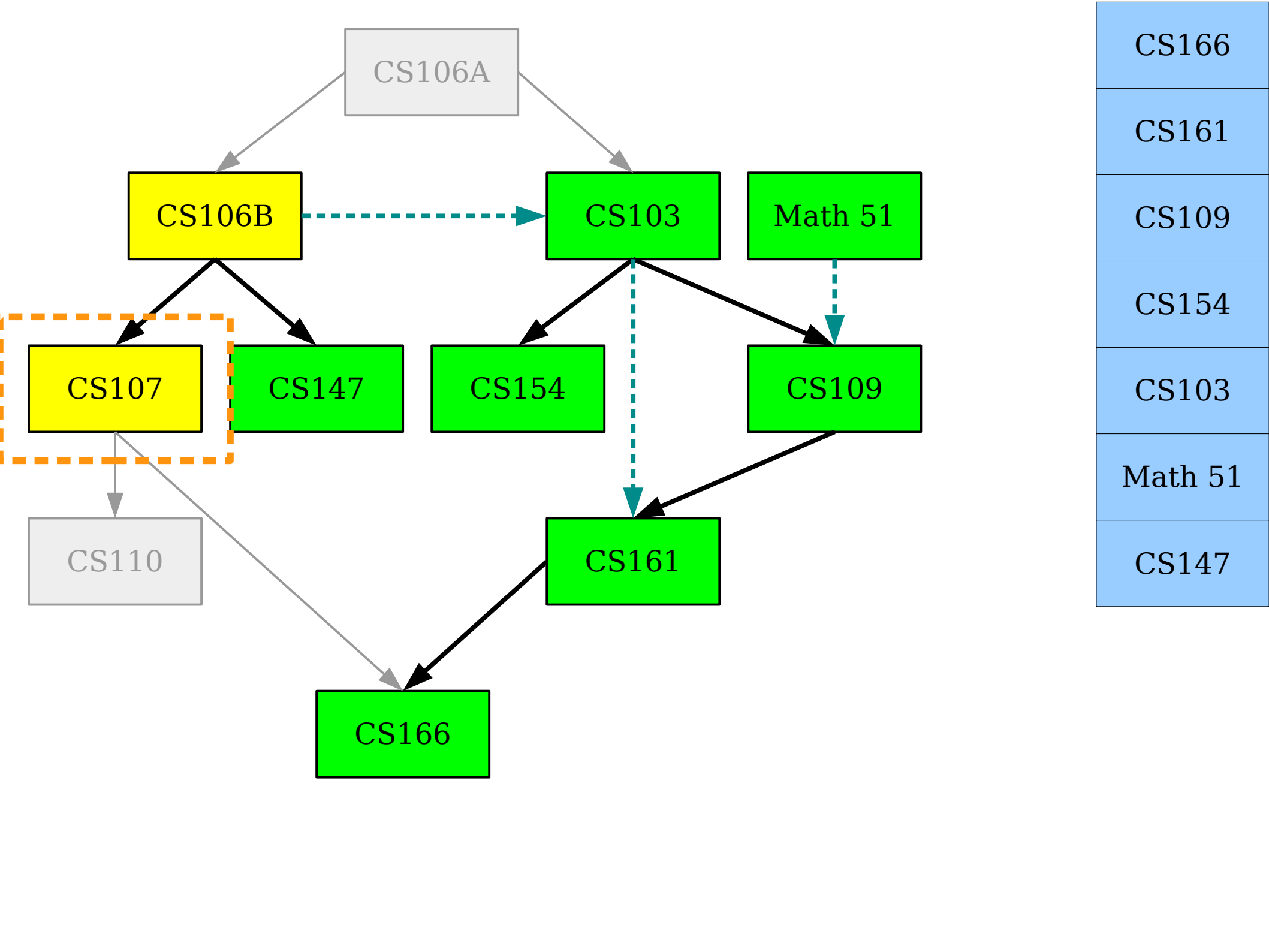
CS166
CS161
CS109
CS154
CS103
Math 51
CS147

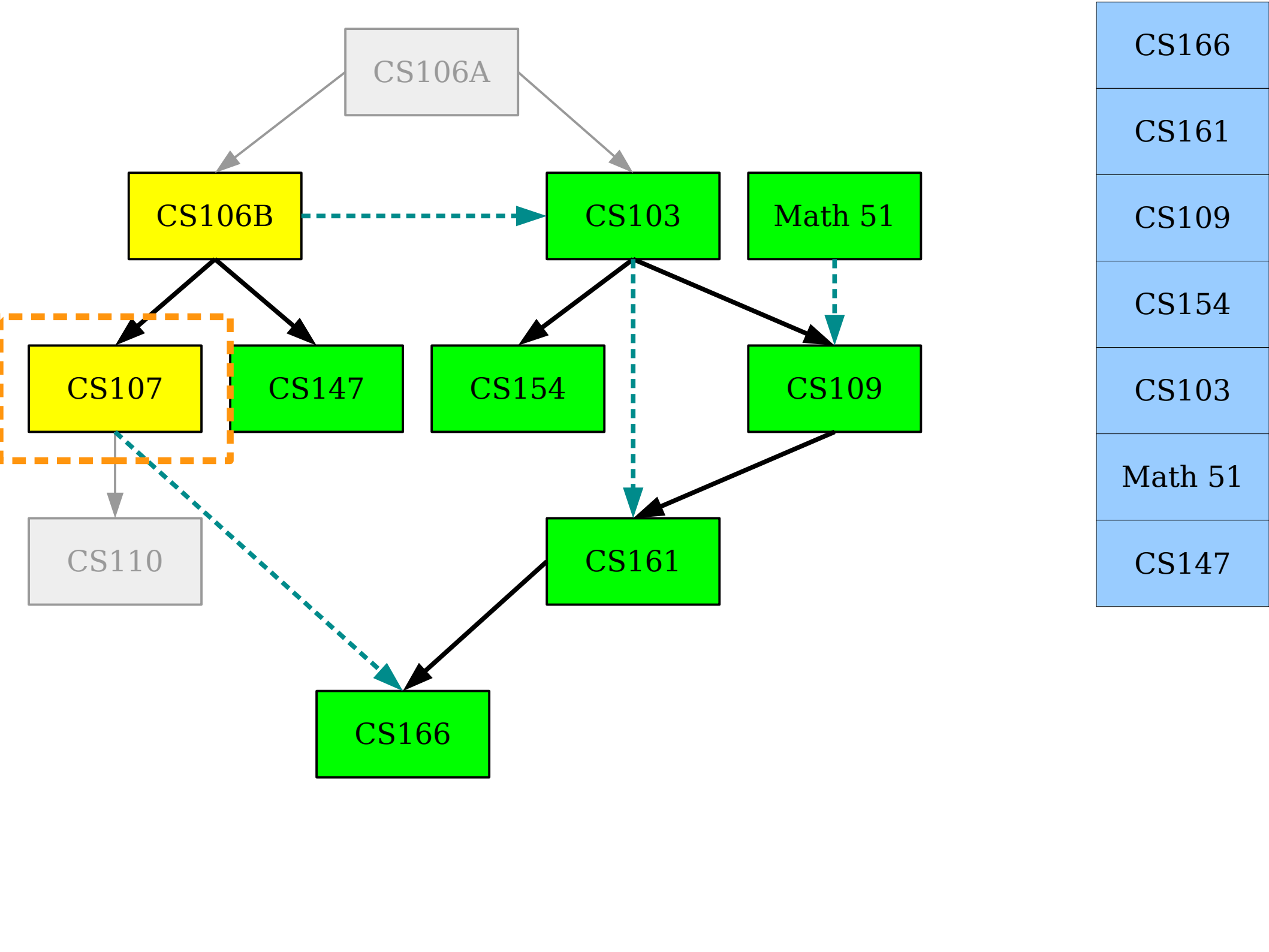


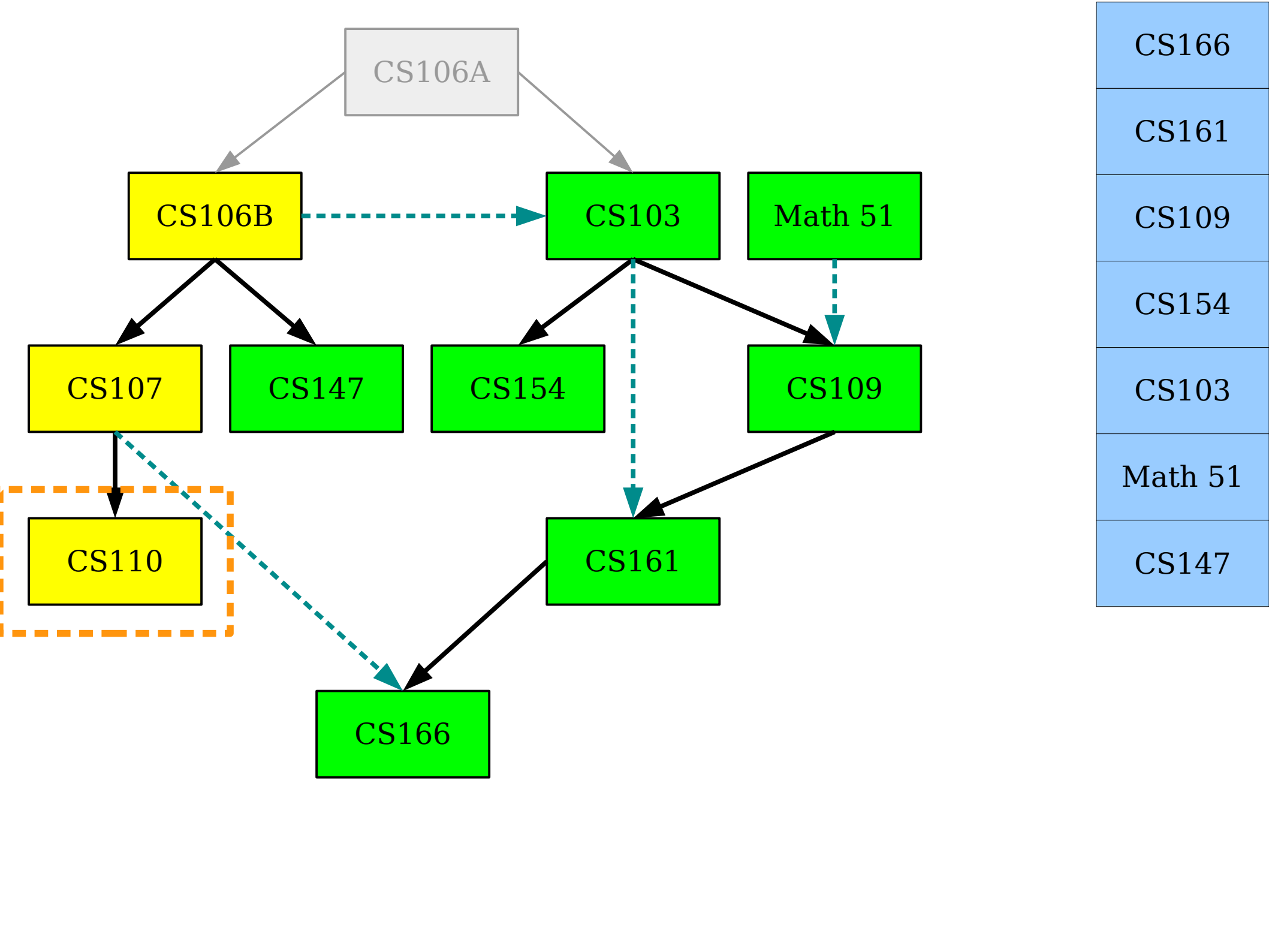
CS166
CS161
CS109
CS154
CS103
Math 51
CS147

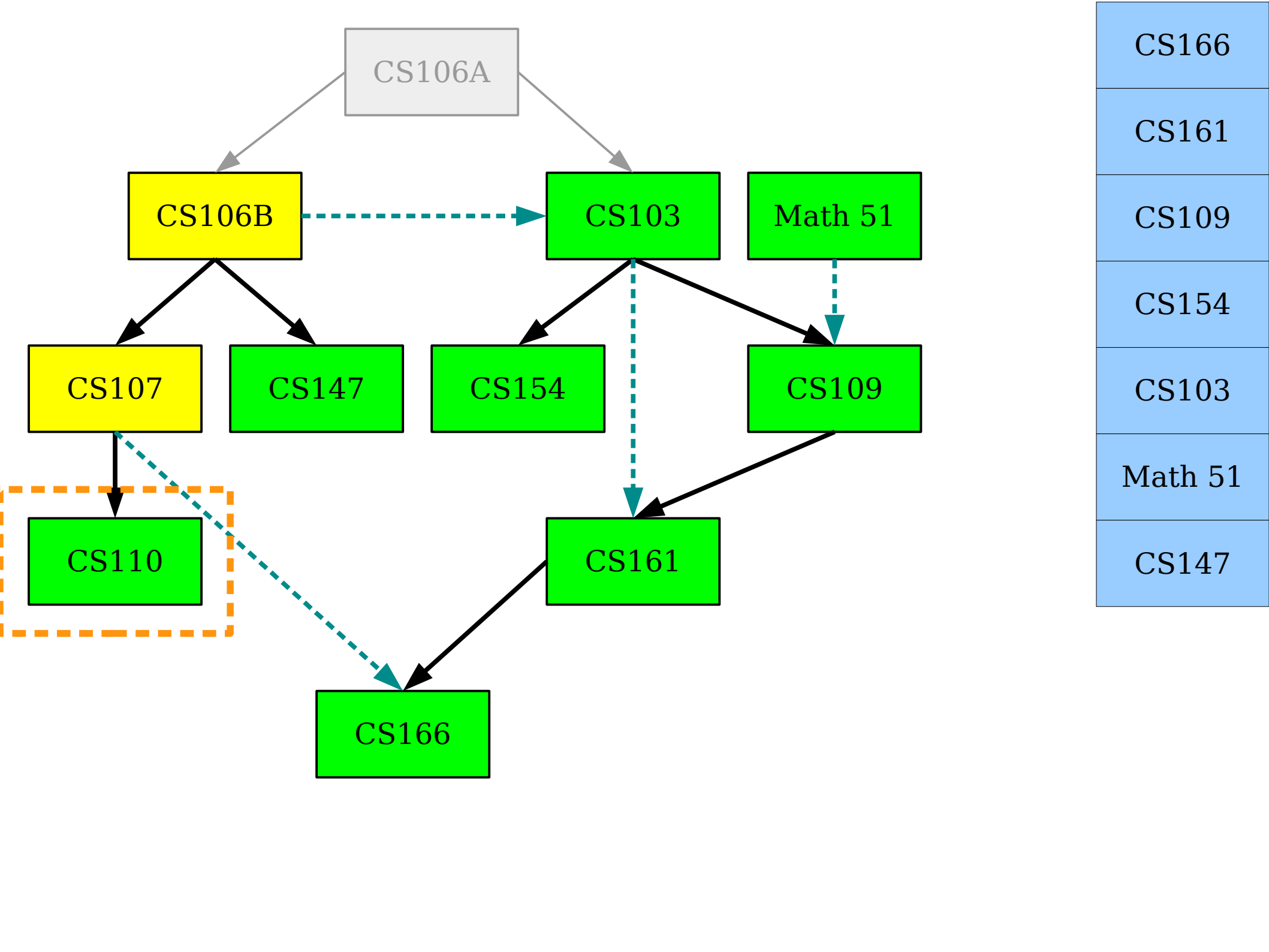


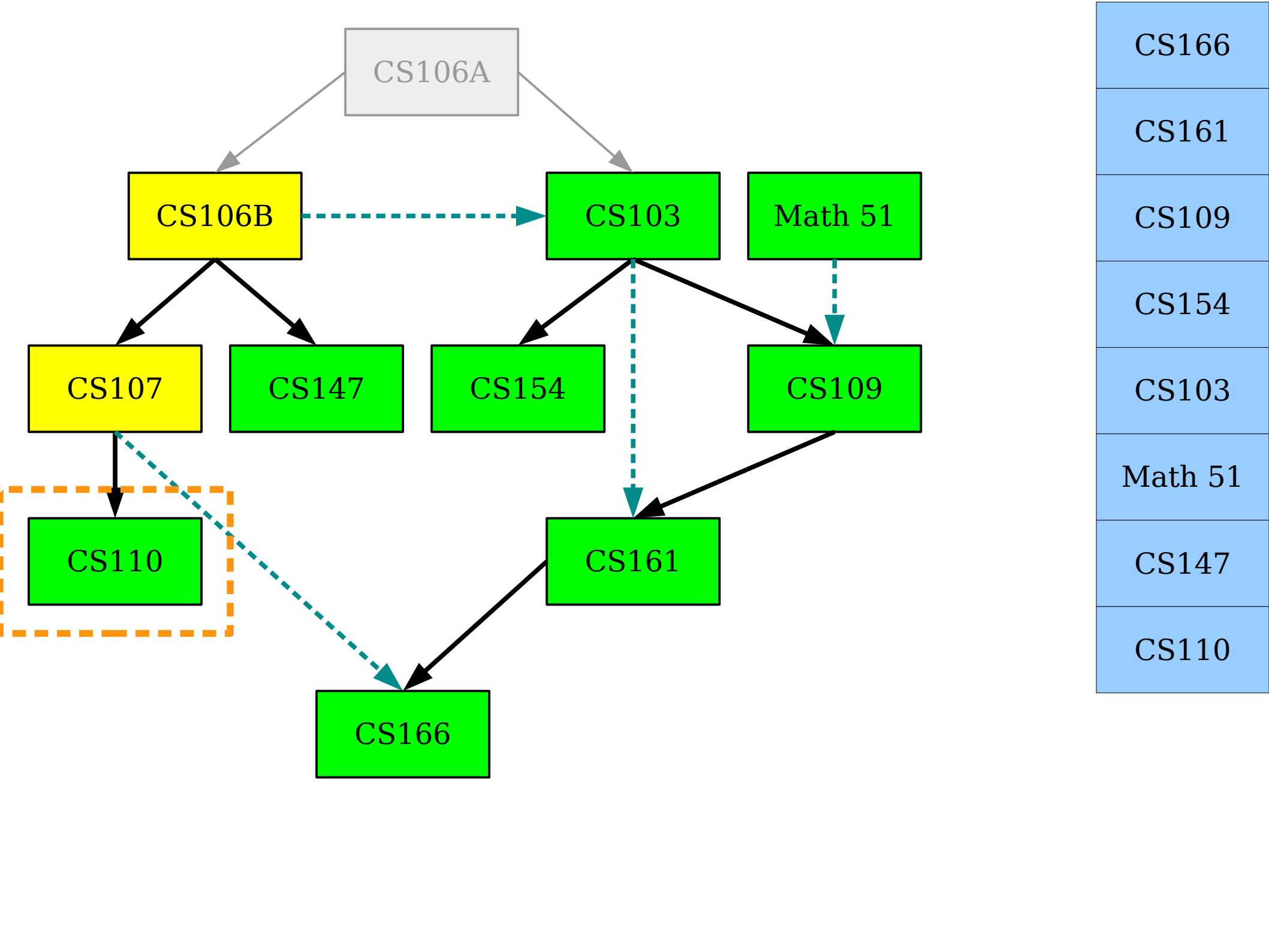
CS166
CS161
CS109
CS154
CS103
Math 51
CS147



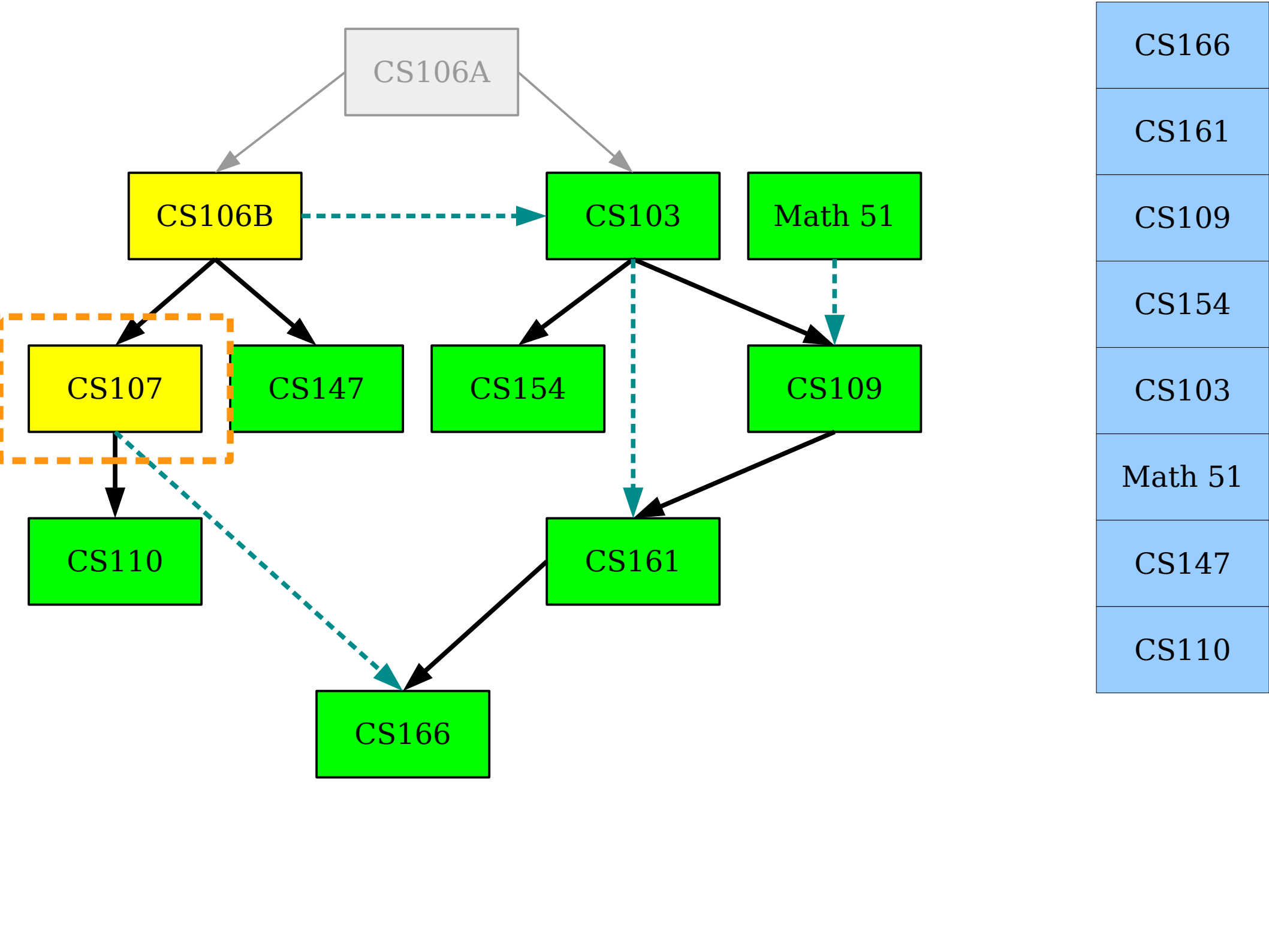


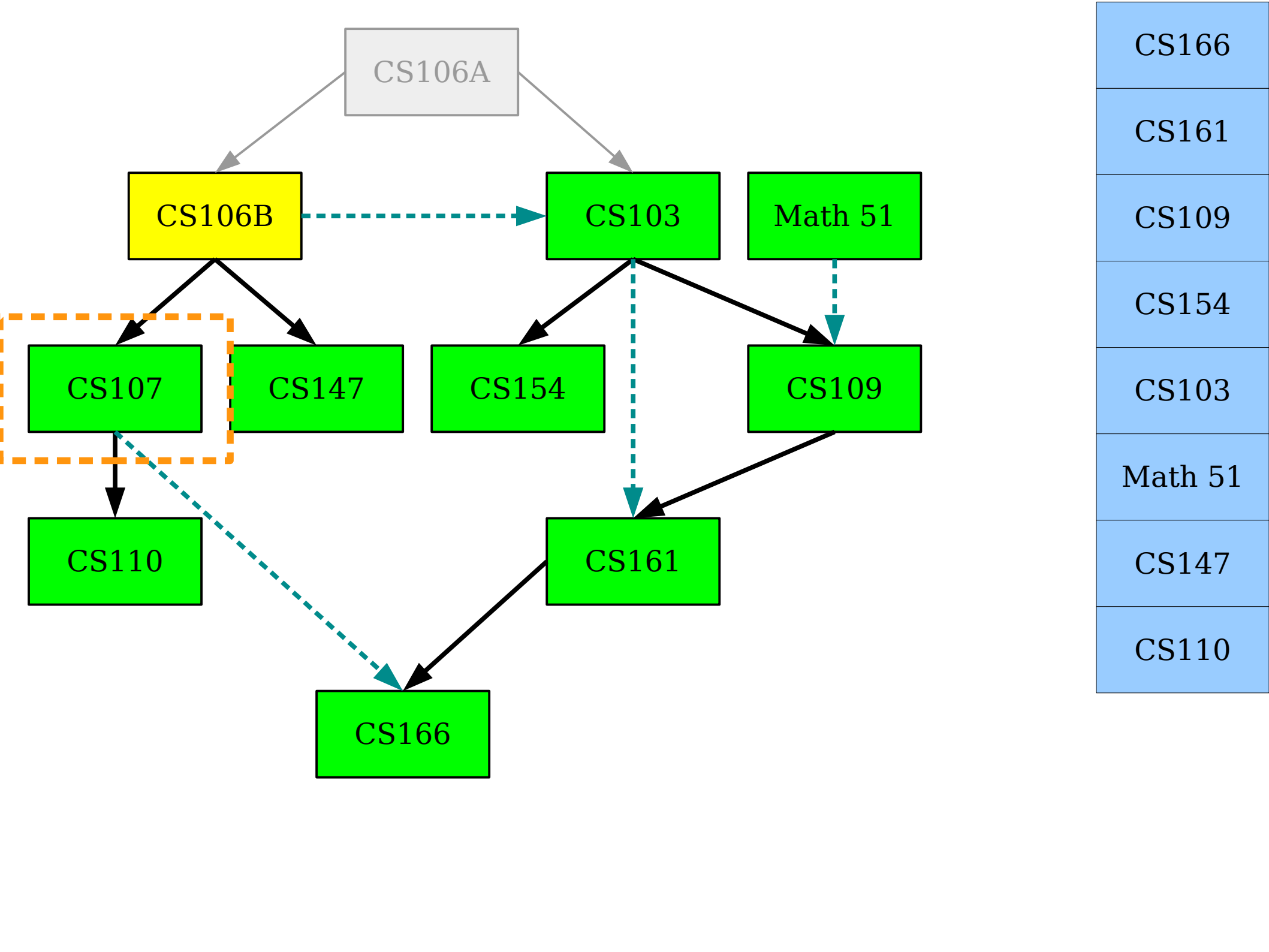


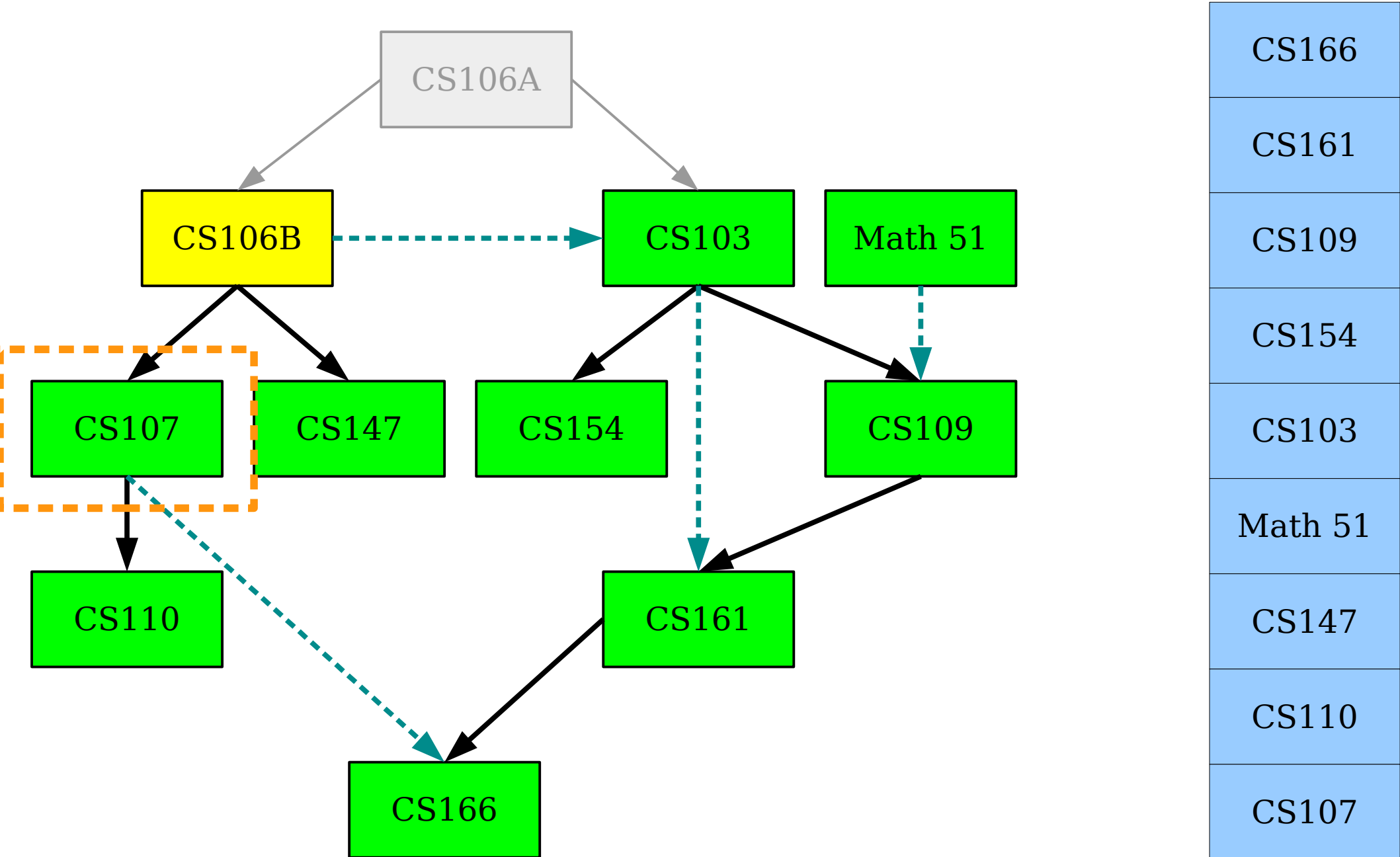


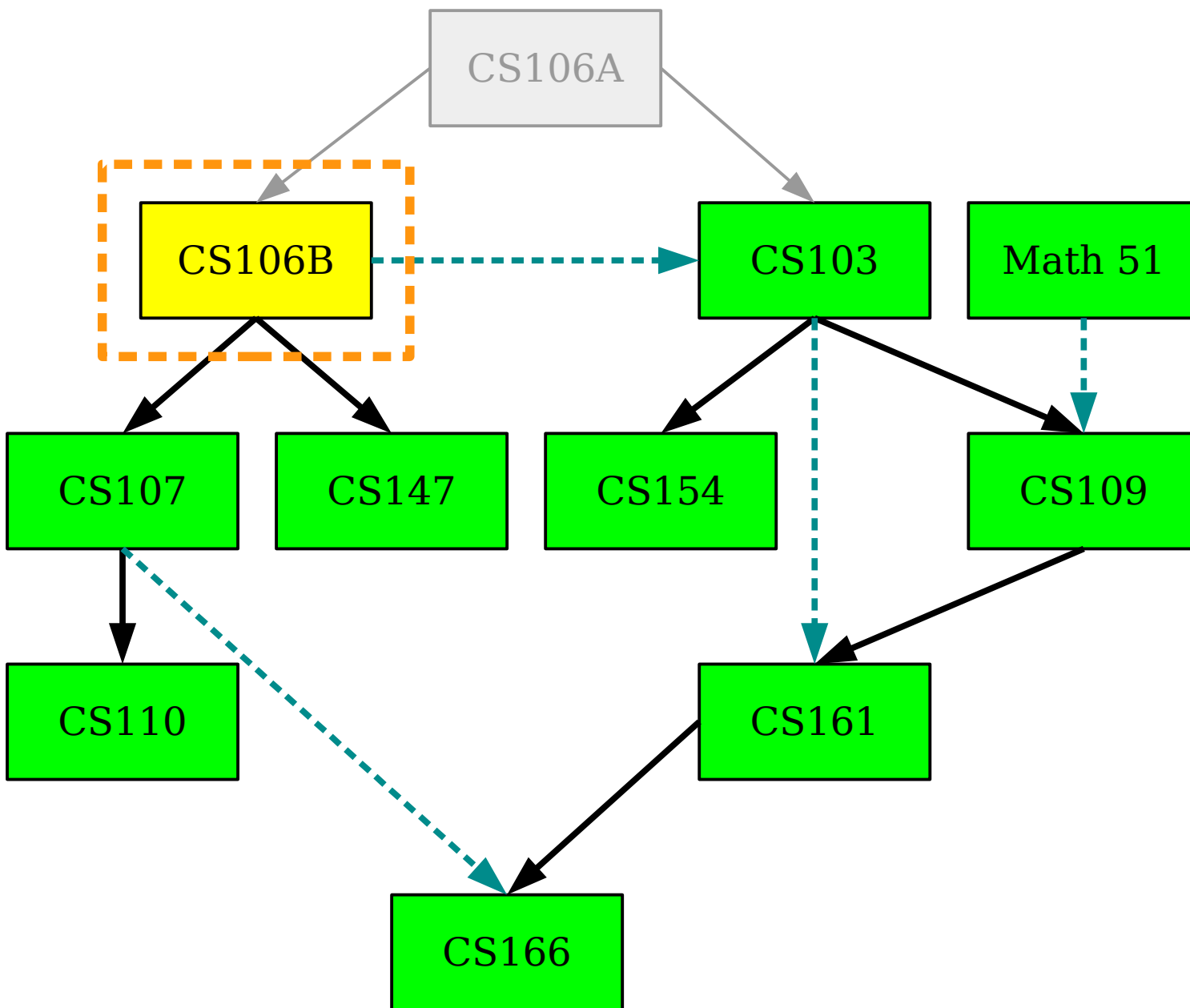


CS166
CS161
CS109
CS154
CS103
Math 51
CS147
CS110

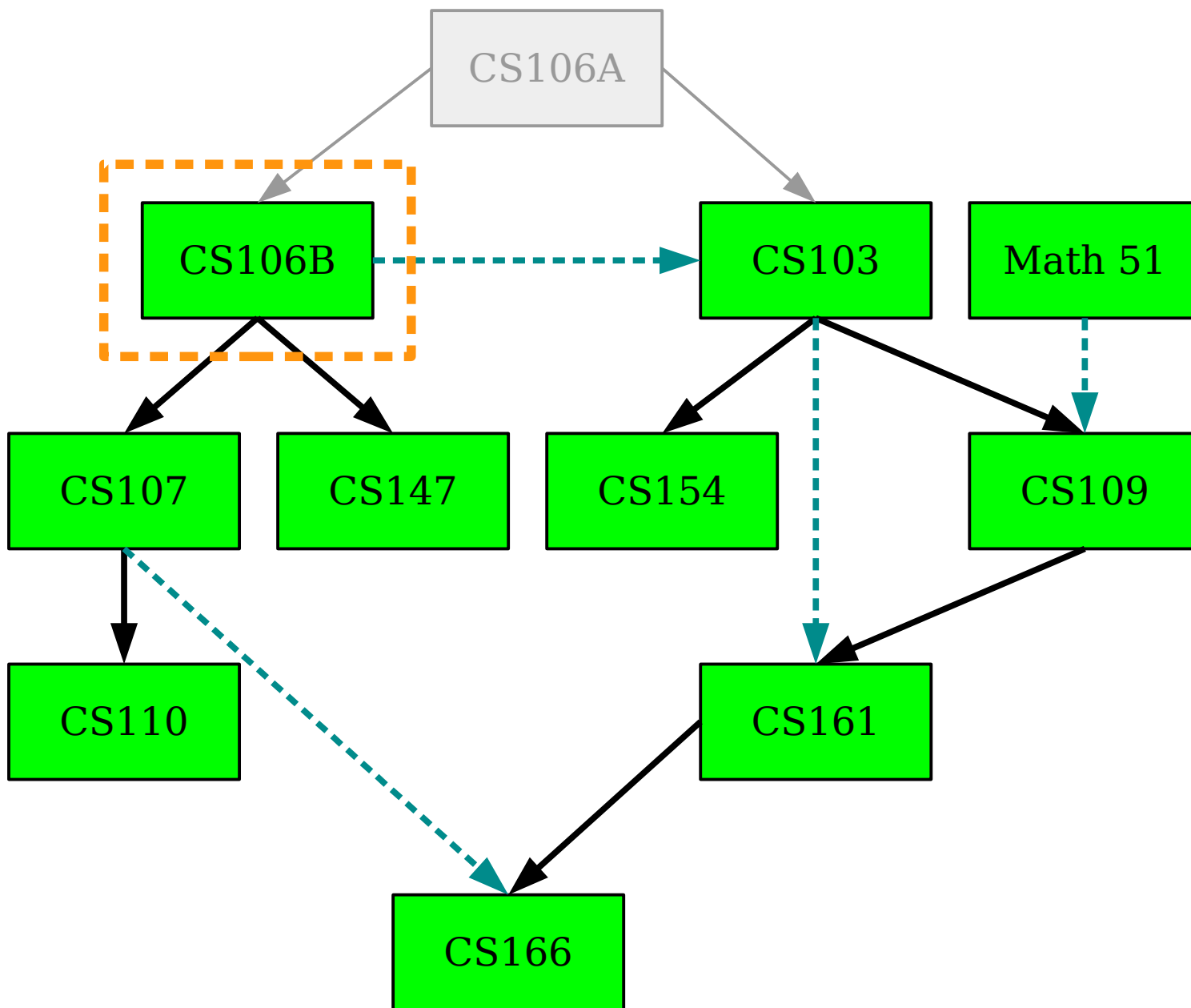




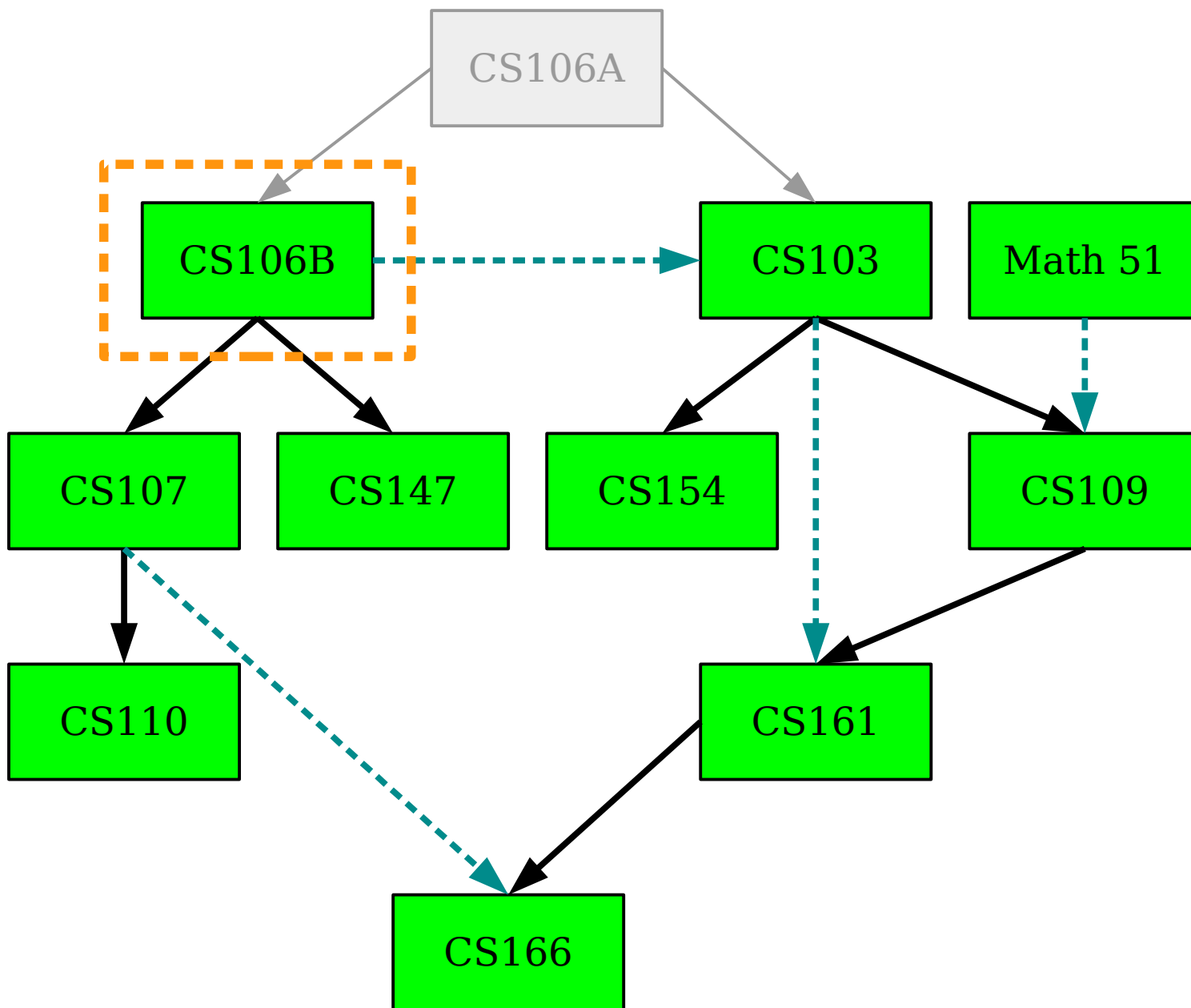




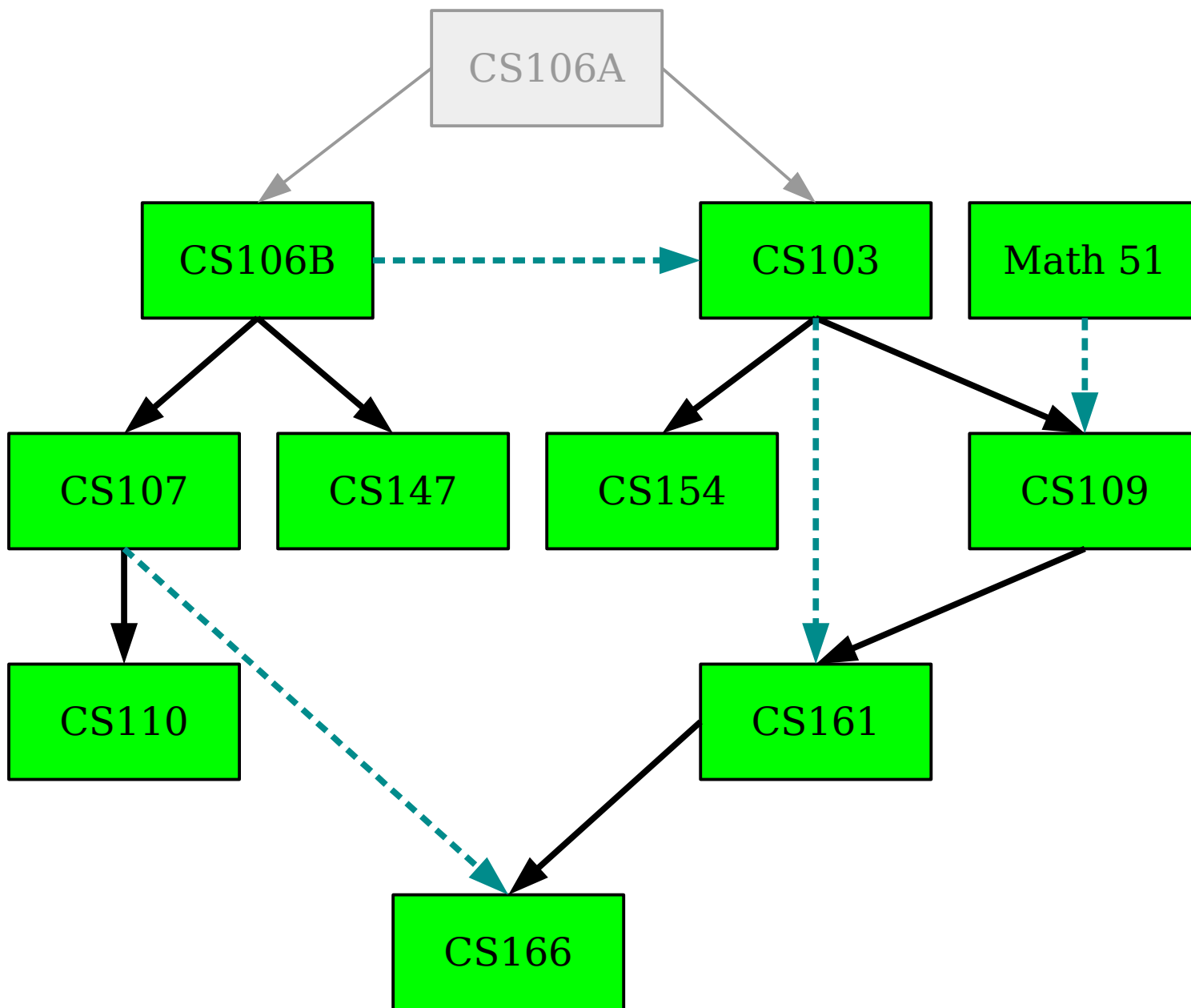
CS166
CS161
CS109
CS154
CS103
Math 51
CS147
CS110
CS107



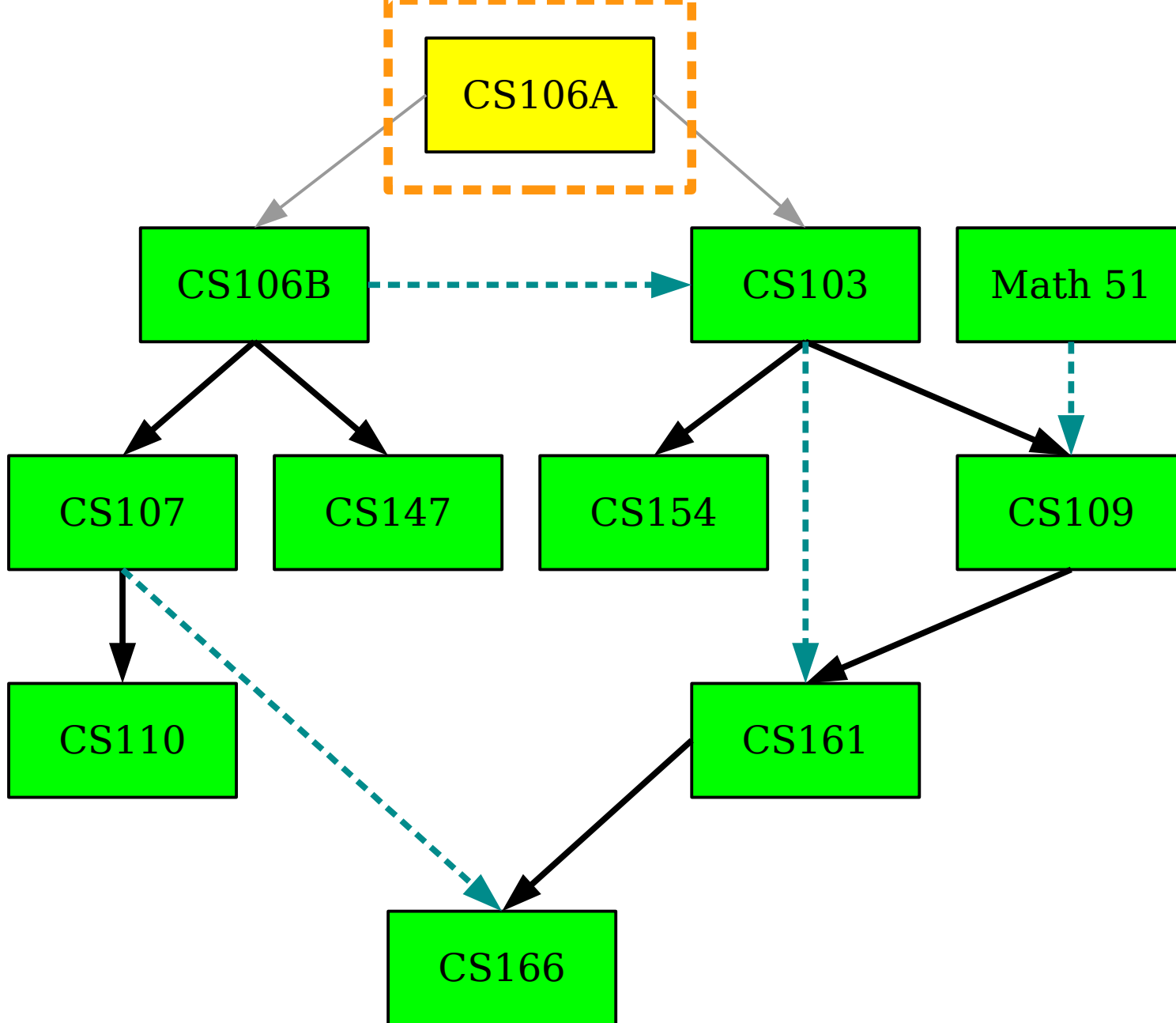
CS166
CS161
CS109
CS154
CS103
Math 51
CS147
CS110
CS107



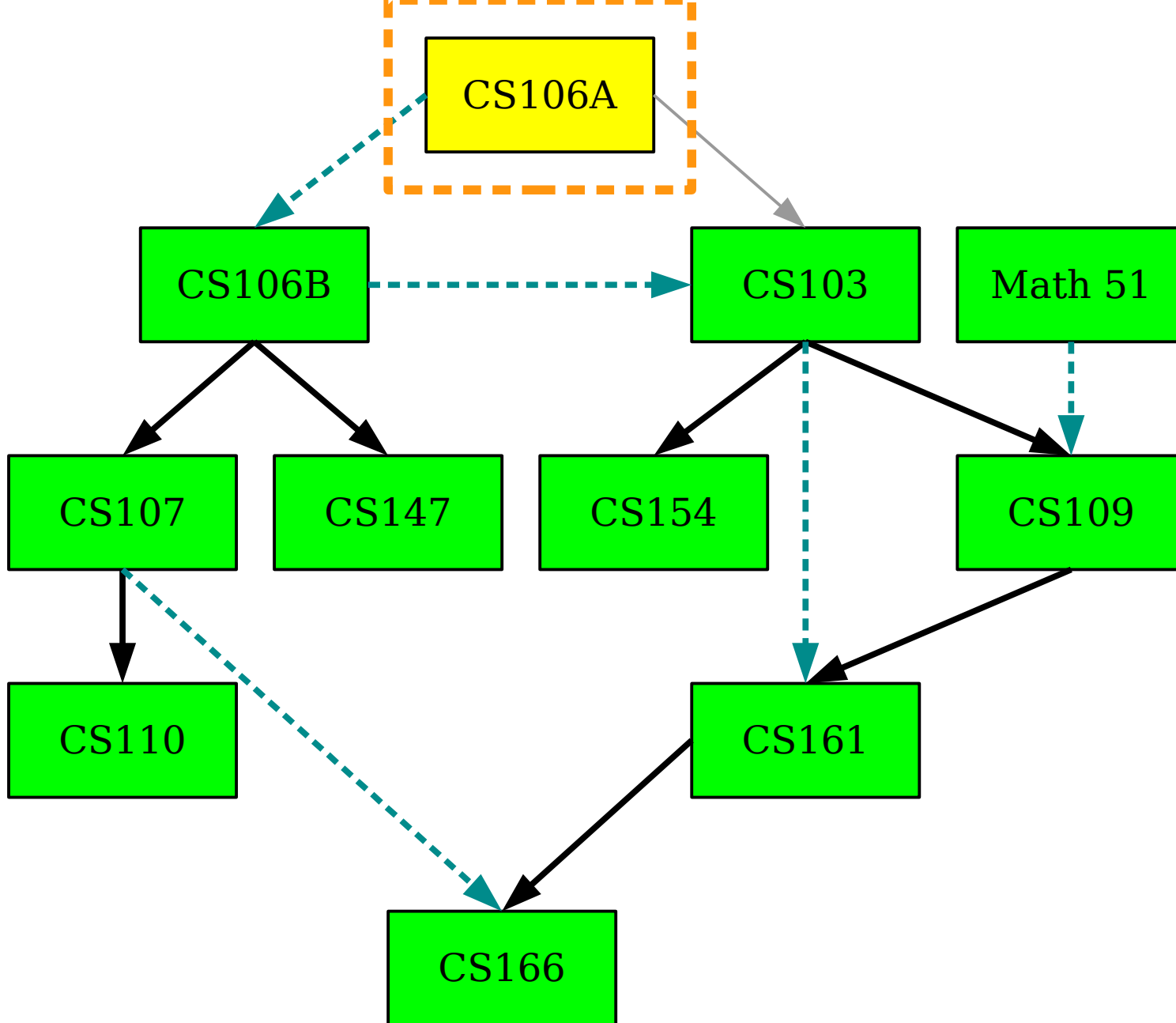
CS166
CS161
CS109
CS154
CS103
Math 51
CS147
CS110
CS107
CS106B



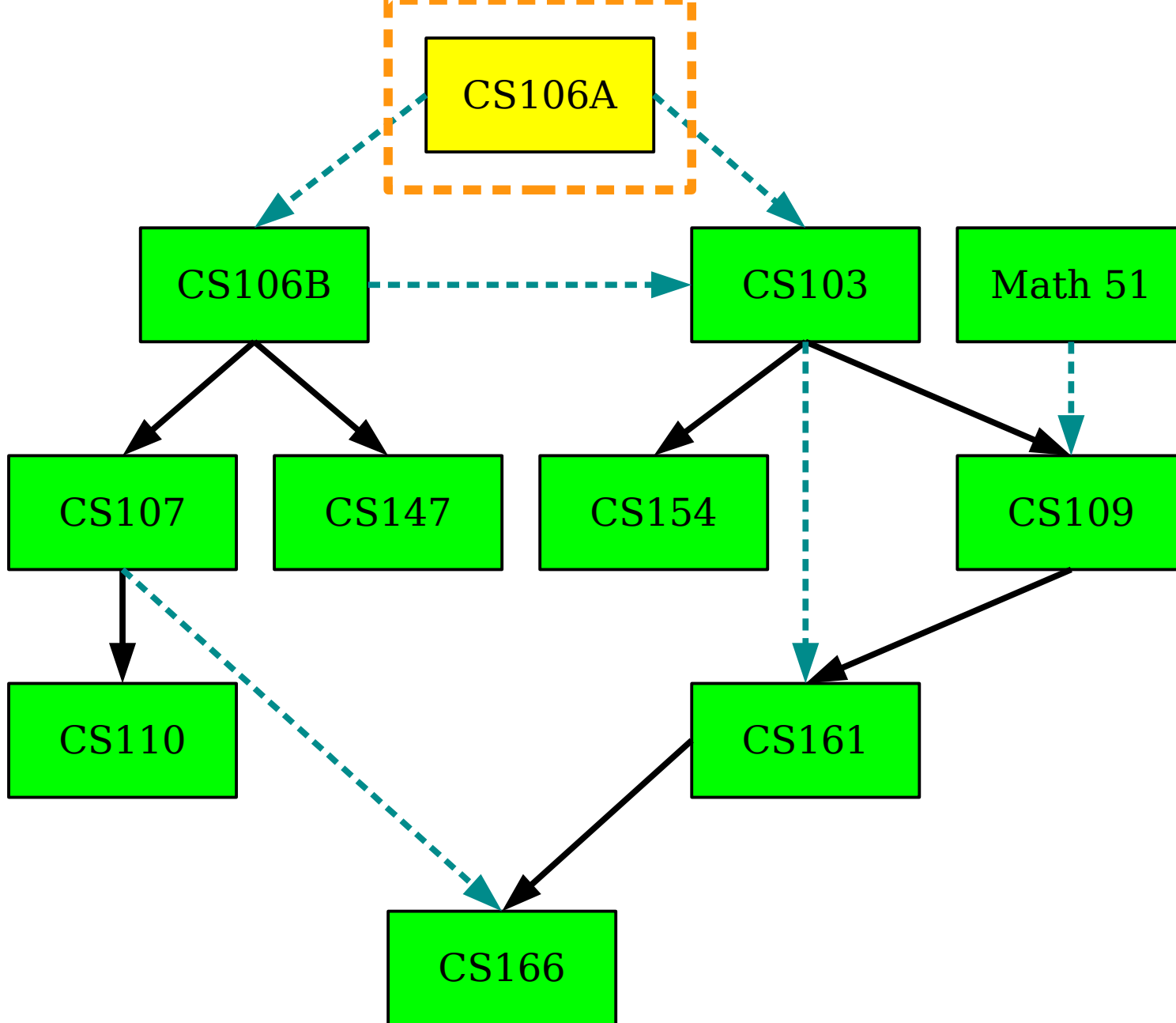
CS166
CS161
CS109
CS154
CS103
Math 51
CS147
CS110
CS107
CS106B



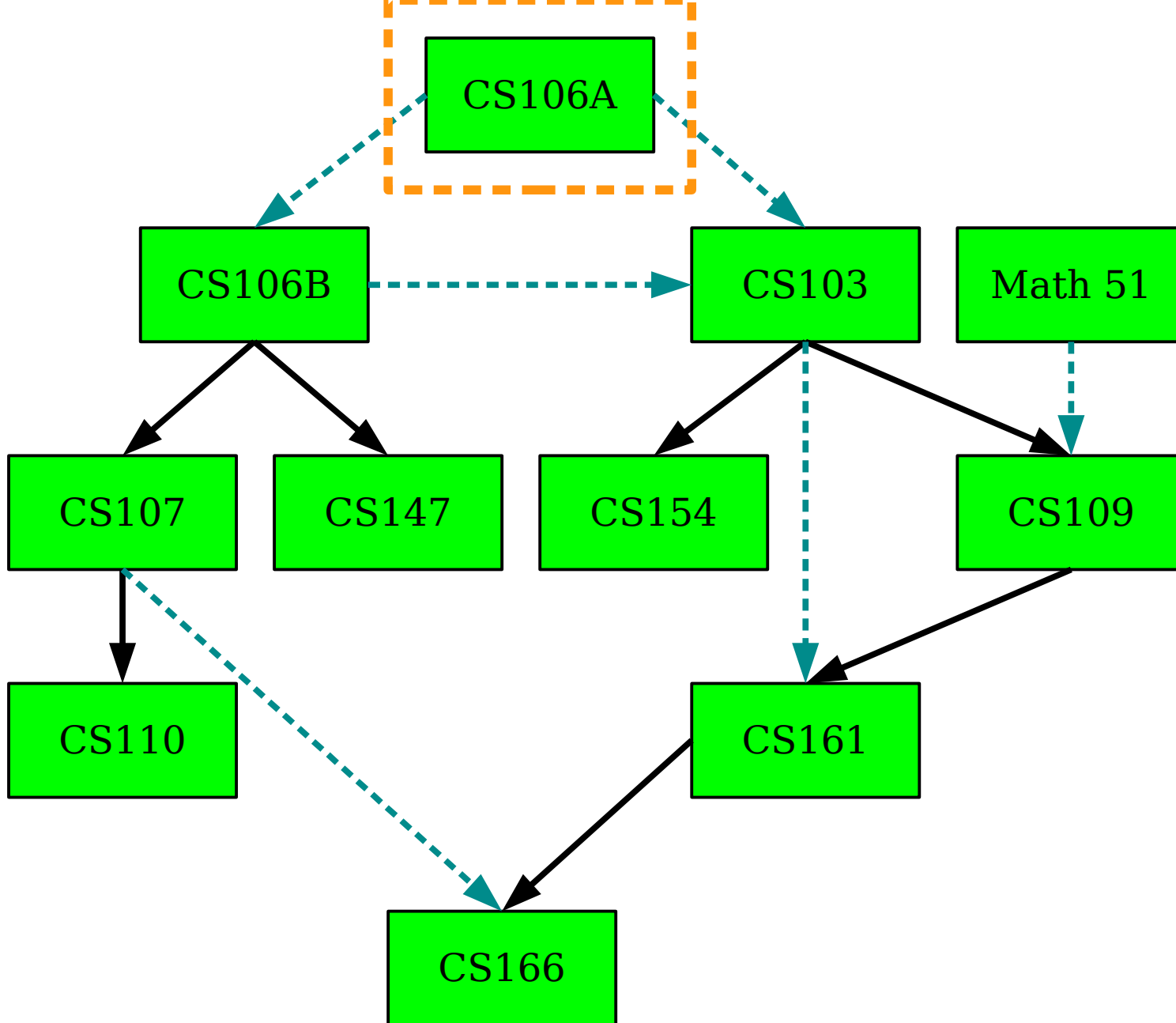
CS166
CS161
CS109
CS154
CS103
Math 51
CS147
CS110
CS107
CS106B



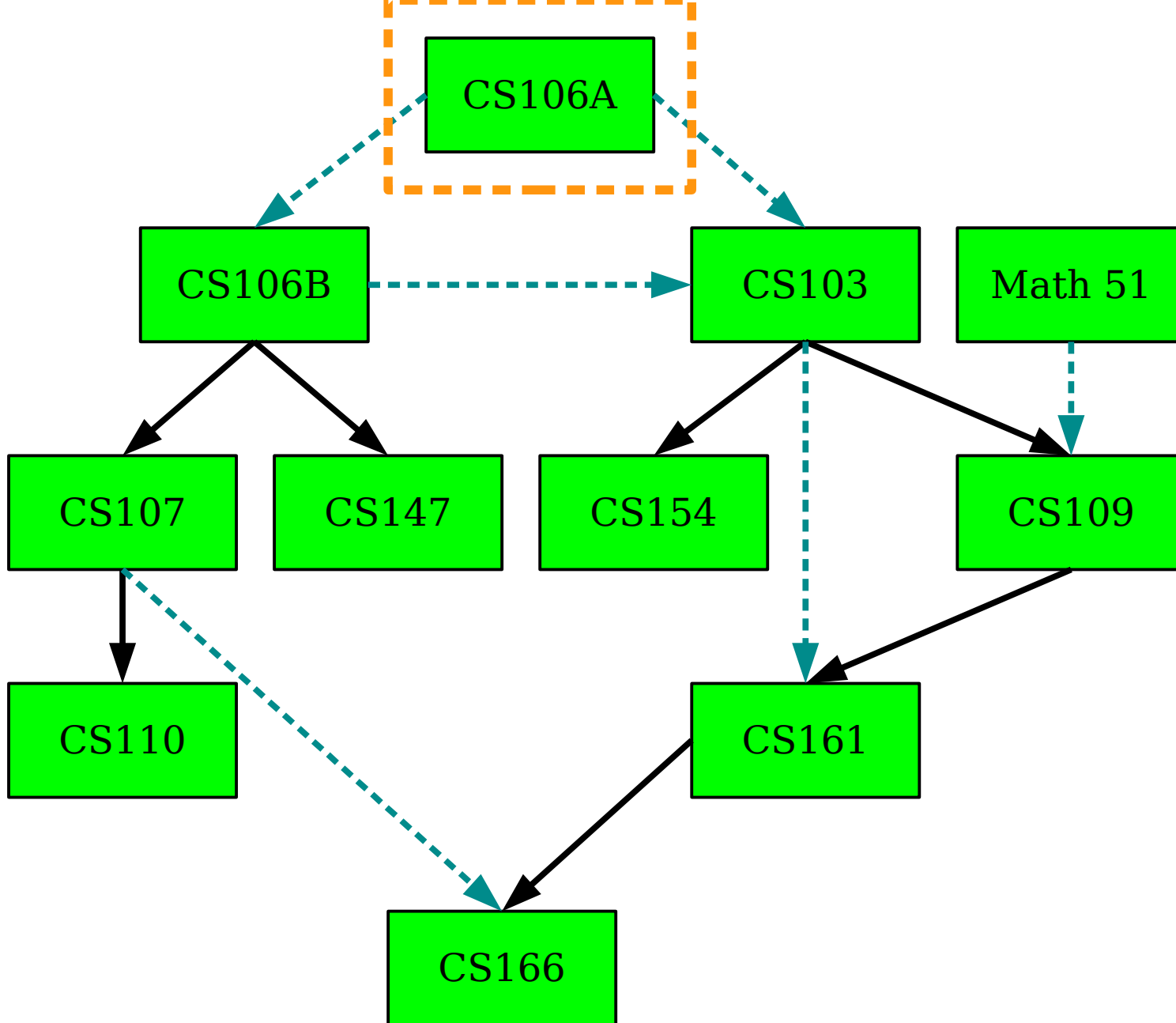
CS166
CS161
CS109
CS154
CS103
Math 51
CS147
CS110
CS107
CS106B



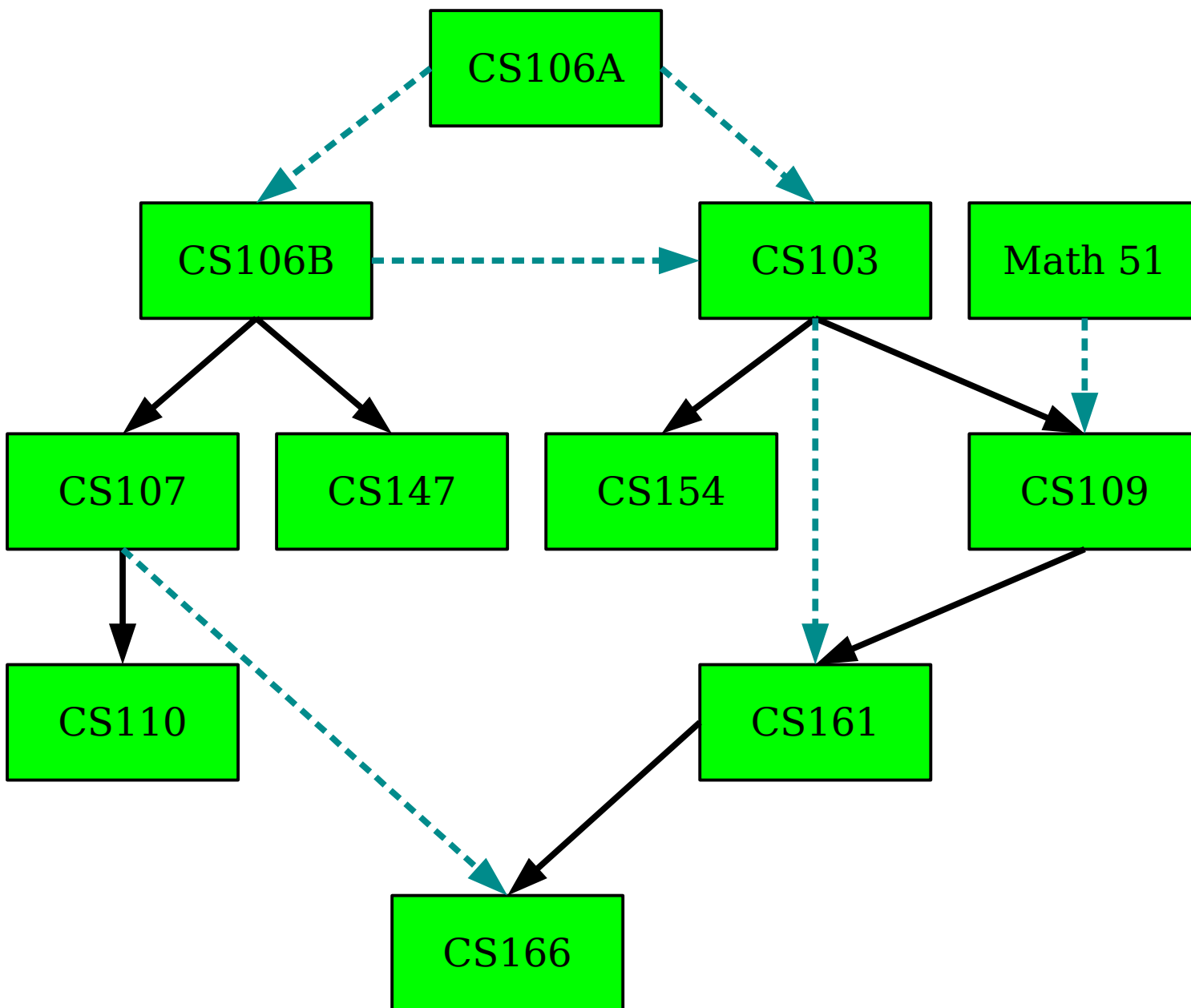
CS166
CS161
CS109
CS154
CS103
Math 51
CS147
CS110
CS107
CS106B



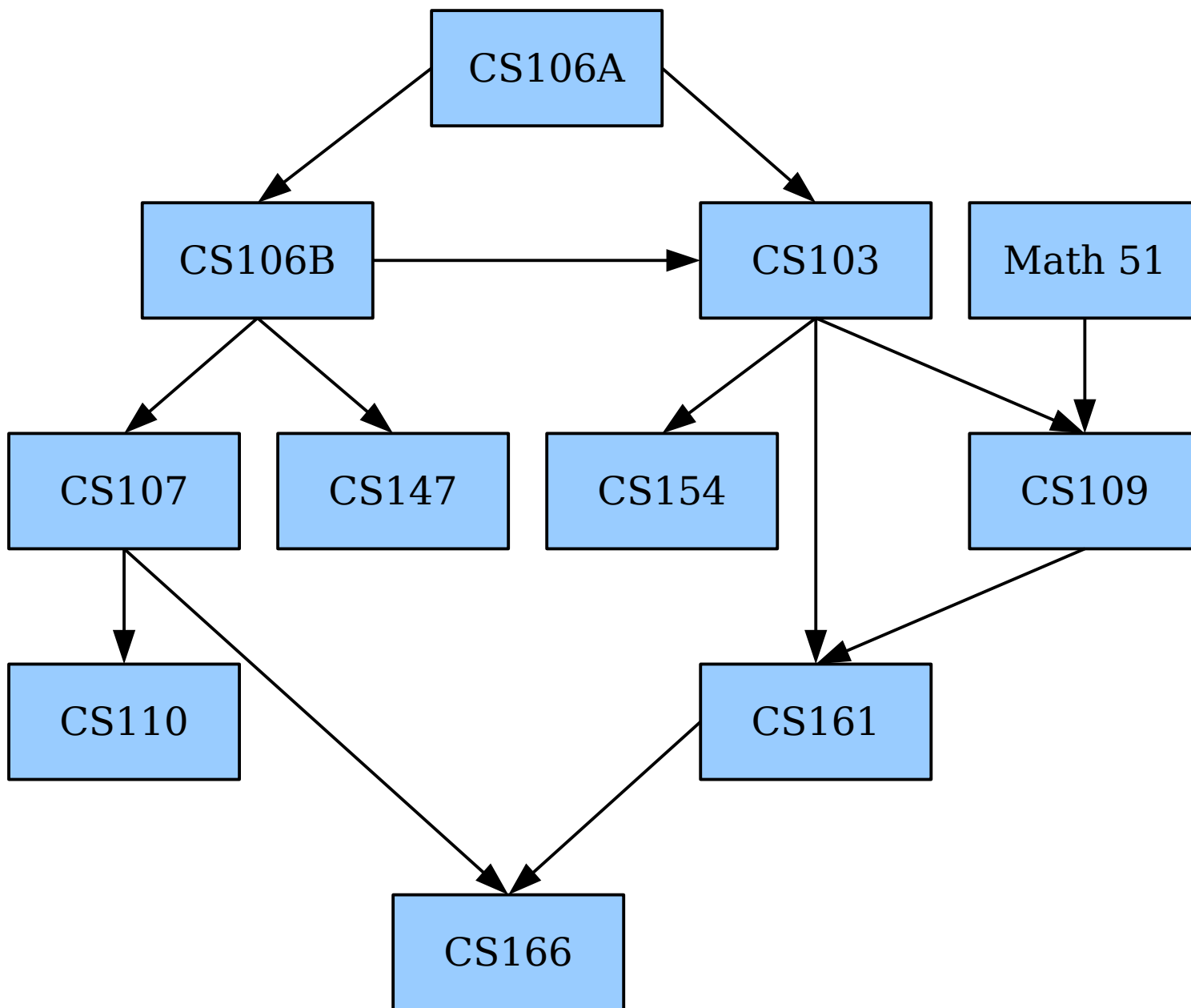
CS166
CS161
CS109
CS154
CS103
Math 51
CS147
CS110
CS107
CS106B



CS166
CS161
CS109
CS154
CS103
Math 51
CS147
CS110
CS107
CS106B
CS106A



CS166
CS161
CS109
CS154
CS103
Math 51
CS147
CS110
CS107
CS106B
CS106A



CS106A
CS106B
CS107
CS110
CS147
Math 51
CS103
CS154
CS109
CS161
CS166

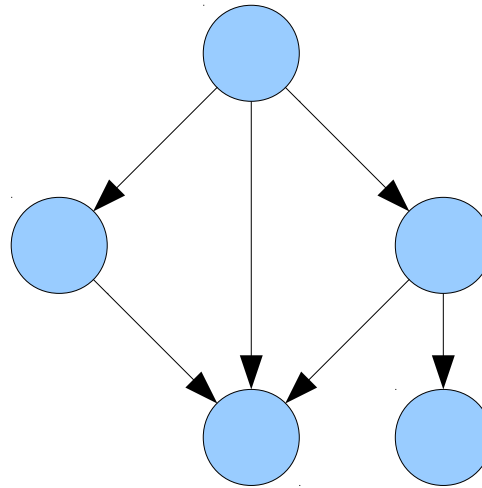
Topological Sort via DFS

- Here's a beautiful algorithm for topologically sorting a graph:

```
dfs-topological-sort() {  
  result = []  
  for each node in the graph, in whatever order sparks joy:  
    run a recursive DFS starting from that node.  
    when you finish visiting a node, append it to result.  
  return the reverse of result
```

- If you're clever with how you implement this, the runtime ends up being **$O(m + n)$** .

Why Does This Work?



- ***Intuition:*** Given a node v , a topological ordering needs to place v before a topological ordering of everything that depends on v .
- Running a DFS orders everything that depends on v before ordering v .
- Reversing things at the end is equivalent to always *prepending* v to the result rather than *appending* v to the result.

Your Action Items

- ***Aim to finish MiniBrowser.***
 - Again, you *can* use late days, but be careful about doing so.
- ***Read Chapter 18 of the textbook.***
 - There's a bunch of goodies in there about graph representations and graph algorithms.

Next Time

- ***More Graph Algorithms***
 - Which ones? Wait and see!