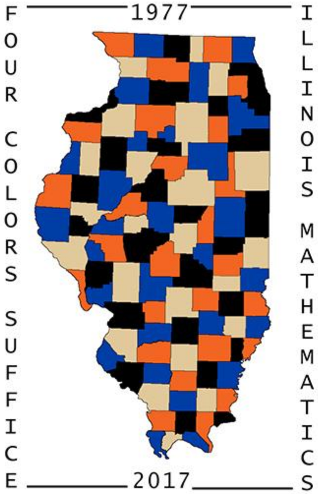
Graph Coloring and Machine Proofs in Computer Science, 1977-2017



Andrew W. Appel

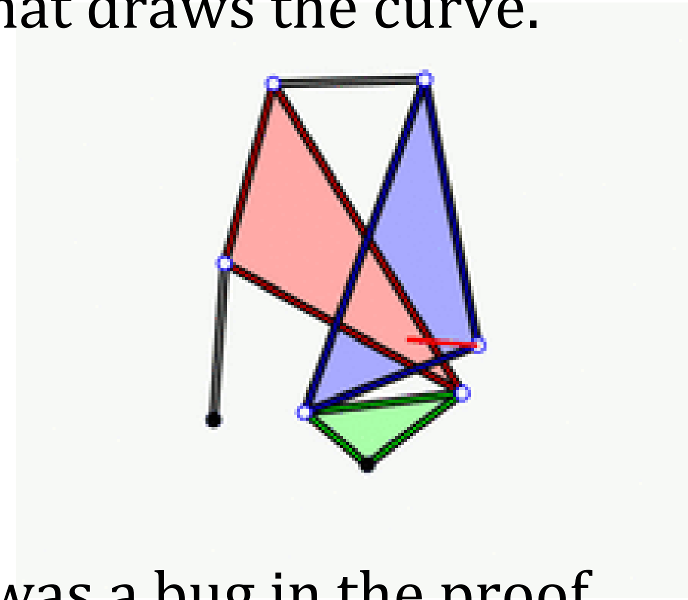
Princeton University

Can it really be a proof if you can’t check it by machine?

Alfred B. Kempe, 1849-1922

Barrister of ecclesiastical law; mathematician

In 1876, Kempe's Universality Theorem: for an arbitrary algebraic plane curve, a linkage can be constructed that draws the curve.



Oops! There was a bug in the proof.

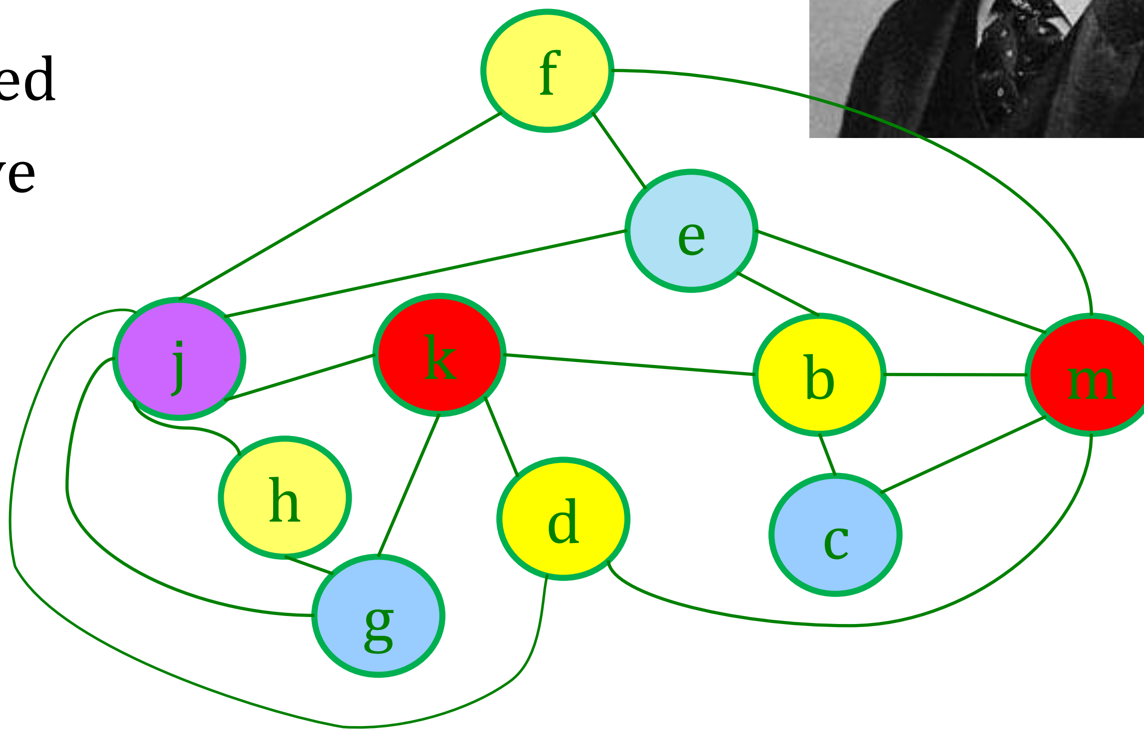
Finally proved in 2002 by Michael Kapovich and John J. Millson

Alfred B. Kempe, 1849-1922

Barrister of ecclesiastical law; mathematician

In 1879, proof of the 4-color theorem: every planar graph can be colored using at most 4 colors.

(Any nodes connected by an edge must have different colors.)



Alfred B. Kempe 1879

6-color theorem:

Every planar graph is 6-colorable.

5-color theorem:

Every planar graph is 5-colorable.

4-color theorem:

Every planar graph is 4-colorable.

Percy J. Heawood

found a bug in the proof, 1890

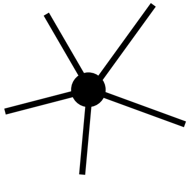
Alfred B. Kempe 1879

6-color theorem:

Every planar graph is 6-colorable.

Proof:

1. Every planar graph has at least one node of degree <6



(by Euler's polyhedron formula): $V-E+F=2$, *average* degree < 6

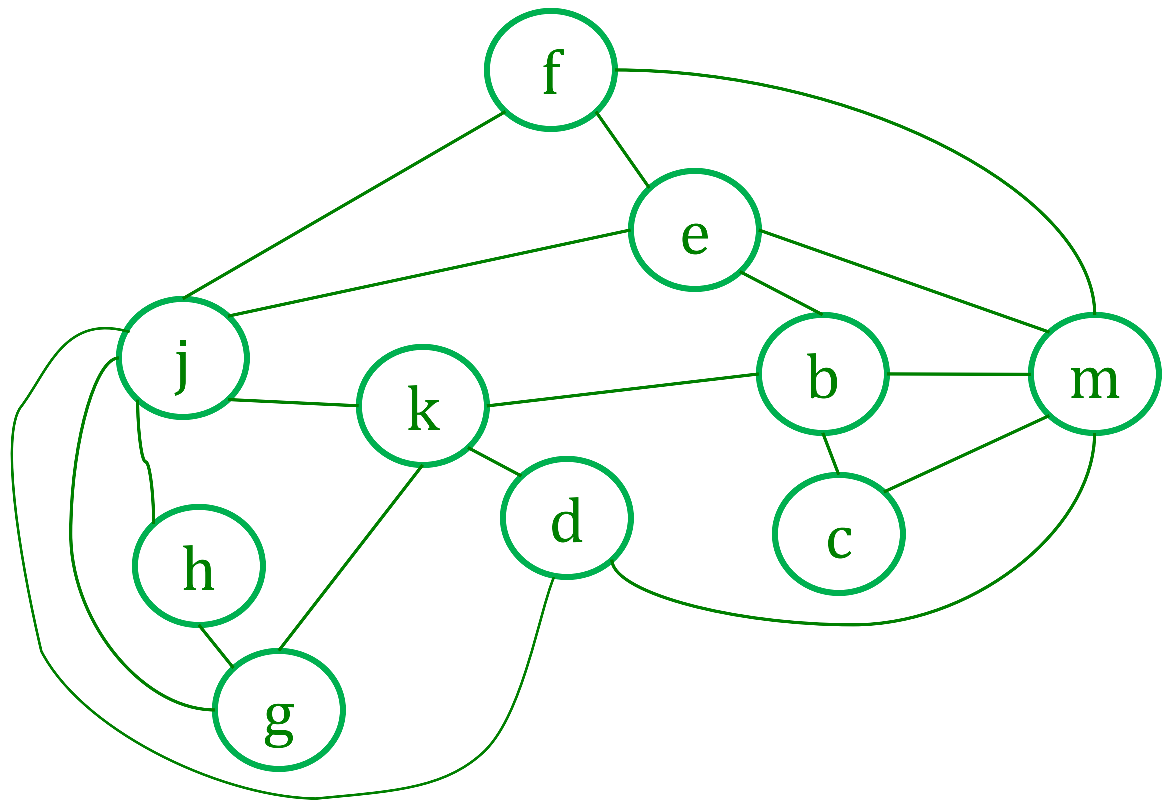
2. If you remove one node from a planar graph, what remains is a planar graph.
3. This leads to an algorithm for coloring graphs . . .

Kempe's graph-coloring algorithm

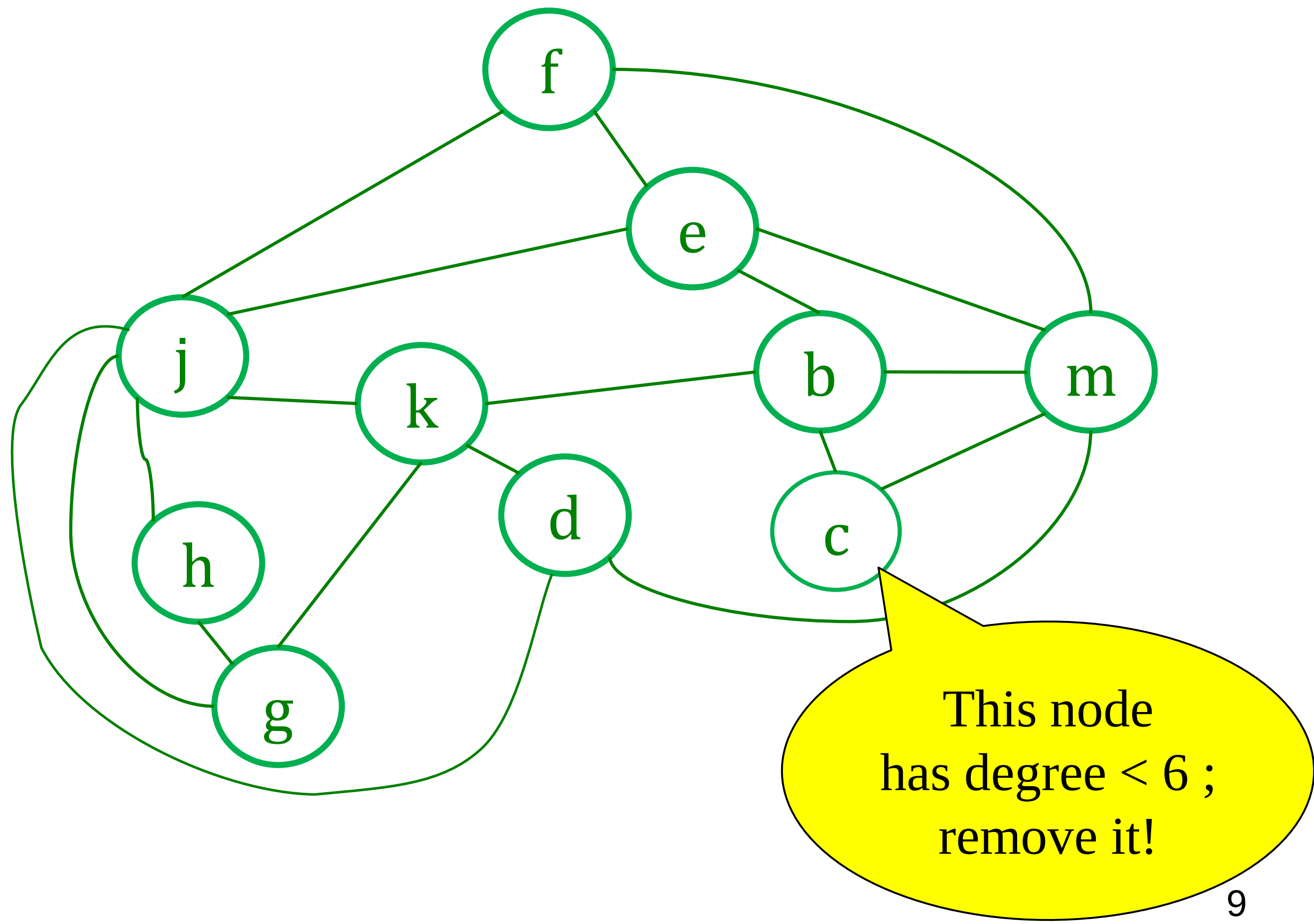
To 6-color a planar graph:

1. Every planar graph has at least one vertex of degree ≤ 5.
2. Remove this vertex.
3. Color the rest of the graph with a recursive call to Kempe's algorithm.
4. Put the vertex back. It is adjacent to at most 5 vertices, which use up at most 5 colors from your "palette." Use the 6^{th} color for this vertex.

Example: 6-color this graph

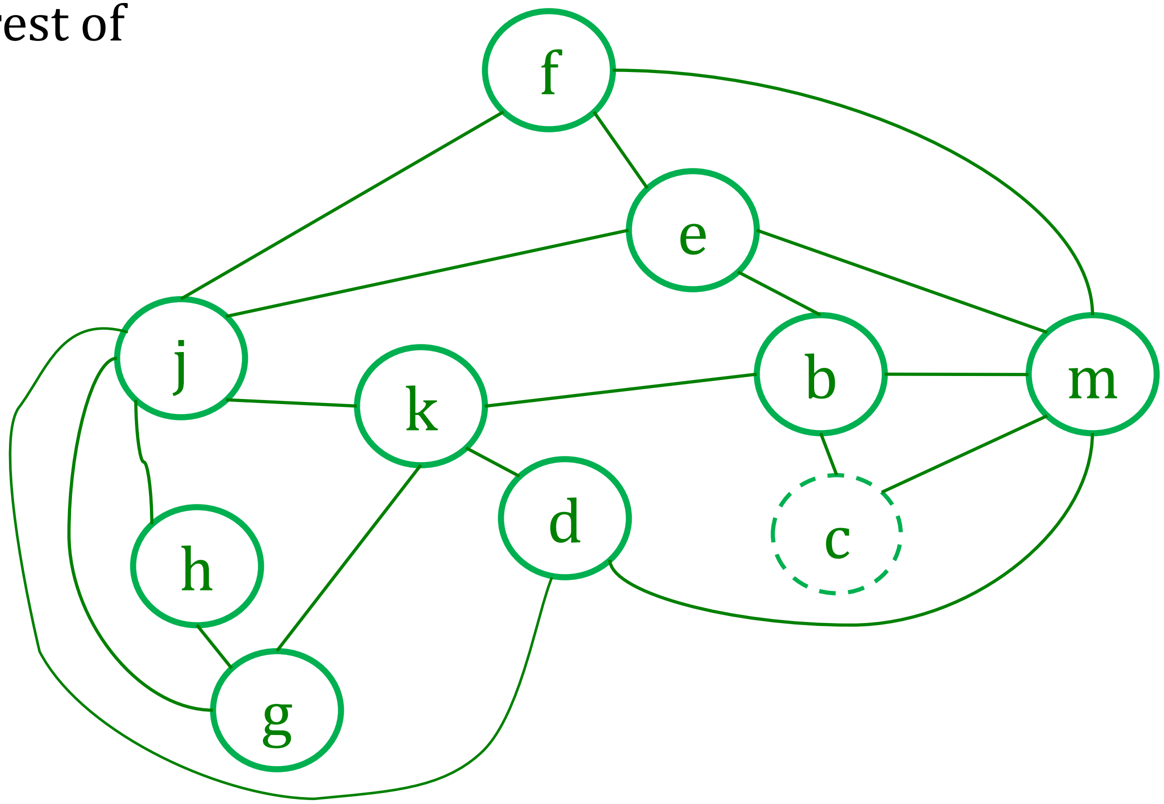


Example: 6-color this graph



Example: 6-color this graph

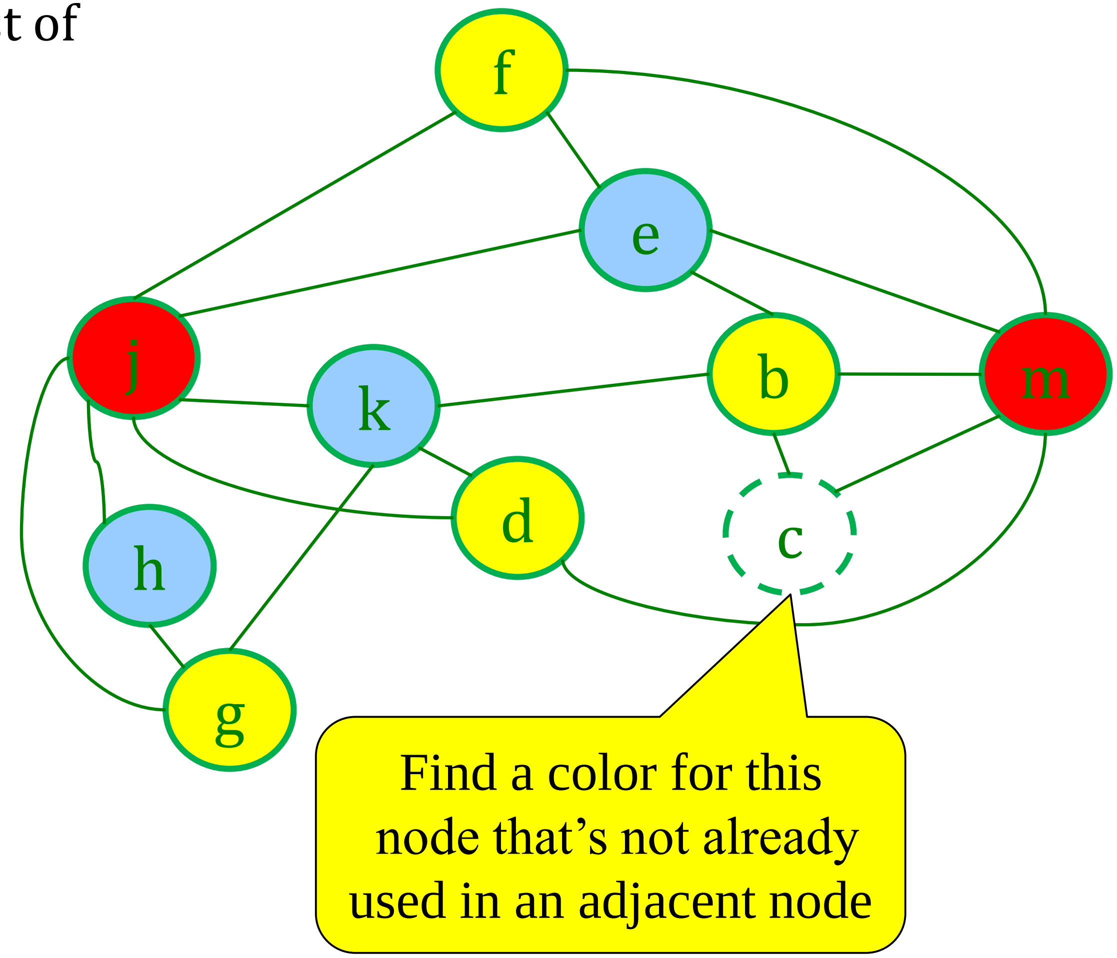
Now, by induction, suppose we could color the rest of the graph



Now, color the residual graph

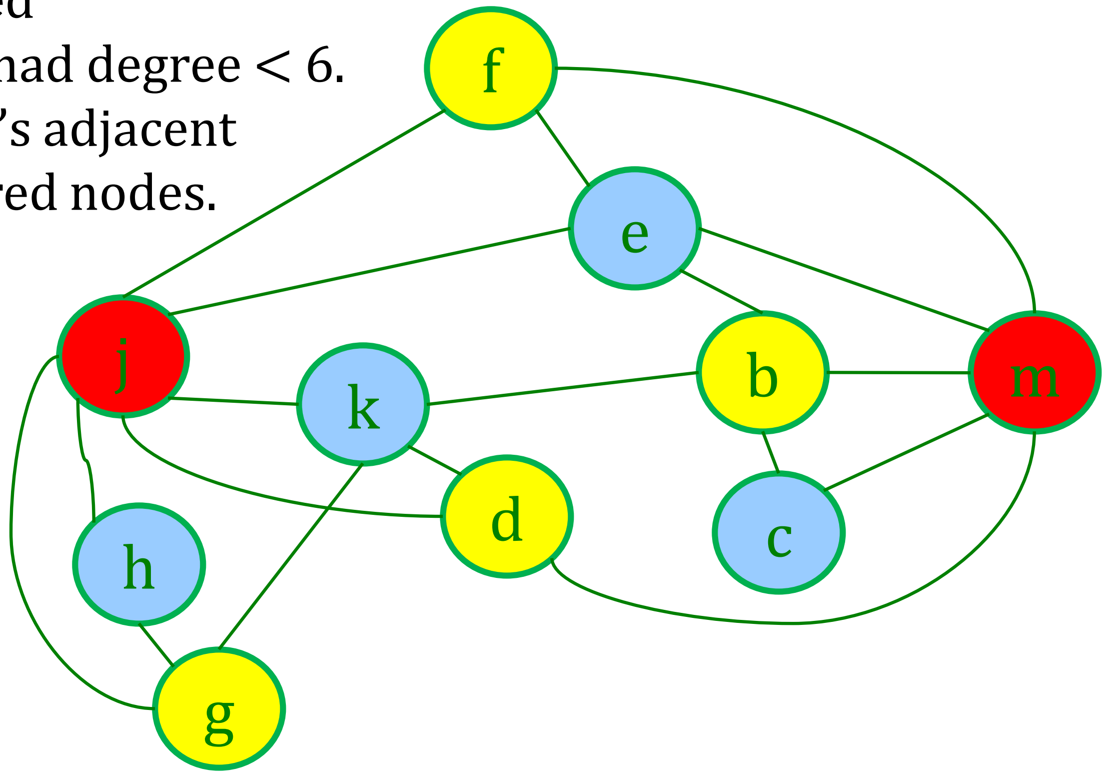
Now, by induction, suppose we could color the rest of the graph

We can surely find a color for c



Put back the node c, and color it

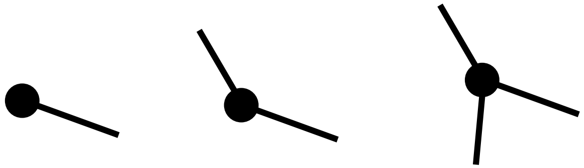
Why did this work?
Because when we removed
each node, at that time it had degree < 6.
So when we put it back, it's adjacent
to at most 5 already-colored nodes.



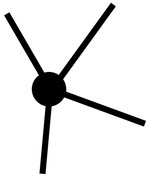
Kempe's 4-coloring algorithm

To 4-color a planar graph:

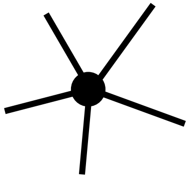
1. Find a vertex of degree ≤ 5 (there must be one)
2. Remove this vertex.
3. Color the rest of the graph with a recursive call to Kempe's algorithm.
4. Put the vertex back.



These cases: easy; you can find a color not used by an adjacent node.



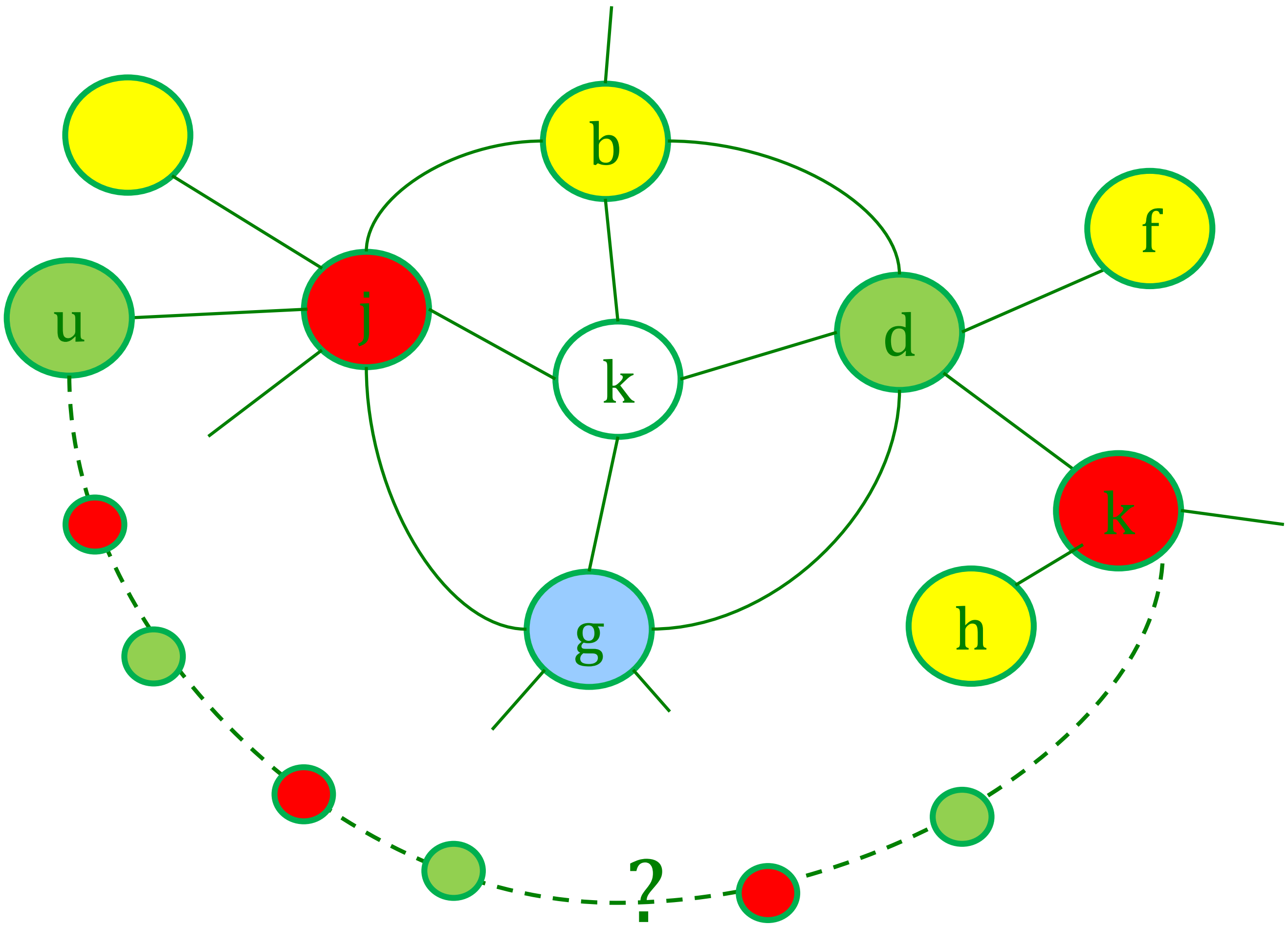
This case: use the method of "Kempe chains"



This case . . .

Kempe chains

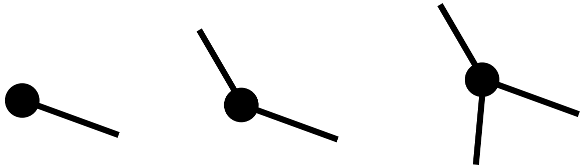
Suppose you are 4-coloring this graph:



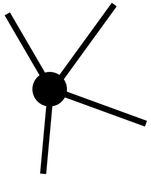
Kempe's 4-coloring algorithm

To 4-color a planar graph:

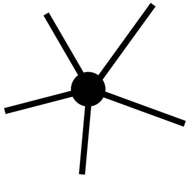
1. Find a vertex of degree ≤ 5 (there must be one)
2. Remove this vertex.
3. Color the rest of the graph with a recursive call to Kempe's algorithm.
4. Put the vertex back.



These cases: easy



This case: use "Kempe chains"

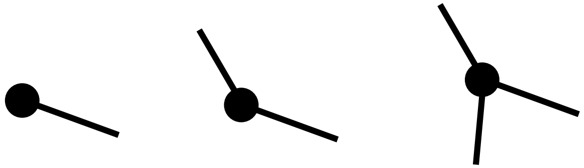


This case: use "simultaneous Kempe chains"

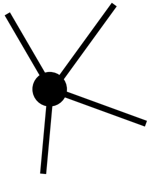
Kempe's 4-coloring algorithm

To 4-color a planar graph:

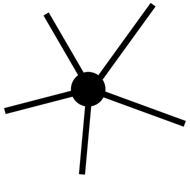
1. Find a vertex of degree ≤ 5 (there must be one)
2. Remove this vertex.
3. Color the rest of the graph with a recursive call to Kempe's algorithm.
4. Put the vertex back.



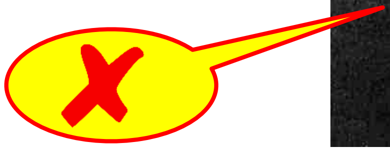
These cases: easy



This case: use "Kempe chains"



"simultaneous Kempe chains"

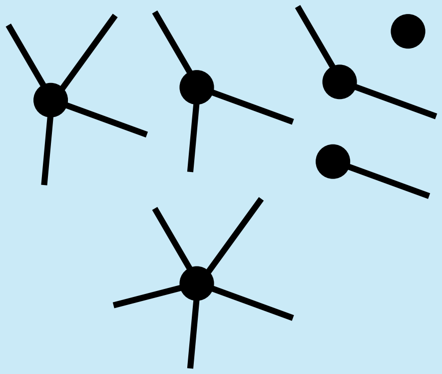


Heawood 1890

Kempe 1879

6-color thm

Every planar graph contains at least 1 of these configurations:

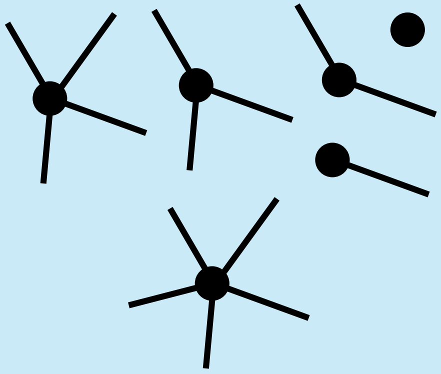


“reduce”: Replace that configuration with a smaller config., color the remaining graph, put the node back, you can find a color for the node!

Kempe 1879

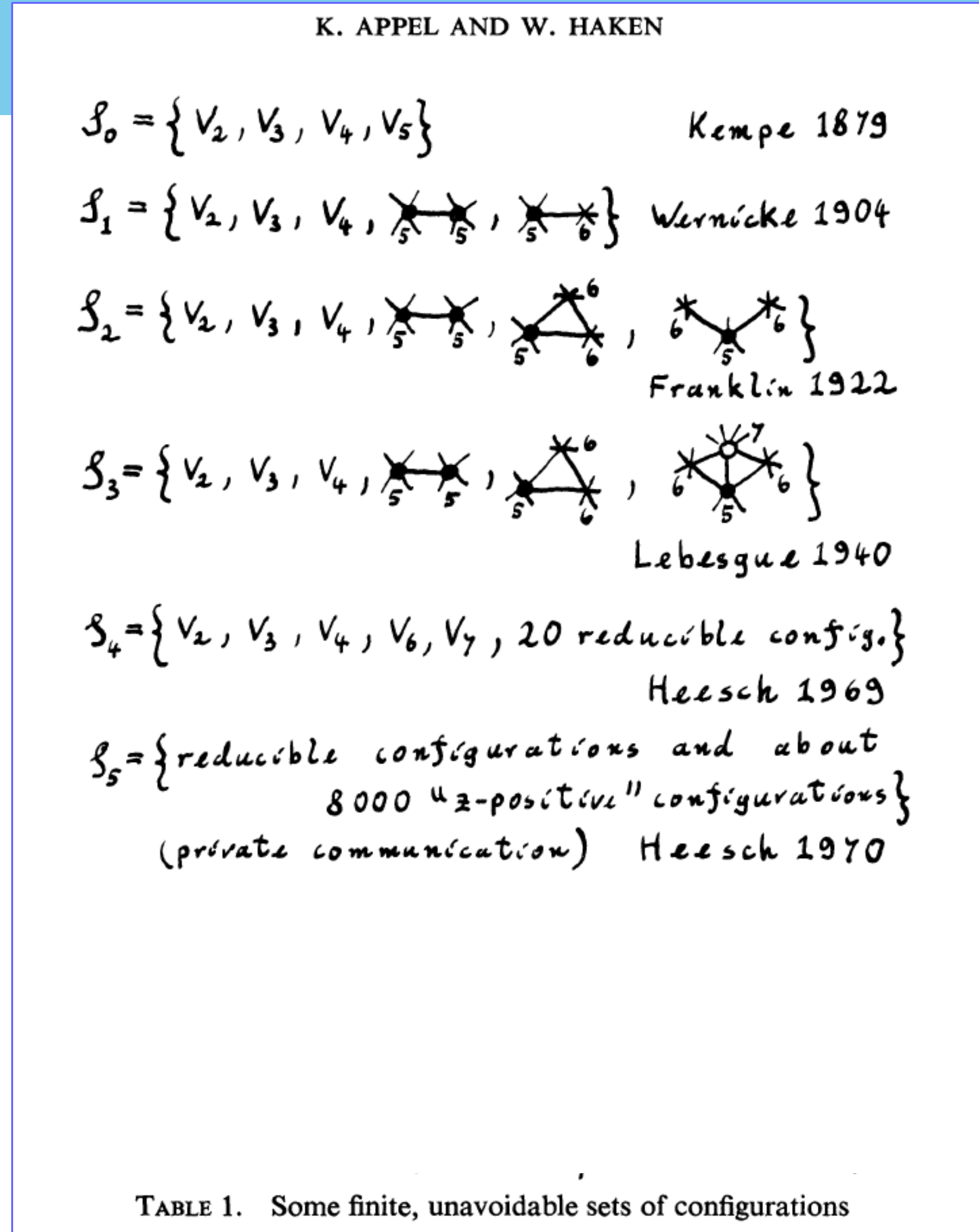
5-color thm

Every planar graph contains at least 1 of these configurations:



“reduce”: Replace that configuration with a smaller config., color the remaining graph, put the node back, you can find a color for the node!

Unavoidable sets



K. APPEL AND W. HAKEN

$\mathcal{S}_0 = \{V_2, V_3, V_4, V_5\}$ Kempe 1879

$\mathcal{S}_1 = \{V_2, V_3, V_4, \ldots\}$ Wernicke 1904

$\mathcal{S}_2 = \{V_2, V_3, V_4, \ldots\}$ Franklin 1922

$\mathcal{S}_3 = \{V_2, V_3, V_4, \ldots\}$ Lebesgue 1940

$\mathcal{S}_4 = \{V_2, V_3, V_4, V_6, V_7,$ 20 reducible config.$\}$ Heesch 1969

$\mathcal{S}_5 = \{$reducible configurations and about 8000 "z-positive" configurations$\}$ (private communication) Heesch 1970

TABLE 1. Some finite, unavoidable sets of configurations

Illinois Journal of Mathematics
1976 (received 1974)

Wernicke, Franklin, Lebesgue, Heesch

Heinrich Heesch
1906-1995

“unavoidable set”

of “reducible configurations”

would prove the 4-color theorem

~1970: [paraphrase]
I estimate that computers will be powerful enough someday, to find an unavoidable set of perhaps 10,000 reducible configurations

4-color thm

Every planar graph contains at least 1 of these configurations:

? ? ? ? ? ? ? ? ? ? ?

“reduce”: Replace that configuration with a smaller config., color the remaining graph, put the node back, you can find a color for the node!

Appel and Haken

1972-1974: Let's use computers to analyze unavoidable sets, and **estimate,**

(1) how many configurations might be in an unavoidable set of reducible configurations?

(2) in what year will future computers be fast enough to calculate this?

THE EXISTENCE OF UNAVOIDABLE SETS OF GEOGRAPHICALLY GOOD CONFIGURATIONS[1]

BY

K. APPEL AND W. HAKEN

Abstract

A set of configurations is unavoidable if every planar map contains at least one element of the set. A configuration $\mathscr{C}$ is called geographically good if whenever a member country M of $\mathscr{C}$ has any three neighbors N_1, N_2, N_3 which are not members of $\mathscr{C}$ then N_1, N_2, N_3 are consecutive (in some order) about M.

The main result is a constructive proof that there exist finite unavoidable sets of geographically good configurations. This result is the first step in an investigation of an approach towards the Four Color Conjecture.

Received December 20, 1974.

Appel and Haken

1974: and the estimate is,

(1) about 2000 configurations

(2) in the year 1972!

IBM System/370 Model 168, 1972

Appel and Haken and Koch

1974-1976: Calculate

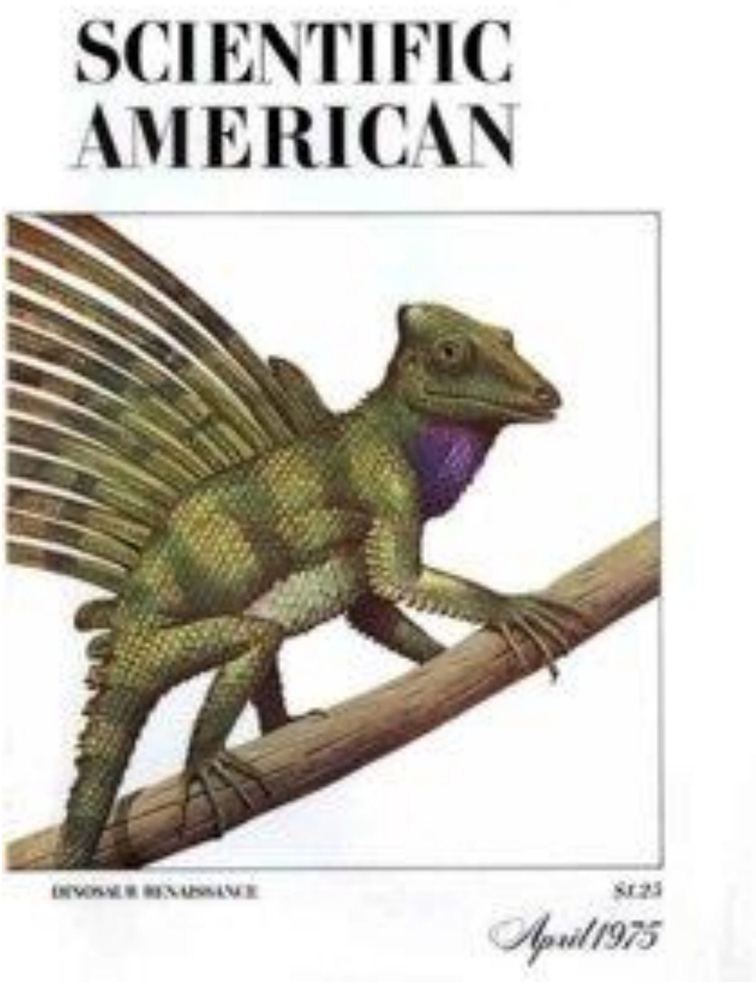
(1) an unavoidable set of 1900 configs (using a version of Heesch's "discharging" procedure)

(2) reducibility proofs for each config., using various reducibility algorithms (implemented with the assistance of C.S. PhD student John Koch)

Teletype model ASR-33

110 bits per second

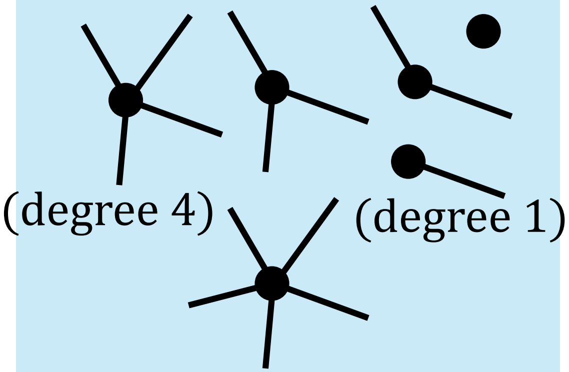
Mathematical Games



Kempe 1879

6-color thm

Every planar graph contains at least 1 of these configurations:



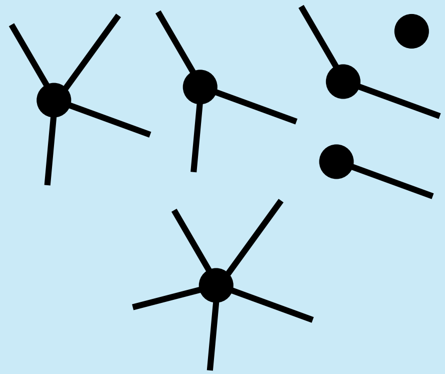
(a degree 5 node)

“reduce”: Replace that configuration with a smaller config., color the remaining graph, put the node back, you can find a color for the node!

Kempe 1879

5-color thm

Every planar graph contains at least 1 of these configurations:

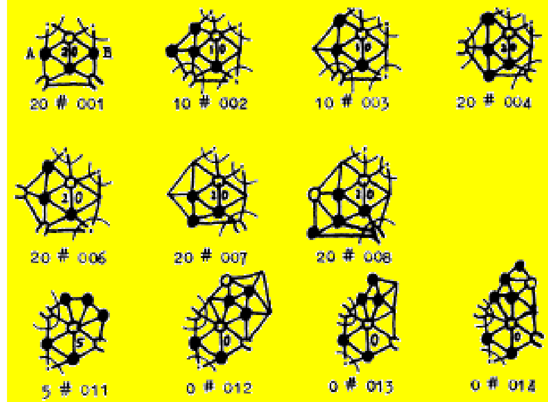


“reduce”: Replace that configuration with a smaller config., color the remaining graph, put the node back, you can find a color for the node!

Appel and Haken 1976

4-color thm

Every planar graph contains at least 1 of these configurations:



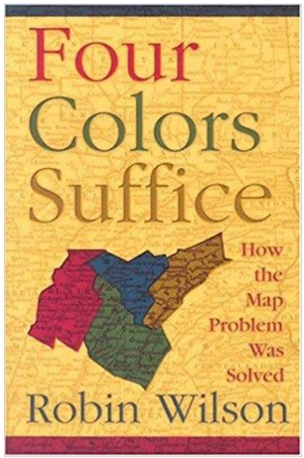
(and 1900 more)

“reduce”: Replace that configuration with a smaller config., color the remaining graph, put the node back, you can find a color for the node!

Math department postage meter

July 22, 1976

My own contribution to the 4CT proofreading:
None.

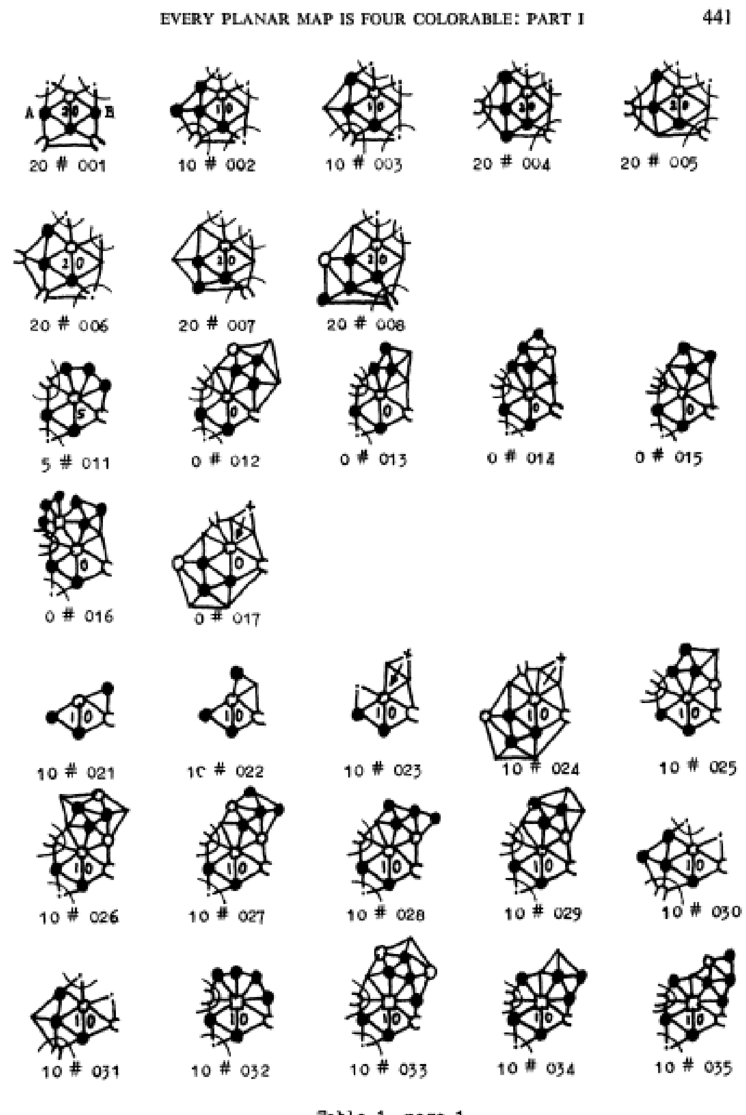


Dorothea Haken Blostein
(1959-)
Professor of C.S.
Queens University

Laurel Appel
(1962-2013)
Adjunct Assoc. Prof. of Biology
Wesleyan University

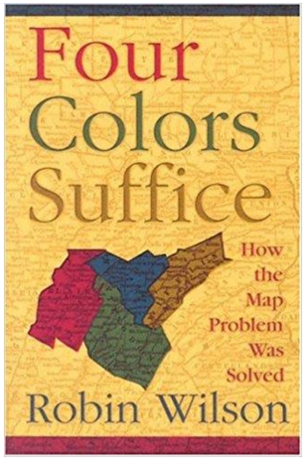
“[with] ~~five~~ two of their children … Dorothea ~~and Armin~~ Haken, and Laurel, ~~Peter, and Andrew~~ Appel, they set to work [proofreading configurations from computer printouts]”

Robin Wilson, 2002



Which part don't you believe?

"Haken's son Armin, by then a graduate student at … Berkeley, gave a lecture on the four-colour problem…. At the end, the audience split into two groups: the over-forties could not be convinced that a proof by computer was correct, while the under-forties could not be convinced that a proof containing 700 pages of hand calculations could be correct."



Princeton University Press 2002

One history

1850 Guthrie

1880 Kempe

Heawood

1904 Wernicke — "unavoidable set"

1920 Birkhoff — "reducible"

1960 Heesch — "discharging" to compute unavoidable set

1976 Appel, Haken — "Every planar map is 4-colorable"

1996 Robertson, Sanders, Seymour, Thomas — improved proof, same basic recipe

2006 Gonthier

One history Another history

1850 Guthrie

1880 Kempe

Heawood

1904 Wernicke

1920 Birkhoff

1960 Heesch

1976 Appel, Haken

1996 Robertson, Sanders, Seymour, Thomas

2006 Gonthier

1700 Leibniz

1850 Babbage

1920 Hilbert

1930 Gödel

Turing

Can we mechanize mathematics?

Proof checking: yes
Proving: not quite

In particular, a short theorem statement might have a very long proof.

Yes, we noticed!

One history Another history

1850	Guthrie
1880	Kempe
	Heawood
1904	Wernicke
1920	Birkhoff
1960	Heesch
1976	Appel, Haken
1996	Robertson, Sanders, Seymour, Thomas
2006	Gonthier

1700	Leibniz
1850	Babbage
1920	Hilbert
1930	Gödel
	Turing
1950	von Neumann
1960	IBM

Can we mechanize mathematics?

Proof checking: yes
Proving: not quite

Let's build those computers!

Thank you!

One history Another history

1850 Guthrie
1880 Kempe
Heawood
1904 Wernicke
1920 Birkhoff
1960 Heesch
1976 Appel, Haken
1996 Robertson, Sanders, Seymour, Thomas
2006 Gonthier

1700 Leibniz
1850 Babbage
1920 Hilbert

Can we mechanize mathematics?

1930 Gödel
Turing

Proof checking: yes
Proving: not quite

1950 von Neumann
1960 IBM

Let's build those computers!

Optimizing compilers

John Cocke 1970s Robin Milner

Proof Assistants

One history Another history

1850	Guthrie
1880	Kempe
	Heawood
1904	Wernicke
1920	Birkhoff
1960	Heesch
1976	Appel, Haken
1996	Robertson, Sanders, Seymour, Thomas
2006	Gonthier

1700	Leibniz
1850	Babbage
1920	Hilbert
1930	Gödel
	Turing
1950	von Ne
1960	IBM

John Cocke 1925-2002
IBM Research

Register Allocation

```
Procedure P (k, j)
 g := mem[j+12]
 h := k-1
 f := g*h
 e := mem [j+8]
 m := mem[j+16]
 b := mem[f]
 c := e+8
 d := c
 k := m+4
 j := b
return (d, k, j)
```

registers

r1	g
r2	j
r3	f
r4	e

memory

0	k
1	h
2	m
3	b
4	c
5	d
6	
7	
8	
9	
10	
11	
12	
13	
14	
.	
.	

One history Another history

1850 Guthrie

1880 Kempe

Heawood

1904 Wernicke

1920 Birkhoff

1960

Heesch

1976

1996

2006

1700 Leibniz

1850 Babbage

1920 Hilbert

1930 Gödel

Turing

1950 von Ne

1960 IBM

John Cocke 1925-2002

IBM Research

1977:
Hmm, this 4-color theorem is interesting.
John, ask Gregory to try Kempe's coloring algorithm in the register allocator of our compiler.

Ashok Chandra
(1948-2014)

Gregory Chaitin
(1947-)

One of the most influential papers in all of computer science

I was recruited to do a coloring register allocator by John Cocke,
IBM's greatest computer architect, who needed it for his RISC project.
He mentioned that Ashok K. Chandra, also at IBM Research at that time

Computer Languages Vol. 6, pp. 47 to 57, 1981

0096-0551/81/010047-11$02.00/0

REGISTER ALLOCATION VIA COLORING

GREGORY J. CHAITIN, MARC A. AUSLANDER, ASHOK K. CHANDRA, JOHN COCKE, MARTIN E. HOPKINS and PETER W. MARKSTEIN

IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, U.S.A.

(*Received* 9 *October* 1980)

Abstract—Register allocation may be viewed as a graph coloring problem. Each node in the graph stands for a computed quantity that resides in a machine register, and two nodes are connected by an edge if the quantities interfere with each other, that is, if they are simultaneously live at some point in the object program. This approach, though mentioned in the literature, was never implemented before. Preliminary results of an experimental implementation in a PL/I optimizing compiler suggest that global register allocation approaching that of hand-coded assembly language may be attainable.

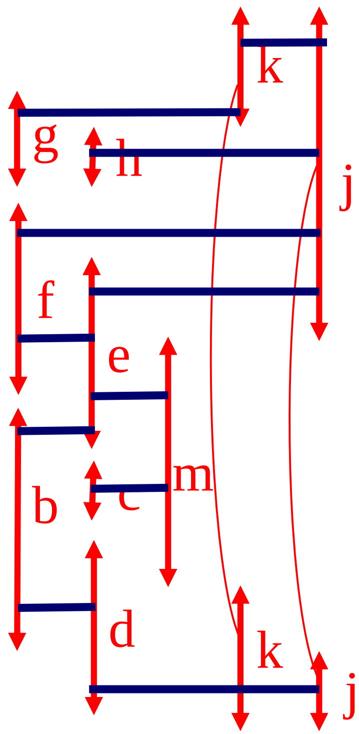
Register Allocation

Chaitin *et al.* 1981

```
Procedure P (k, j)
 g := mem[j+12]
 h := k-1
 f := g*h
 e := mem [j+8]
 m := mem[j+16]
 b := mem[f]
 c := e+8
 d := c
 k := m+4
 j := b
return (d, k, j)
```

Live ranges

Interferences
(some not shown)



Interference Graph

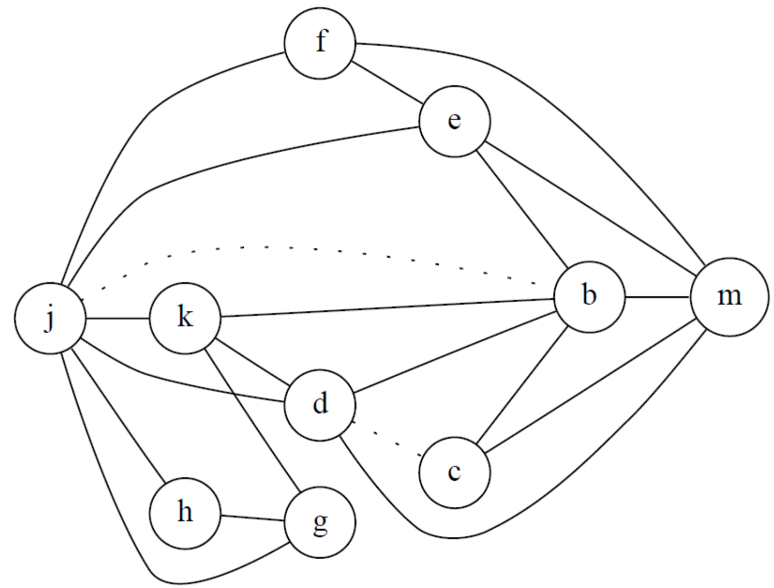


figure 11.1 from *Modern Compiler Implementation in ML*,
Andrew W. Appel, Cambridge University Press 1998

Chaitin's

Heuristic hack of Kempe's algorithm

To mostly K-color a graph (whether planar or not!)

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 $\geq$ K vertices. How many colors do these vertices use among them?

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

If $\geq K$:

Briggs's version of Chaitin's

Heuristic hack of Kempe's algorithm

To mostly K-color a graph (whether planar or not!)

Is there a vertex of degree < K ?

If so:

Remove this vertex.

Color the rest of the graph with a r

Put the vertex back. It is adjac

them) at most K-1 colors. Th

If not:

Remove this vertex.

Color the rest of the graph with a re

Put the vertex back. It is adjacen … do these vertices use among them?

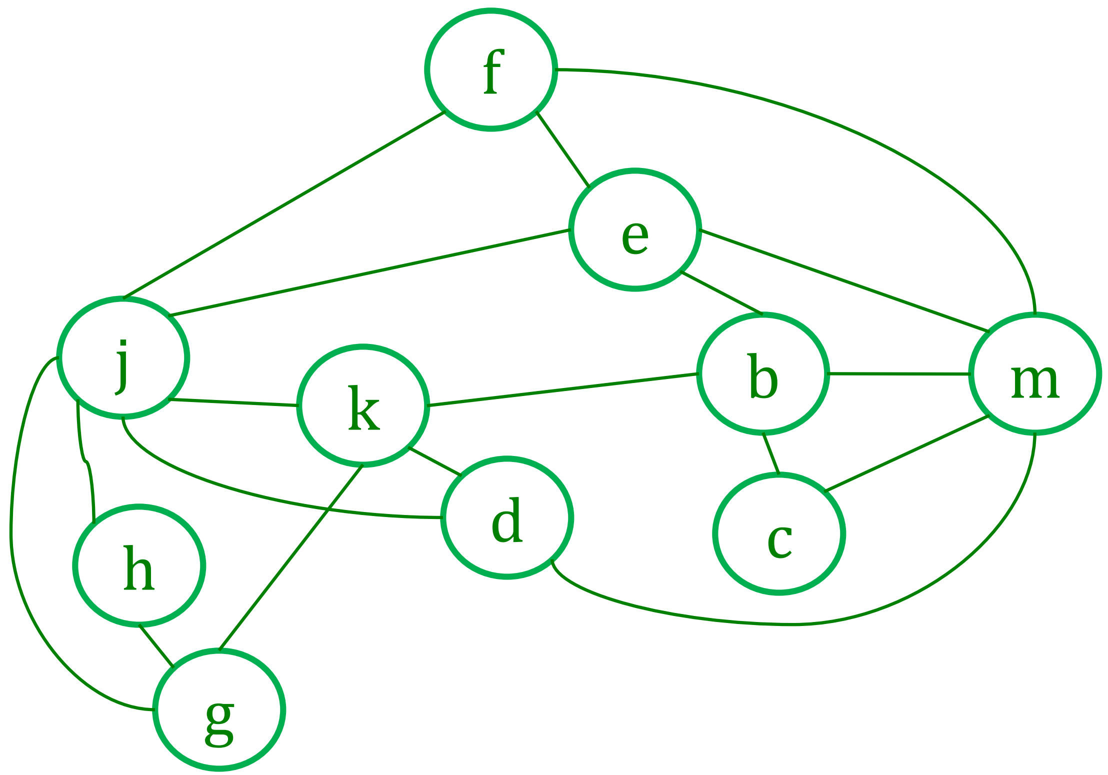
If < K : there is an unuse color to use for this vertex

If ≥ K: leave this vertex uncolored.

What?
Are we allowed to do that?

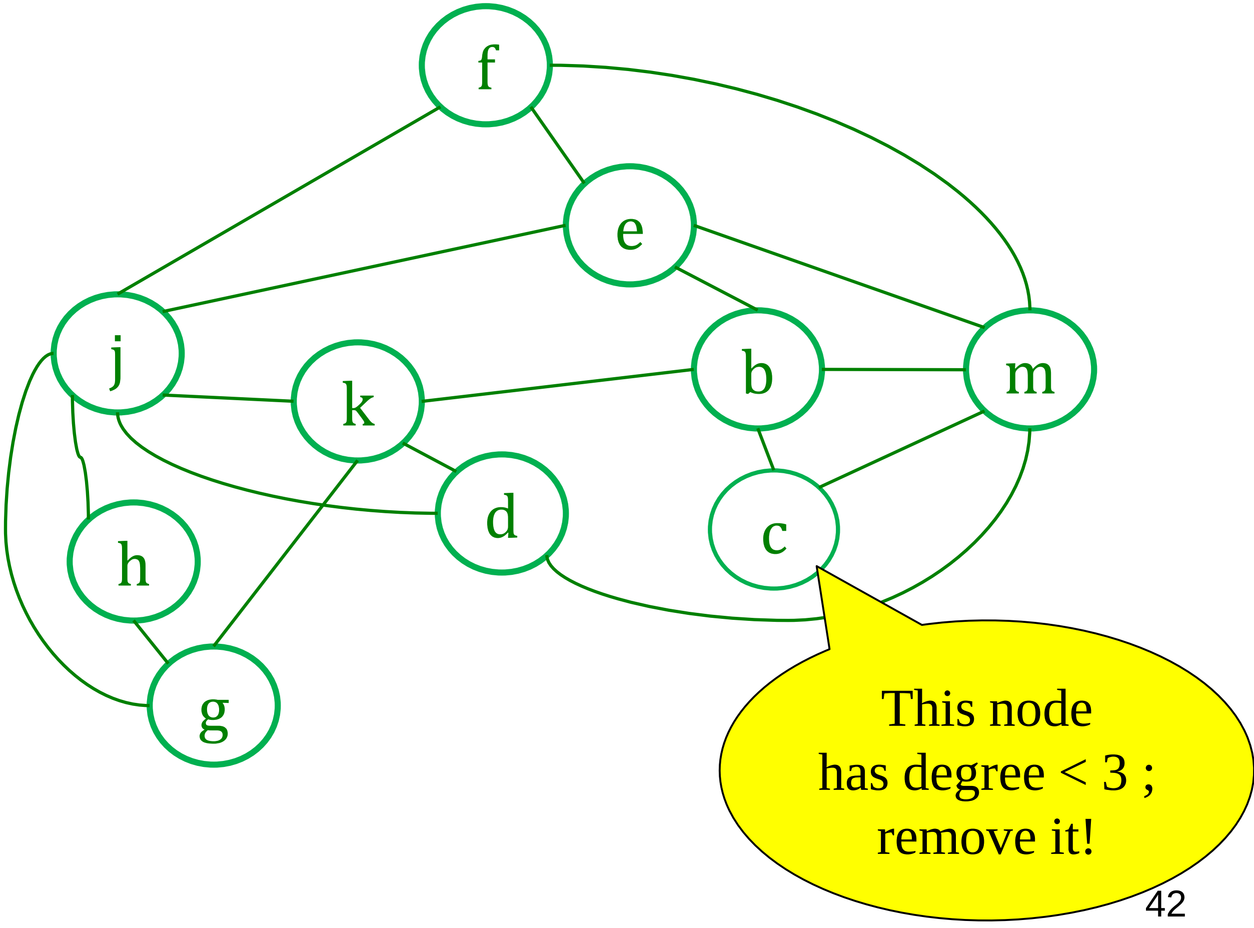
Yes!
This is an algorithm to "mostly K-color" a graph.

Example: 3-color this graph



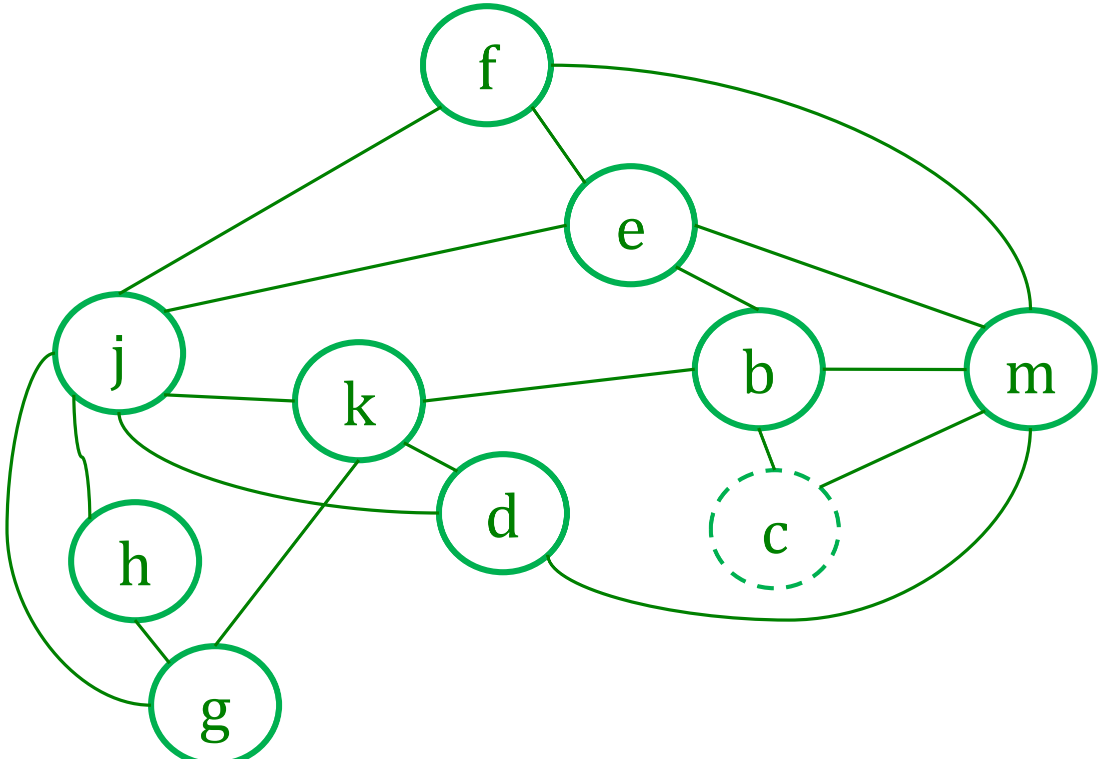
Stack:

Example: 3-color this graph



Stack:

Example: 3-color this graph

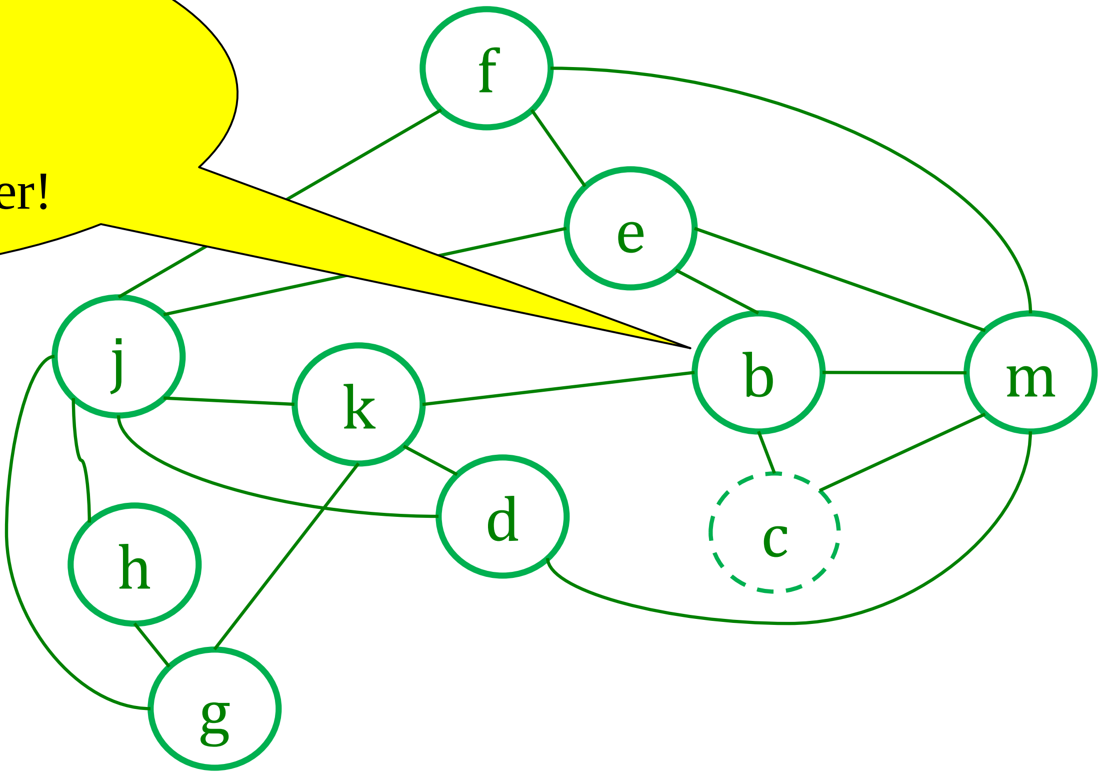


Stack: c

Push node c on the stack

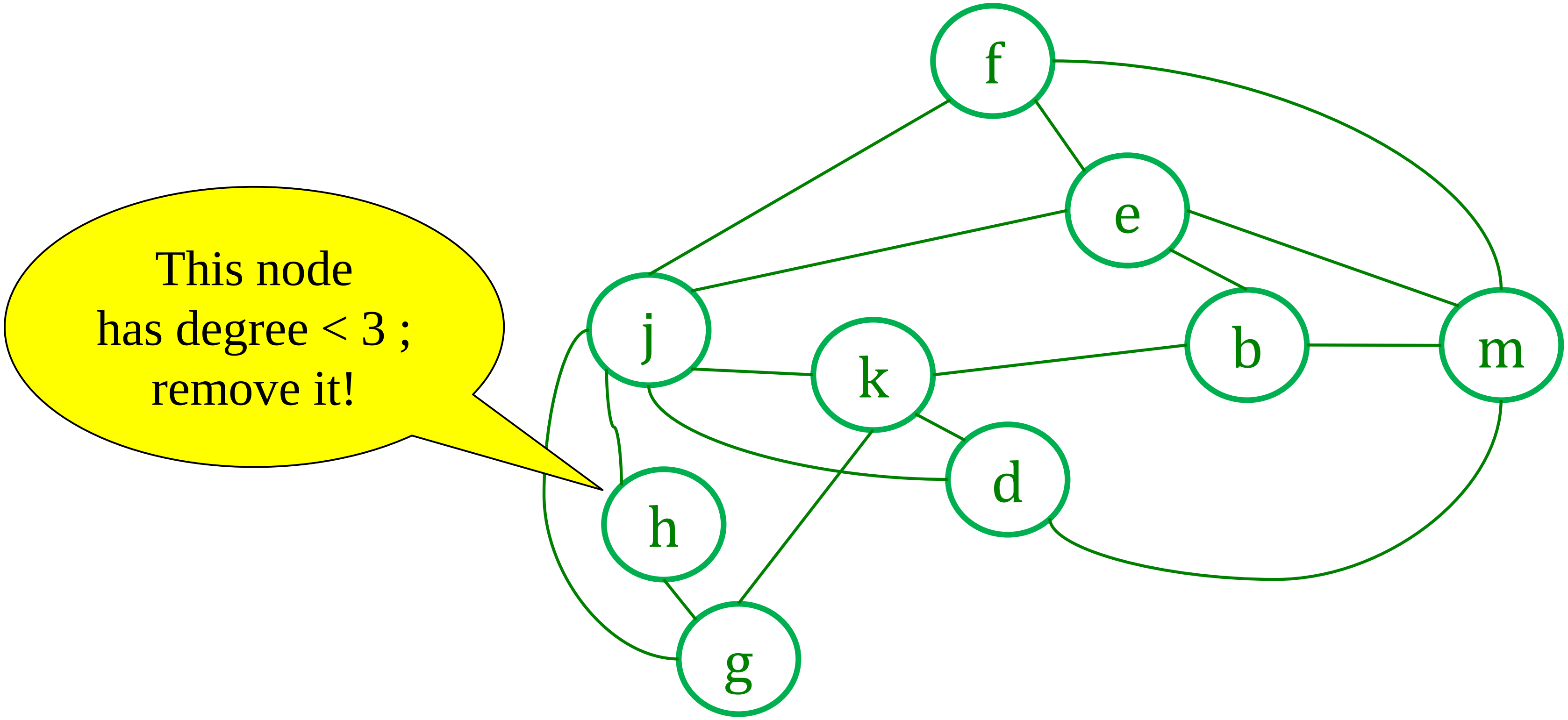
Example: 3-color this graph

Removing c
lowers the degree
of nodes b and m;
that will be helpful later!



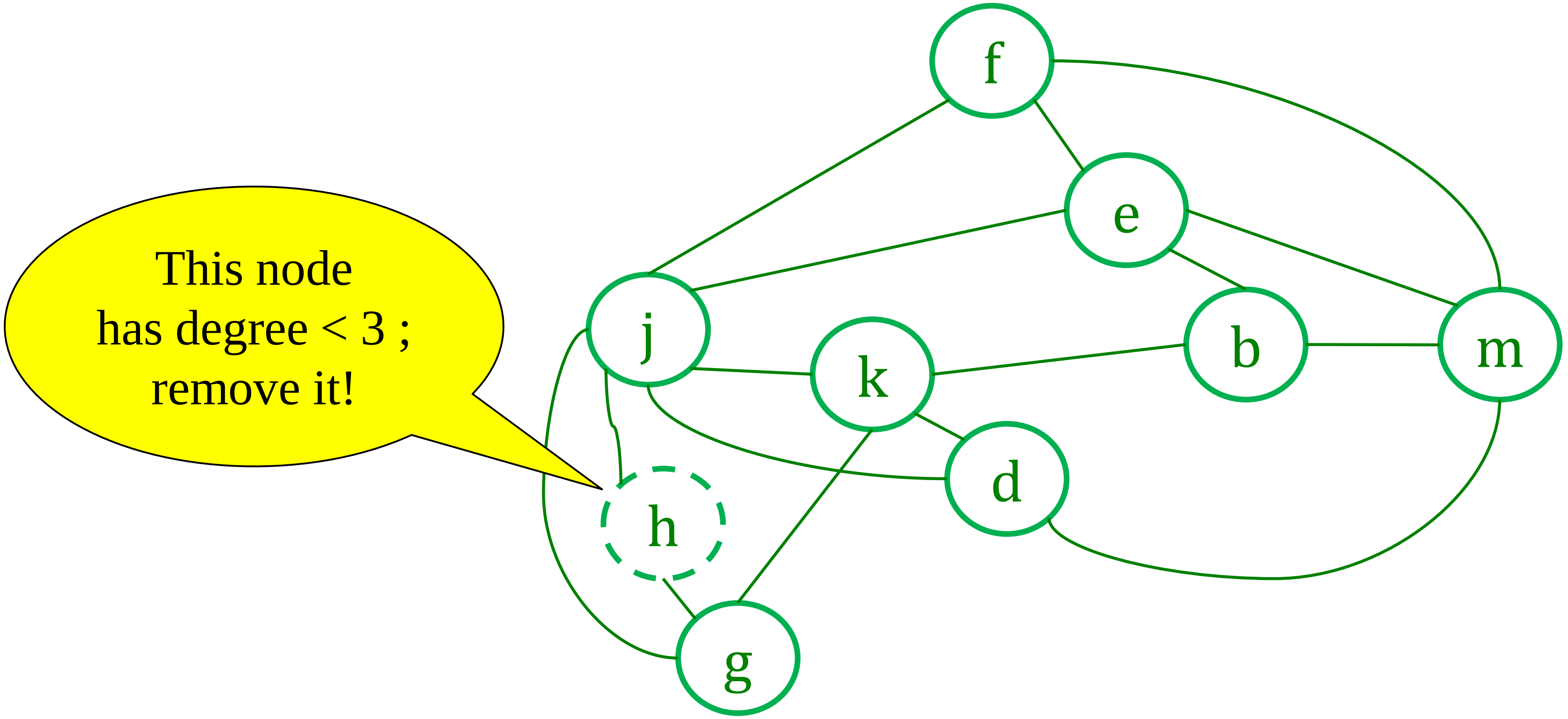
Stack: c

Example: 3-color this graph



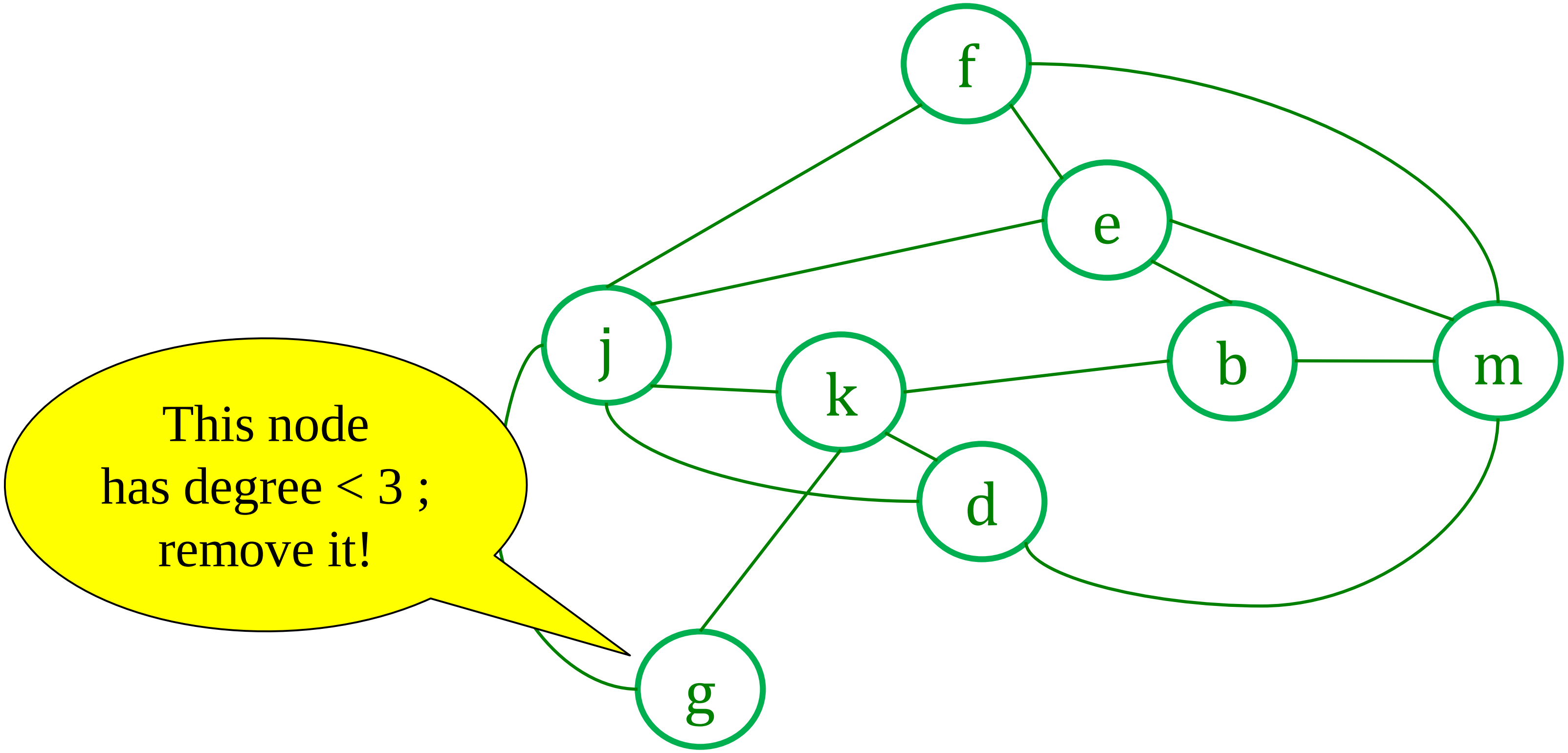
Stack: c

Example: 3-color this graph



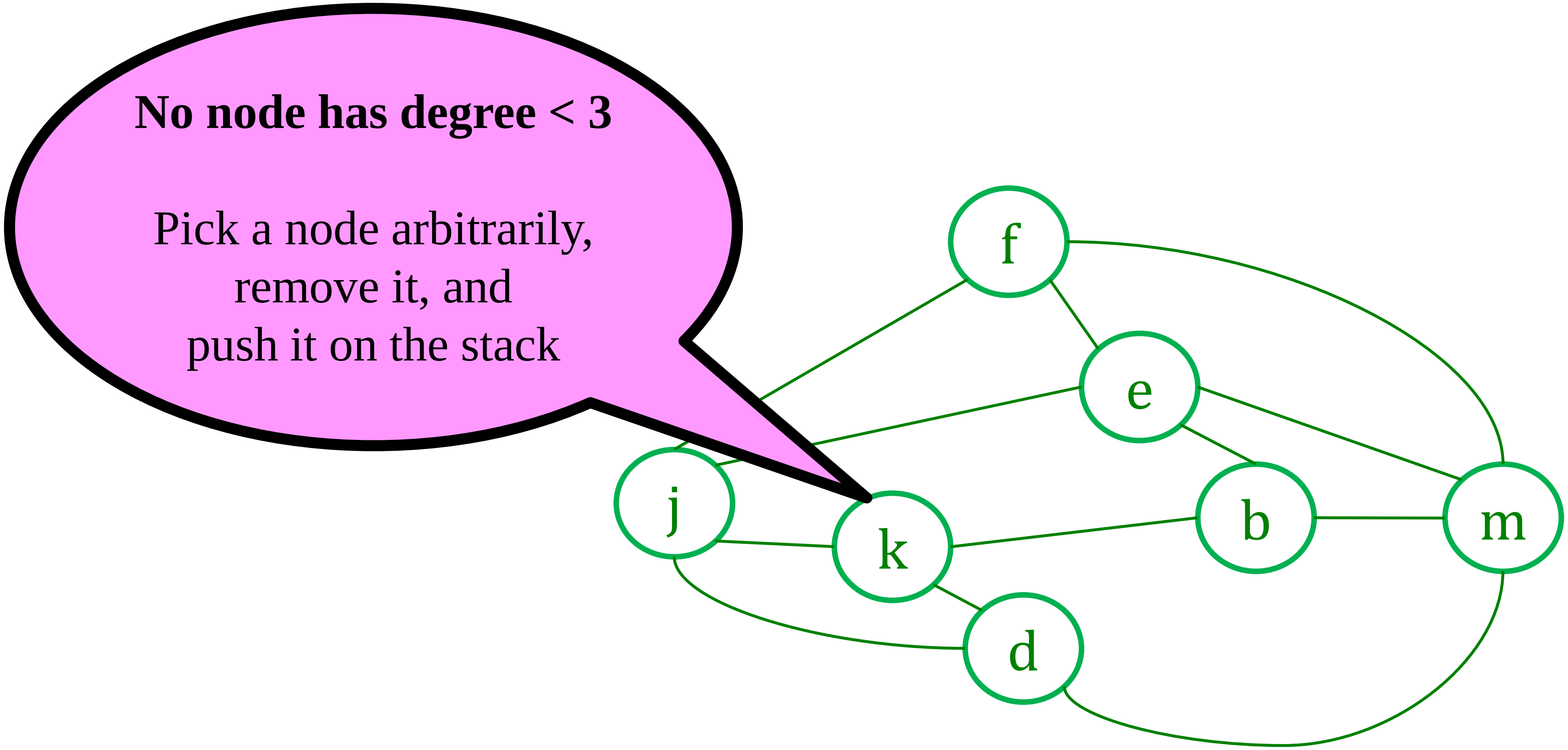
Stack: h c

Example: 3-color this graph



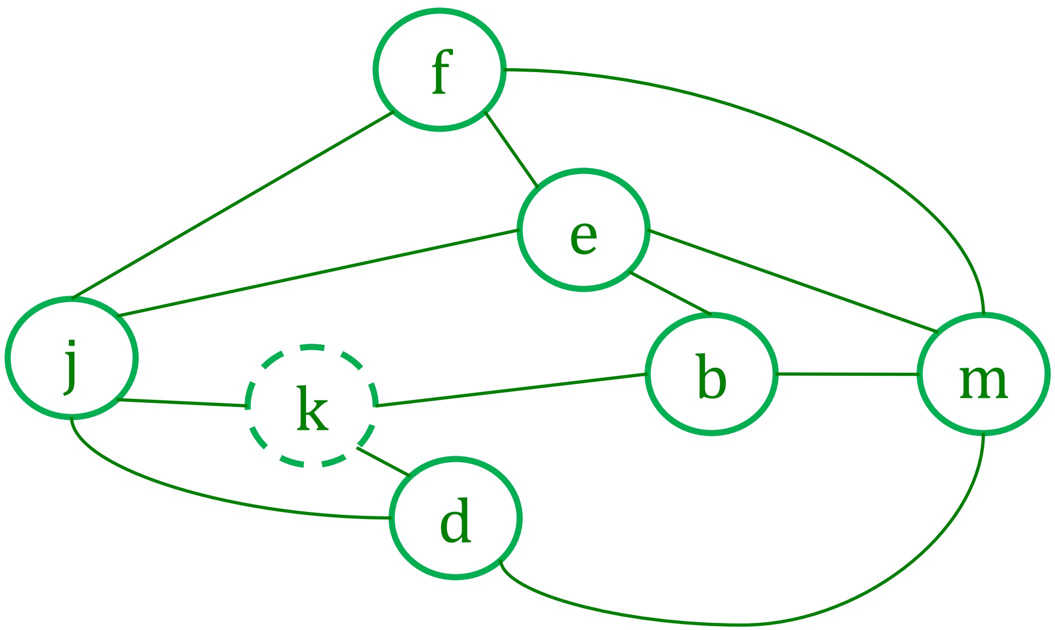
Stack: h c

Example: 3-color this graph



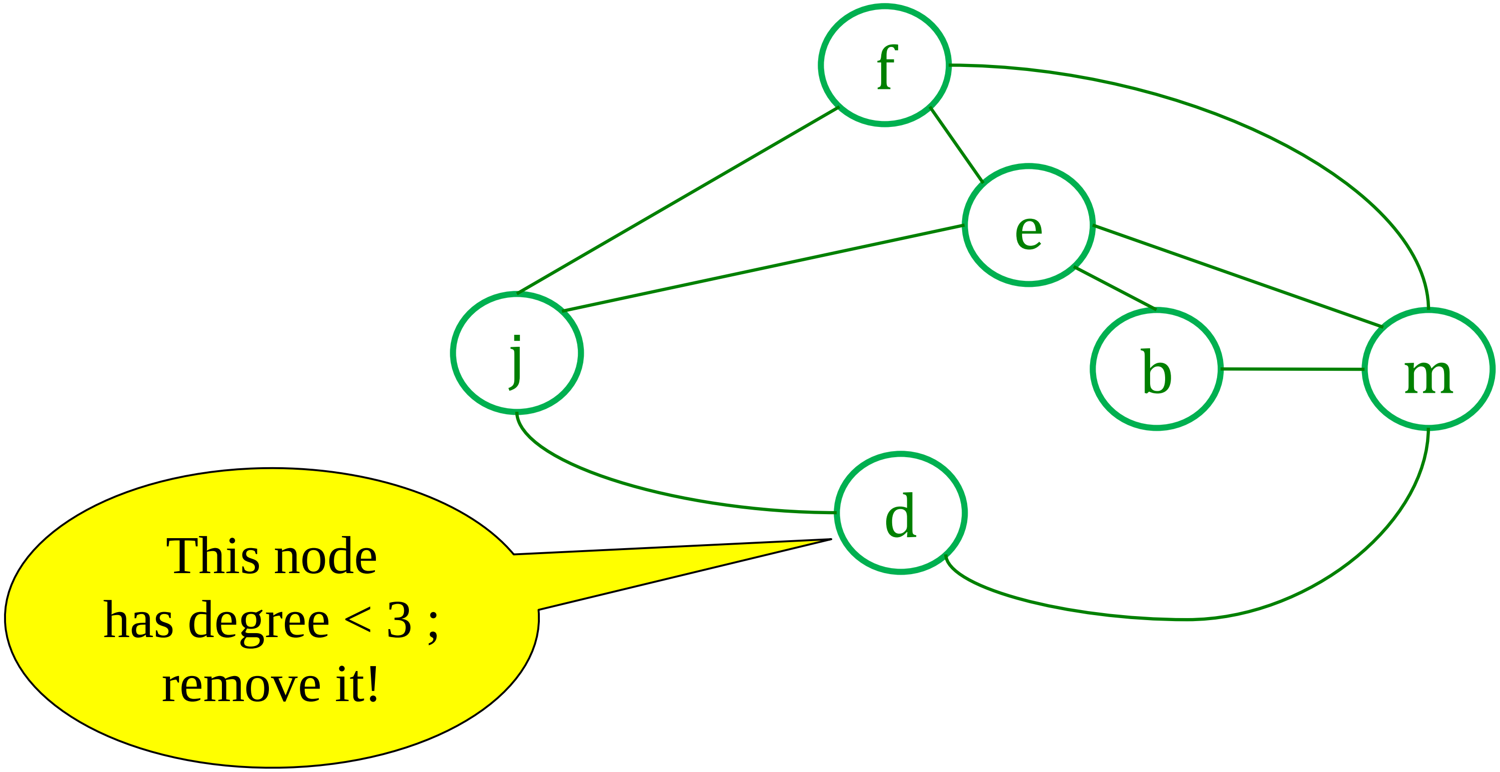
Stack: g h c

Example: 3-color this graph



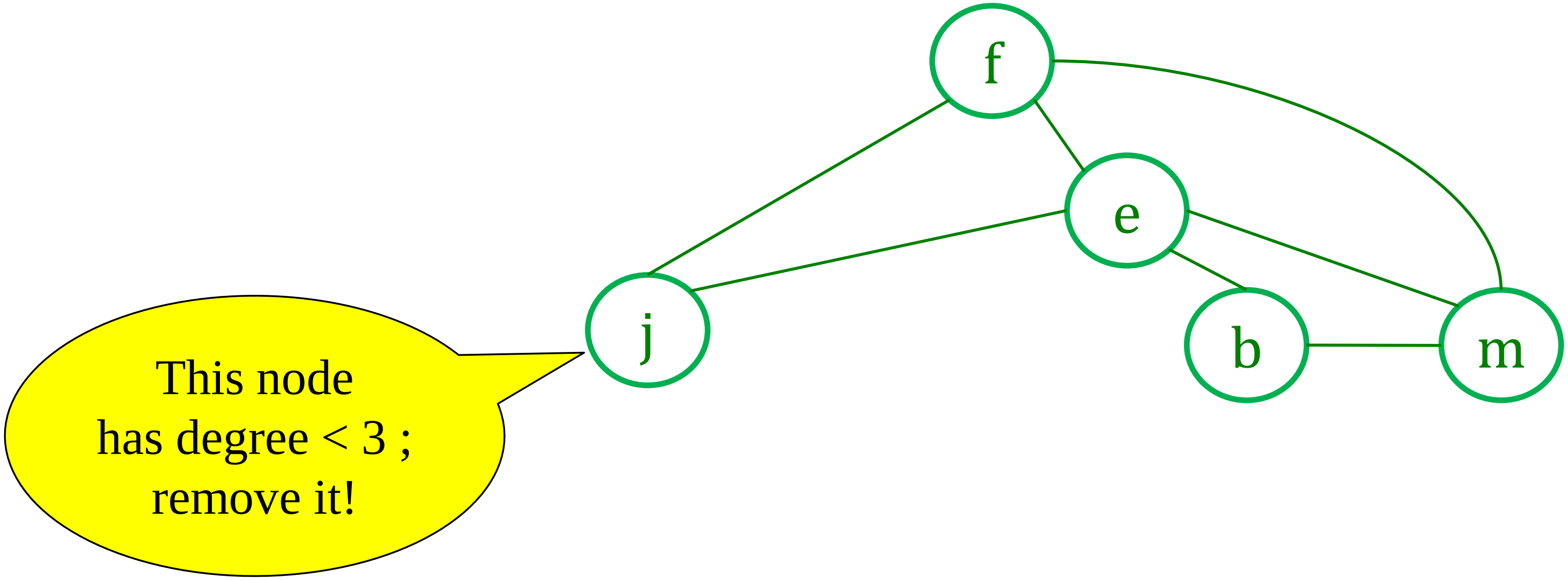
Stack: k g h c

Example: 3-color this graph



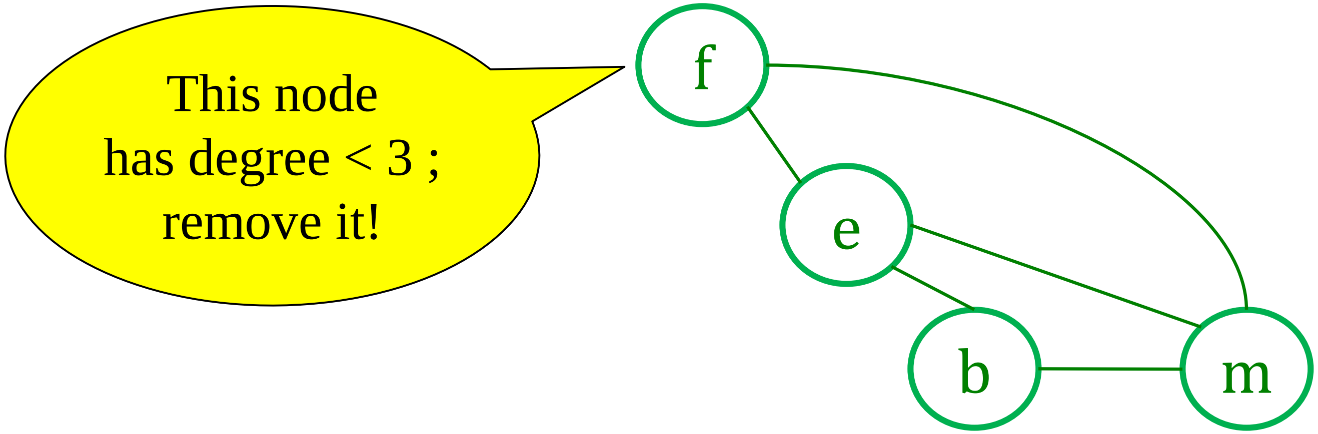
Stack: k g h c

Example: 3-color this graph



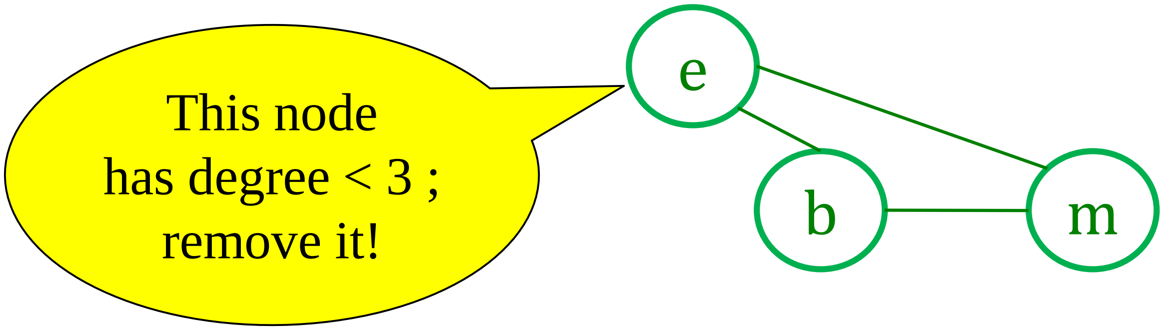
Stack: d k g h c

Example: 3-color this graph



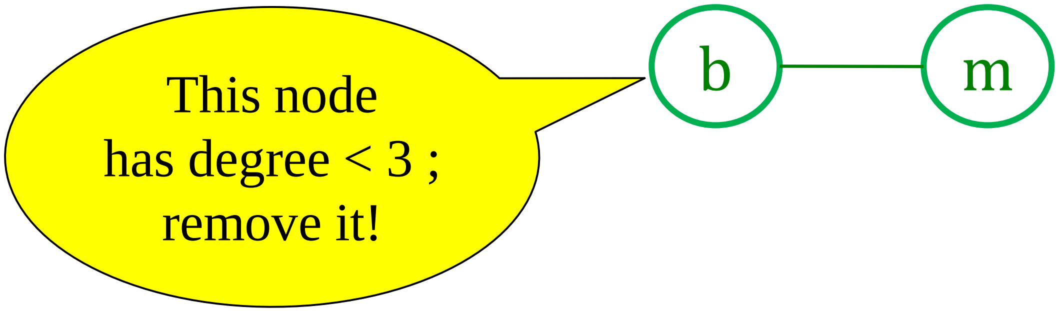
Stack: j d k g h c

Example: 3-color this graph



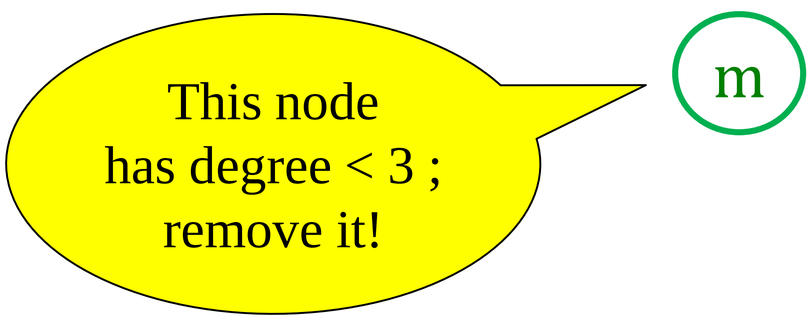
Stack: f j d k g h c

Example: 3-color this graph



Stack: e f j d k g h c

Example: 3-color this graph



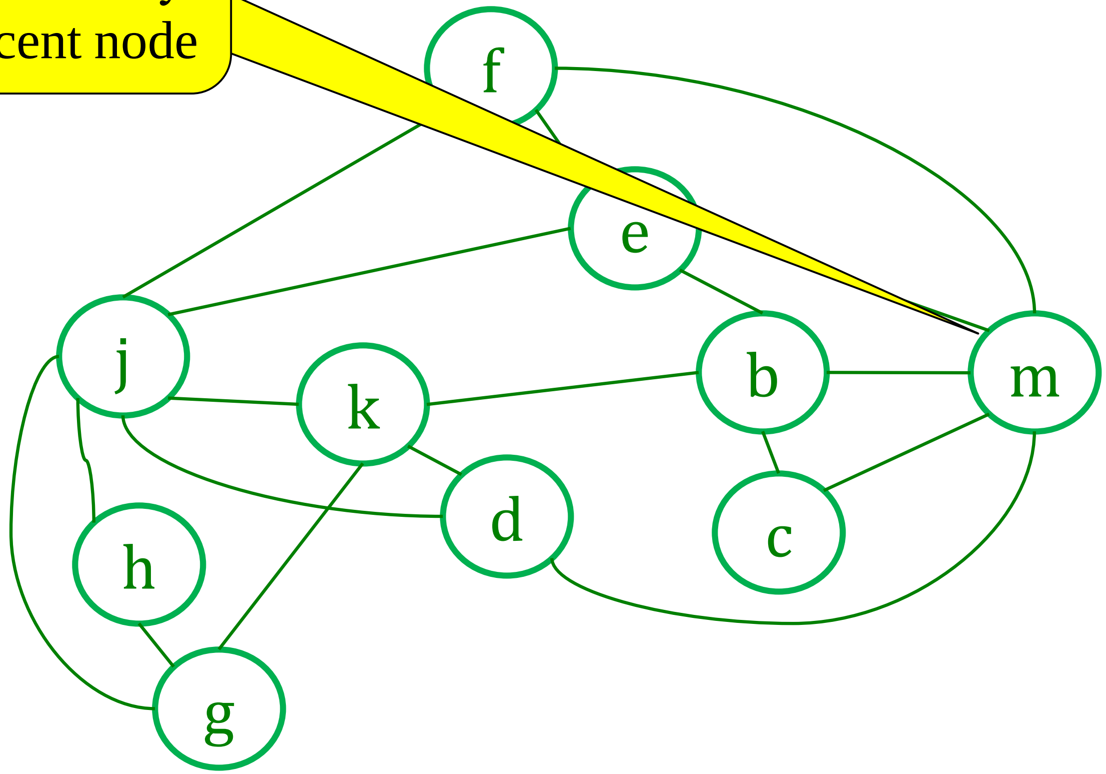
Stack: b e f j d k g h c

Example: 3-color this graph

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

Now, color the nodes in stack order

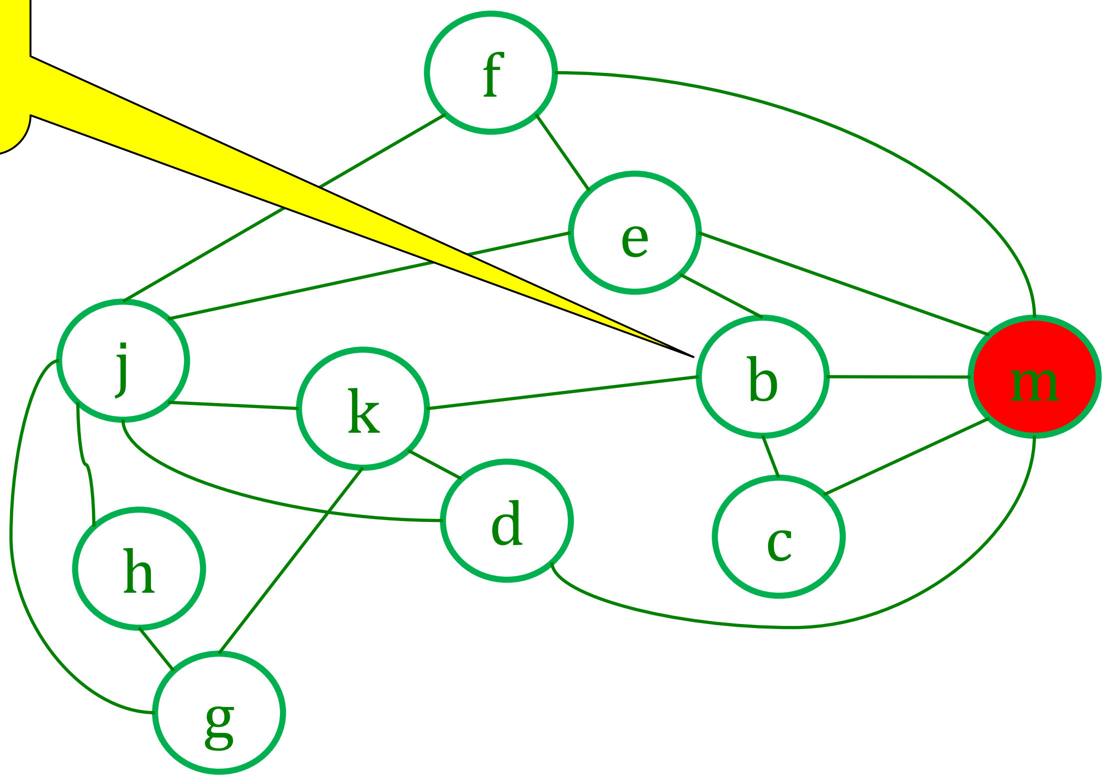
Find a color for this node that's not already used in an adjacent node



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

Now, color the nodes in stack order

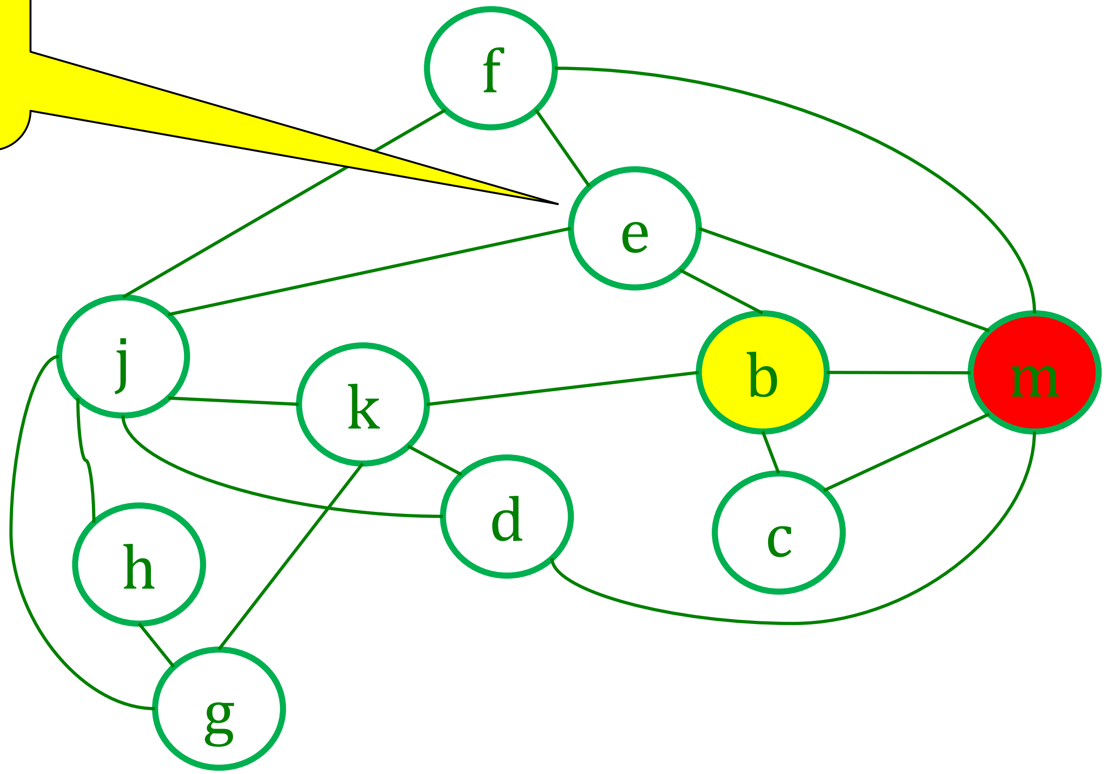
Find a color for this node that's not already used in an adjacent node



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

Now, color the nodes in stack order

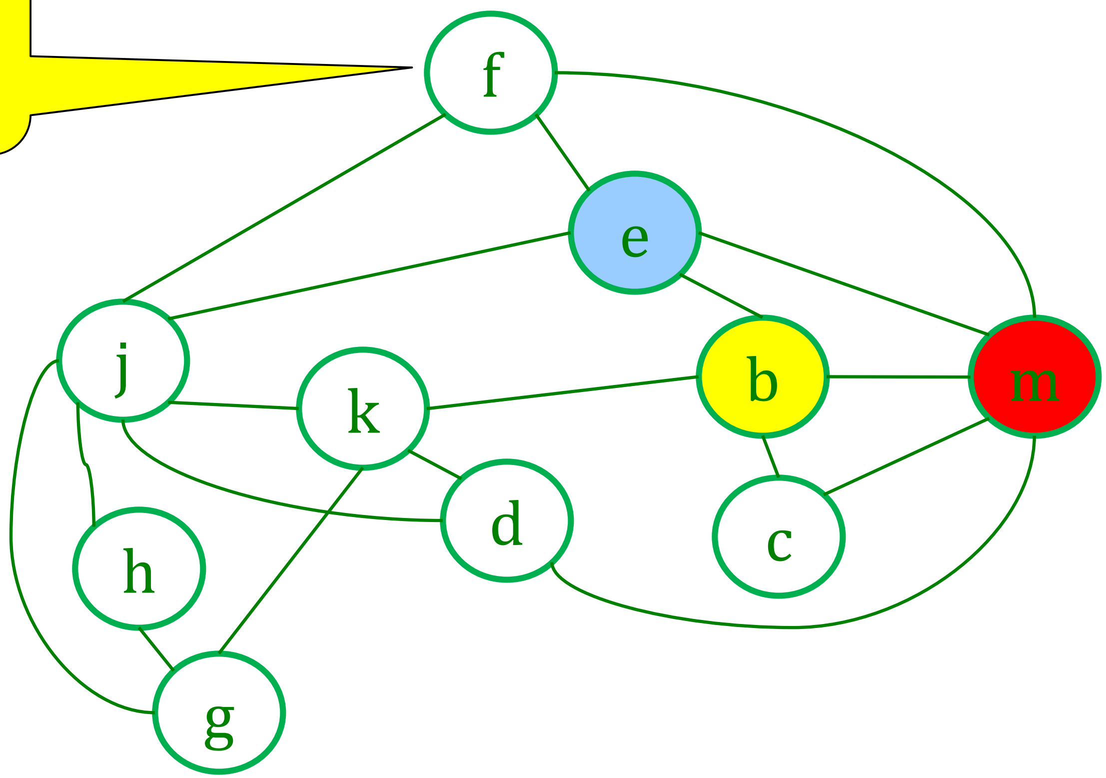
Find a color for this node that's not already used in an adjacent node



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

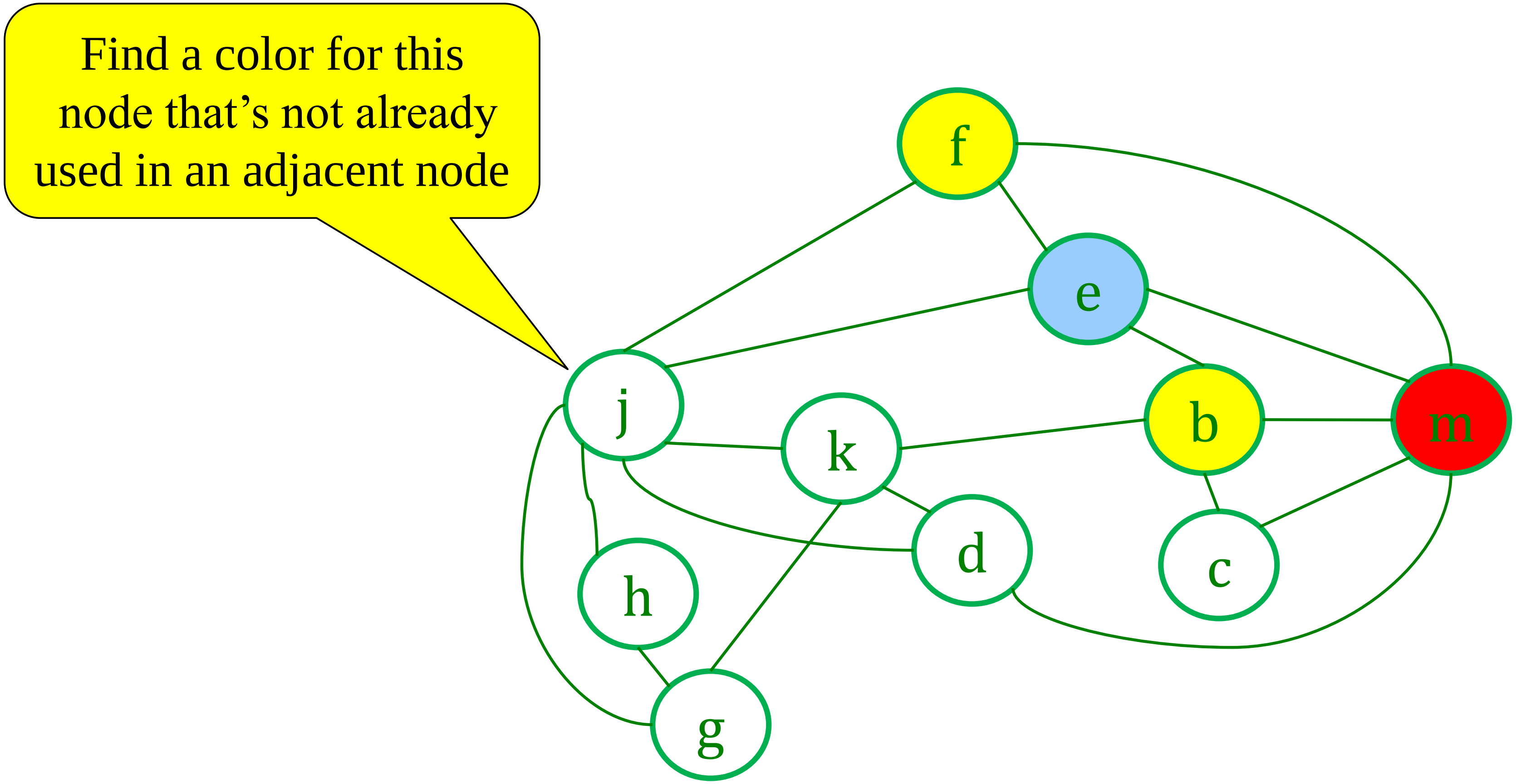
Now, color the nodes in stack order

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



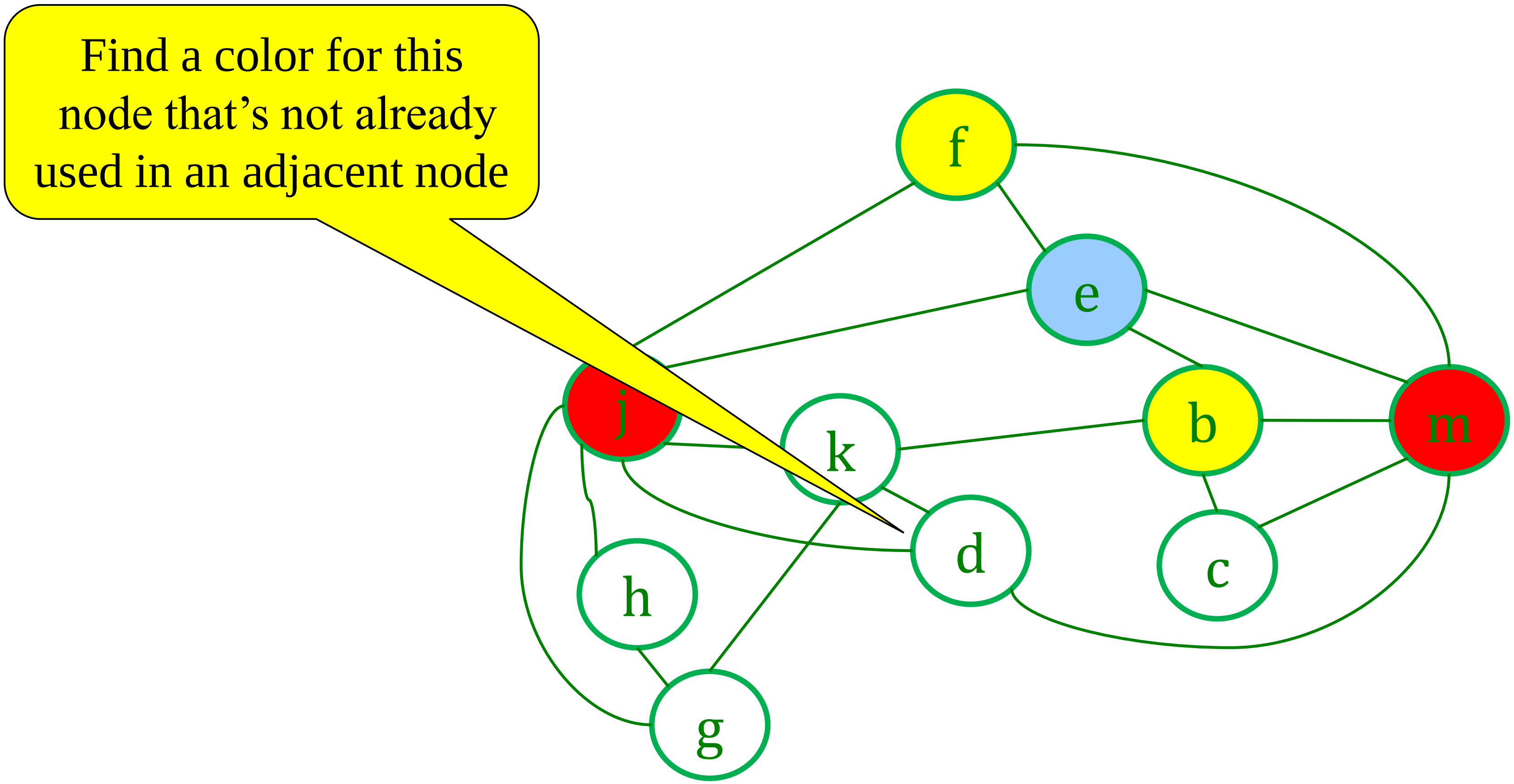
Stack: ~~m b e~~ f j d k g h c

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

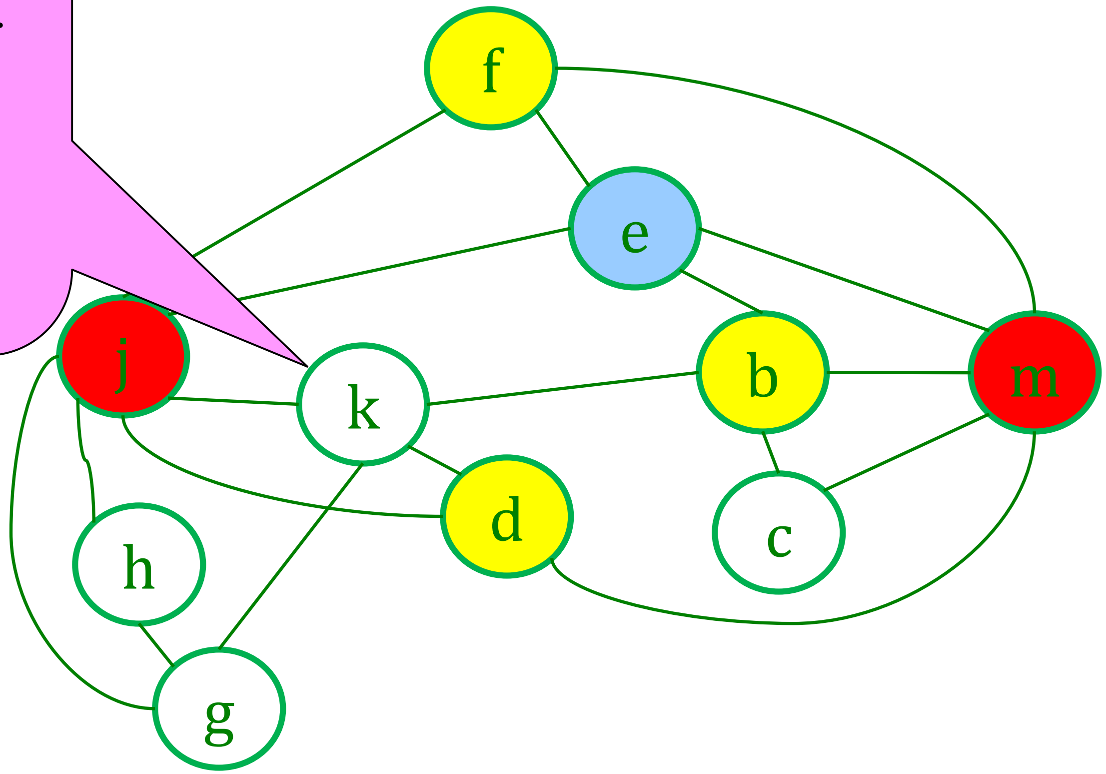


Stack: ~~m b e f j~~ d k g h c

Now, color the nodes in stack order

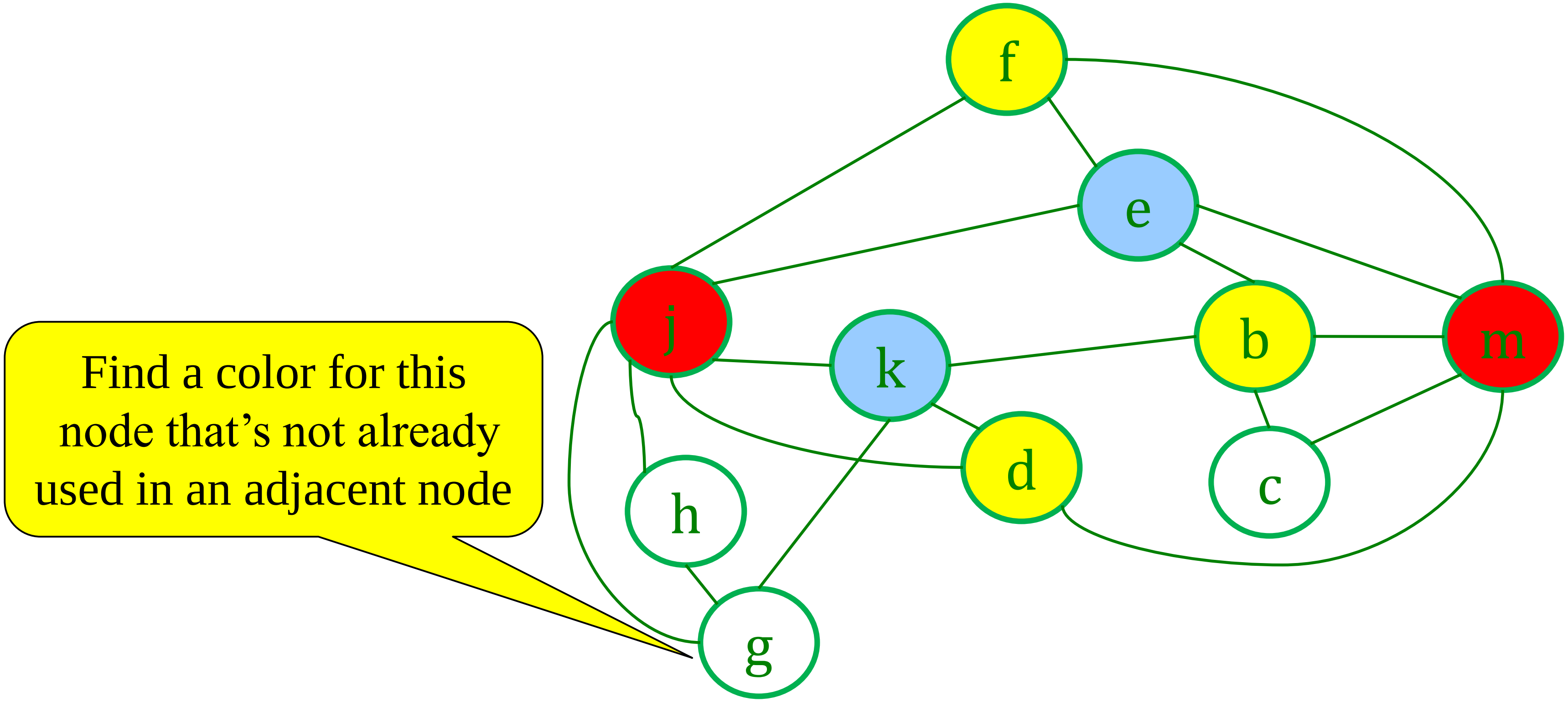
We're about to color node k.
This was the only one that was degree ≥ 3 when we removed it.
Hence, it is not guaranteed that we can find a color for it now.

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



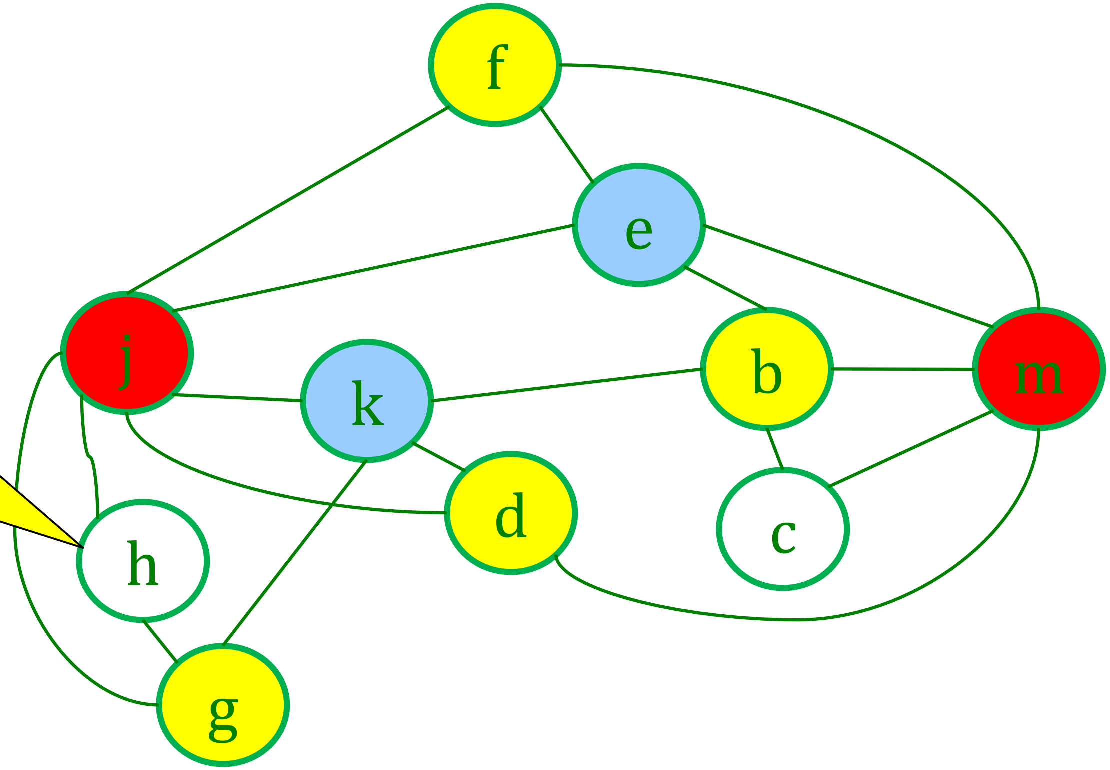
Stack: ~~m b e f j d~~ k g h c

Now, color the nodes in stack order



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

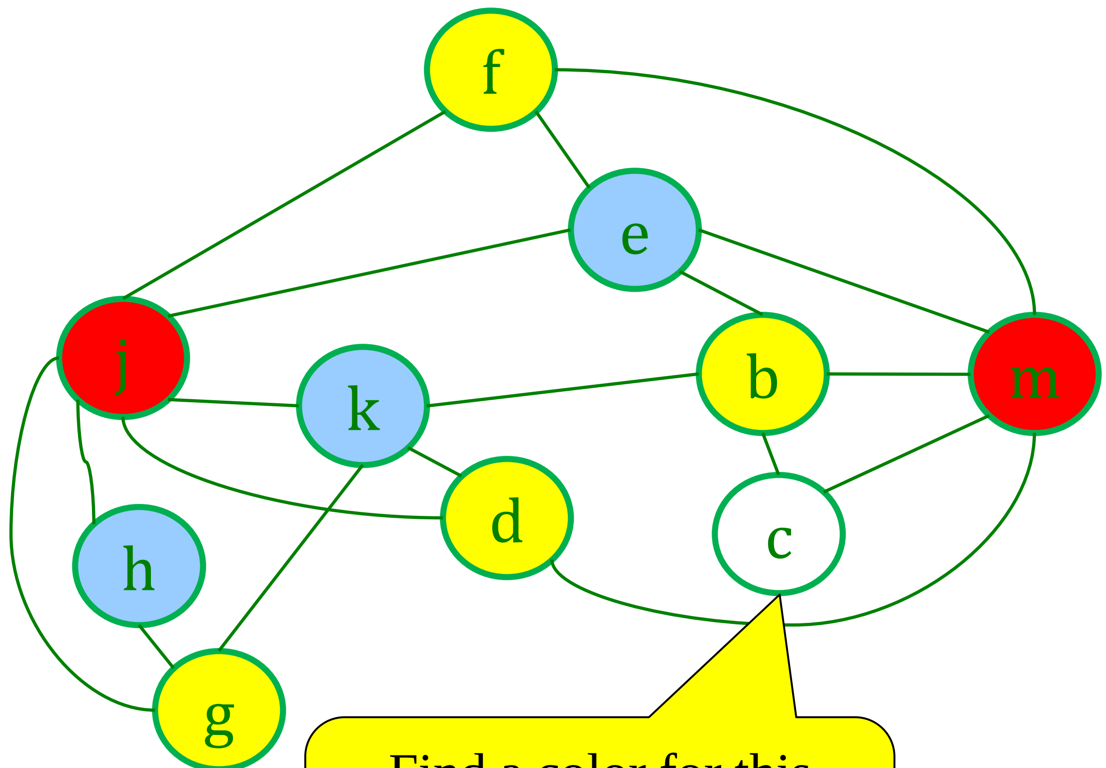
Now, color the nodes in stack order



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

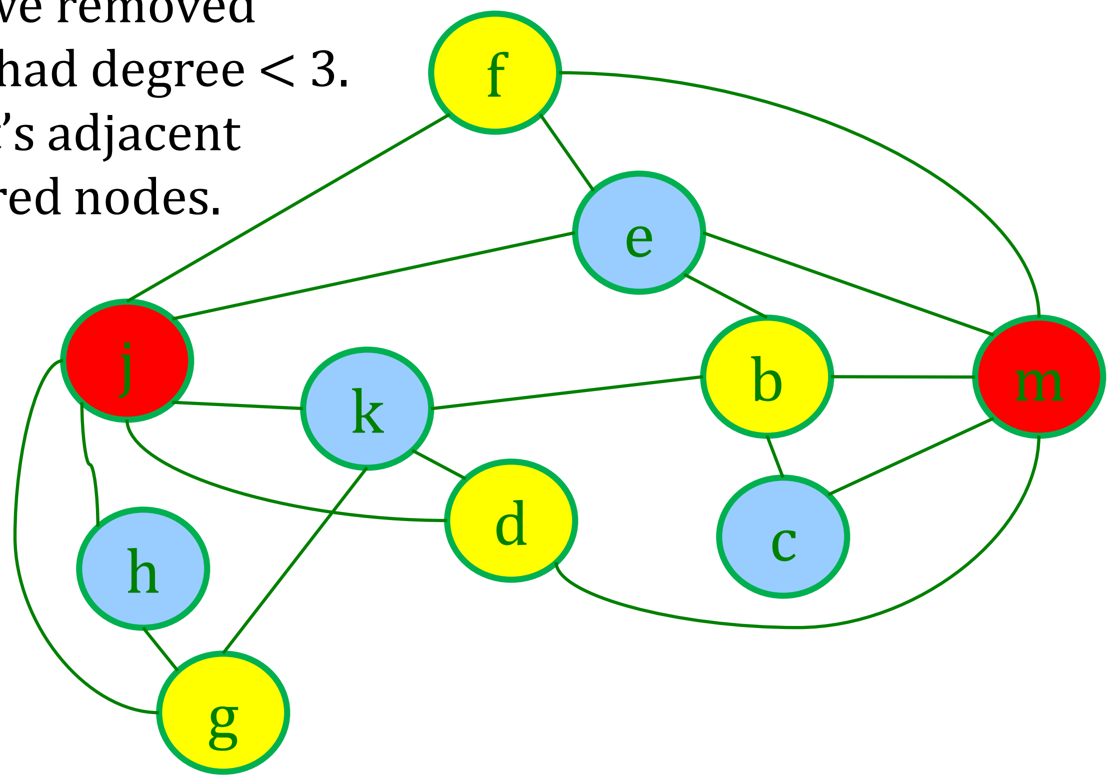


Find a color for this node that's not already used in an adjacent node

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

Now, color the nodes in stack order

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~~

Improvements to the Chaitin algorithm

Kempe 1879 graph coloring algorithm

Chaitin *et al.* 1981 register allocation by coloring

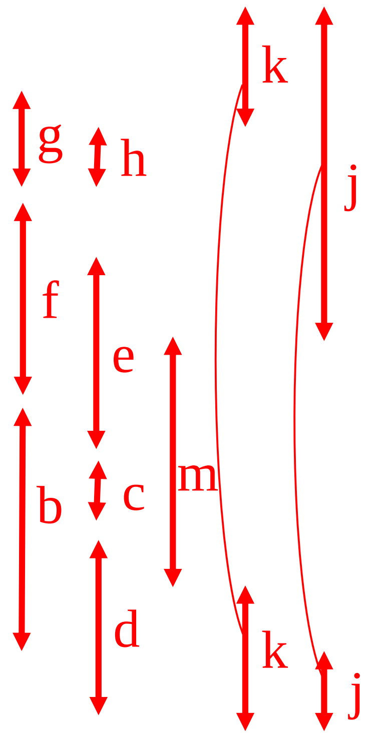
Chaitin 1982: spilling ("leave some nodes uncolored")

Briggs *et al.* 1984: coalescing + improved spilling

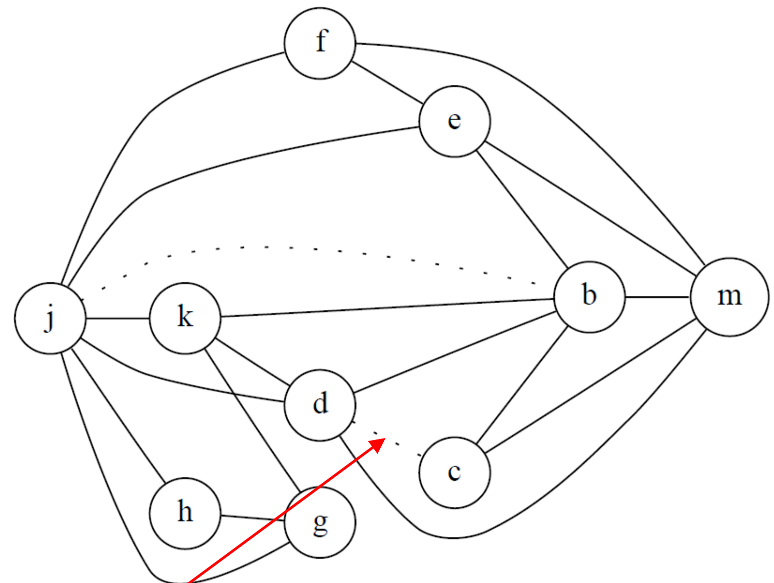
Move coalescing

Procedure P (k, j)
 g := mem[j+12]
 h := k-1
 f := g∗h
 e := mem [j+8]
 m := mem[j+16]
 b := mem[f]
 c := e+8
 d := c
 k := m+4
 j := b
return (d, k, j)

Live ranges



Interference Graph



If these nodes can be colored the same color, then you can delete the *move* instruction

figure 11.1 from *Modern Compiler Implementation in ML*, Andrew W. Appel, Cambridge University Press 1998

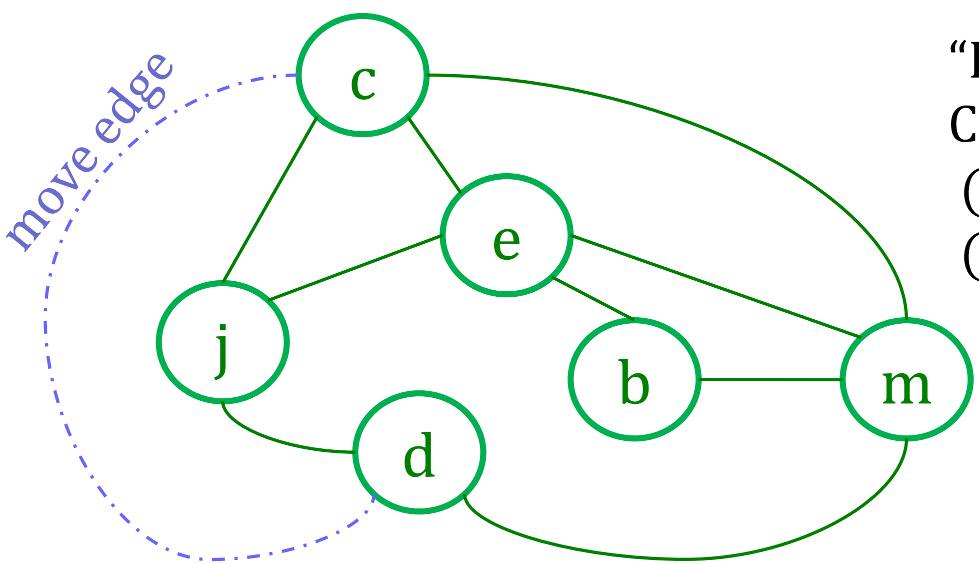
Improvements to the Chaitin algorithm

Kempe 1879 graph coloring algorithm

Chaitin *et al.* 1981 register allocation by coloring

Chaitin 1982: spilling ("leave some nodes uncolored")

Briggs *et al.* 1984: coalescing + improved spilling



"Briggs reduction:"
Coalesce a move edge c-d, if
(1) no interference edge c-d
(2) coalesced node cd has degree <K

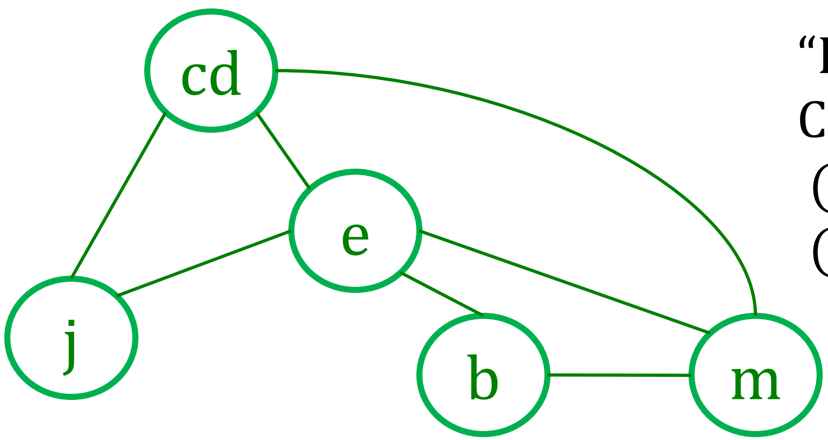
Improvements to the Chaitin algorithm

Kempe 1879 graph coloring algorithm

Chaitin *et al.* 1981 register allocation by coloring

Chaitin 1982: spilling ("leave some nodes uncolored")

Briggs *et al.* 1984: coalescing + improved spilling



"Briggs reduction:"
Coalesce a move edge c-d, if
(1) no interference edge c-d
(2) coalesced node cd has degree <K

Improvements to the Chaitin algorithm

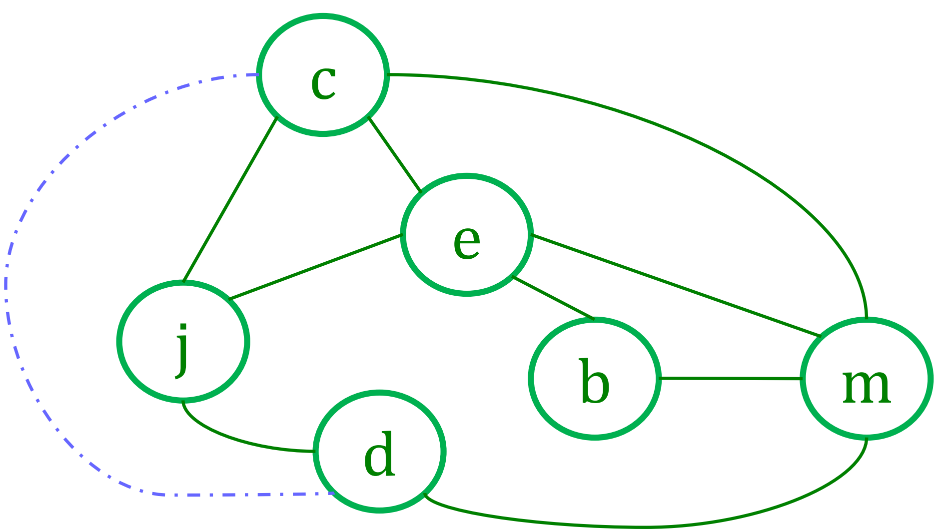
Kempe 1879 graph coloring algorithm

Chaitin 1981 register allocation by coloring

Chaitin 1982: spilling ("leave some nodes uncolored")

Briggs *et al.* 1984: coalescing + improved spilling

L. George & A.W. Appel 1996: **Iterated Register Coalescing**



Interleave Briggs reductions with Kempe reductions

but also:

"George reduction:"

Coalesce a move edge c-d, if

(1) no interference edge c-d

(2) neighbors(d) ⊂ neighbors(c)

ACM Transactions on Programming Languages and Systems, Vol. 18, No. 3, May 1996, Pages 300–324.

Iterated Register Coalescing

LAL GEORGE
Lucent Technologies, Bell Labs Innovations
and
ANDREW W. APPEL
Princeton University

An important function of any register allocator is to target registers so as to eliminate copy instructions. Graph-coloring register allocation is an elegant approach to this problem. If the source and destination of a move instruction do not interfere, then their nodes can be coalesced in the interference graph. Chaitin's coalescing heuristic could make a graph uncolorable (i.e., introduce spills); Briggs et al. demonstrated a conservative coalescing heuristic that preserves colorability. But Briggs's algorithm is *too* conservative and leaves too many move instructions in our programs. We show how to interleave coloring reductions with Briggs's coalescing heuristic, leading to an algorithm that is safe but much more aggressive.

Histories

1850	Guthrie
1880	Kempe
	Heawood
1904	Wernicke
1920	Birkhoff
1960	Heesch
1976	K. Appel, Haken
1996	Robertson, Sanders, Seymour, Thomas
2006	Gonthier

4-color theorem

1700 Leibniz
1850 Babbage
1920 Hilbert
1930 Gödel
Turing
1950 von Neumann
1960 IBM
Cocke
1980 Chaitin
Briggs
1996 L. George & A. Appel

computing

Histories: Logic

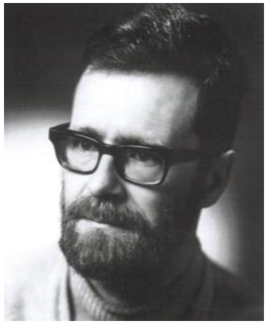
1850	Guthrie
1880	Kempe
	Heawood
1904	Wernicke
1920	Birkhoff
1960	Heesch
1976	K. Appel, Haken
1996	Robertson, Sanders, Seymour, Thomas
2006	Gonthier

4-color theorem

1700 Leibniz
1850 Babbage
1920 Hilbert
1930 Gödel Church
Turing
1950 von Neumann
1960 IBM
1970 Dijkstra, Floyd, Hoare
Cocke
1977 Milner
1980 Chaitin
Briggs
1996 L. George
& A. Appel

computing

Program verification



Edsger Dijkstra
1930-2002

Robert Floyd
1936-2001

Tony Hoare
1934-

David Gries
1939-

Proofs (written by hand, checked by hand) about programs

Edinburgh LCF, the first **Proof Assistant**

1978

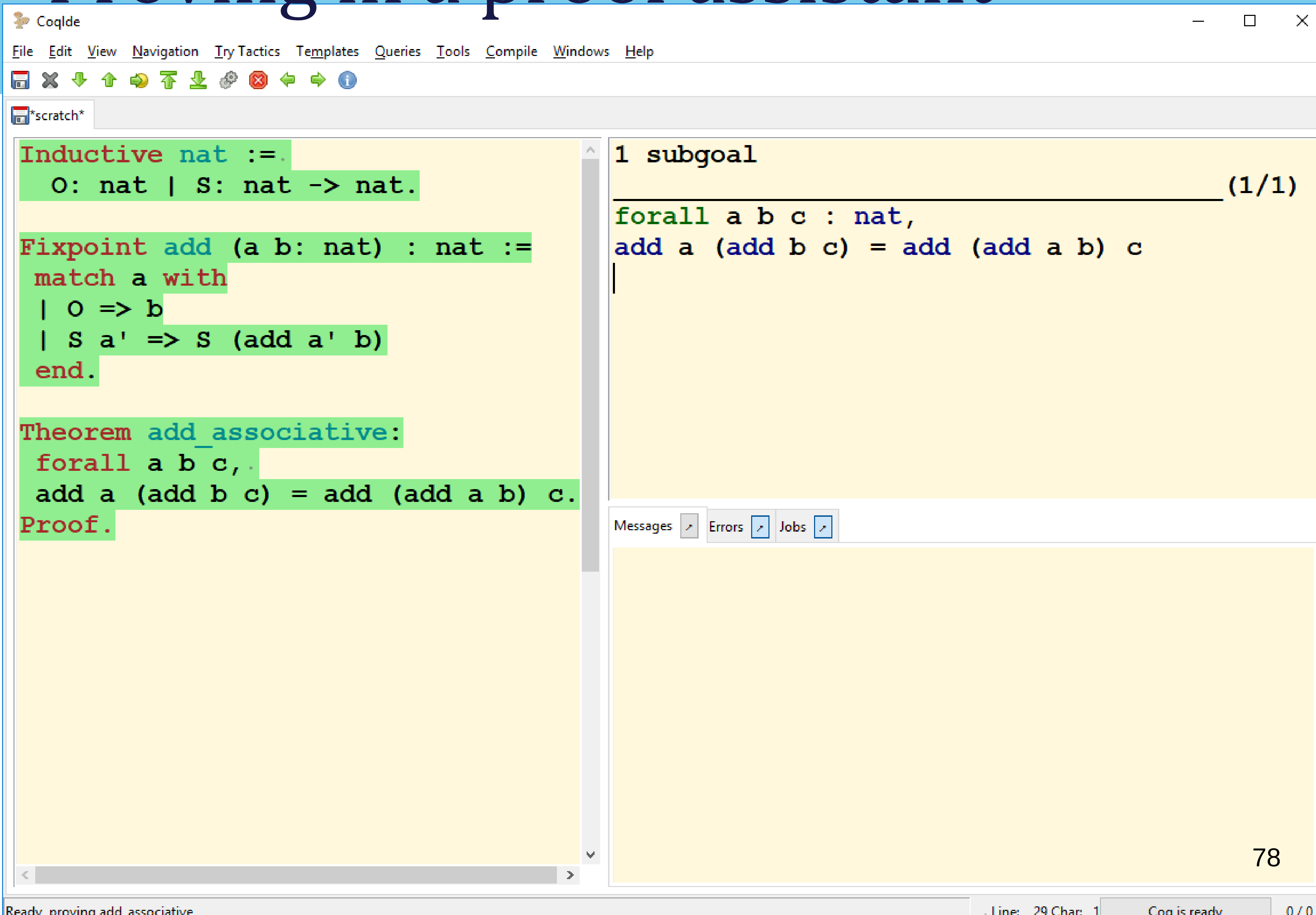
Construct proofs in a "proof language" by hand (like programs).

Proof-checker program ("kernel") checks each step of the proof as you build it.

"Tactic" language permits you to write programs to fill in the trivial parts of the proofs.

Robin Milner
1934-2010

Proving in a proof assistant



Proving in a proof assistant

CoqIde

File Edit View Navigation Try Tactics Templates Queries Tools Compile Windows Help

scratch

```
Inductive nat :=
  O: nat | S: nat -> nat.

Fixpoint add (a b: nat) : nat :=
 match a with
  | O => b
  | S a' => S (add a' b)
 end.

Theorem add_associative:
 forall a b c,
 add a (add b c) = add (add a b) c.
Proof.
intros.
```

```
1 subgoal
a, b, c : nat
______________________________________(1/1)
add a (add b c) = add (add a b) c
```

Messages Errors Jobs

Ready, proving add_associative Line: 14 Char: 8 Coq is ready 0 / 0

Proving in a proof assistant

CoqIde

File Edit View Navigation Try Tactics Templates Queries Tools Compile Windows Help

scratch

```
Inductive nat :=
  O: nat | S: nat -> nat.

Fixpoint add (a b: nat) : nat :=
 match a with
 | O => b
 | S a' => S (add a' b)
 end.

Theorem add_associative:
 forall a b c,
 add a (add b c) = add (add a b) c.
Proof.
intros.
induction a.
* simpl. reflexivity.
* simpl.
  reflexivity.
 rewrite IHa.
  reflexivity.
Qed.
```

```
2 subgoals
b, c : nat
______________________________________(1/2)
add O (add b c) = add (add O b) c
______________________________________(2/2)
add (S a) (add b c) = add (add (S a) b) c
```

Messages Errors Jobs

Ready, proving add_associative Line: 15 Char: 13 Coq is ready 0 / 0

Proving in a proof assistant

CoqIde

File Edit View Navigation Try Tactics Templates Queries Tools Compile Windows Help

scratch

```
Inductive nat :=
  O: nat | S: nat -> nat.

Fixpoint add (a b: nat) : nat :=
 match a with
 | O => b
 | S a' => S (add a' b)
 end.

Theorem add_associative:
 forall a b c,
 add a (add b c) = add (add a b) c.
Proof.
intros.
induction a.
* simpl. reflexivity.
* simpl.
  reflexivity.
 rewrite IHa.
  reflexivity.
Qed.
```

```
1 subgoal
b, c : nat
______________________________________(1/1)
add b c = add b c
```

Messages Errors Jobs

Ready, proving add_associative | Line: 16 Char: 9 | Coq is ready | 0 / 0

Proving in a proof assistant

CoqIde

File Edit View Navigation Try Tactics Templates Queries Tools Compile Windows Help

scratch

```
Inductive nat :=
  O: nat | S: nat -> nat.

Fixpoint add (a b: nat) : nat :=
 match a with
  | O => b
  | S a' => S (add a' b)
 end.

Theorem add_associative:
 forall a b c,
 add a (add b c) = add (add a b) c.
Proof.
intros.
induction a.
* simpl. reflexivity.
* simpl.
  reflexivity.
 rewrite IHa.
  reflexivity.
Qed.
```

```
1 subgoal
a, b, c : nat
IHa : add a (add b c) = add (add a b) c
______________________________________(1/1)
add (S a) (add b c) = add (add (S a) b) c
```

Messages Errors Jobs

Ready, proving add_associative   Line: 17 Char: 2   Coq is ready   0 / 0

Proving in a proof assistant

CoqIde

File Edit View Navigation Try Tactics Templates Queries Tools Compile Windows Help

scratch

```
Inductive nat :=
  O: nat | S: nat -> nat.

Fixpoint add (a b: nat) : nat :=
 match a with
  | O => b
  | S a' => S (add a' b)
 end.

Theorem add_associative:
 forall a b c,
 add a (add b c) = add (add a b) c.
Proof.
intros.
induction a.
* simpl. reflexivity.
* simpl.
  reflexivity.
 rewrite IHa.
  reflexivity.
Qed.
```

```
1 subgoal
a, b, c : nat
IHa : add a (add b c) = add (add a b) c
______________________________________(1/1)
S (add a (add b c)) = S (add (add a b) c)
```

Messages Errors Jobs

```
Error:
In environment
a, b, c : nat
IHa : add a (add b c) = add (add a b) c
Unable to unify "S (add (add a b) c)" with
 "S (add a (add b c))".
```


Ready, proving add_associative Line: 17 Char: 9 Coq is ready 0 / 0

Proving in a proof assistant

CoqIde

File Edit View Navigation Try Tactics Templates Queries Tools Compile Windows Help

scratch

```
Inductive nat :=
  O: nat | S: nat -> nat.

Fixpoint add (a b: nat) : nat :=
 match a with
 | O => b
 | S a' => S (add a' b)
 end.

Theorem add_associative:
 forall a b c,
 add a (add b c) = add (add a b) c.
Proof.
intros.
induction a.
* simpl. reflexivity.
* simpl.
  rewrite IHa.
  reflexivity.
Qed.
```

```
1 subgoal
a, b, c : nat
IHa : add a (add b c) = add (add a b) c
______________________________________(1/1)
S (add (add a b) c) = S (add (add a b) c)
```

Messages Errors Jobs

Ready, proving add_associative Line: 18 Char: 15 Coq is ready 0 / 0

Proving in a proof assistant

CoqIde

File Edit View Navigation Try Tactics Templates Queries Tools Compile Windows Help

scratch

```
Inductive nat :=
  O: nat | S: nat -> nat.

Fixpoint add (a b: nat) : nat :=
 match a with
 | O => b
 | S a' => S (add a' b)
 end.

Theorem add_associative:
 forall a b c,
 add a (add b c) = add (add a b) c.
Proof.
intros.
induction a.
* simpl. reflexivity.
* simpl.
  rewrite IHa.
  reflexivity.
Qed.
```

Messages Errors Jobs

```
add_associative is defined
```

Ready Line: 20 Char: 5 Coq is ready 0 / 0

What's it good for?

Robin Milner's observation (along with the thousands of people who have worked in this field after 1978, including me):

Machine-checked proofs (and proof assistants) are really good for theorems *about computer programs!*

Landmarks of program verification
a personal selection

Andrew Appel
1960-
Foundational
Proof-Carrying Code
2005

Verified
Software Toolchain
2014

Verified SHA/HMAC
cryptographic authentication
2015

Xavier Leroy
1968-
CompCert
optimizing C compiler
2006

Gerwin Klein
1975-
seL4
operating system
2013

Zhong Shao
1968-
CertiKOS
operating system
2015

Math Computing Logic

1850 Guthrie
1880 Kempe
Heawood
1904 Wernicke
1920 Birkhoff
1960 Heesch
1976 K. Appel, Haken
1996 Robertson, Sanders, Seymour, Thomas
2006 Gonthier

4-color theorem

1700 Leibniz
1850 Babbage
1920 Hilbert
1930 Gödel Church
Turing
1950 von Neumann
1960 IBM
Cocke
1980 Chaitin
Briggs
1996 L. George & A. Appel

computing

1970 Dijkstra, Floyd, Hoare
1977 Milner
1990s numerous
21st century more numerous!

software verification

Which part don't you believe?

"Haken's son Armin, by then a graduate student at … Berkeley, gave a lecture on the four-colour problem…. At the end, the audience split into two groups: the over-forties could not be convinced that a proof by computer was correct, while the under-forties* could not be convinced that a proof containing 700 pages of hand calculations could be correct."

*By now that would be, "people under 80"

2005 (see also Notices of the AMS 2008)

A computer-checked proof of the Four Colour Theorem

Georges Gonthier
Microsoft Research Cambridge

This report gives an account of a successful formalization of the proof of the Four Colour Theorem, which was fully checked by the Coq v7.3.1 proof assistant [13]. This proof is largely based on the mixed mathematics/computer proof [26] of Robertson et *al*, but contains original contributions as well. This document is organized as follows: section 1 gives a historical introduction to the problem and positions our work in this setting; section 2 defines more precisely what was proved; section 3 explains the broad outline of the proof; section 4 explains how we exploited the features of the Coq assistant to conduct the proof, and gives a brief description of the tactic shell that we used to write our proof scripts; section 5 is a detailed account of the formal proof (for even more details the actual scripts can be consulted); section 6 is a chronological account of how the formal proof was developed; finally, we draw some general conclusions in section 7.

Georges Gonthier
1962-

“Trusted base:” 141 lines

Definition real_model := . . . 100 lines of Coq axiomatizing real numbers...

Definition map : Type :=
Definition simple_map: Type → Prop :=
Definition map_colorable: ℕ → map → Prop :=

40 lines of elementary topology

Theorem four_color:
∀ R : real_model, ∀m : map R, simple_map m → map_colorable 4 m.
Proof.
. . . 60,000 lines of Coq ...
Qed.

Math Computing Logic

	Math	Computing	Logic
		1700 Leibniz	
1850	Guthrie	1850 Babbage	
1880	Kempe	1920 Hilbert	
1904	Heawood Wernicke	1930 Gödel Church Turing	
1920	Birkhoff	1950 von Neumann	
1960	Heesch	1960 IBM	
1976	K. Appel, Haken	Cocke 1980 Chaitin Briggs	1977 Milner
1996	Robertson, Sanders, Seymour, Thomas	1996 L. George & A. Appel	1990s numerous
		computing	
2005	Gonthier		

4-color theorem machine-checked proof

Formal Verification of Coalescing Graph-Coloring Register Allocation

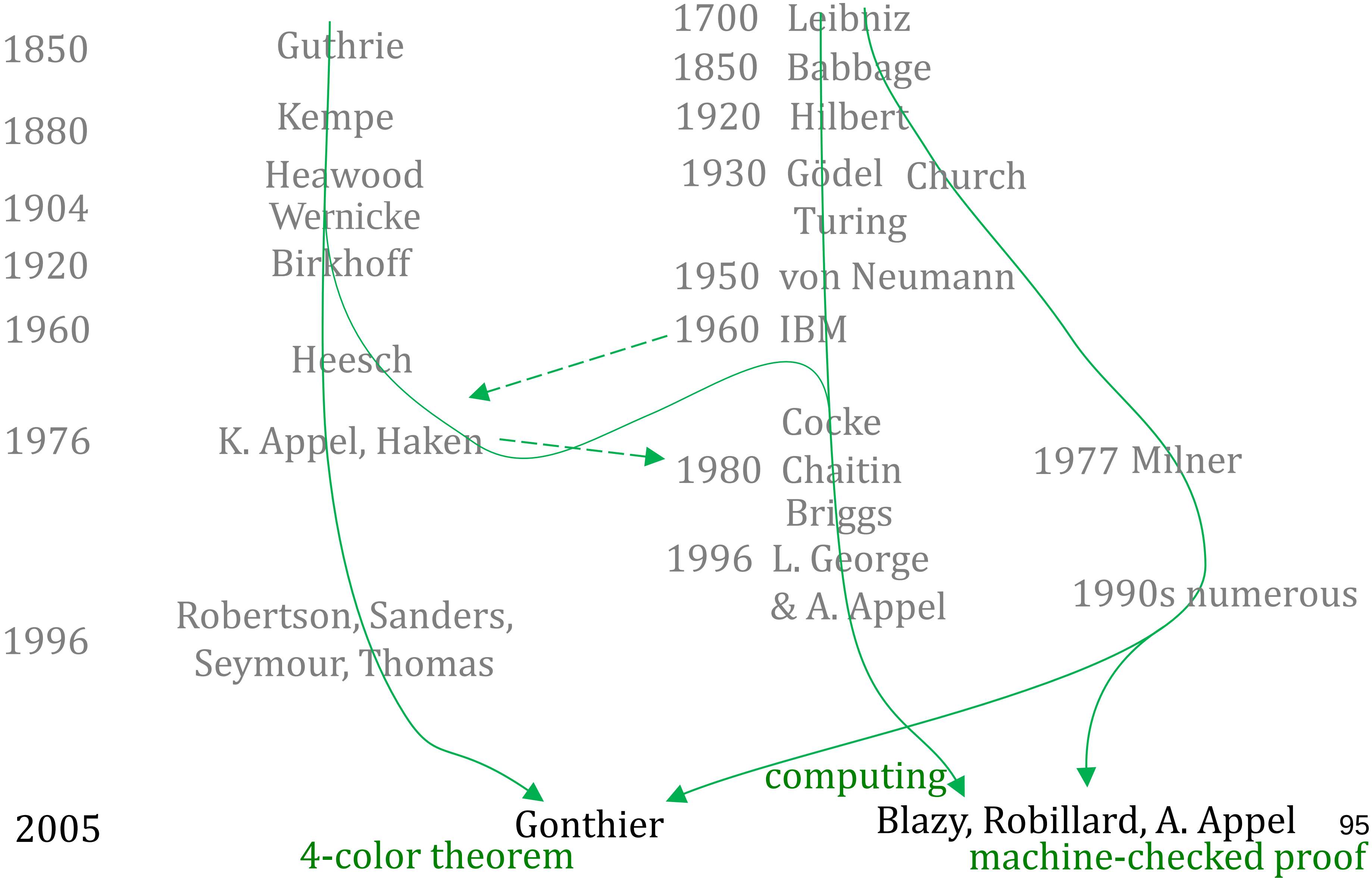
Sandrine Blazy[1], Benoît Robillard[2], and Andrew W. Appel[3]

[1] IRISA - Université Rennes 1
[2] CEDRIC - ENSIIE
[3] Princeton University

19th European Symposium on Programming, 2010

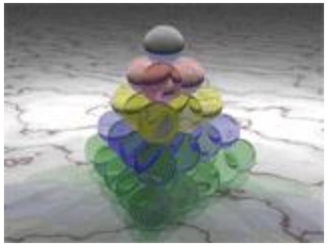
Abstract. Iterated Register Coalescing (IRC) is a widely used heuristic for performing register allocation via graph coloring. Many implementations in existing compilers follow (more or less faithfully) the imperative algorithm published in 1996. Several mistakes have been found in some of these implementations.
In this paper, we present a formal verification (in Coq) of the whole IRC algorithm. We detail a specification that can be used as a reference for IRC. We also define the theory of register-interference graphs; we implement a purely functional version of the IRC algorithm, and we prove the total correctness of our implementation. The automatic extraction of our IRC algorithm into Caml yields a program with competitive performance. This work has been integrated into the CompCert verified compiler.

Math Computing Logic



In mathematics, as well

Kepler conjecture (1611):
Face-centered cubic is
densest possible sphere packing



Hales proof (1998):
5000 planar graphs,
each with a computerized
nonlinear optimization calculation

Referees: we're 99% sure it's correct

Hales *et al.* 2004-2014:
Flyspec project- Formal verification
in HOL Light proof assistant

Thomas Hales
1958-

- Project Director: Thomas Hales
- Project Managers: Ta Thi Hoai An, Mark Adams
- HOL Light libraries and support: John Harrison,
- Isabelle Tame Graph Classification:
  Gertrud Bauer, Tobias Nipkow,
- Chief Programmer: Alexey Solovyev,
  - Nonlinear inequalities: Victor Magron, Sean McLaughlin, Roland Zumkeller,
  - Linear Programming: Steven Obua,
  - Microsoft Azure Cloud support: Daron Green, Joe Pleso, Dan Synek, Wenming Ye,
- Chief Formalizer: Hoang Le Truong,
  - Text formalization: Jason Rute, Dang Tat Dat, Nguyen Tat Thang, Nguyen Quang Truong, Tran Nam Trung, Trieu Thi Diep, Vu Khac Ky, Vuong Anh Quyen,
- Student Projects: Catalin Anghel, Matthew Wampler-Doty, Nicholas Volker, Nguyen Duc Tam, Nguyen Duc Thinh, Vu Quang Thanh,
- Proof Automation: Cezary Kaliszyk, Josef Urban,
- Editing: Erin Susick, Laurel Martin, Mary Johnston,
- External Advisors and Design: Freek Wiedijk, Georges Gonthier, Jeremy Avigad, Christian Marchal,
- Institutional Support: NSF, Microsoft Azure Research, William Benter Foundation, University of Pittsburgh, Radboud University, Institute of Math (VAST), VIASM.

Conclusions

- Graph coloring, with or without proofs, is widespread in Computer Science
- Computer-checked proofs are widespread, and important, in Computer Science
- Computer-checked proofs are even becoming important in Mathematics