# Graph Theory

Dr. Irfan Yousuf
Department of Computer Science (New Campus)
UET, Lahore

(Lecture # 21; April 04, 2023)

# Outline

- Max Flow Min Cut Theorem
- Graph Coloring

# Max Flow Min Cut Theorem

- In computer science and optimization theory, the max-flow min-cut theorem states that

In a flow network, the maximum amount of flow passing from the source to the sink is equal to the total weight of the edges in a minimum cut, i.e., the smallest total weight of the edges which if removed would disconnect the source from the sink.

# Max Flow Min Cut Theorem

- The second half of the max-flow min-cut theorem refers to the collection of cuts.
- An s-t cut C = (S, T) is a partition of V such that $s \in S$ and $t \in T$.
- That is, an s-t cut is a division of the vertices of the network into two parts, with the **source in one part** and **the sink in the other**.

# Max Flow Min Cut Theorem

- The maximum value of an s-t flow is equal to the minimum capacity over all s-t cuts.

- The max-flow min-cut theorem states that the **maximum flow** through any network from a given source to a given sink is exactly equal to **the minimum sum of a cut**.

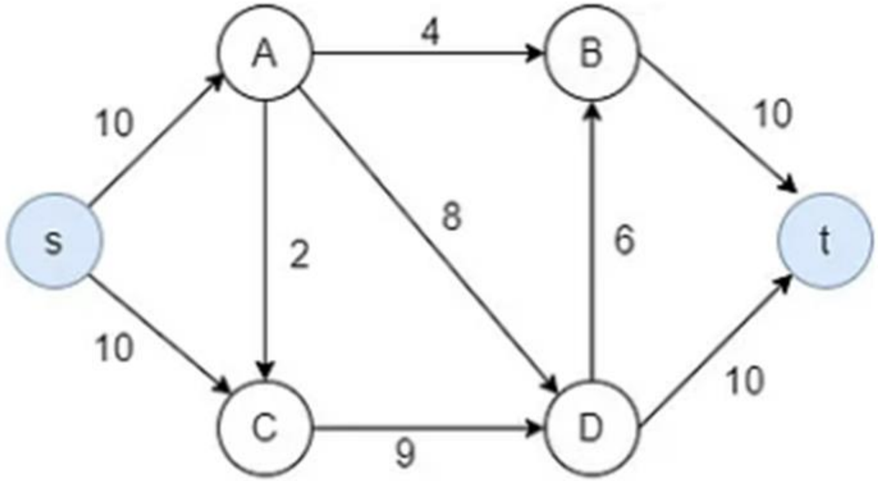
# How to Find Max Flow Min Cut?

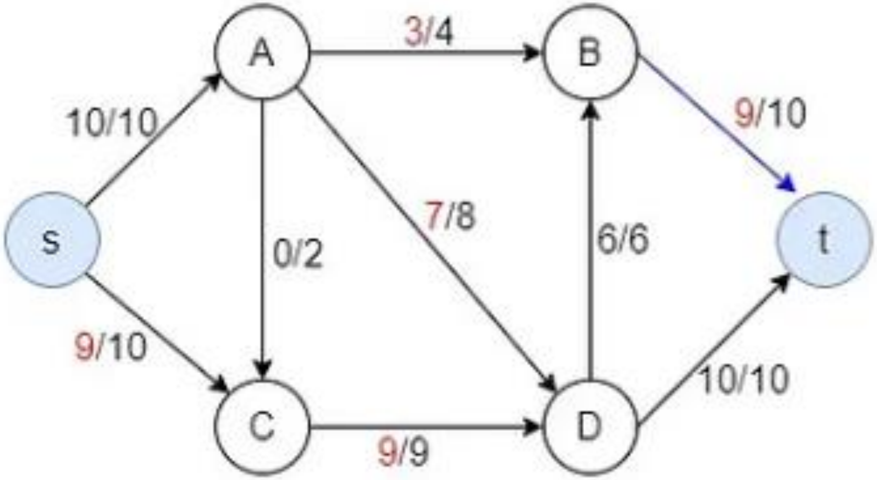
- Run Ford-Fulkerson algorithm and consider the final residual graph.

- Find the set of vertices that are <u>reachable</u> from source in the residual graph.

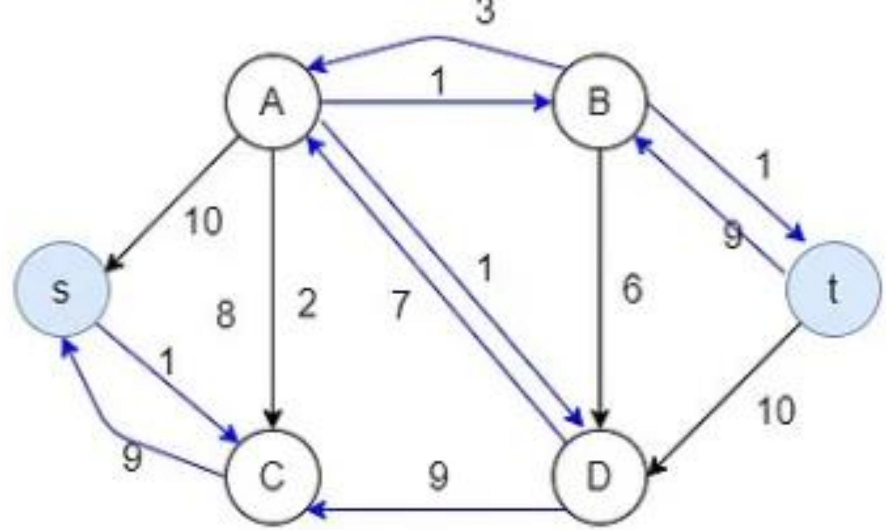
- All the edges **<u>(in the original graph)</u>** which are from a reachable vertex to non-reachable vertex are minimum cut edges.

# How to Find Max Flow Min Cut?



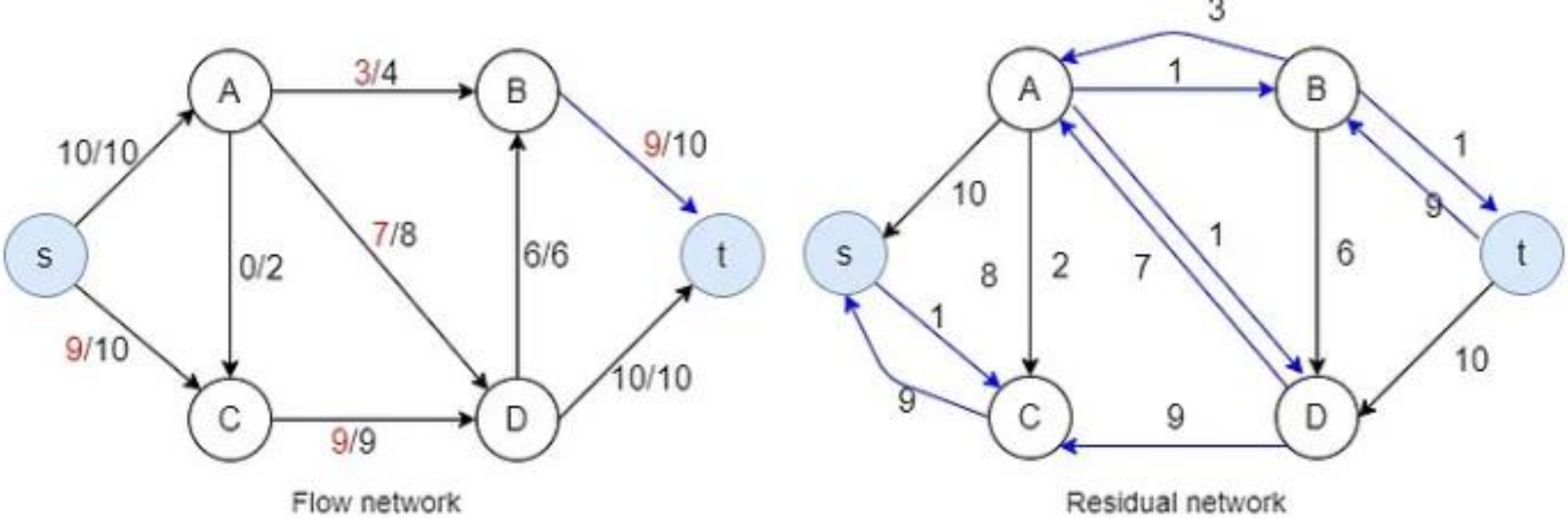


Flow network



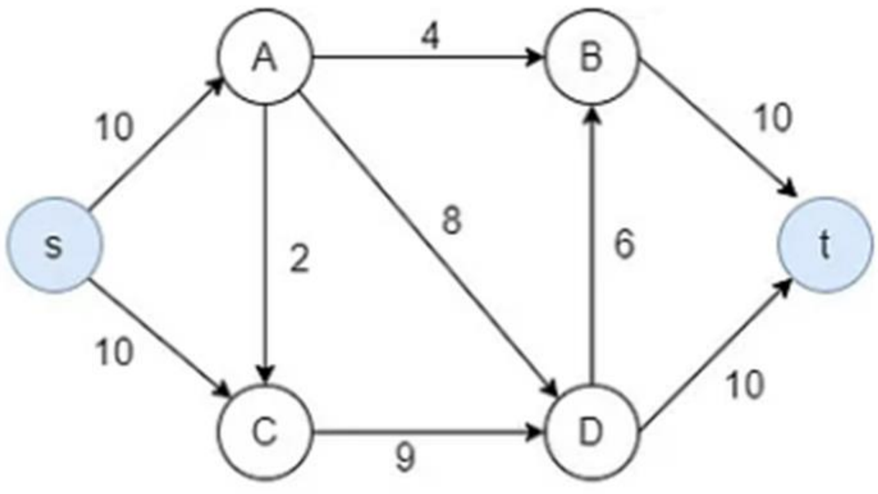
Residual network

- Now there are no paths left from the s to t in the residual graph. So, there is no possibility to add flow.
- Since the maximum flow is equal to the flow coming out of the source, in this example, the maximum flow is 10+9 = 19.



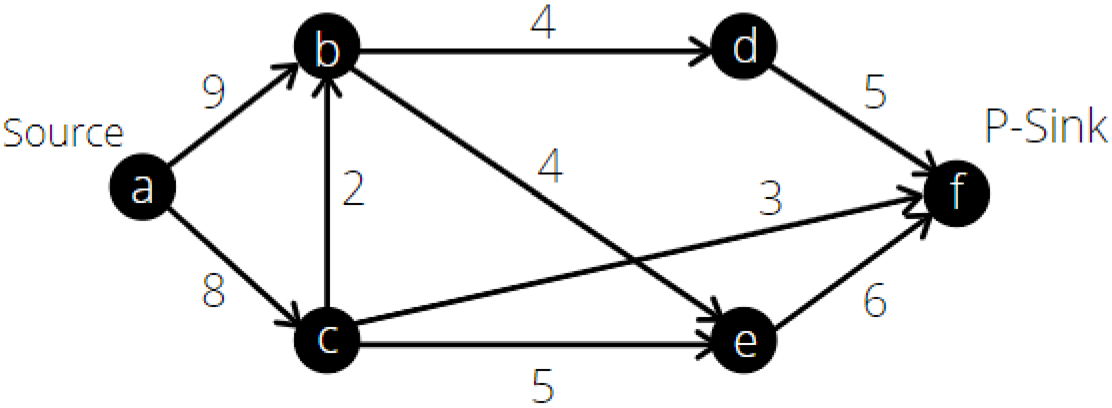

- Reachable vertices from Source (S)
  - C vertex
  - S and C form one partition and all other nodes form second partition.

# How to Find Max Flow Min Cut?



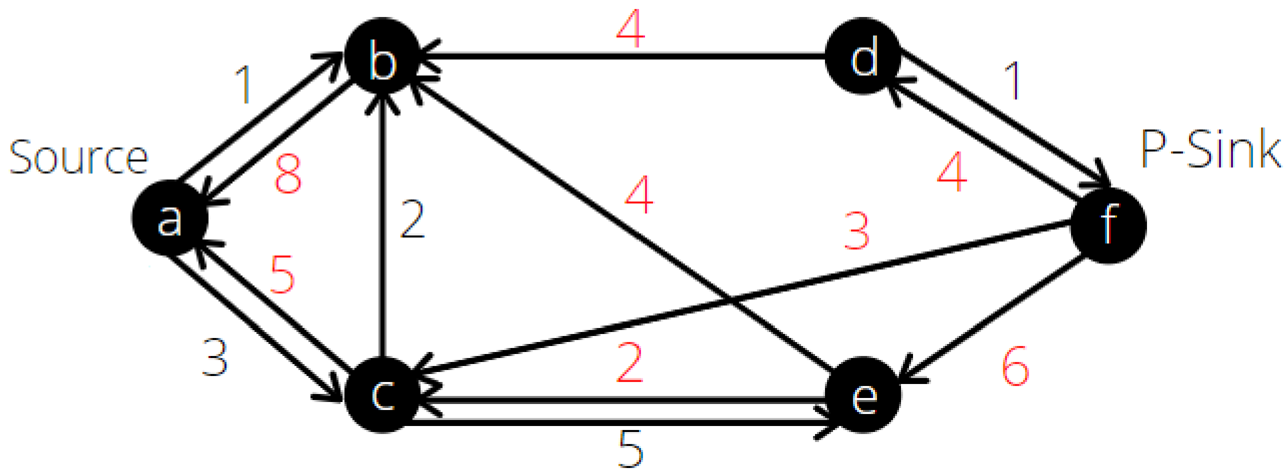

- S and C form one partition and all other nodes form second partition.

# How to Find Max Flow Min Cut?



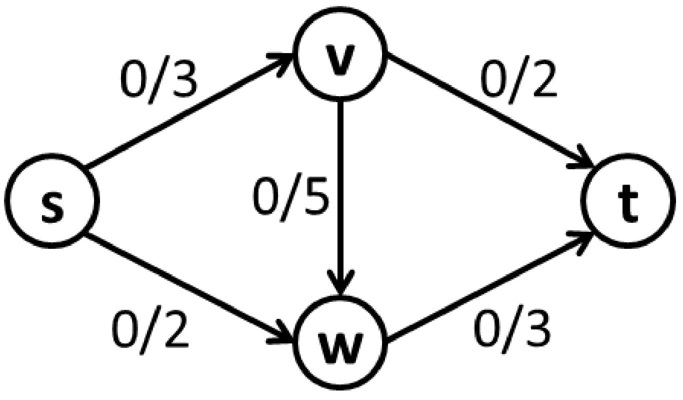

**Max flow = 13**

Residual Network

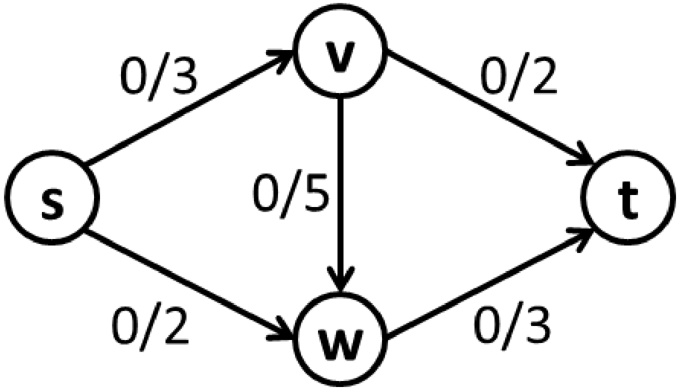


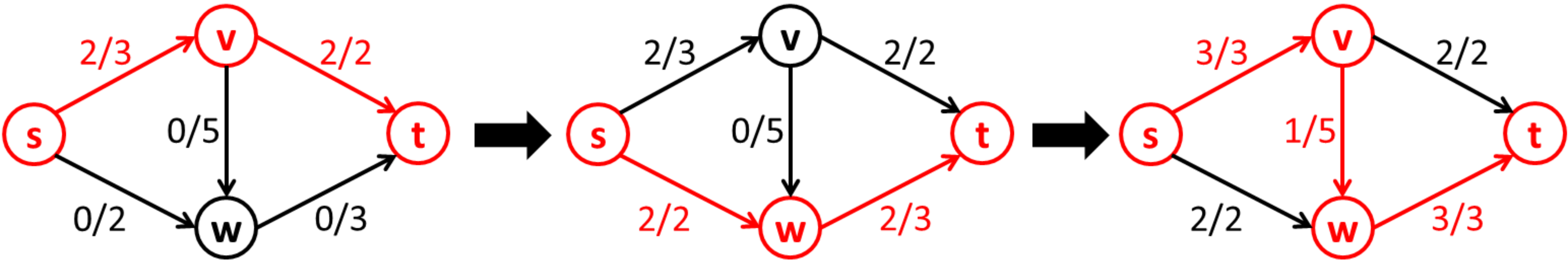
# Residual Graph Construction?

- Why do we construct a residual graph in Ford Fulkerson Algorithm?



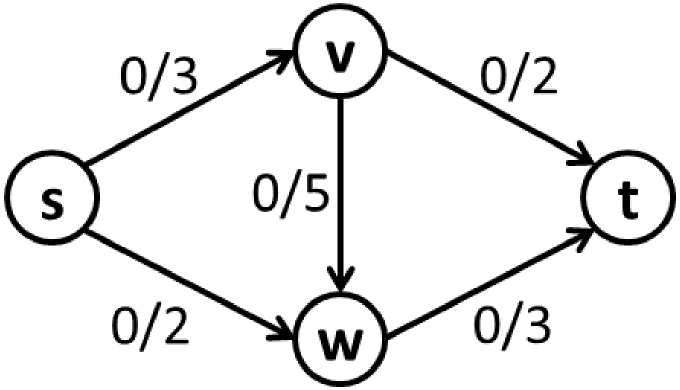
# Residual Graph Construction?

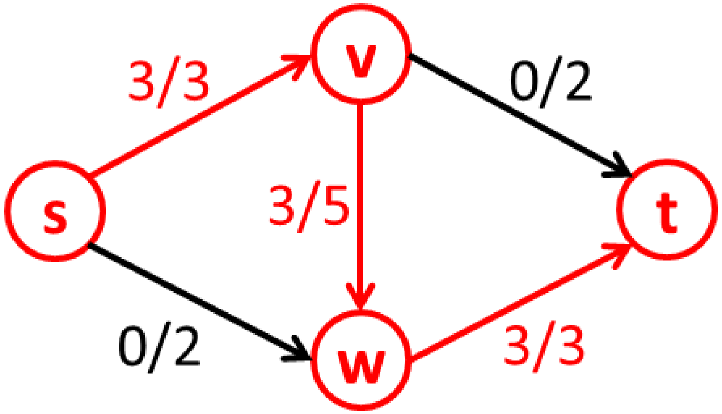



$P_1$ = [s, v, t]    $P_2$ = [s, w, t]    $P_3$ = [s, v, w, t]

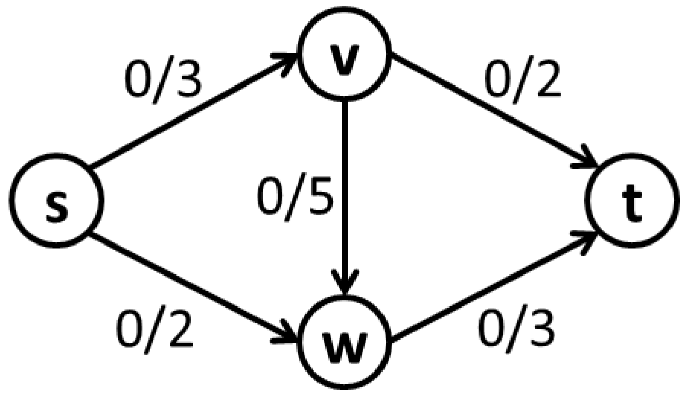
# Residual Graph Construction?

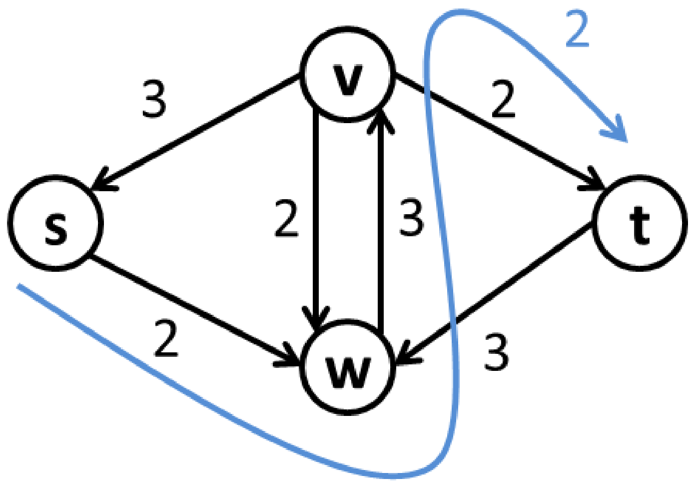



We get what is called a **blocking flow**: no more augmenting paths exist. In this case, the total flow is 3, which is not optimal.

# Residual Graph Construction?



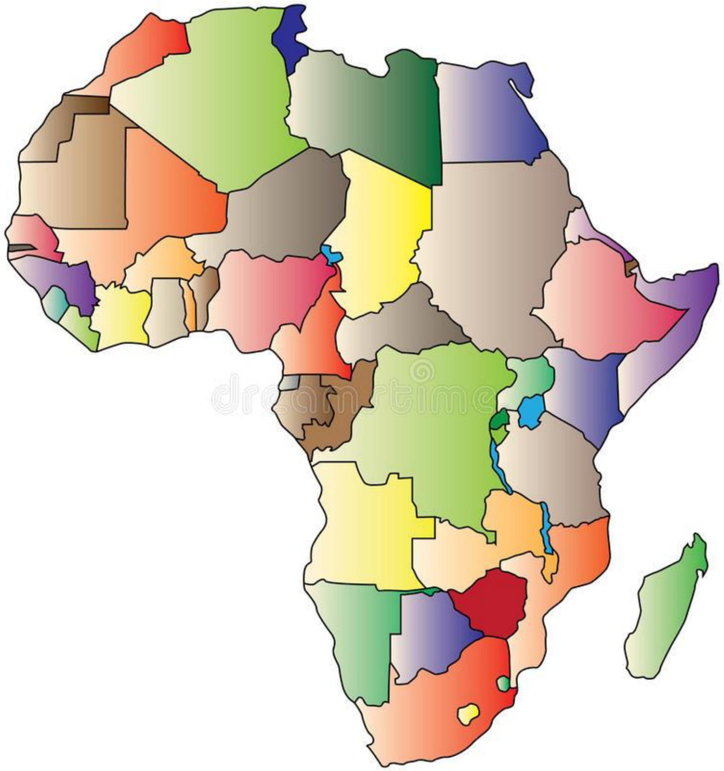
# Why Ford Fulkerson Algorithm Works?

- The Ford–Fulkerson method proceeds in exactly the same way as the greedy approach described above, but it only stops when there are <u>no more augmenting paths</u> in the residual graph (not in the original network).
- The method is correct (i.e., it always computes a maximum flow) because the residual graph establishes the following optimality condition:

  - Given a network G, a flow f is maximum in G if there is no s–t path in the residual graph.

# Graph Coloring

# Graph Coloring

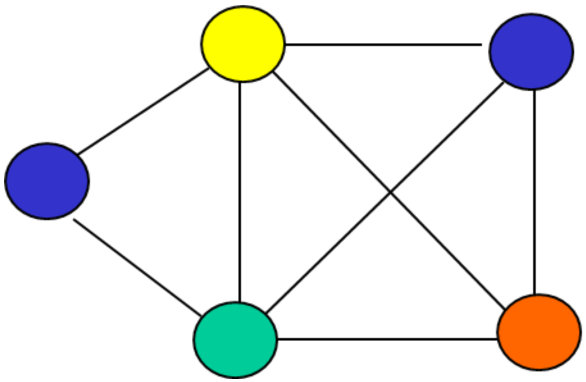
# Graph Coloring

- Color a map such that two regions with a common border are assigned different colors.

- Each map can be represented by a graph:
  - Each region of the map is represented by a vertex;
  - Edges connect two vertices if the regions represented by these vertices have a common border.
- The resulting graph is called the dual graph of the map.

# Graph Coloring

- A graph has been colored if a color has been assigned to each vertex in such a way that adjacent vertices have different colors.



- The chromatic number of a graph is the smallest number of colors with which it can be colored.

In the example above, the chromatic number is 4.

# Graph Coloring

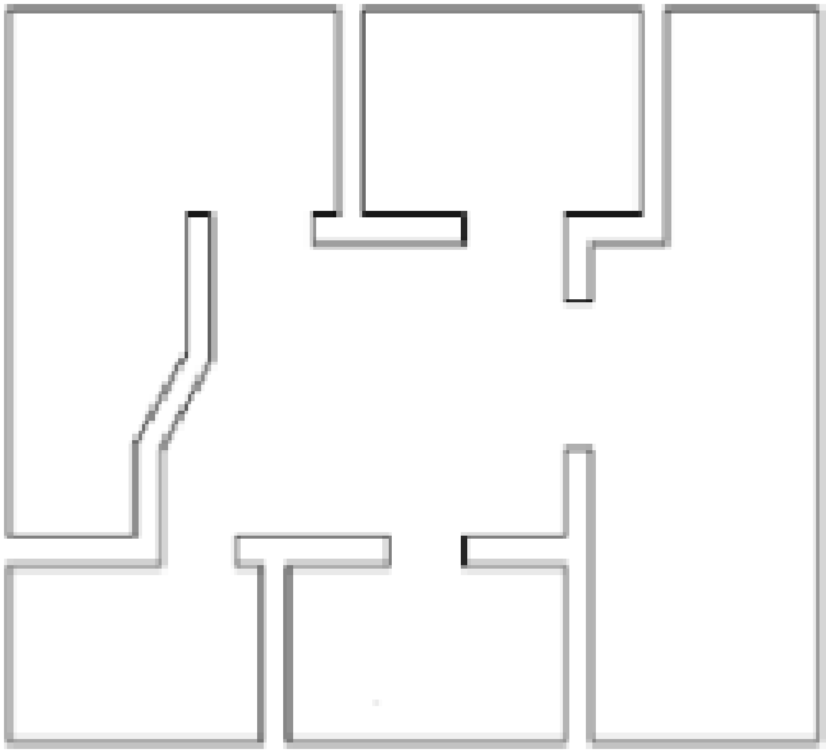
Suppose that in a particular quarter there are students taking each of the following combinations of courses:

- *Math, English, Biology, Chemistry*
- *Math, English, Computer Science, Geography*
- *Biology, Psychology, Geography*, Spanish
- *Biology, Computer Science, History, French*
- *English, Psychology, Computer Science, History*
- *Psychology, Chemistry, Computer Science, French*
- *Psychology, Geography*, *History, Spanish*

What is the <u>minimum number of examination slots</u> required for the exams in the ten courses specified so that students taking any of the given combinations of courses have no conflicts? Find a schedule that uses this minimum number of slots.

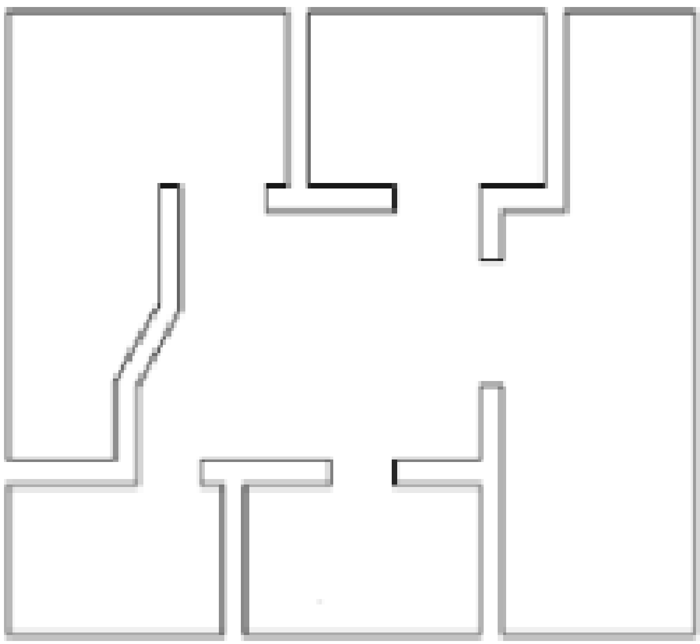
# Graph Coloring

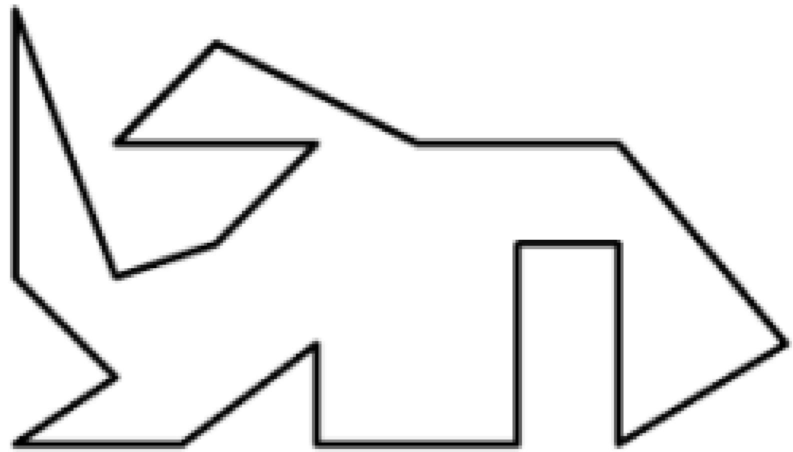
- Minimum number of cameras?
- Location of each camera?



# Graph Coloring

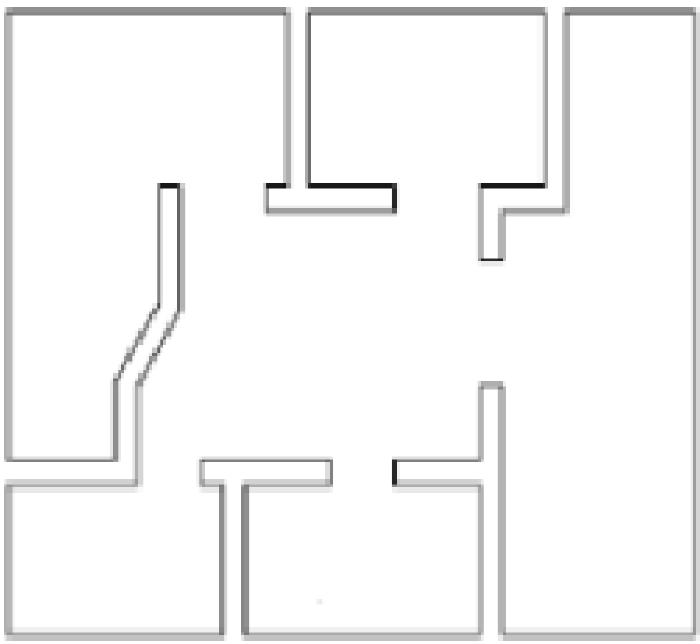
- Minimum number of cameras?
- Location of each camera?

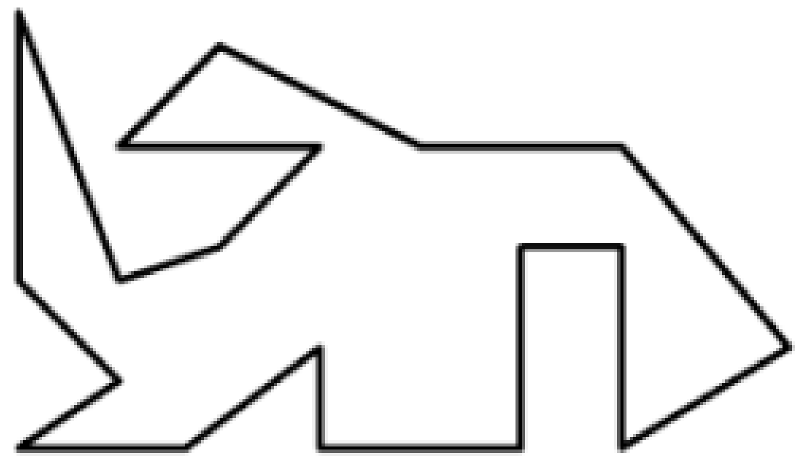


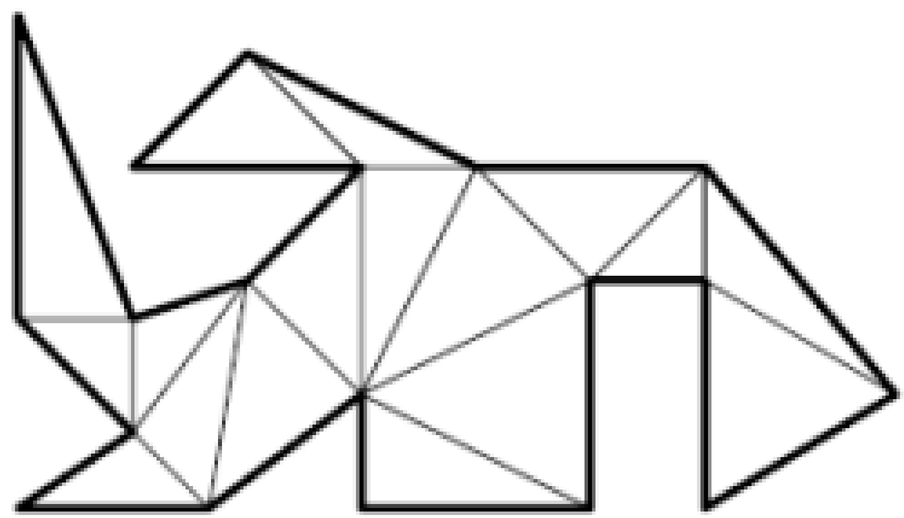


# Graph Coloring

- Minimum number of cameras?
- Location of each camera?

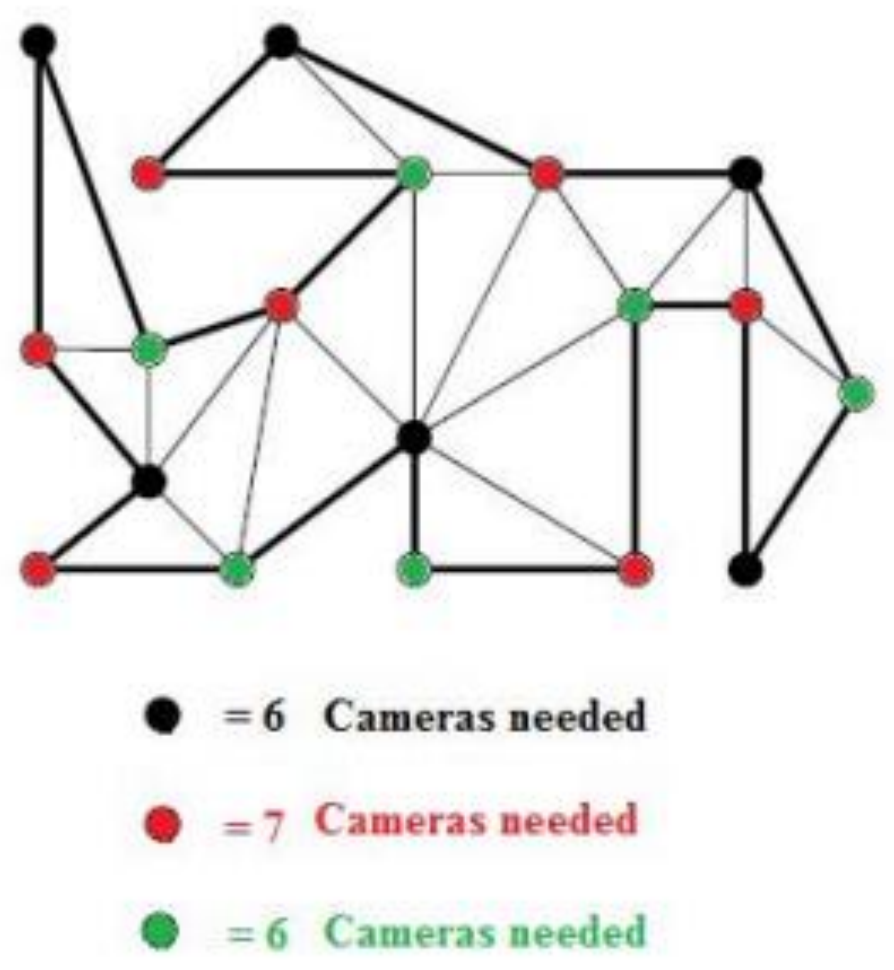






# Graph Coloring

- Minimum number of cameras?
- Location of each camera?



# Graph Coloring

(Vertex Coloring). Let $G = (V, E)$ be a graph and let $C = \{c_1, \ldots, c_k\}$ be a finite set of *colors* (labels). A *vertex coloring* is a mapping $c : V \to C$ with the property that if $\{v_1, v_2\} \in E$, then $c(v_1) \neq c(v_2)$.

($k$-Colorable). A graph $G = (V, E)$ is a $k$-colorable if there is a vertex coloring with $k$ colors.

(Chromatic Number). Let $G = (V, E)$ be a graph. The *chromatic number* of $G$, written $\chi(G)$ is the minimum integer $k$ such that $G$ is $k$-colorable.

# Kempe's Algorithm

To mostly K-color a graph

Is there a vertex of degree < K ?

If so:

Remove this vertex.

Color the rest of the graph with a recursive call to the algorithm.

Put the vertex back. It is adjacent to at most K-1 vertices. They use (among them) at most K-1 colors. That leaves one of your colors for this vertex.

If not:

Remove this vertex.

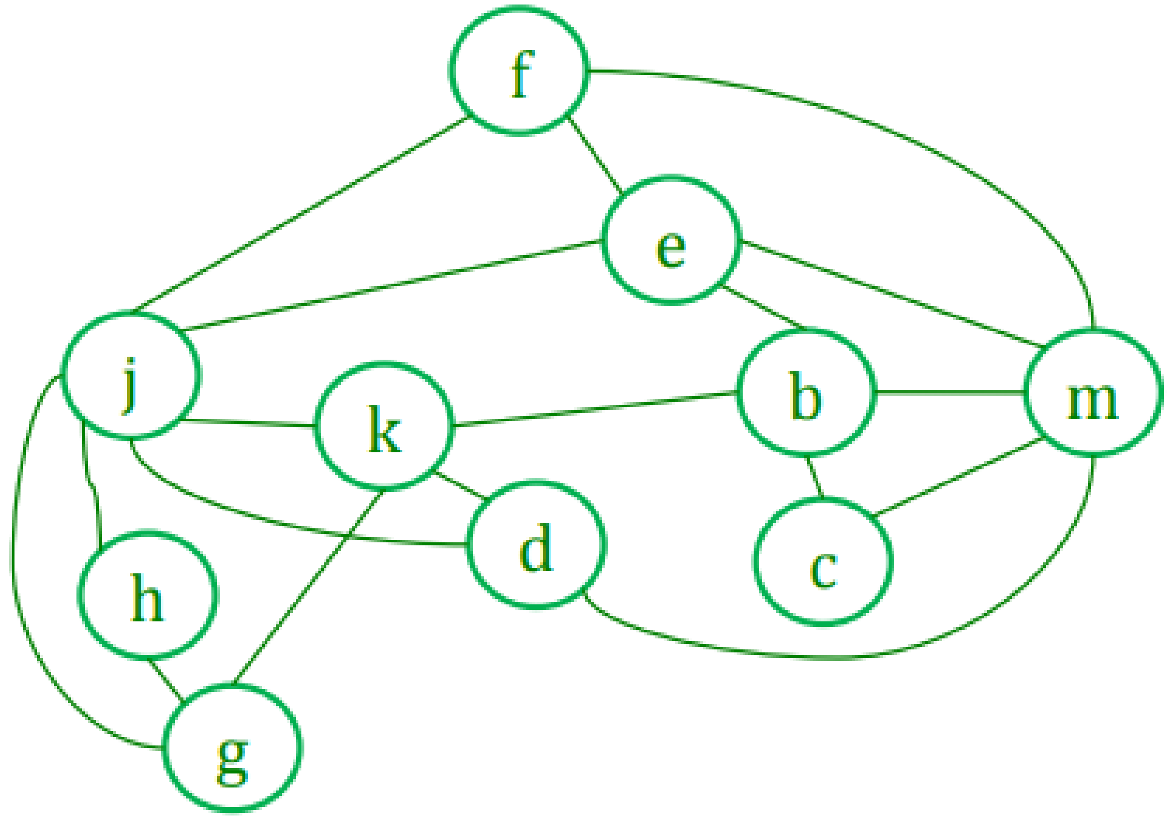
Color the rest of the graph with a recursive call.

Put the vertex back. It is adjacent to ≥ K vertices. How many colors do these vertices use among them?

If < K : there is an unused color to use for this vertex

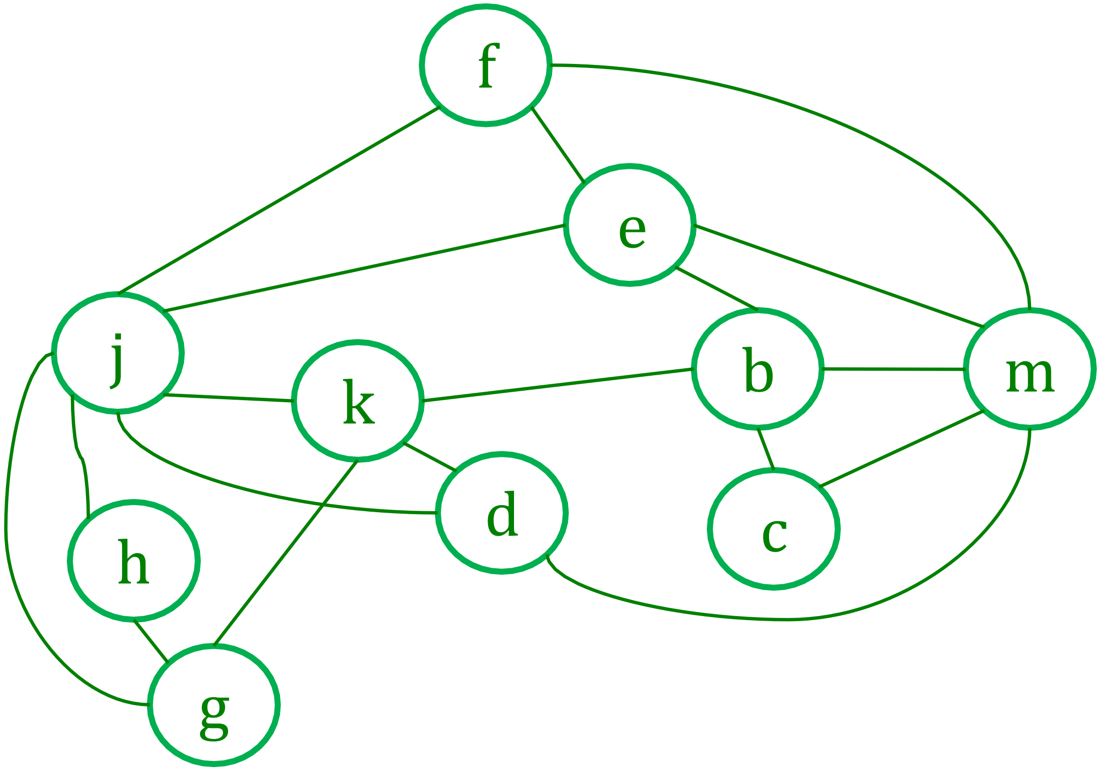
If ≥ K: leave this vertex uncolored.

# Kempe's Algorithm: 3-Color




Stack:

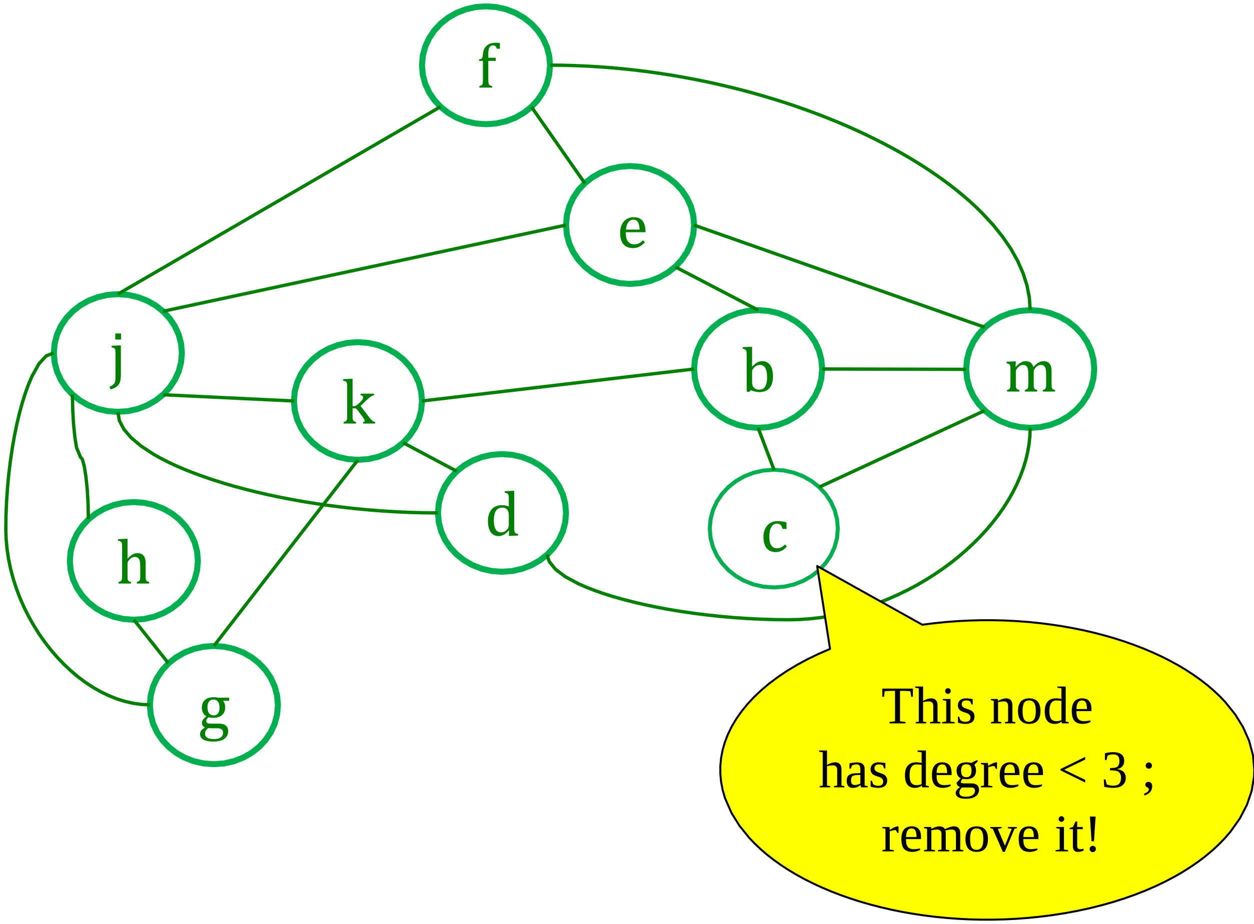
# Example:  3-color this graph




Stack:

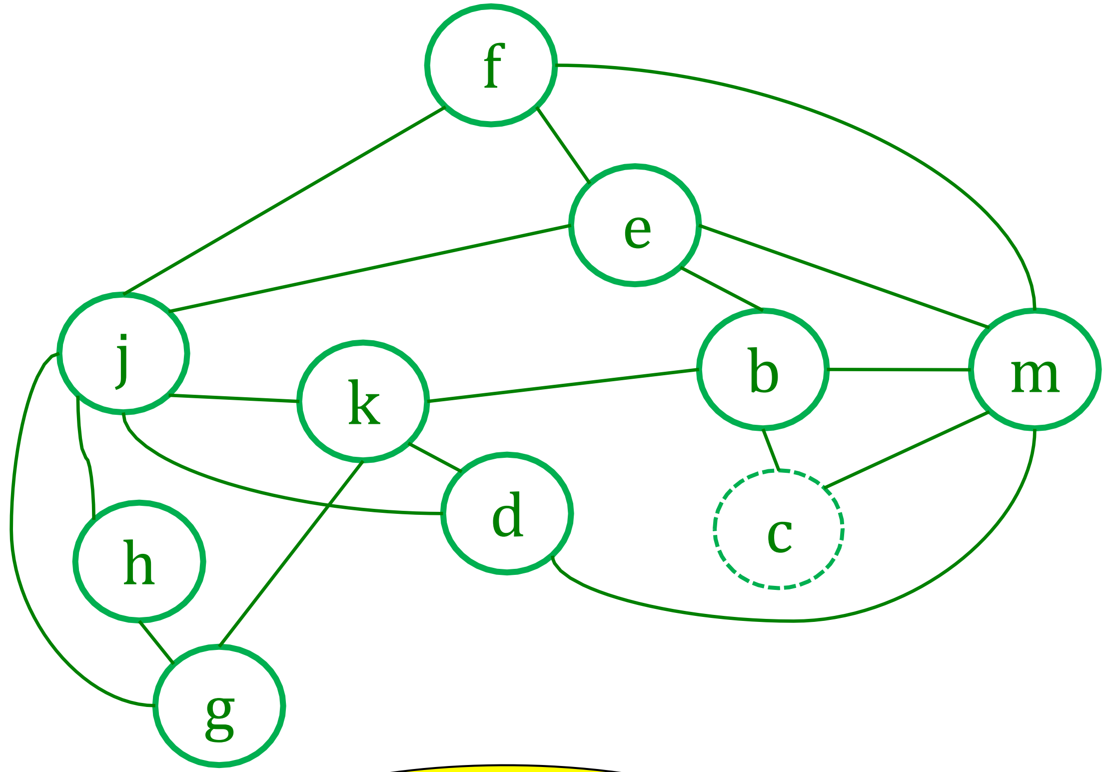
* Copied from Princeton University

# Example: 3-color this graph




Stack:

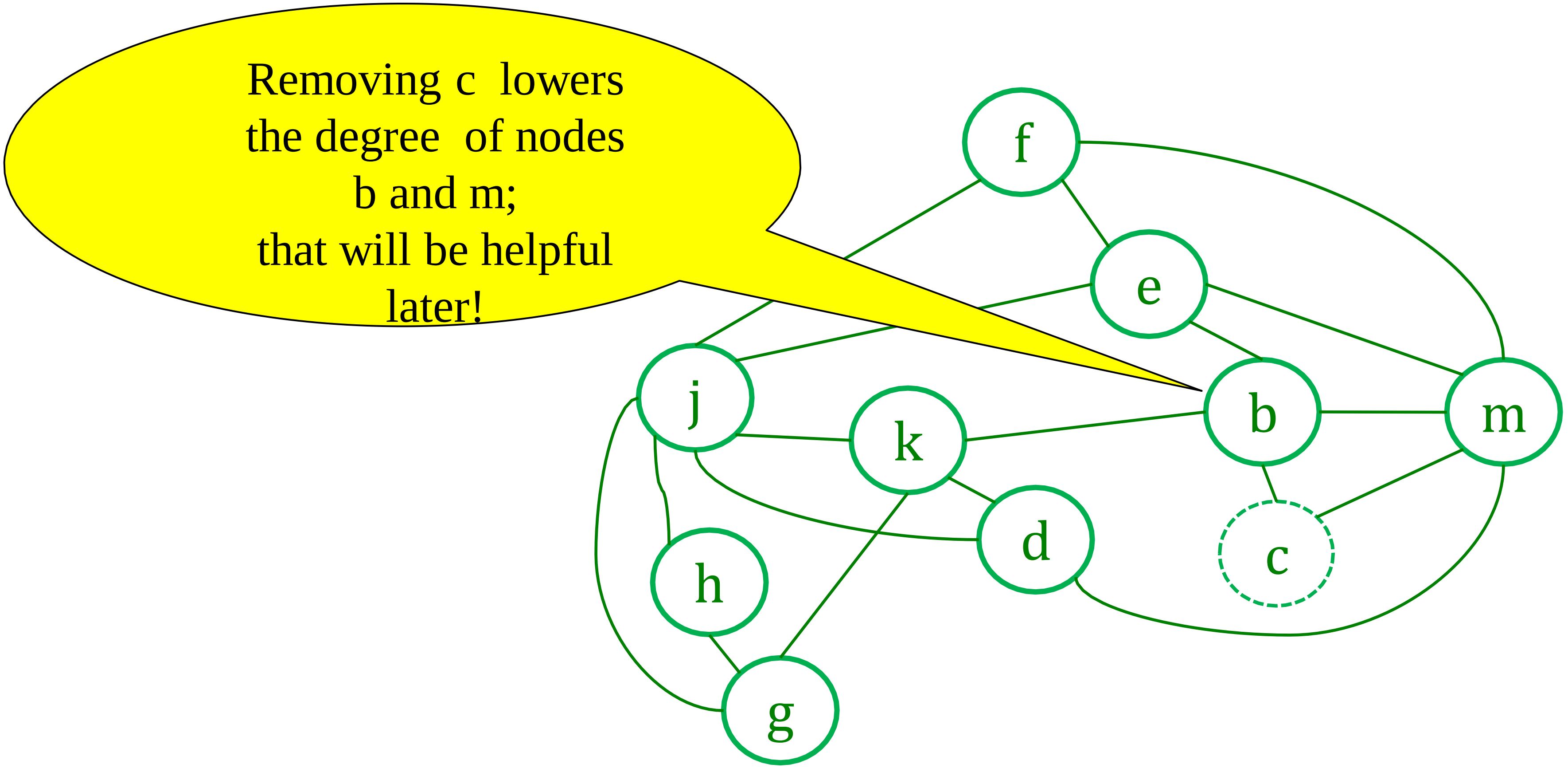
# Example: 3-color this graph




Stack: c

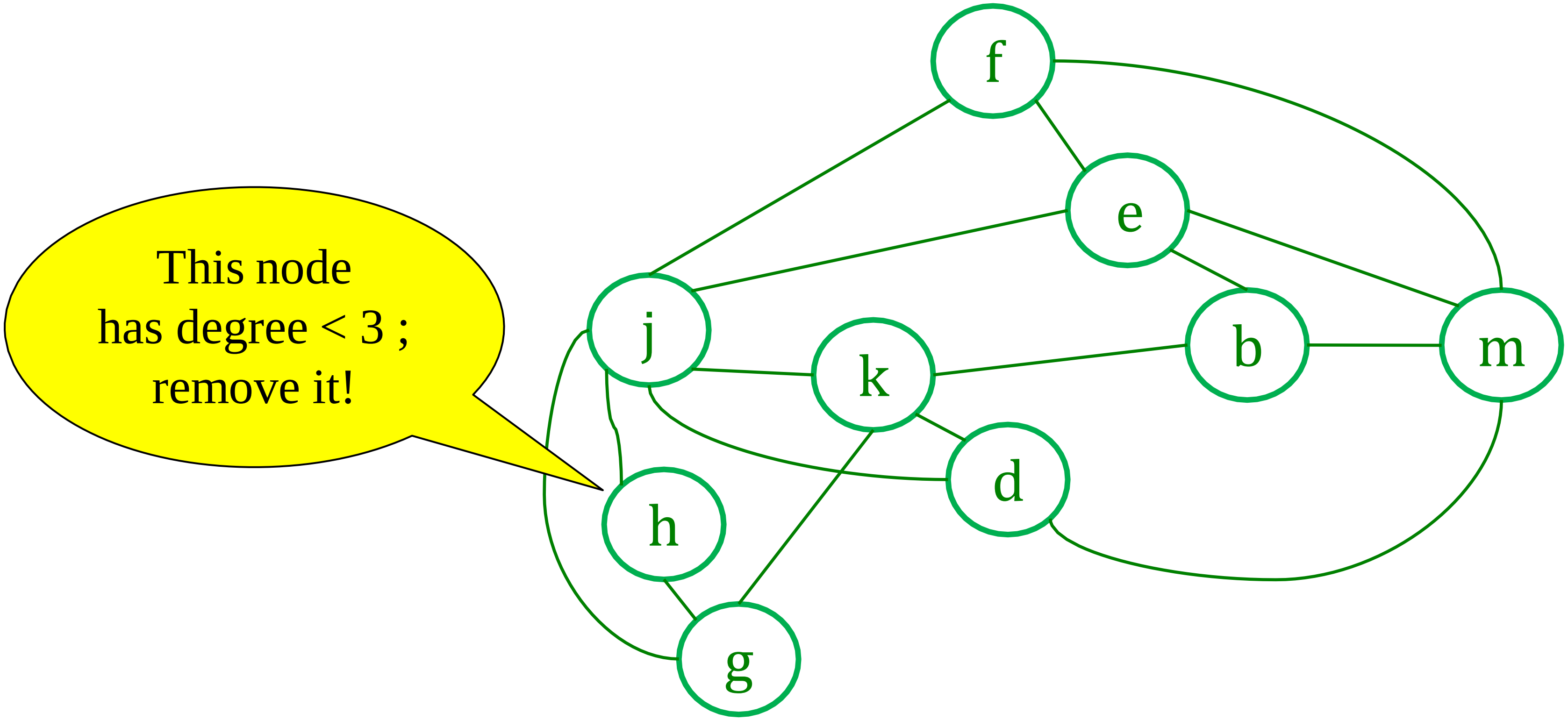
Push node c on the stack

# Example: 3-color this graph



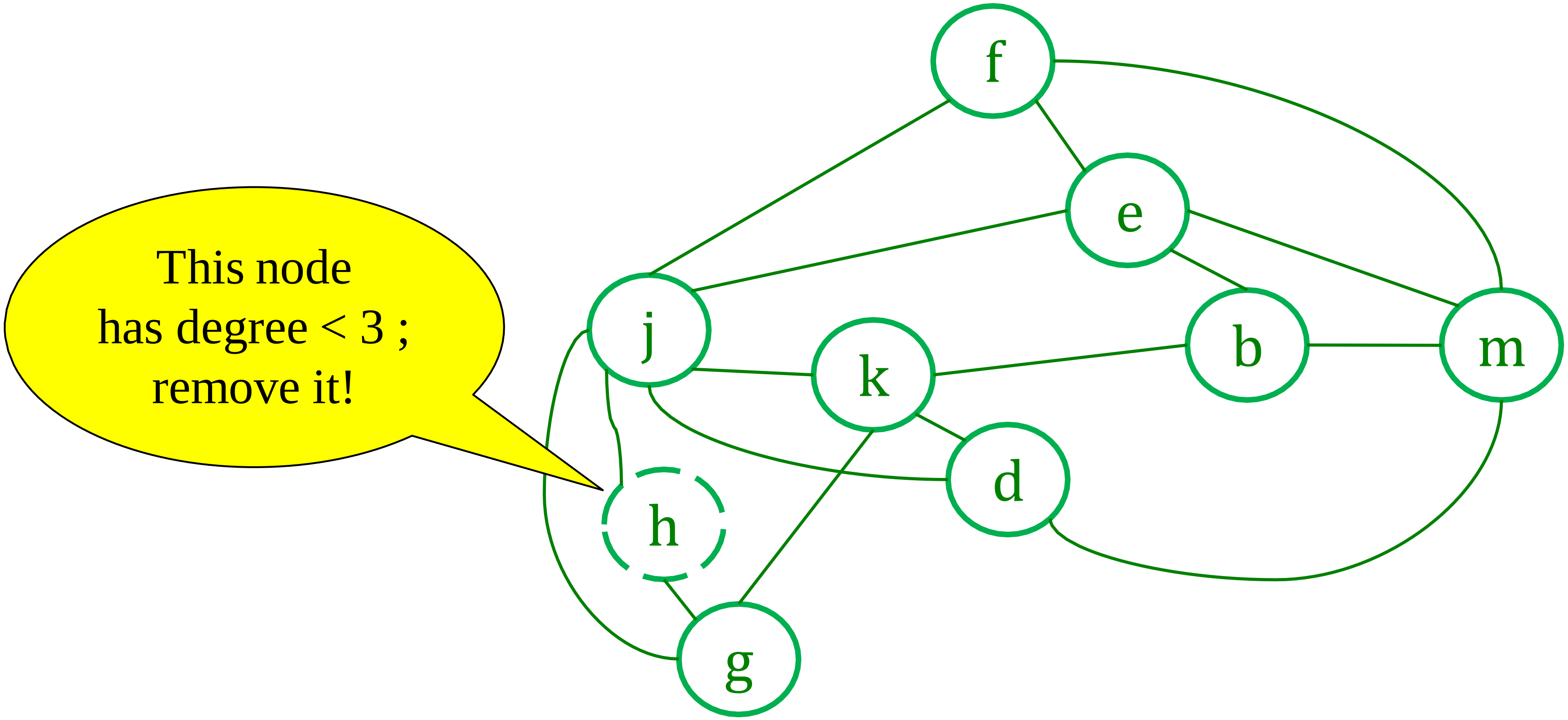

Stack: c

# Example:  3-color this graph



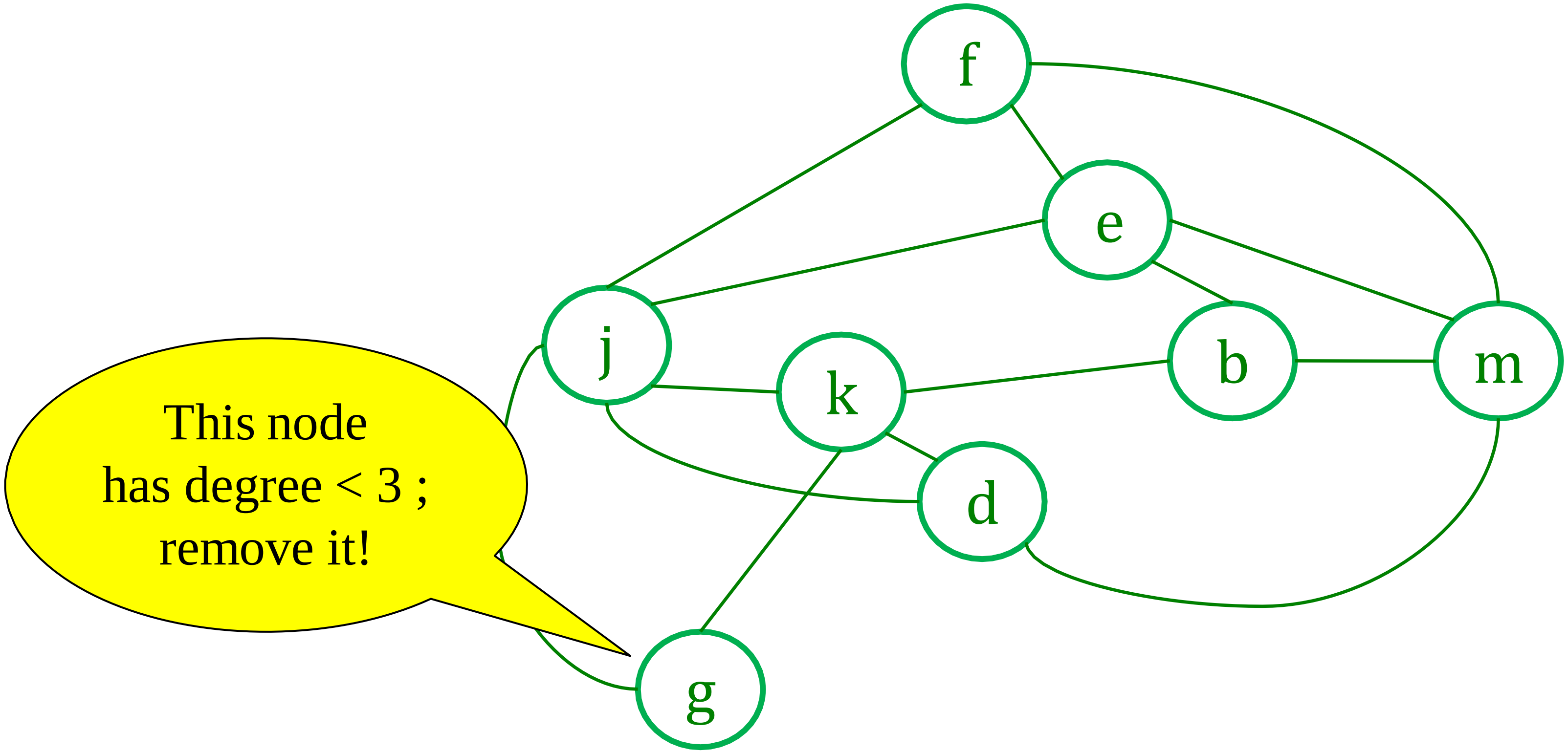

Stack:  c

# Example:  3-color this graph



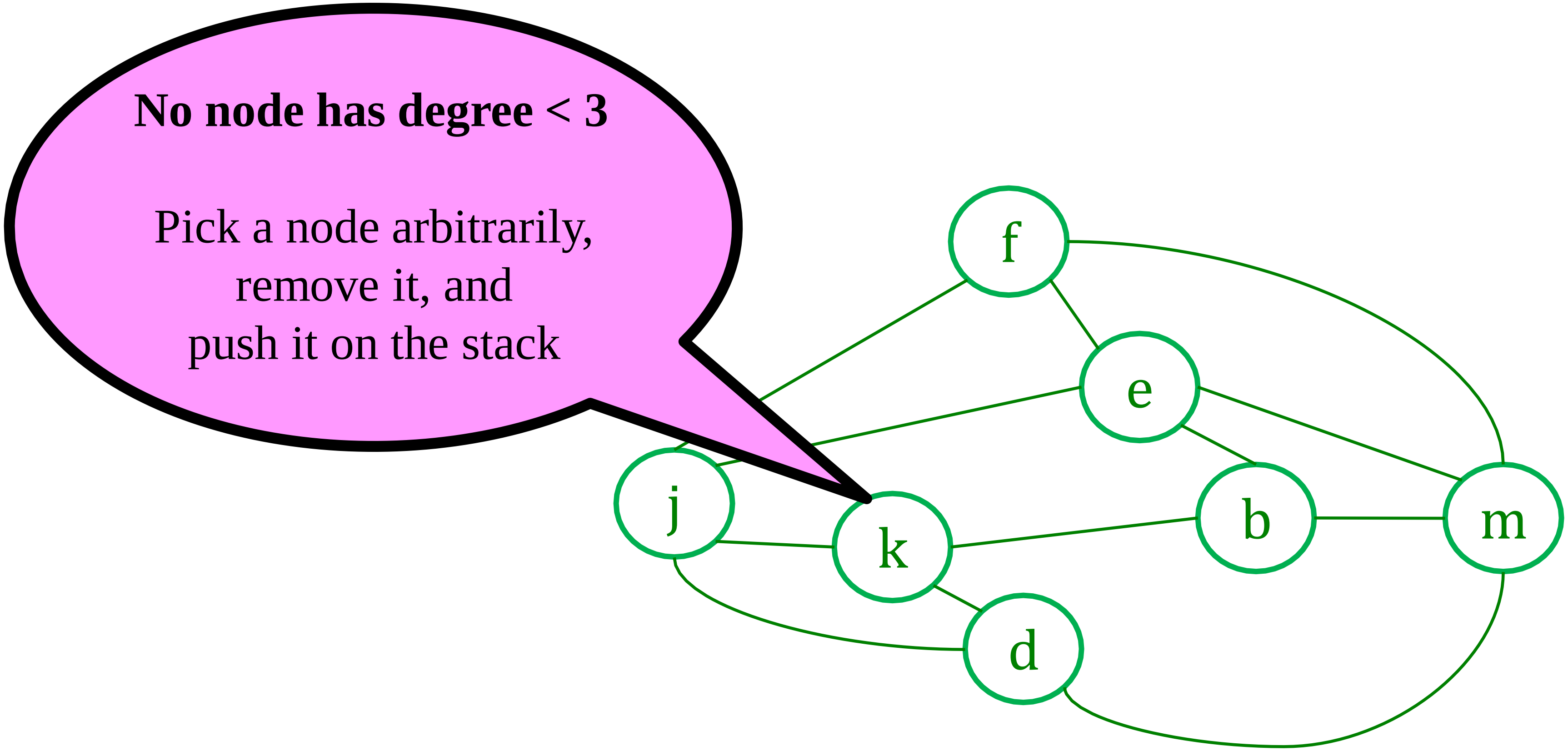

Stack:  h c

# Example: 3-color this graph




Stack: h c

# Example: 3-color this graph

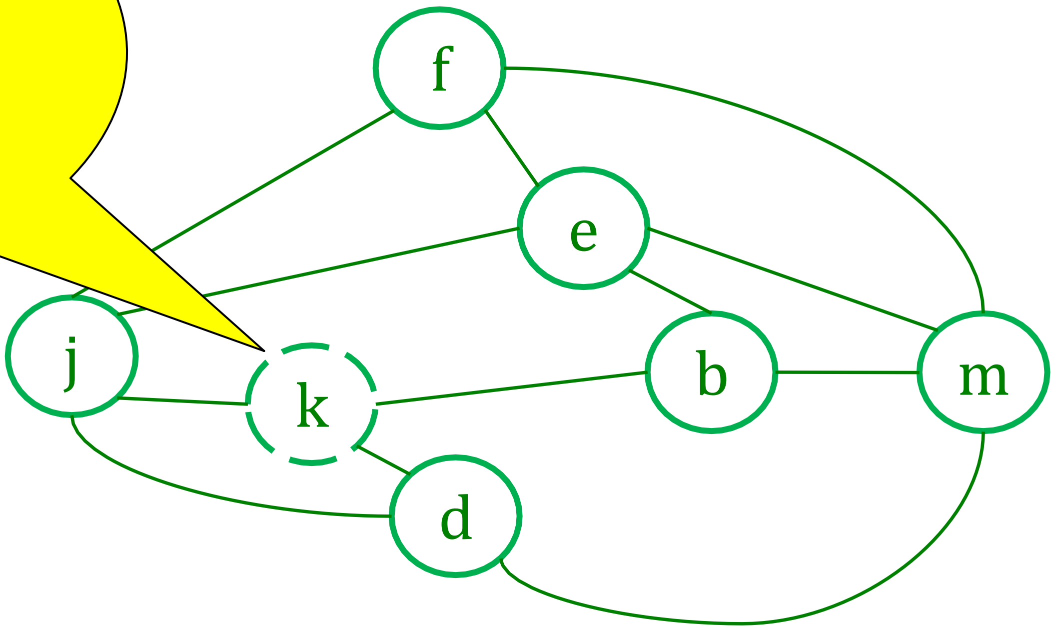



Stack: g h c

# Example: 3-color this graph

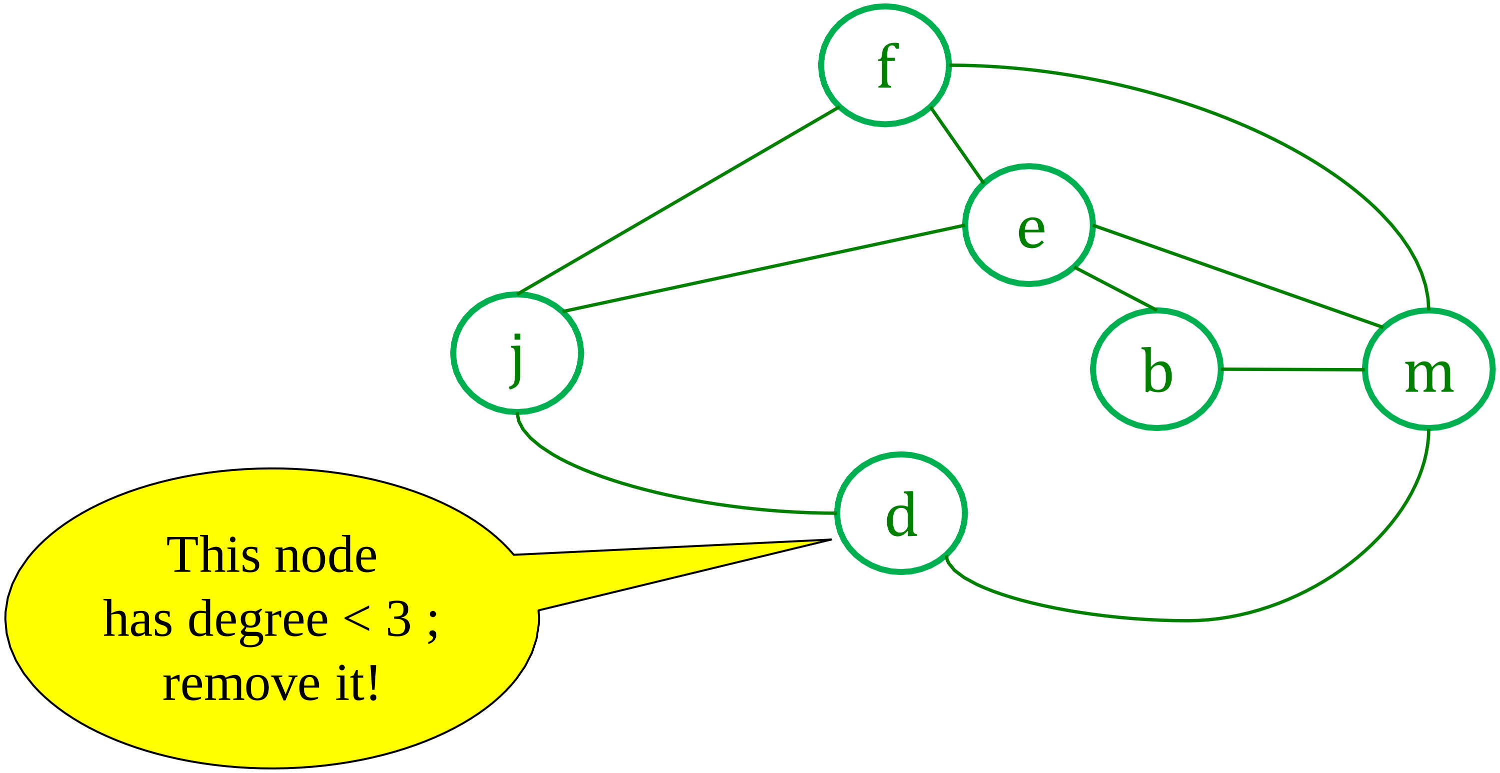
No node has degree < 3

Pick a node arbitrarily,
remove it, and
push it on the stack



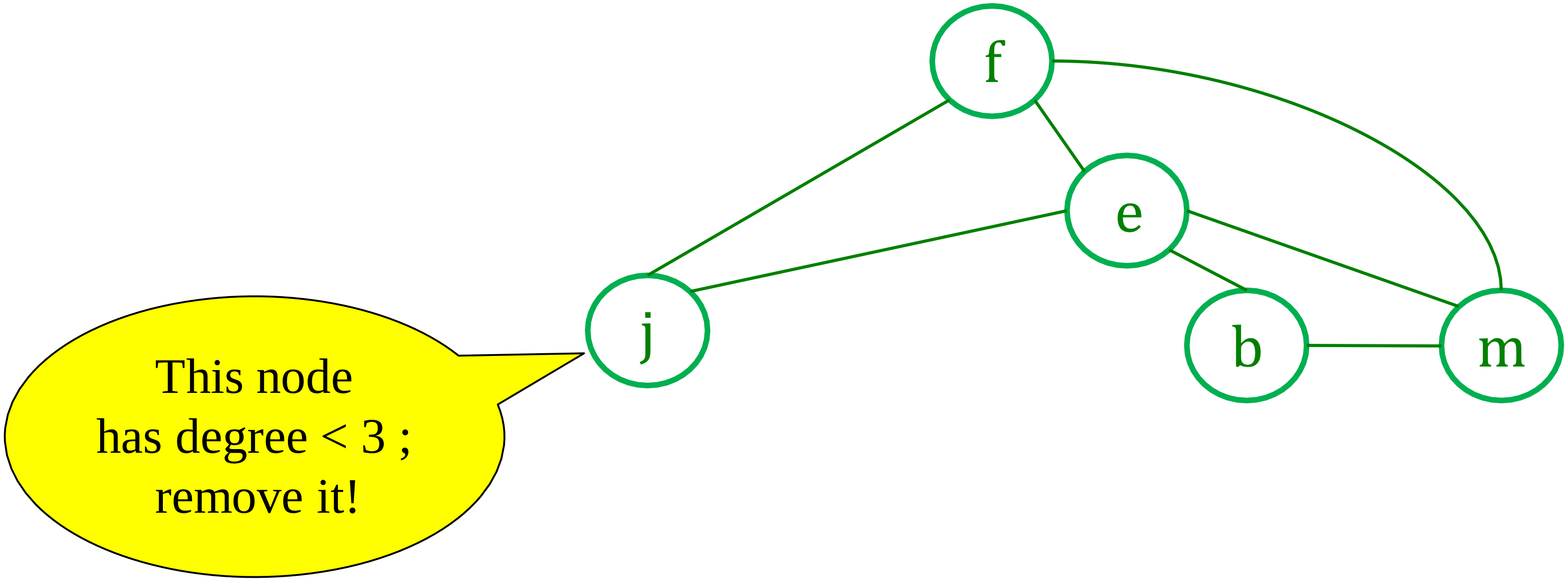

Stack: k g h c

# Example: 3-color this graph



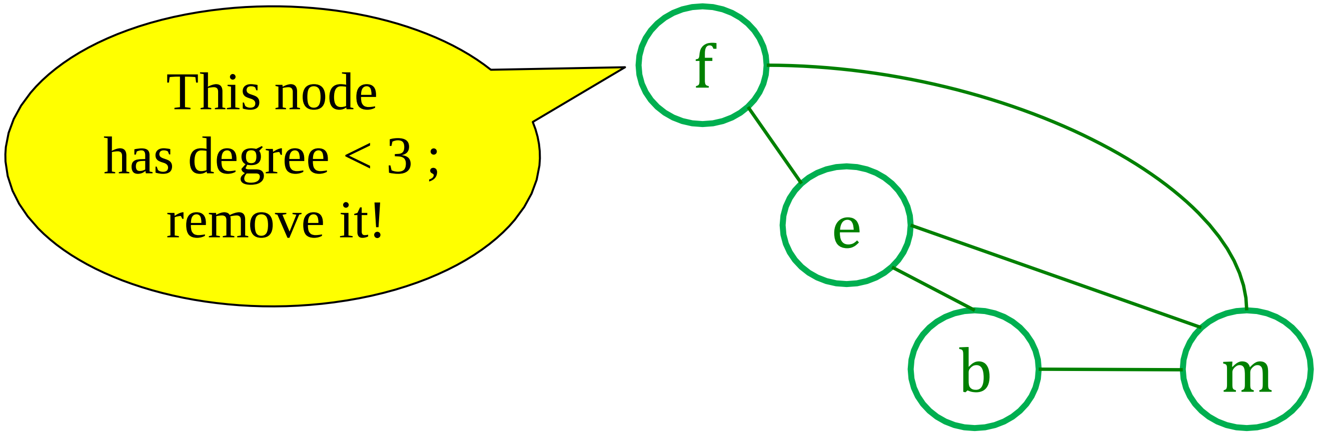

Stack: k g h c

# Example: 3-color this graph



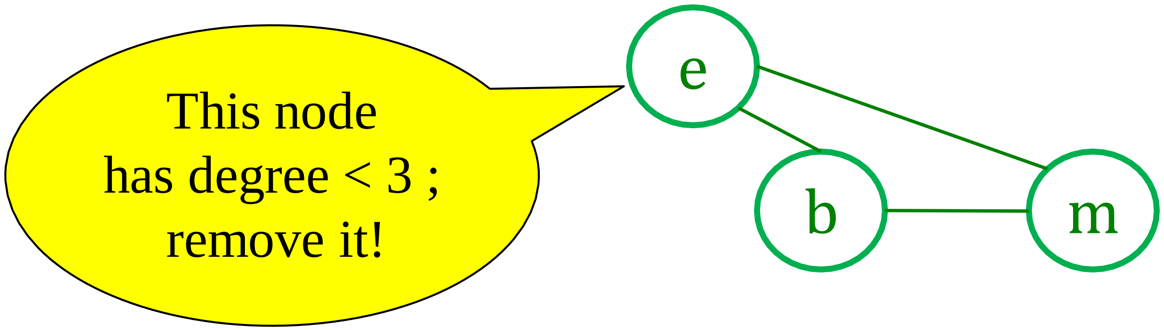

Stack: d k g h c

# Example: 3-color this graph



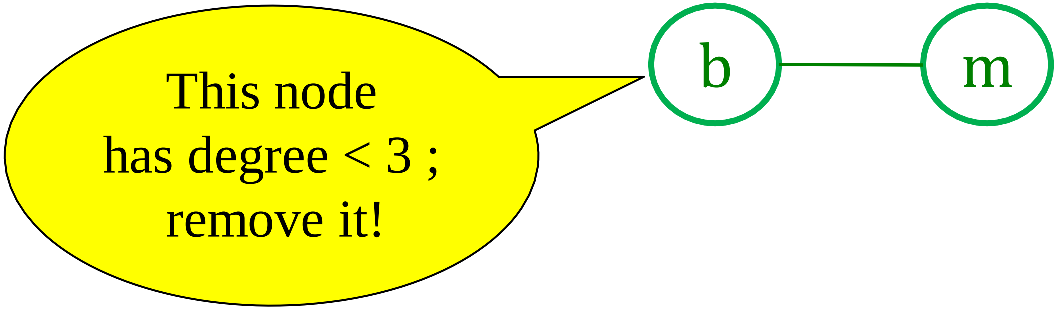

Stack: j d k g h c

# Example: 3-color this graph



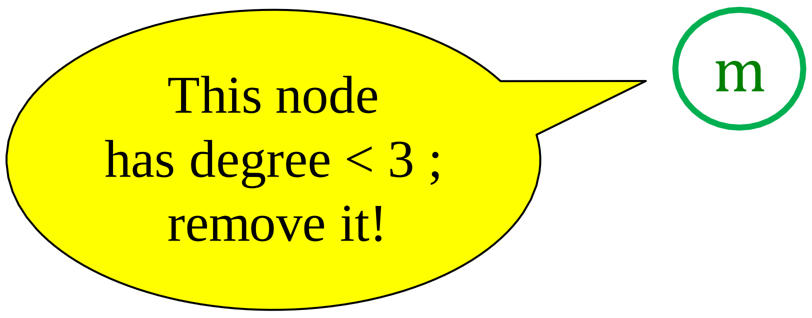

Stack: f j d k g h c

# Example: 3-color this graph




Stack: e f j d k g h c

# Example:  3-color this graph



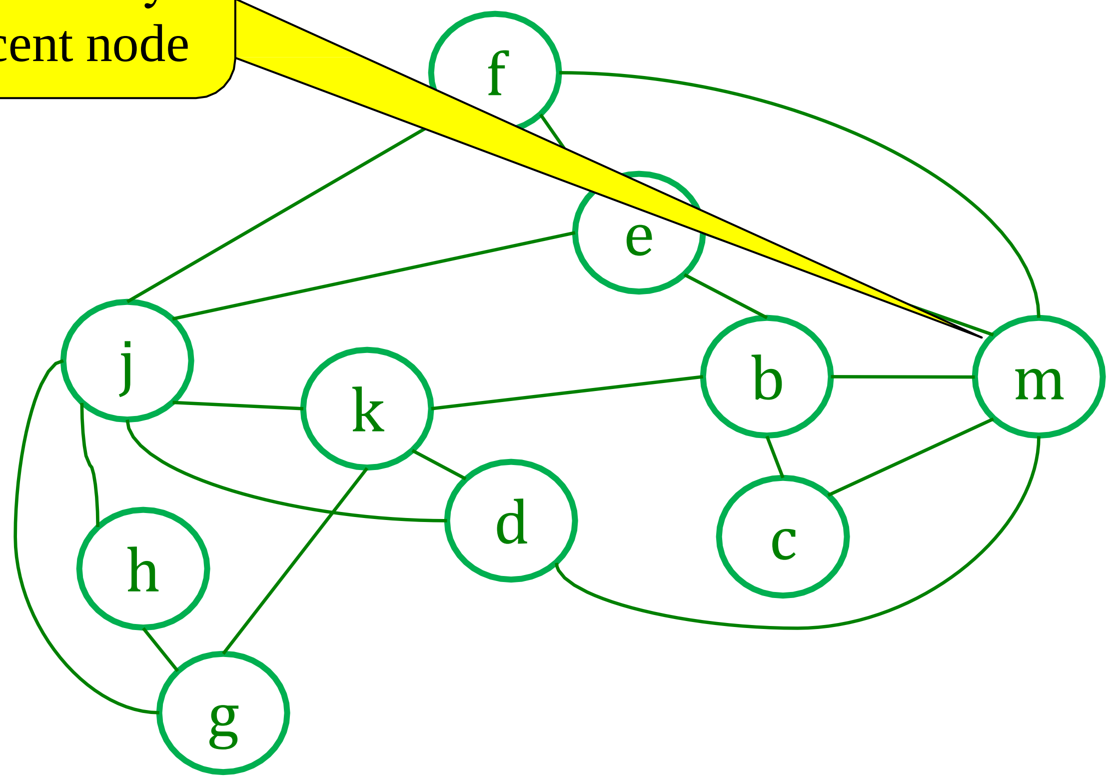

Stack:  b e f j d k g h c

# Example: 3-color this graph

Stack: m b e f j d k g h c

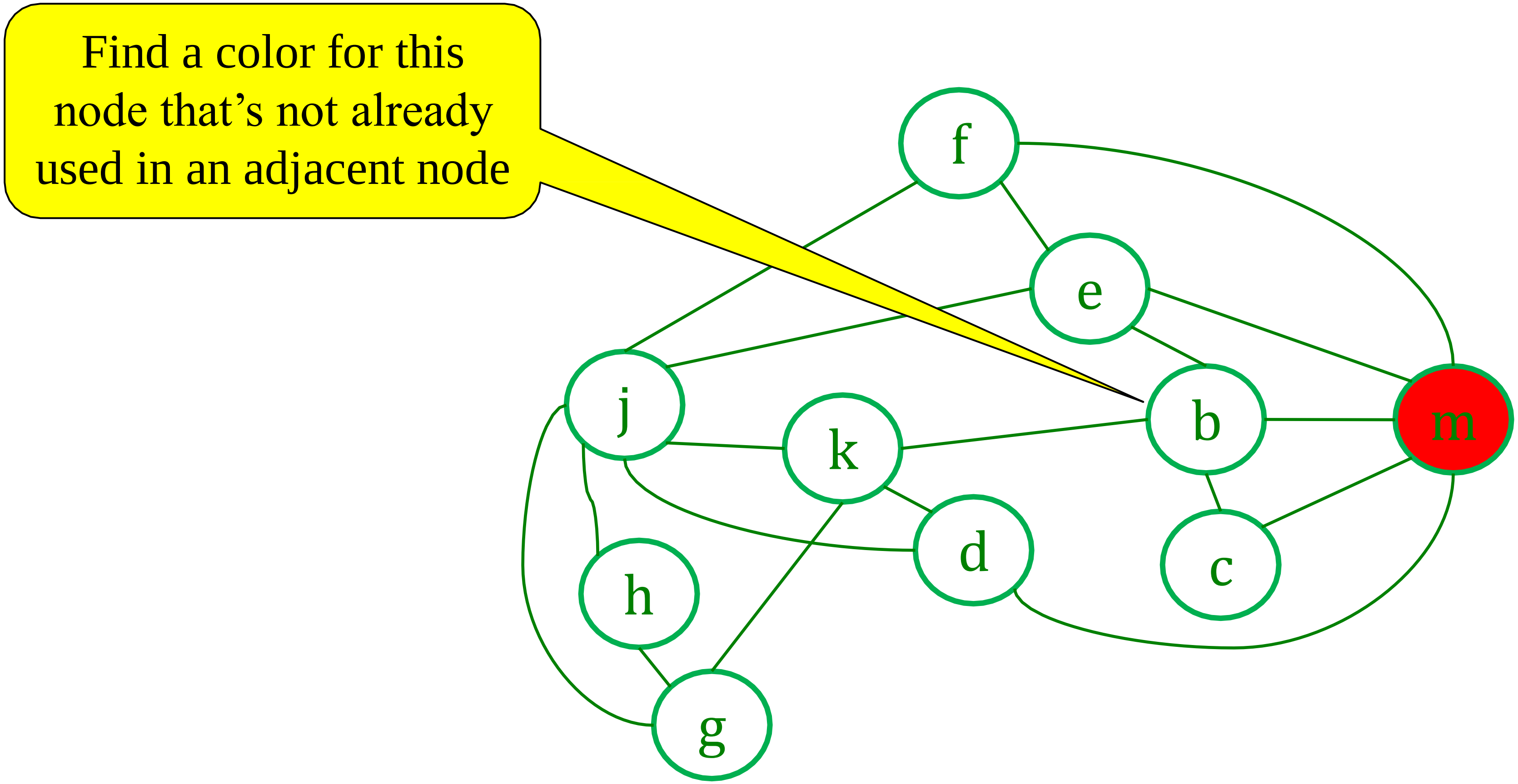
# Now, color the nodes in stack order

Find a color for this node that's not already used in an adjacent node



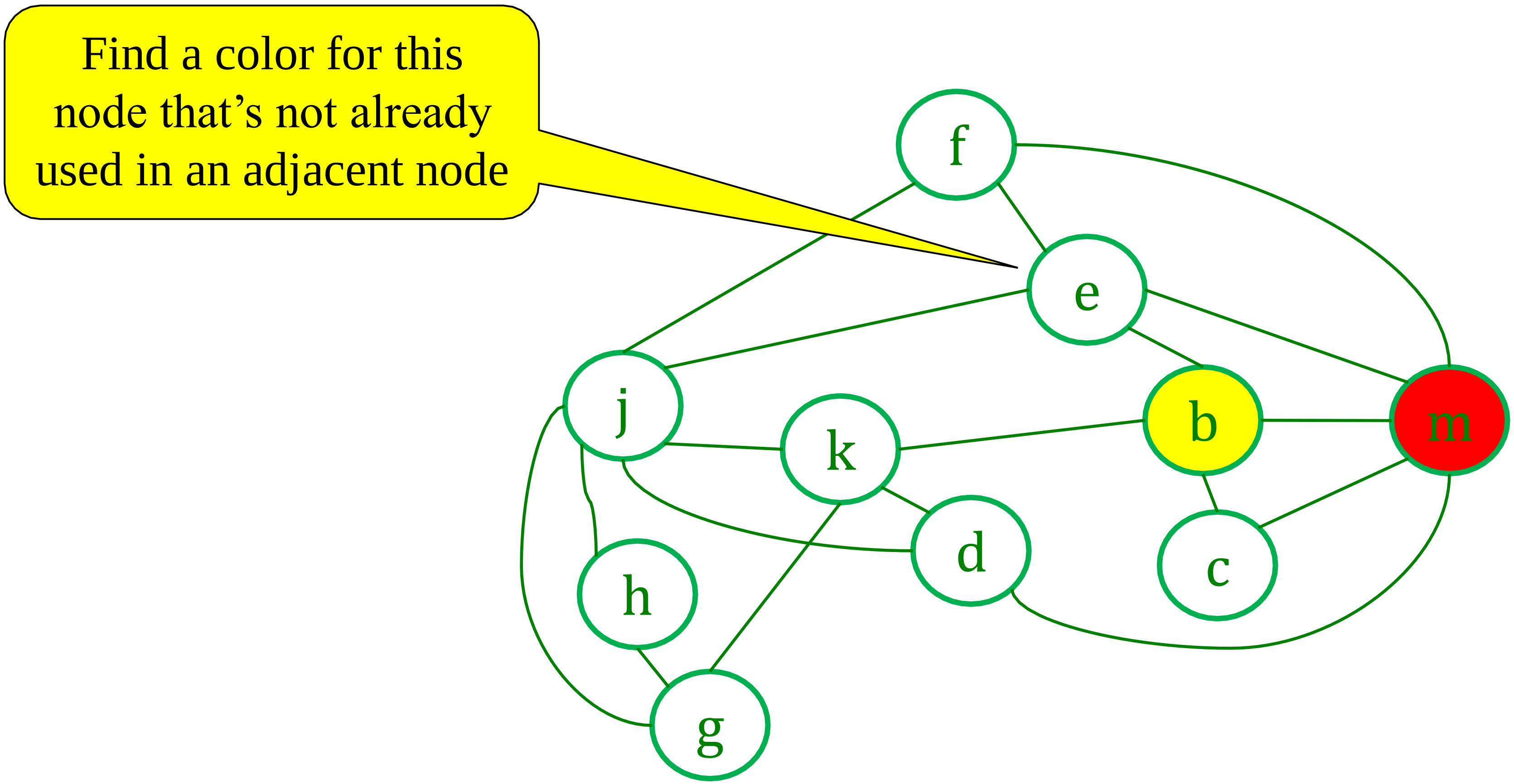

Stack: m b e f j d k g h c

# Now, color the nodes in stack order



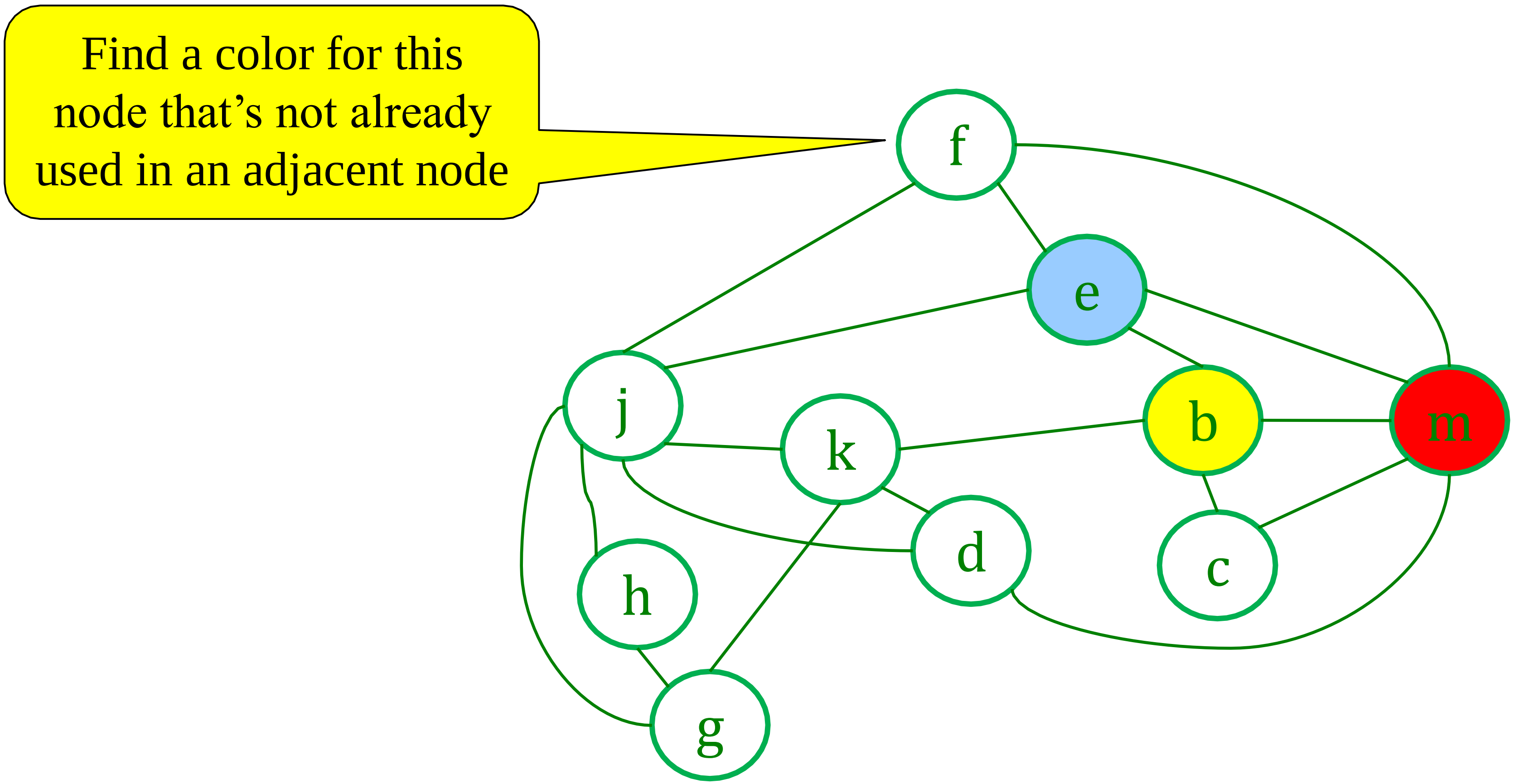

Stack: m b e f j d k g h c

# Now, color the nodes in stack order



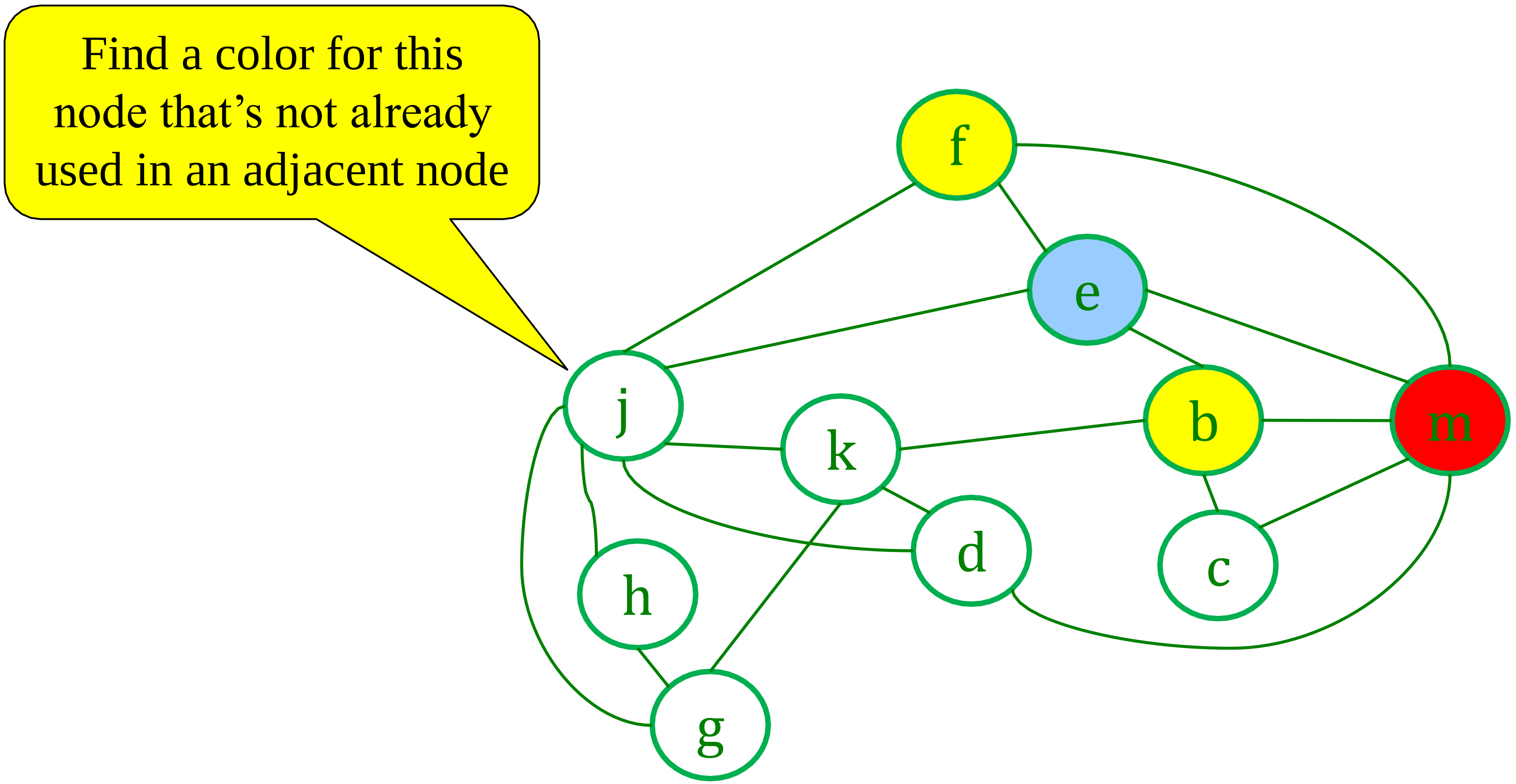

Stack: m b e f j d k g h c

# Now, color the nodes in stack order



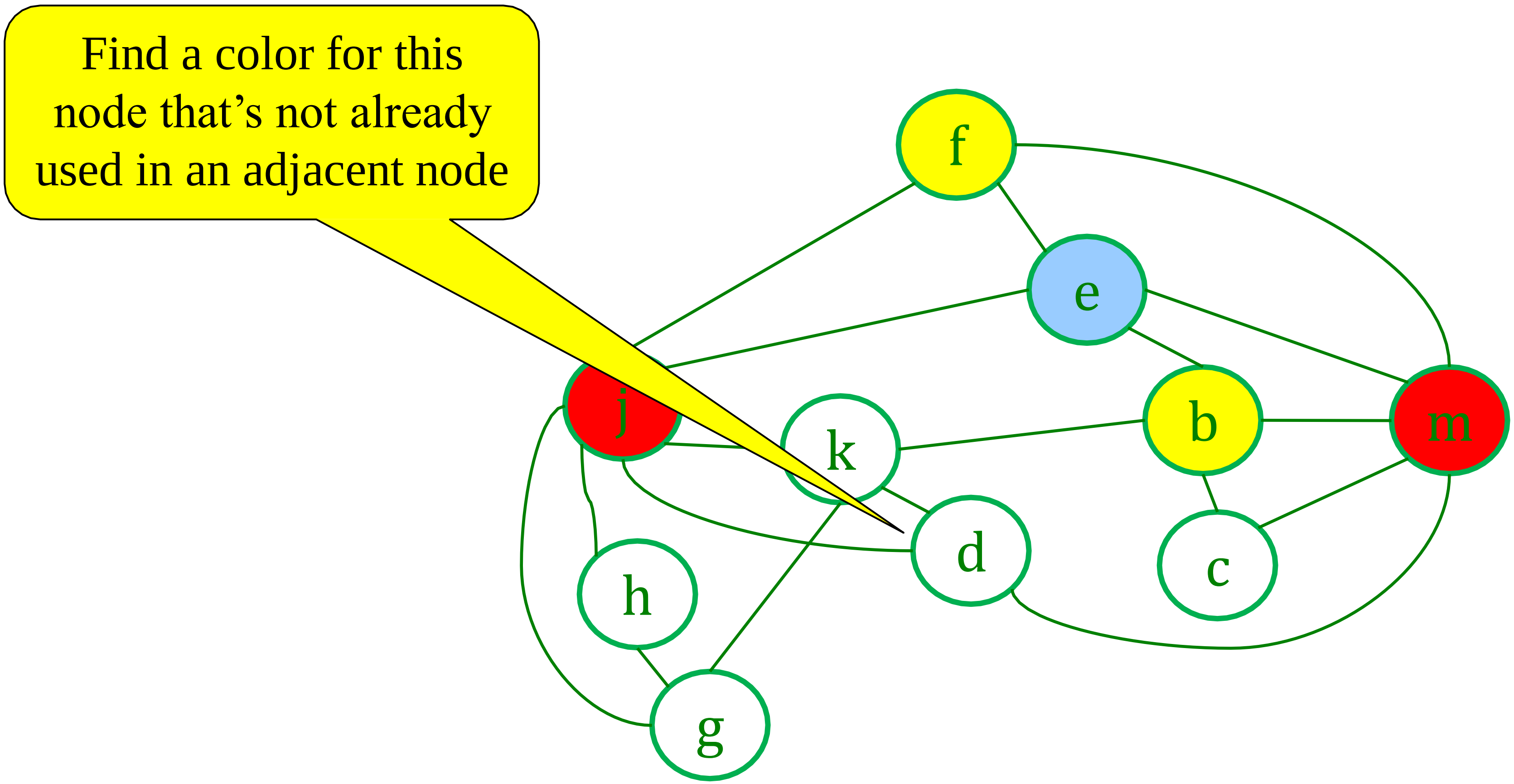

Stack: m b e f j d k g h c

# Now, color the nodes in stack order




Stack: m b e f j d k g h c

# Now, color the nodes in stack order

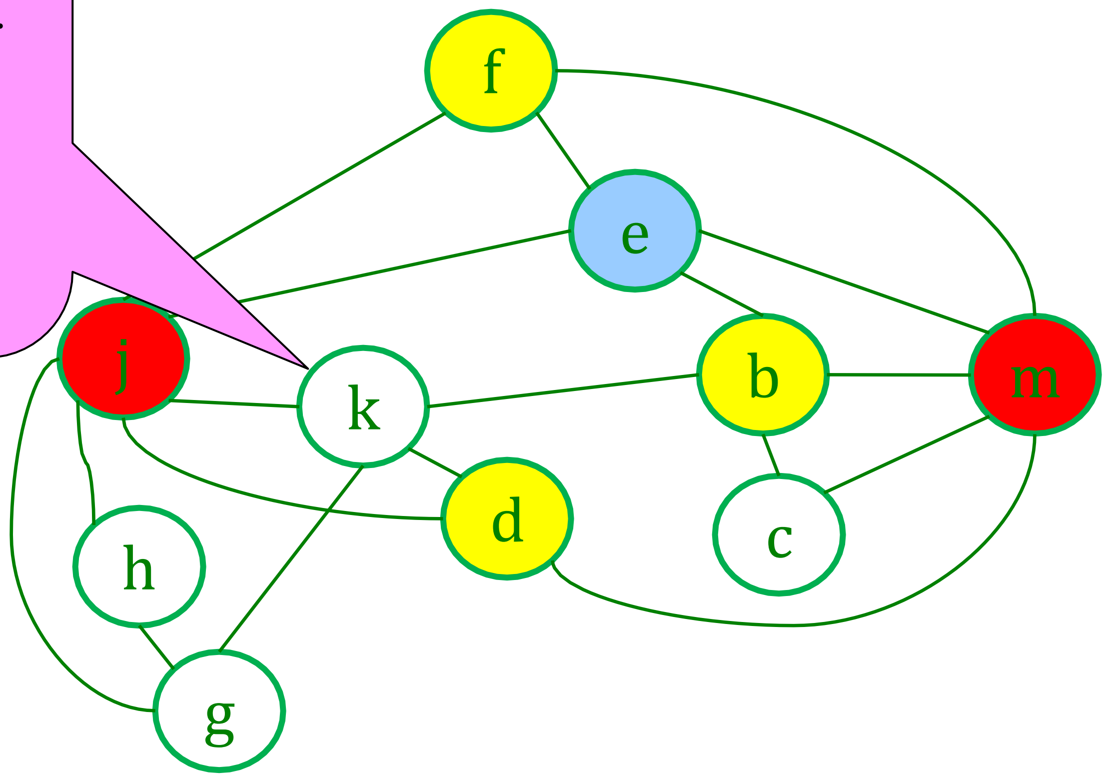



Stack: m b e f j d k g h c

# Now, color the nodes in stack order

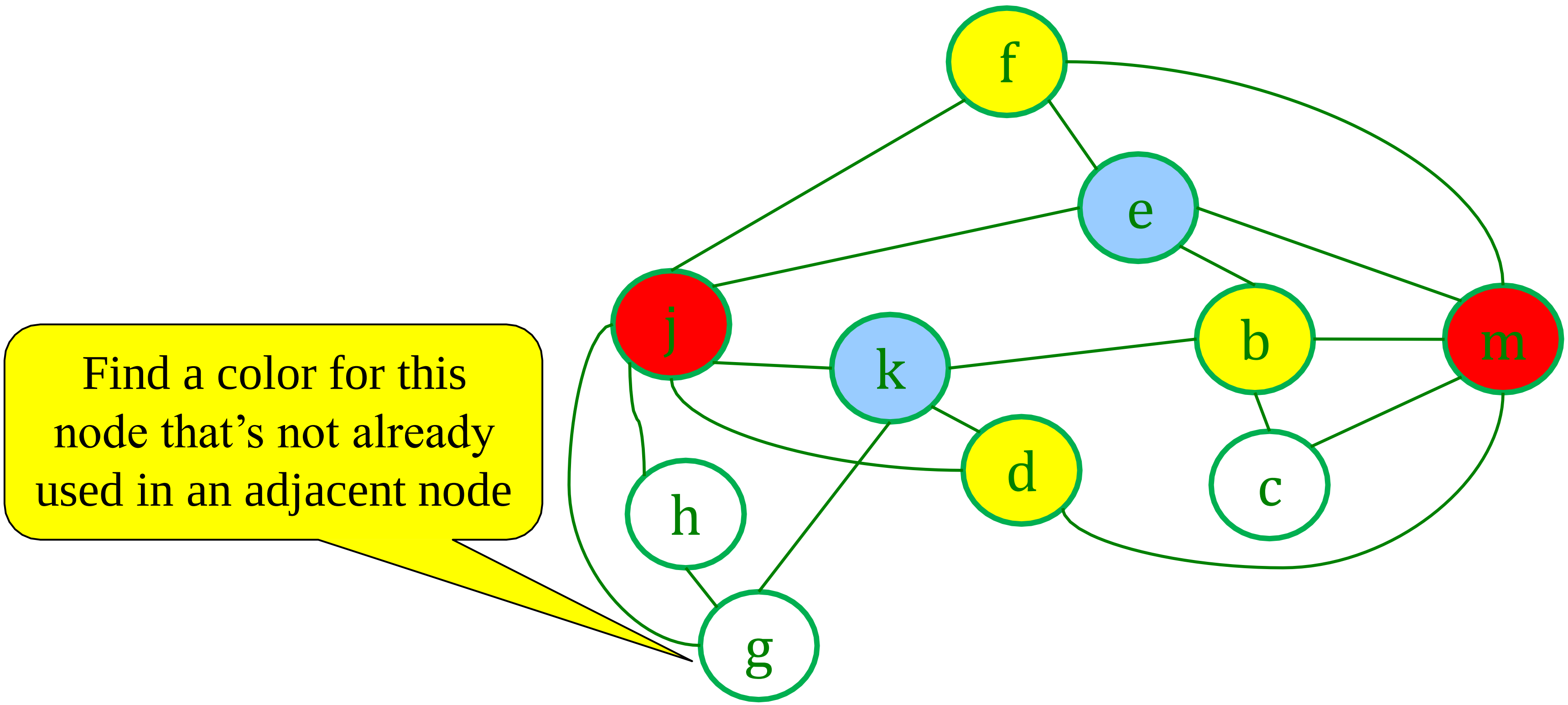
We're about to color node k.
This was the only one that was degree ≥ 3 when we removed it.
Hence, it is not guaranteed that we can find a color for it now.

But we got lucky, because
b and d have the same color!



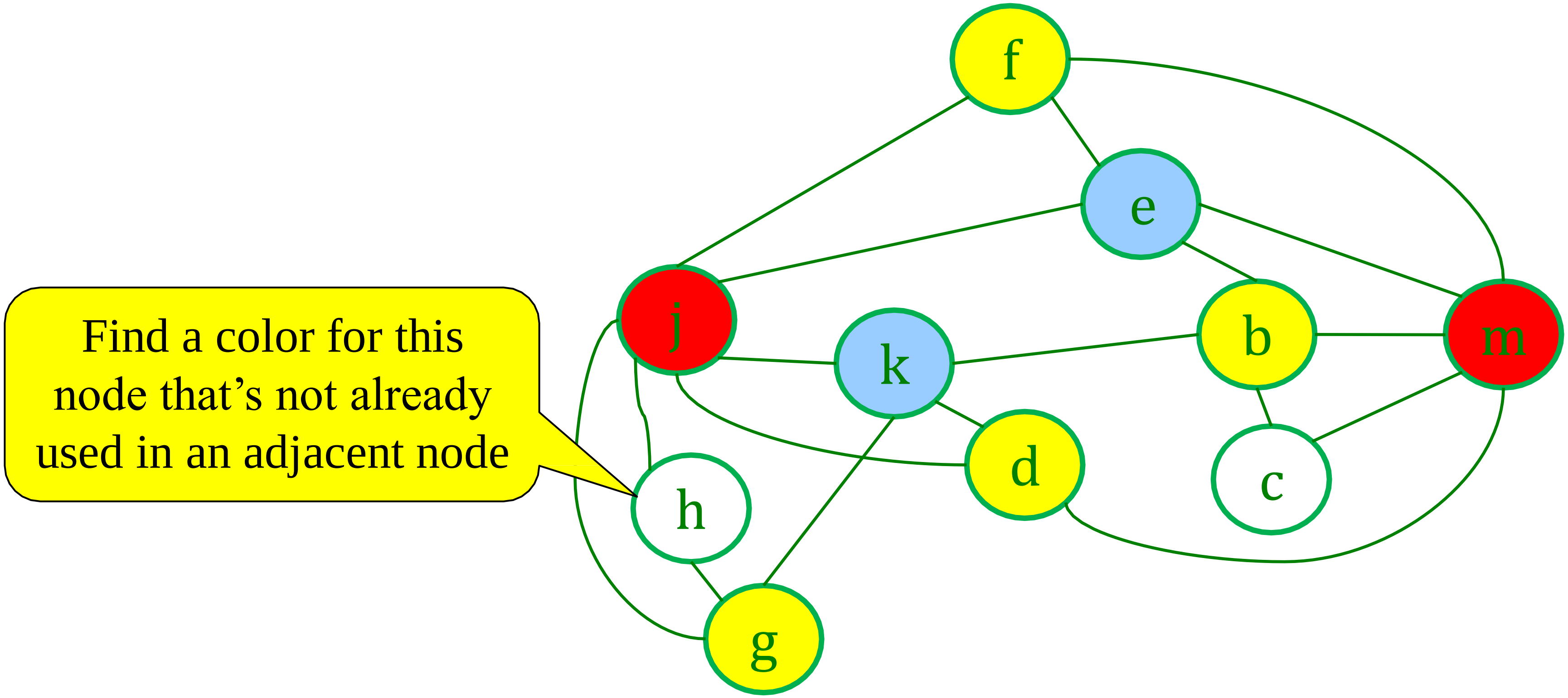

Stack: m b e f j d k g h c

# Now, color the nodes in stack order



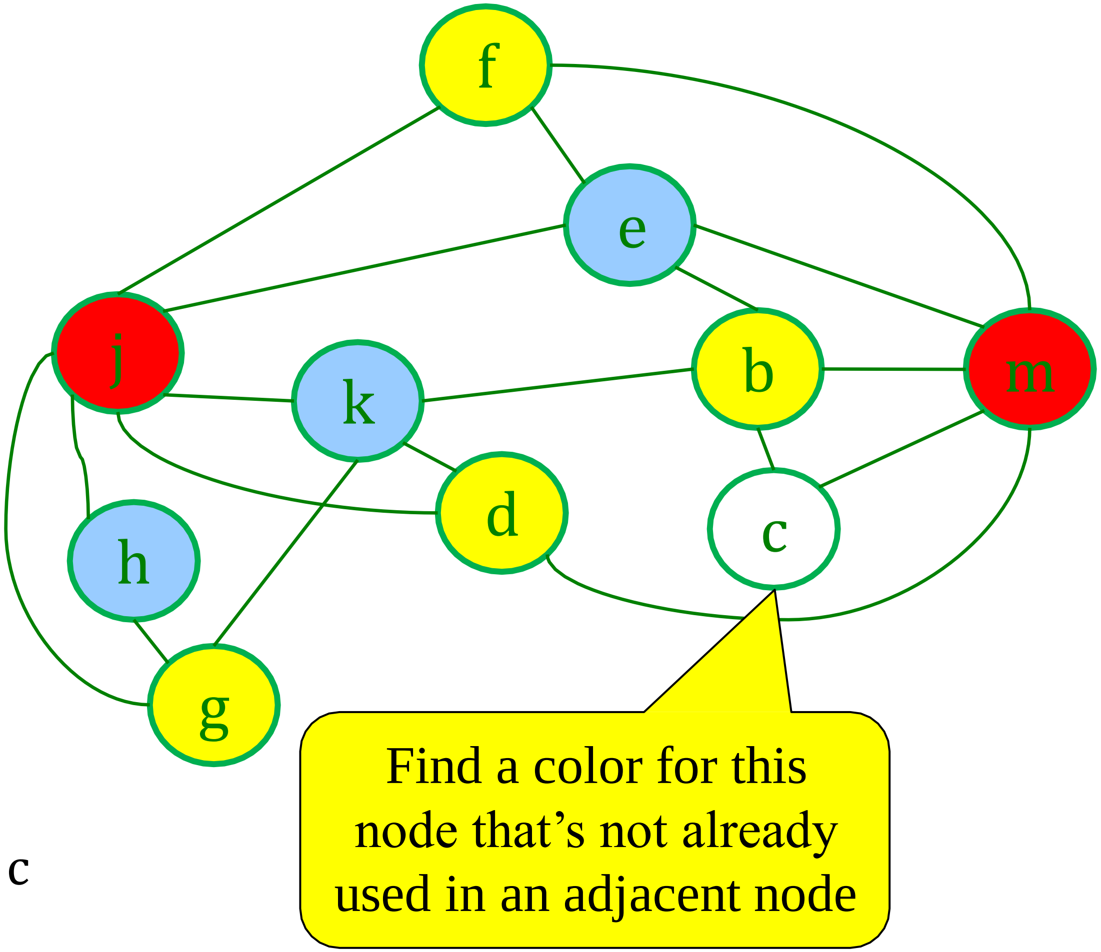

Stack: m b e f j d k g h c

# Now, color the nodes in stack order




Stack: m b e f j d k g h c

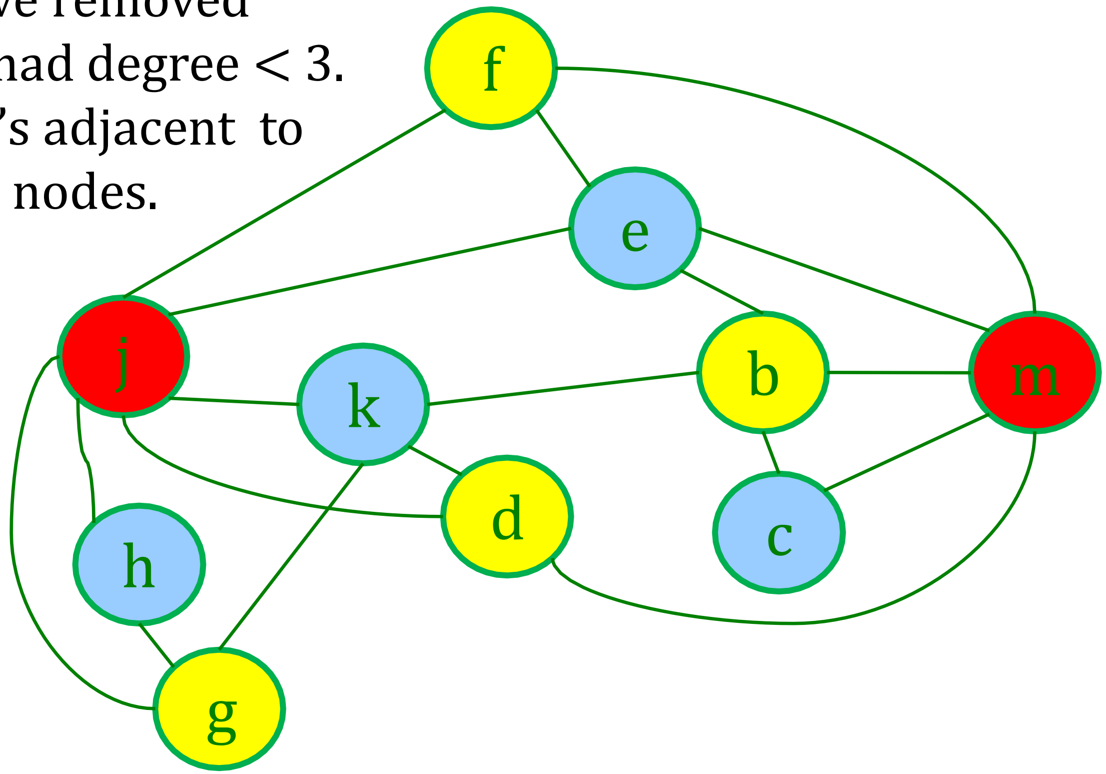
# Now, color the nodes in stack order




Stack: ~~m b e f j d k g h~~ c

# Now, color the nodes in stack order

Why did this work?
Because (usually) when we removed each node, at that time it had degree < 3. So when we put it back, it's adjacent to at most 2 already-colored nodes.

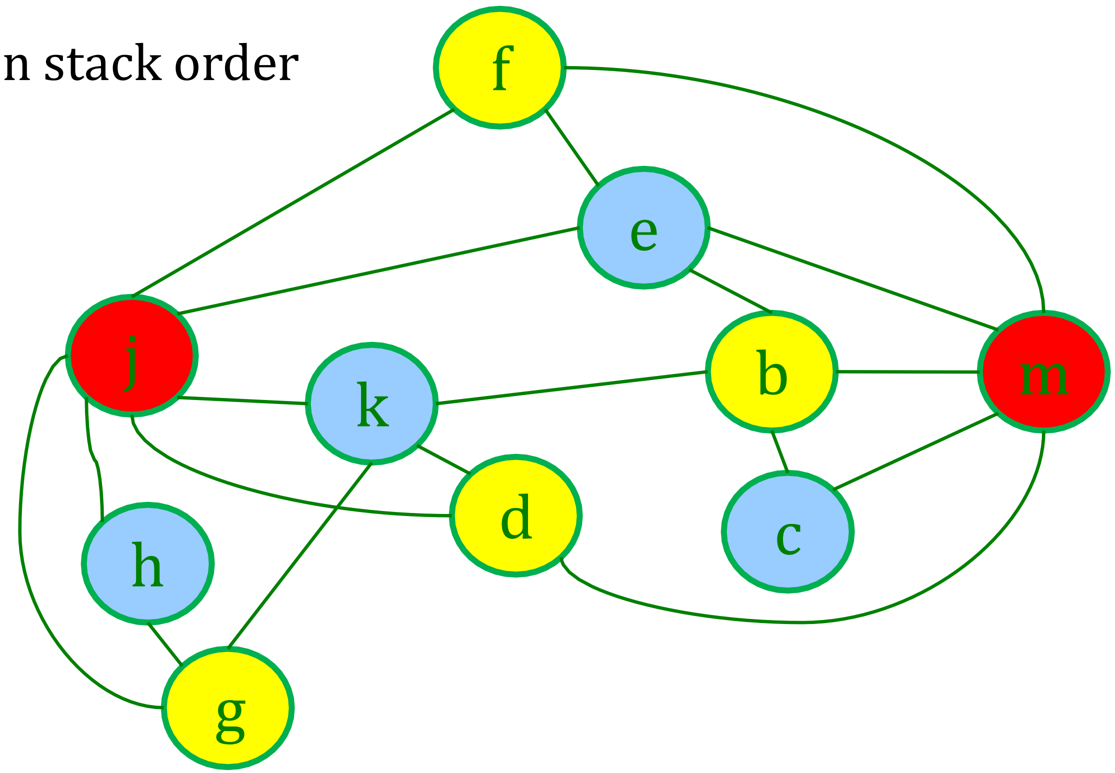



Stack: ~~m b e f j d k g h c~~

# Two-phase algorithm:

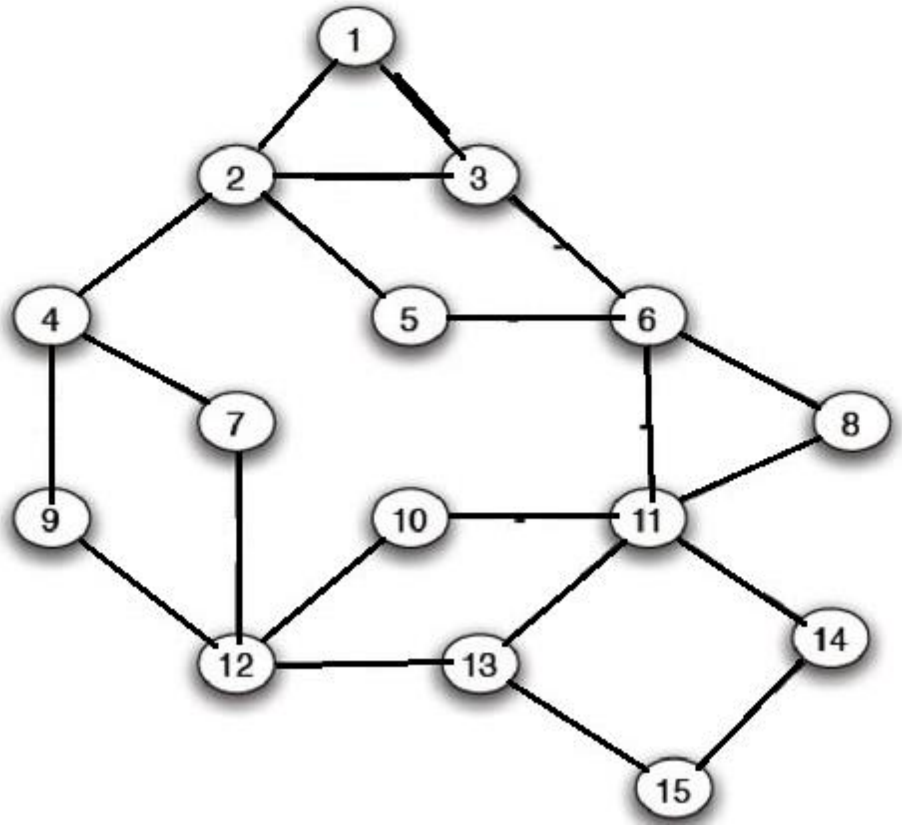
Phase 1: list the nodes in some order ("the stack")

Phase 2: color the nodes in stack order




Stack: ~~m b e d k j f h g c~~

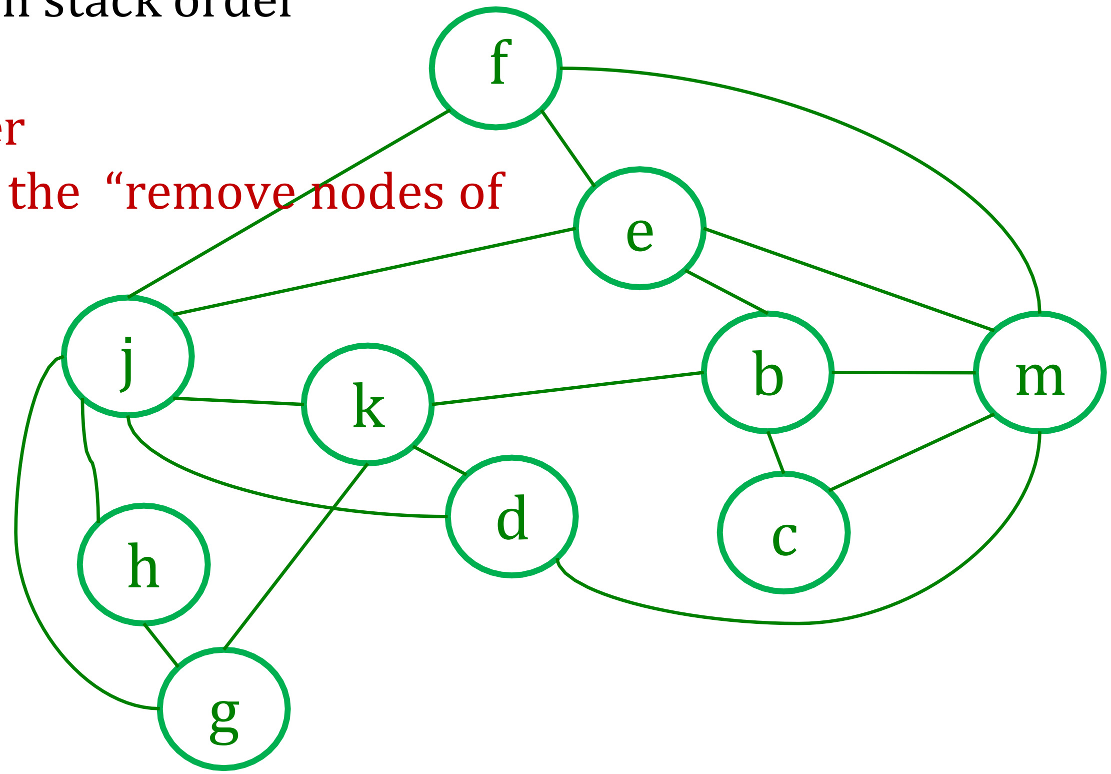
# Graph Coloring: Exercise



# Two phase algorithm

Phase 1: list the nodes in some order ("the stack")
Phase 2: color the nodes in stack order

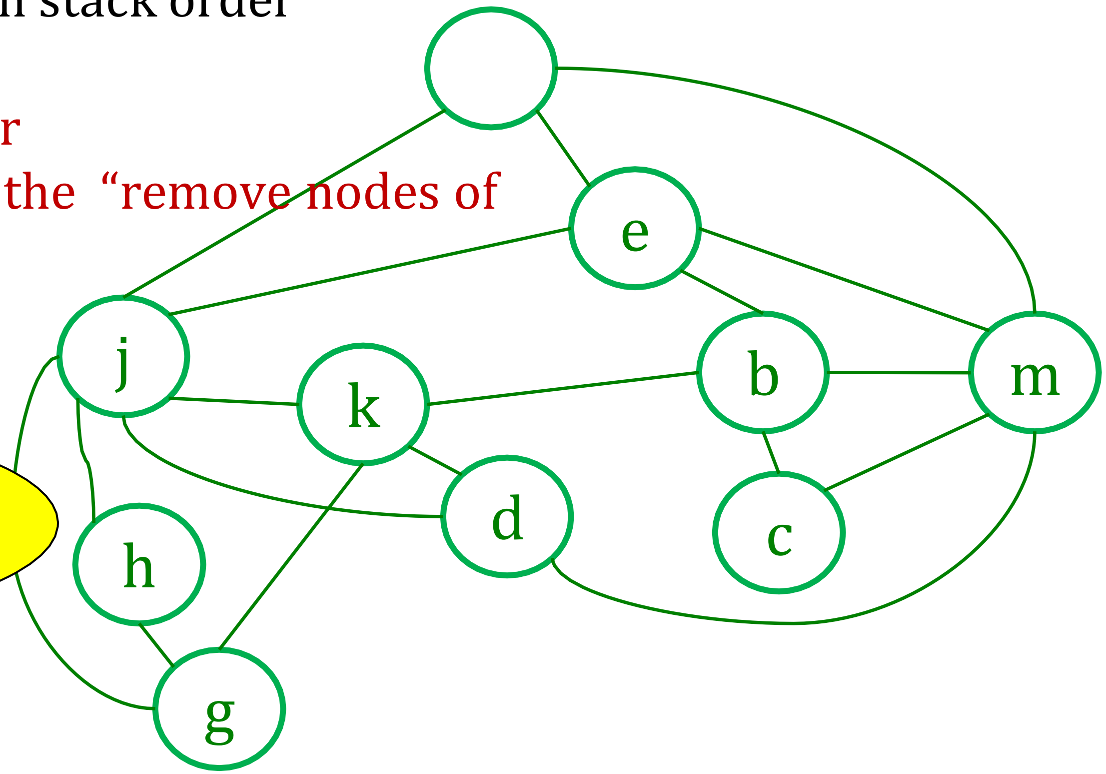
What if we use some other coloring order, instead of the "remove nodes of low-degree" order?




Coloring order:

# Two phase algorithm

Phase 1: list the nodes in some order ("the stack")
Phase 2: color the nodes in stack order
f
What if we use some other coloring order, instead of the "remove nodes of low-degree" order?



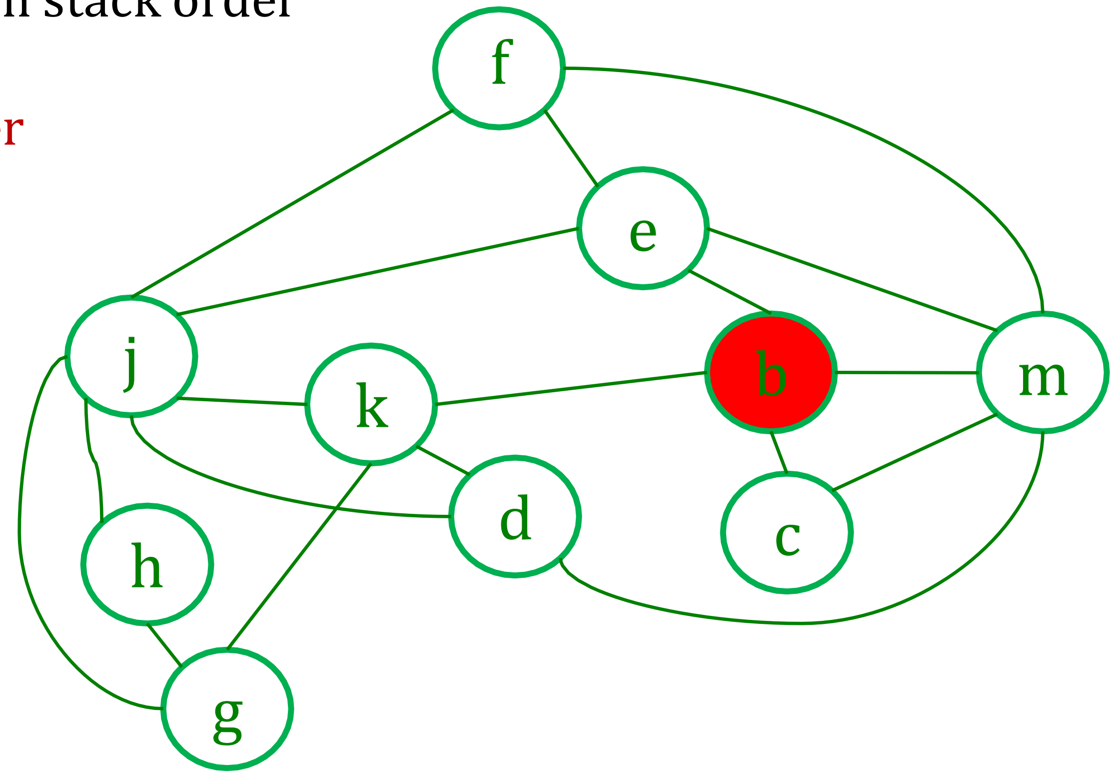

Just for fun, let's use alphabetical order.

Coloring order: b c d e f g h j k m

# Two phase algorithm

Phase 1: list the nodes in some order ("the stack")
Phase 2: color the nodes in stack order

What if we use some other coloring order?

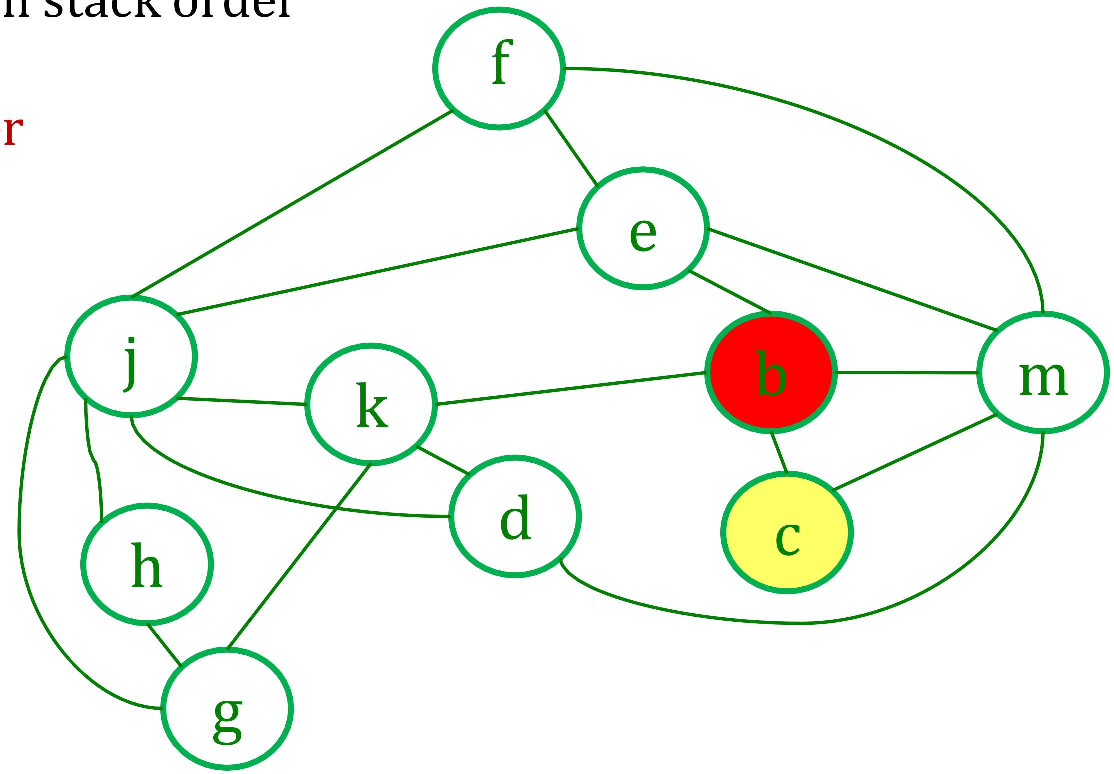



Coloring order: b c d e f g h j k m

# Two phase algorithm

Phase 1: list the nodes in some order ("the stack")
Phase 2: color the nodes in stack order

What if we use some other coloring order?

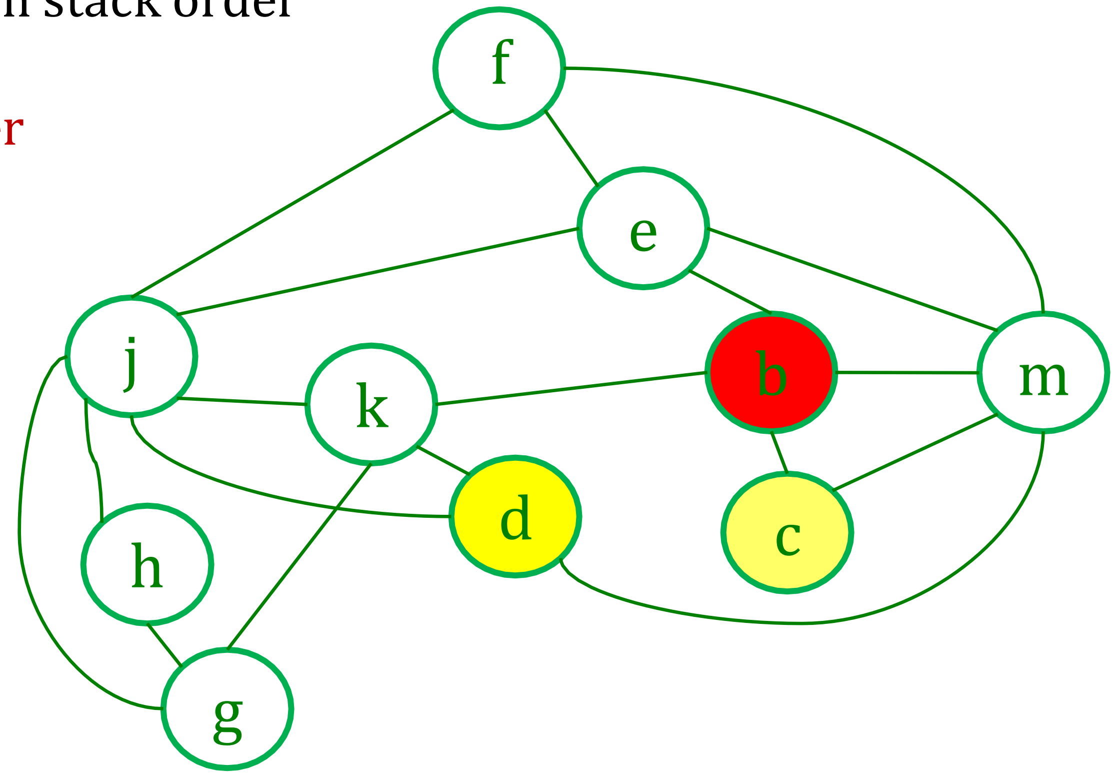



Coloring order: b c d e f g h j k m

# Two phase algorithm

Phase 1: list the nodes in some order ("the stack")
Phase 2: color the nodes in stack order

What if we use some other coloring order?

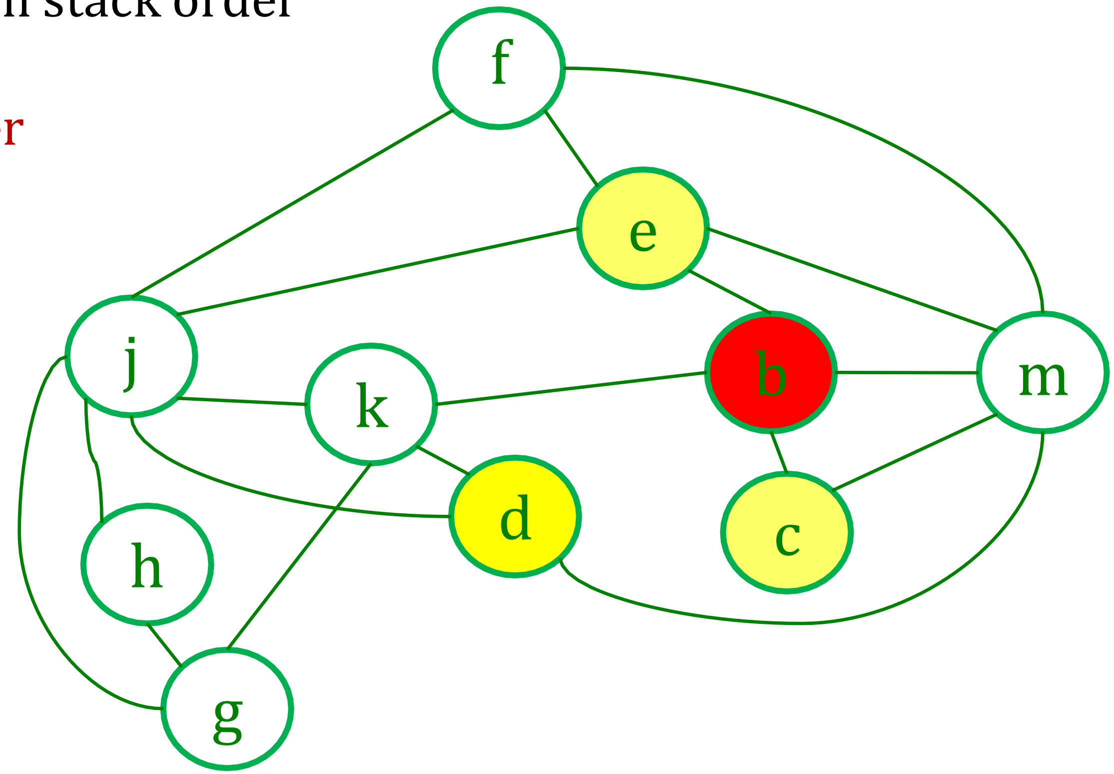



Coloring order: b c d e f g h j k m

# Two phase algorithm

Phase 1: list the nodes in some order ("the stack")
Phase 2: color the nodes in stack order

What if we use some other coloring order?

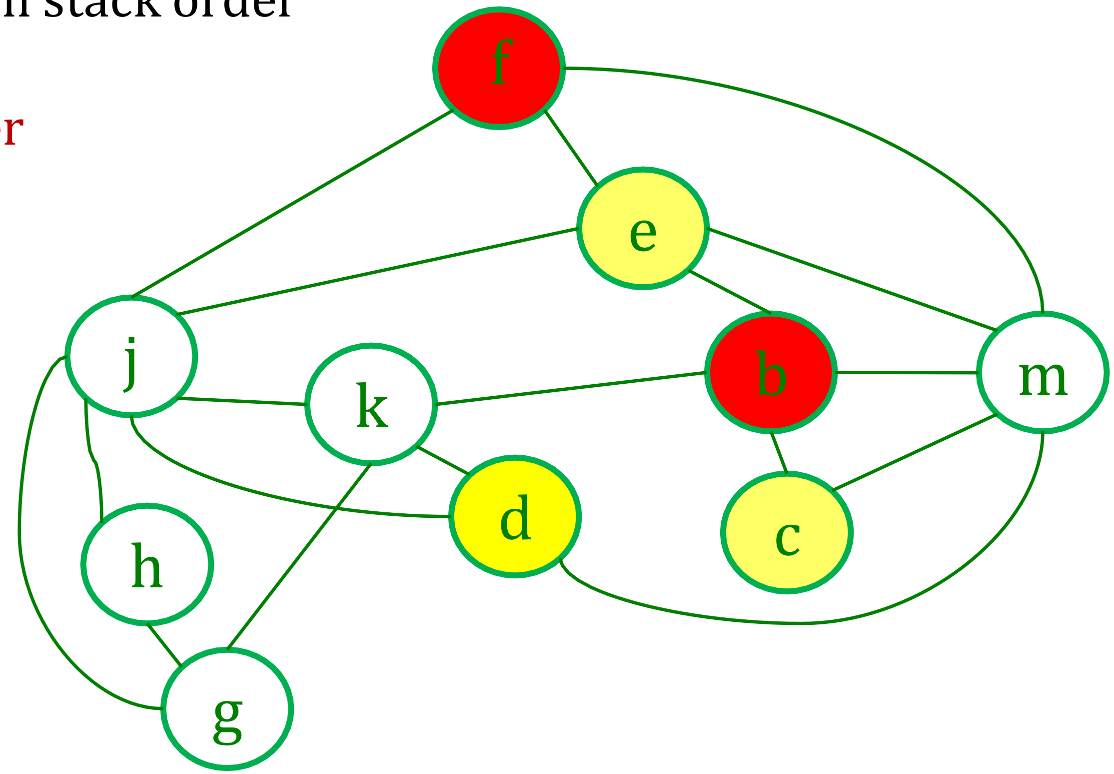



Coloring order: b c d e f g h j k m

# Two phase algorithm

Phase 1:  list the nodes in some order (“the stack”)
Phase 2: color the nodes in stack order

What if we use some other coloring order?

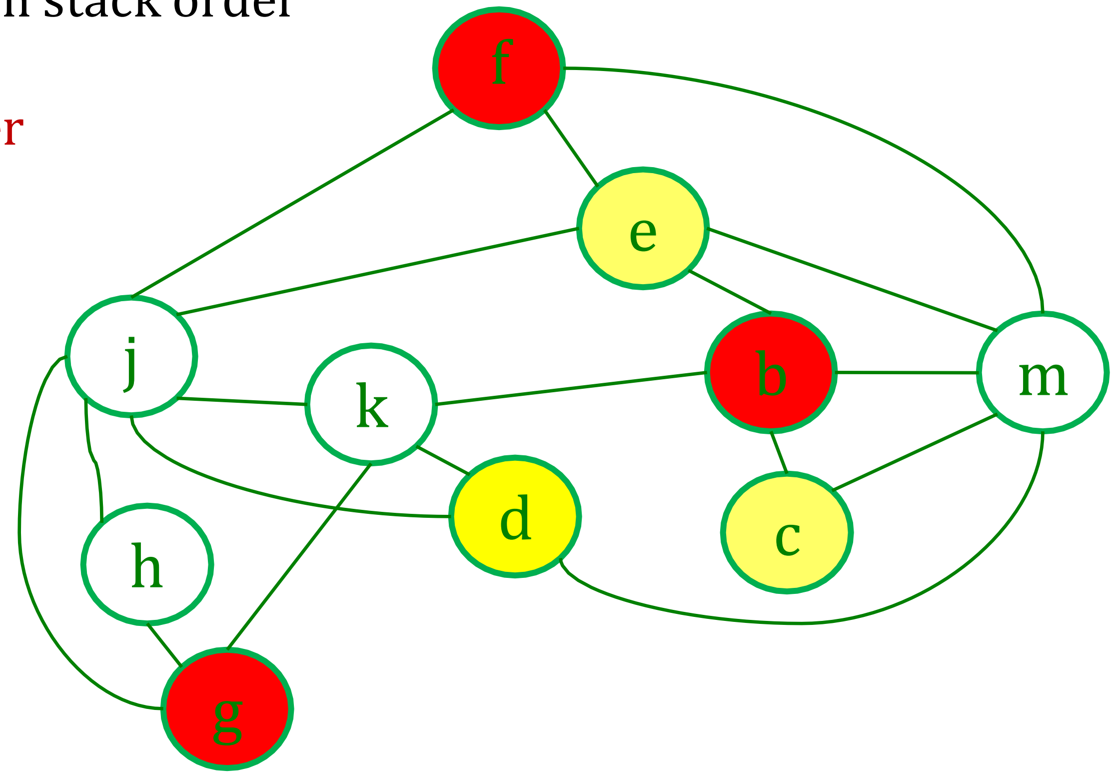



Coloring order: b c d e f g h j k m

# Two phase algorithm

Phase 1: list the nodes in some order ("the stack")
Phase 2: color the nodes in stack order

What if we use some other coloring order?

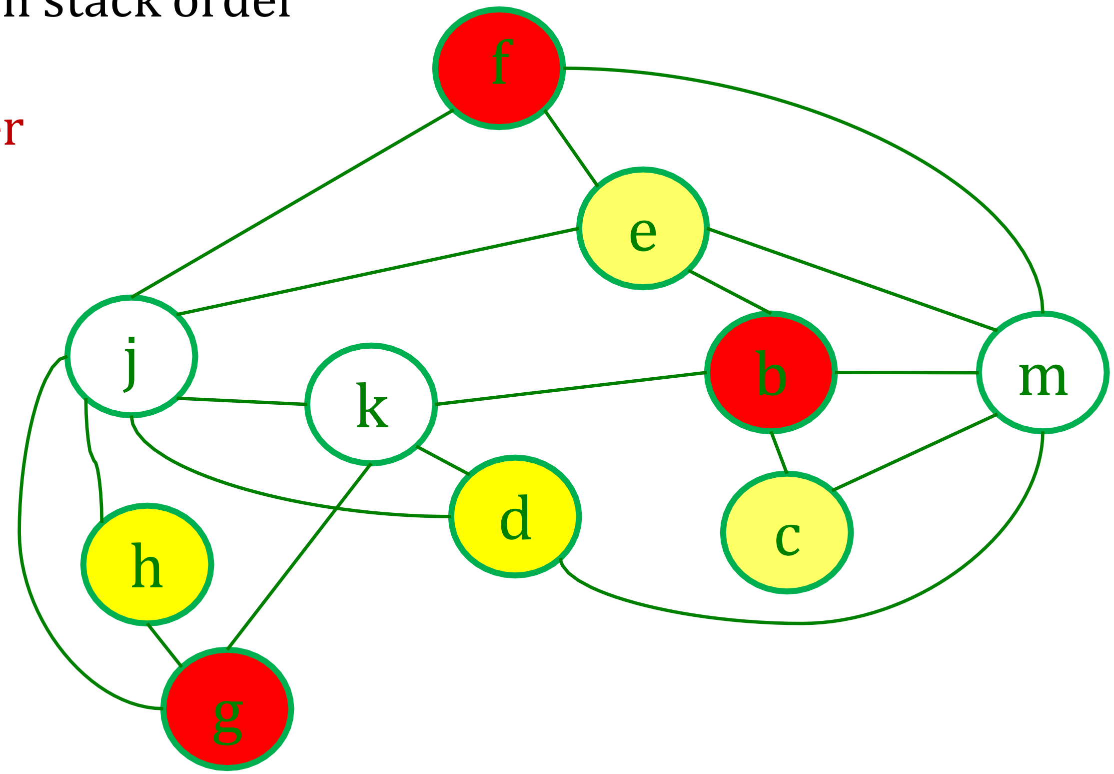



Coloring order: b c d e f g h j k m

# Two phase algorithm

Phase 1: list the nodes in some order ("the stack")
Phase 2: color the nodes in stack order

What if we use some other coloring order?

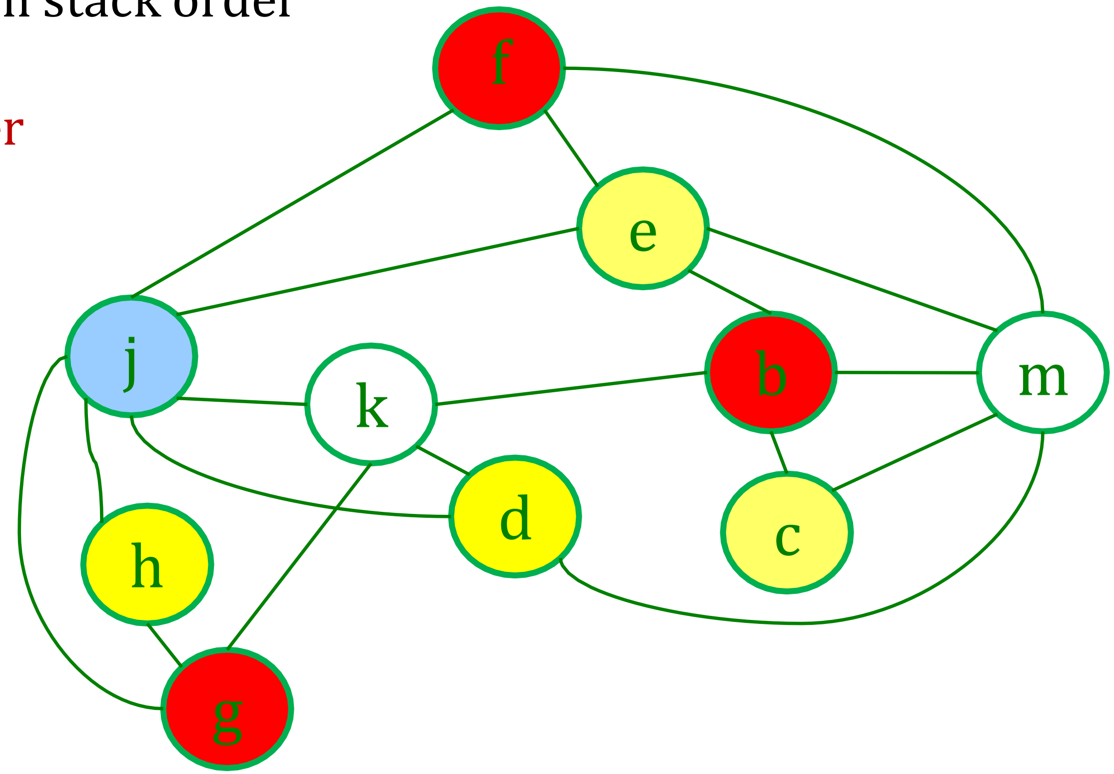



Coloring order: b c d e f g h j k m

# Two phase algorithm

Phase 1:  list the nodes in some order ("the stack")
Phase 2: color the nodes in stack order

What if we use some other coloring order?

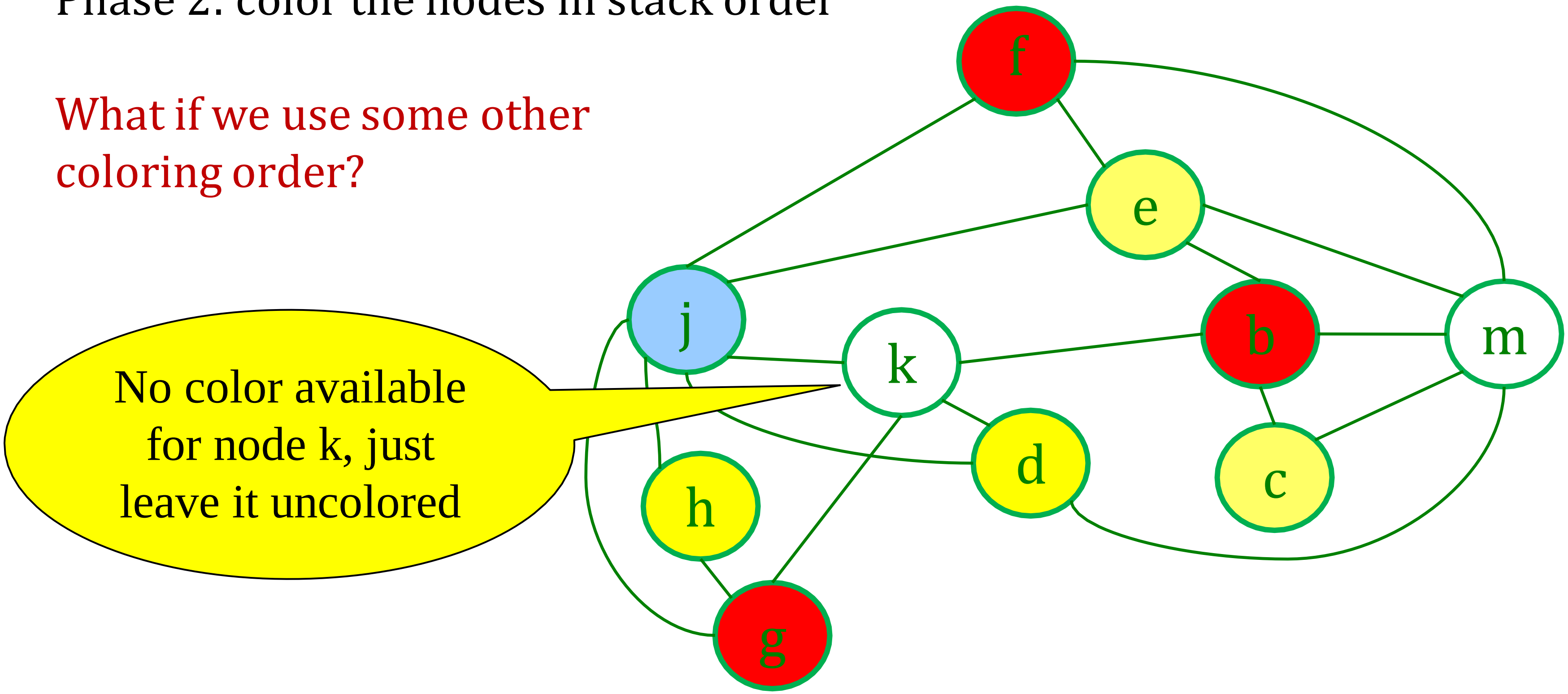



Coloring order: b c d e f g h j k m

# Two phase algorithm

Phase 1: list the nodes in some order ("the stack")
Phase 2: color the nodes in stack order

What if we use some other coloring order?

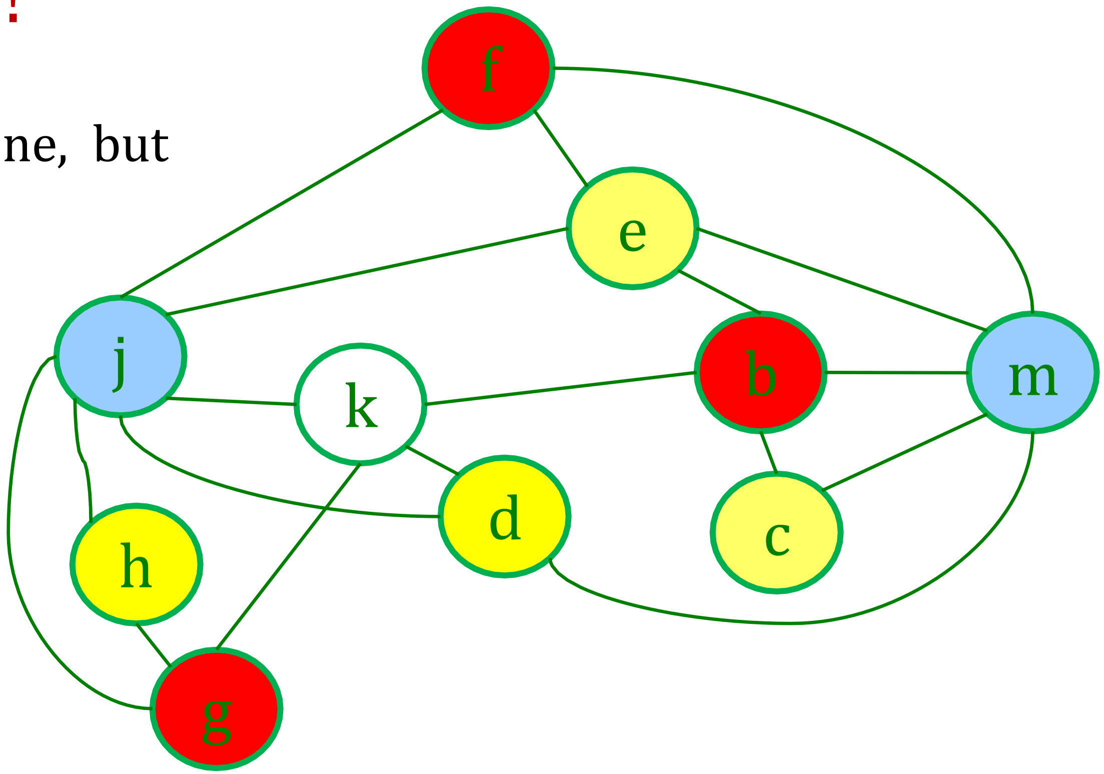



Coloring order: b c d e f g h j k m

# Two phase algorithm

This is a correct partial coloring of the graph!

It's not as good as the other one, but it is correct.

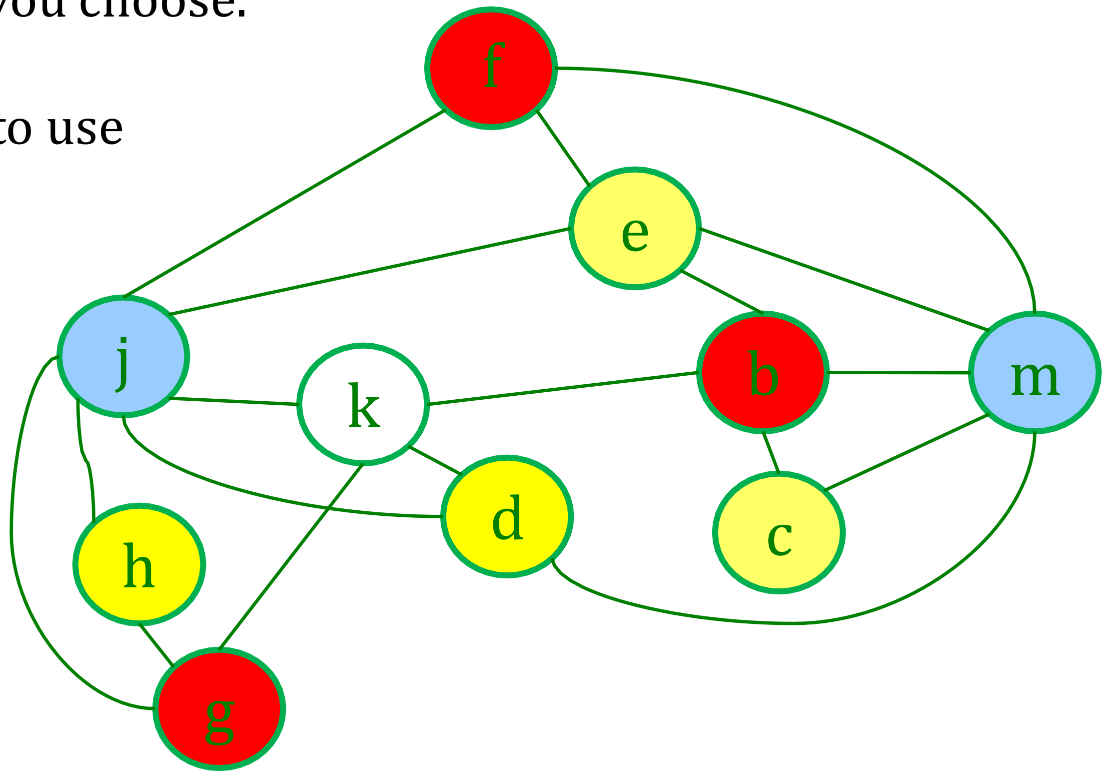



Coloring order: b c d e f g h j k m

# Two phase algorithm

Moral: The two-phase algorithm is correct no matter what ordering you choose.

In phase 1, not necessary to use Kempe's algorithm, although that may give better results.




Coloring order: b c d e f g h j k m

# Summary

- Graph Coloring