Graph-Coloring Register Allocation

CS4410: Spring 2013

Last Time:

Dataflow analysis on CFG's

Find iterative solution to equations:

- Available Expressions (forwards):
  - $Din[L] = Dout[L_1] \cap \ldots \cap Dout[L_n]$
    where $pred[L] = \{L_1,\ldots,L_n\}$
  - $Dout[L] = (Din[L] - Kill[L]) \cup Gen[L]$
- live variable sets (backwards):
  - $LiveIn[L] = Gen[L] \cup (LiveOut[L] - Kill[L])$
  - $LiveOut[L] = LiveIn[L_1] \cup \ldots \cup LiveIn[L_n]$
    where $succ[L] = \{L_1,\ldots,L_n\}$

Register Allocation

Goal is to assign each temp to one of k registers.

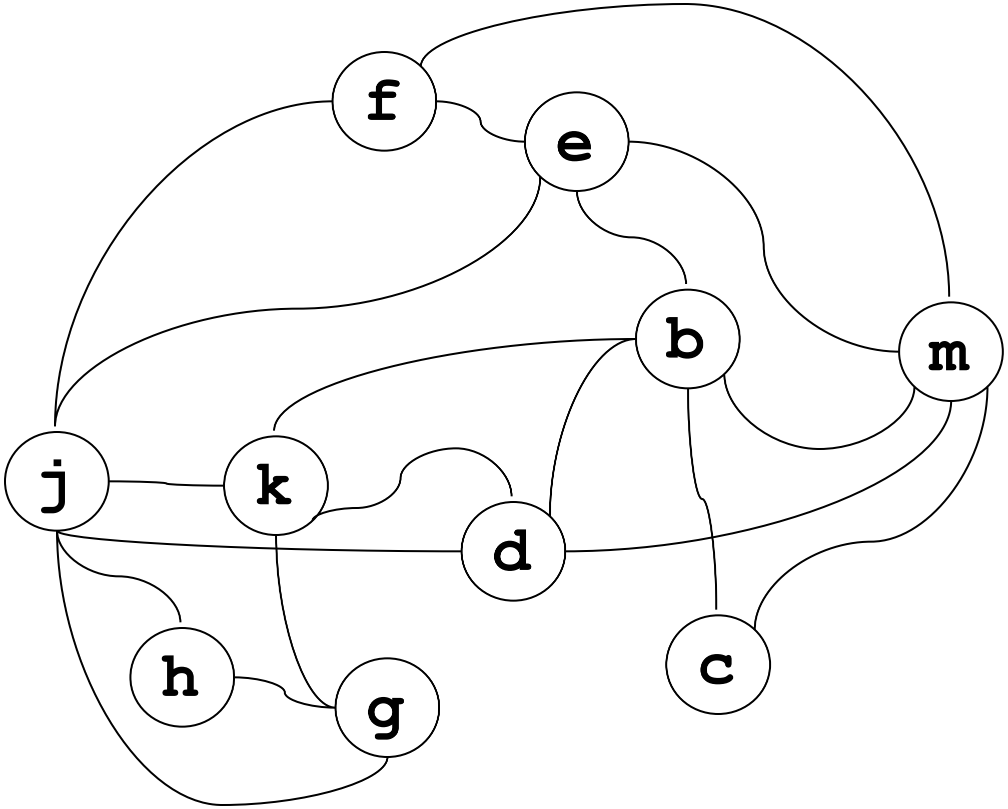
In general, an NP-complete problem.

So we use a greedy heuristic:

- Build interference graph G
  - G(x,y)=true if x & y are live at same point.
- Simplify the graph G
  - If x has degree < k, push x and simplify G-{x}
  - if no such x, then we need to *spill* some temp.
- Once graph is empty, start popping temps and assigning them registers.
  - Always have a free register since sub-graph G-{x} can't have >= k interfering temps.

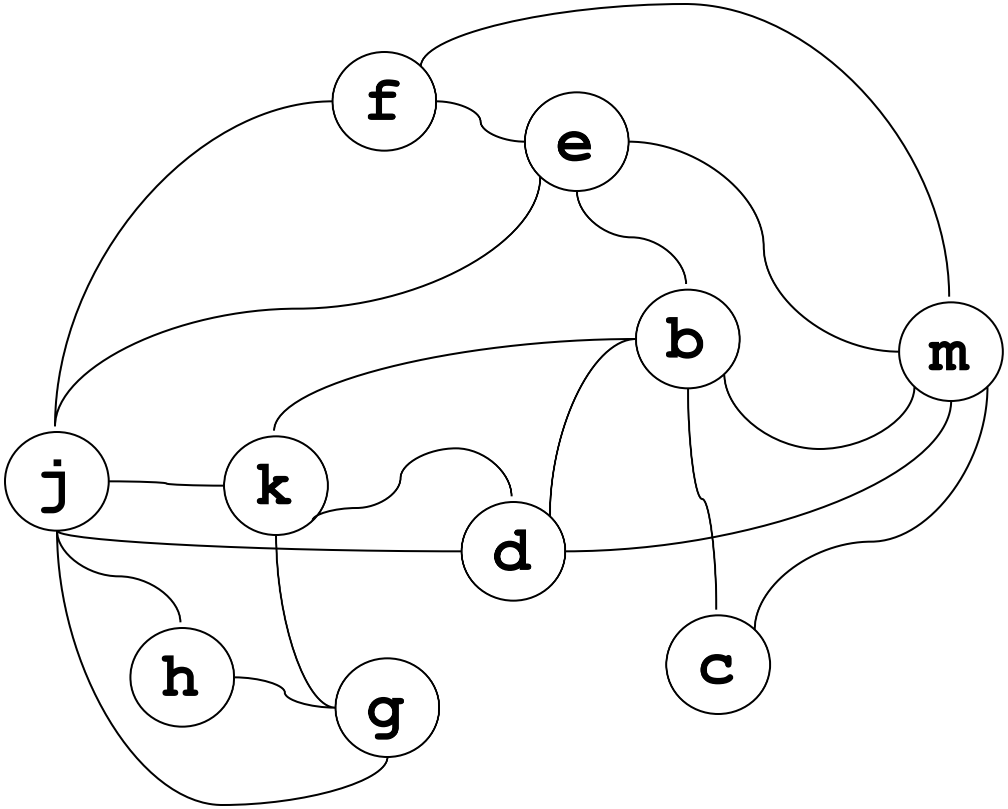
Example from book

```
{live-in: j, k}
g := *(j+12)
h := k - 1
f := g * h
e := *(j+8)
m := *(j+16)
b := *(f+0)
c := e + 8
d := c
k := m + 4
j := b
{live-out: d,j,k}
```



Simplification (4 regs)

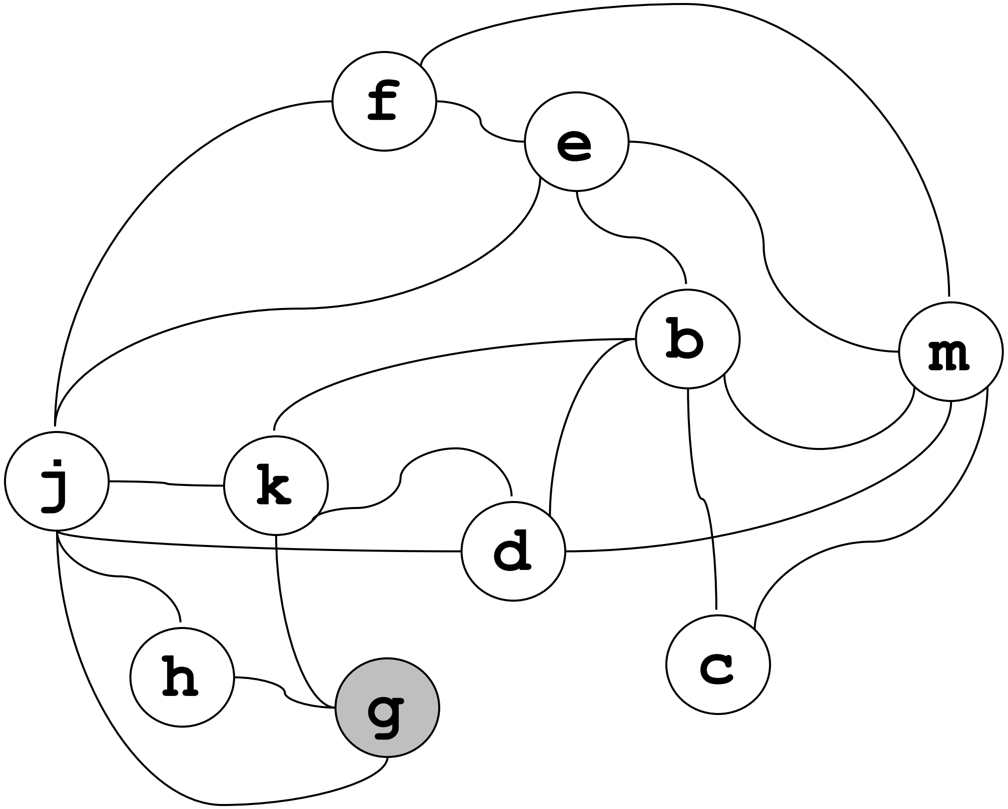
Stack:



Simplification

Stack:

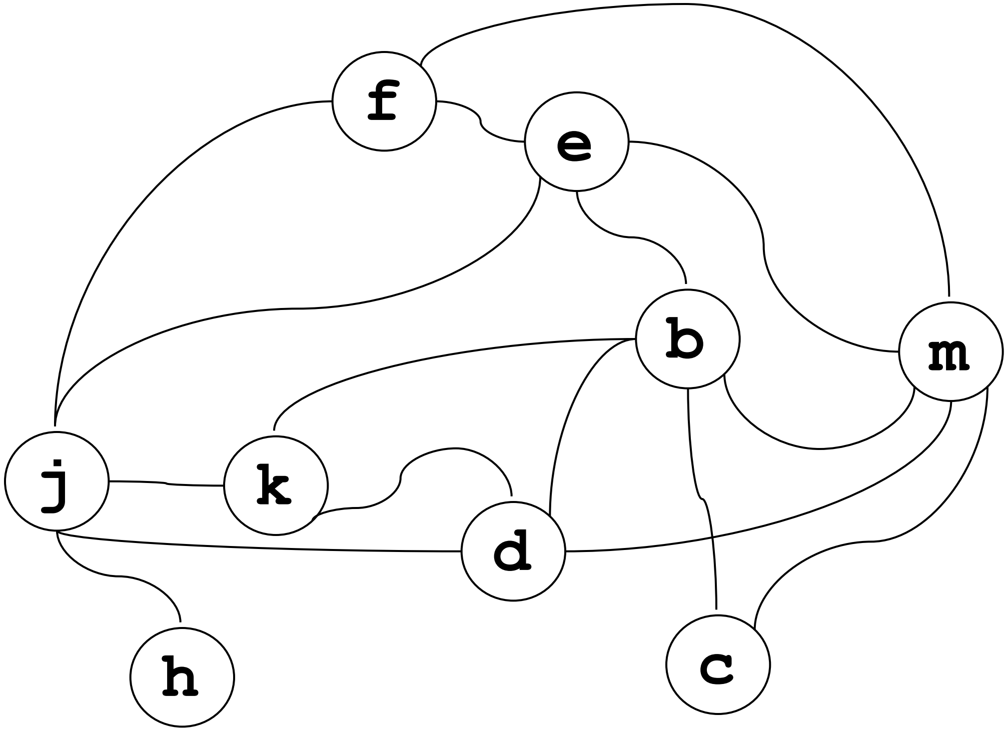
g



Simplification

Stack:

g

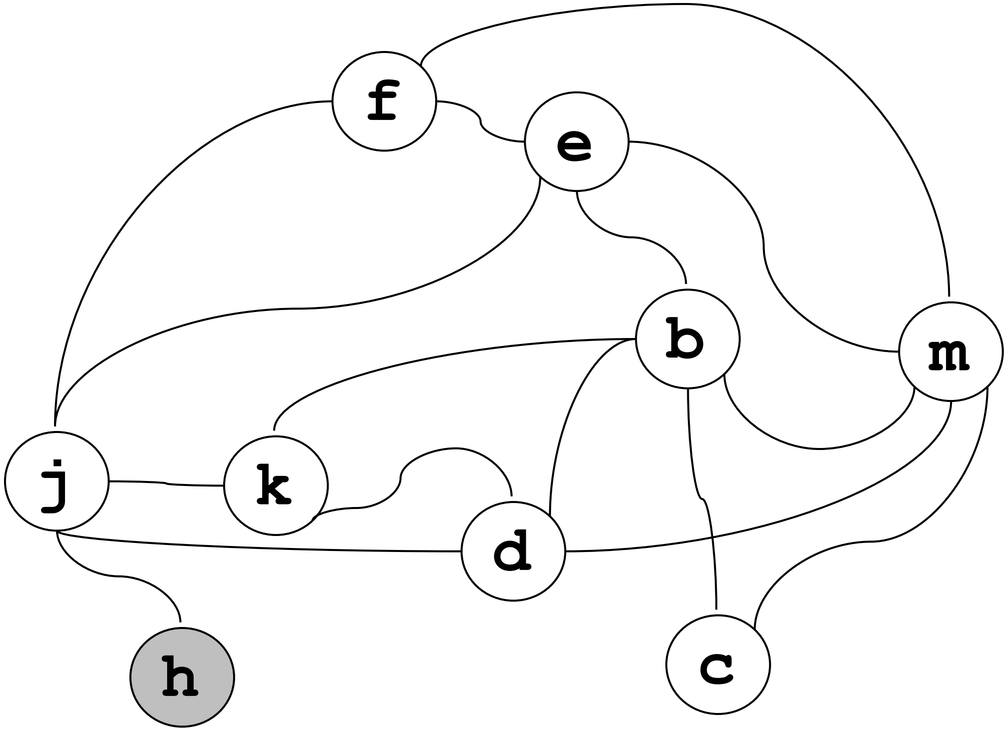


Simplification

Stack:

g

h

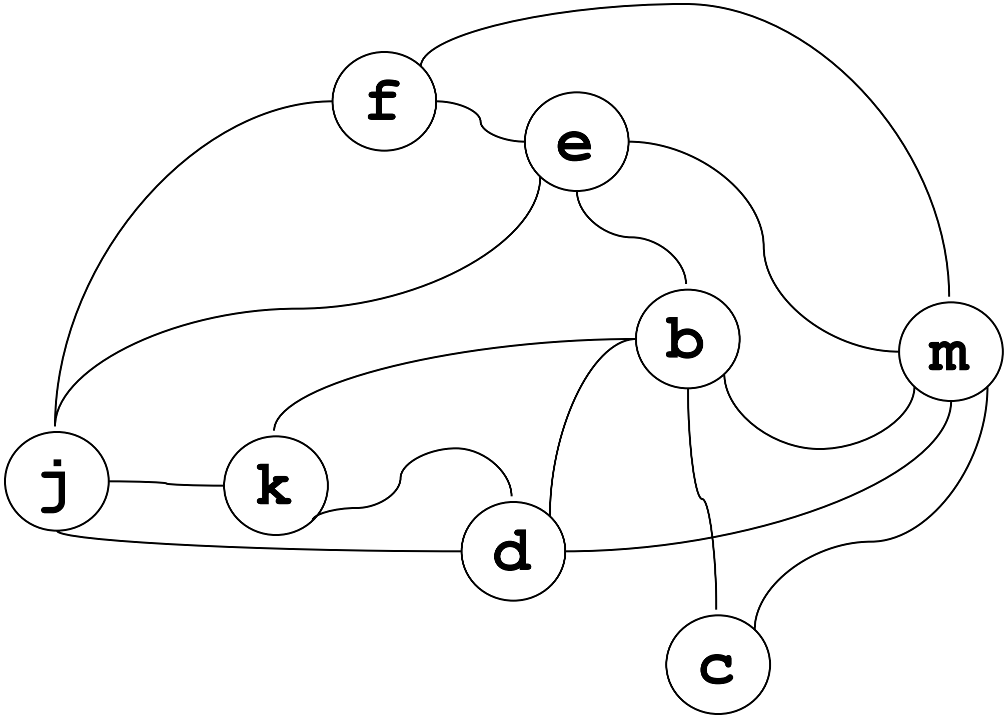


Simplification

Stack:

g

h



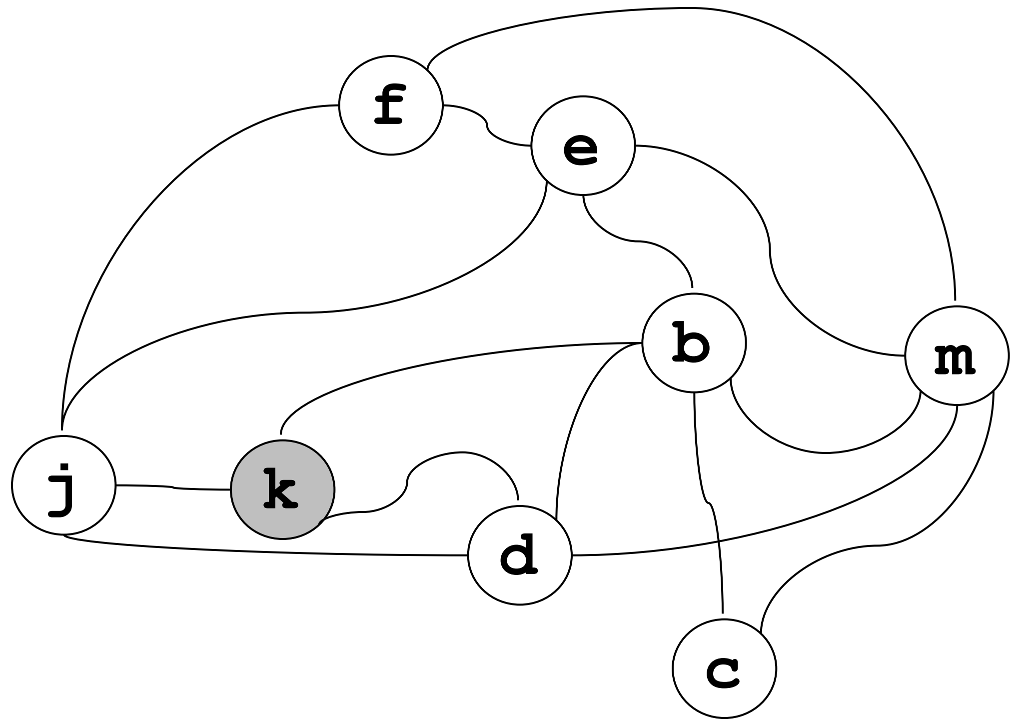
Simplification

Stack:

g

h

k



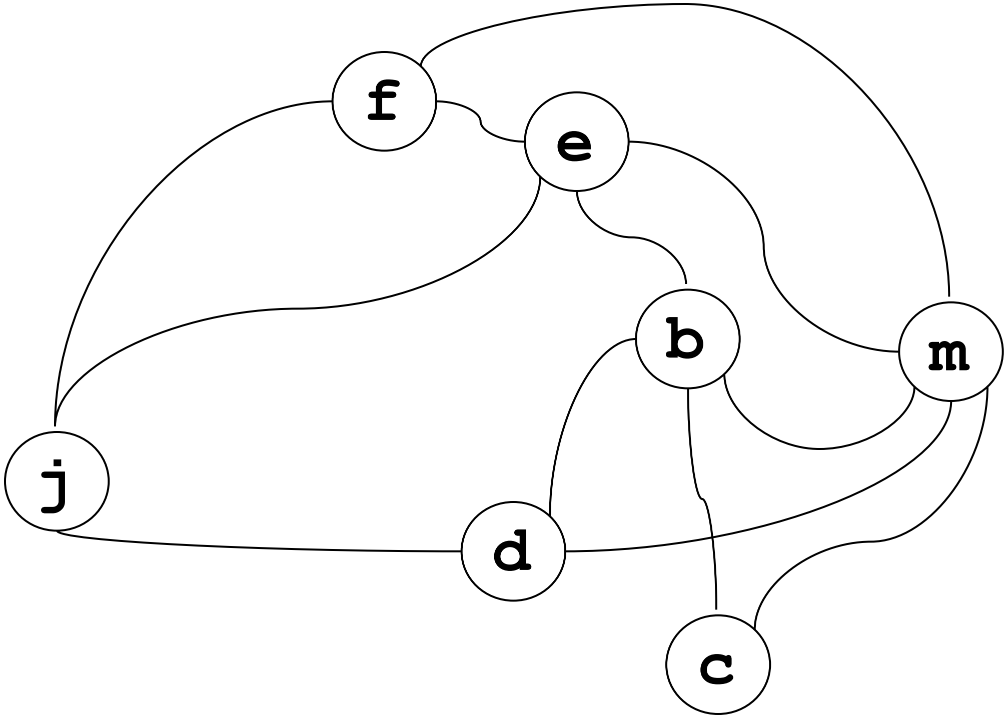
Simplification

Stack:

g

h

k



Simplification

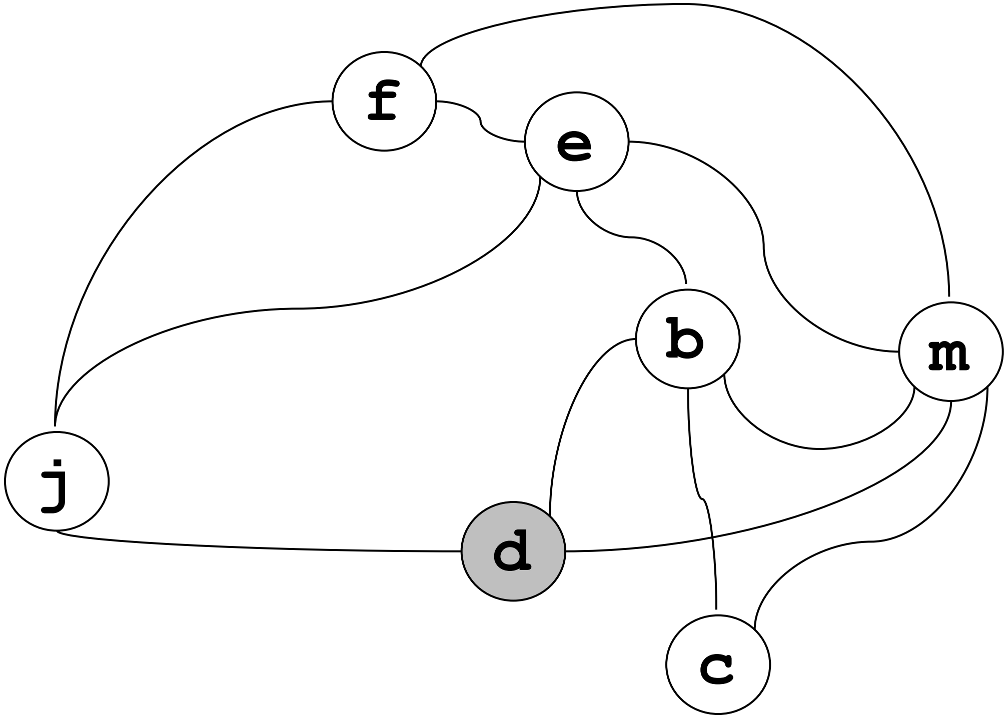
Stack:

g

h

k

d



Simplification

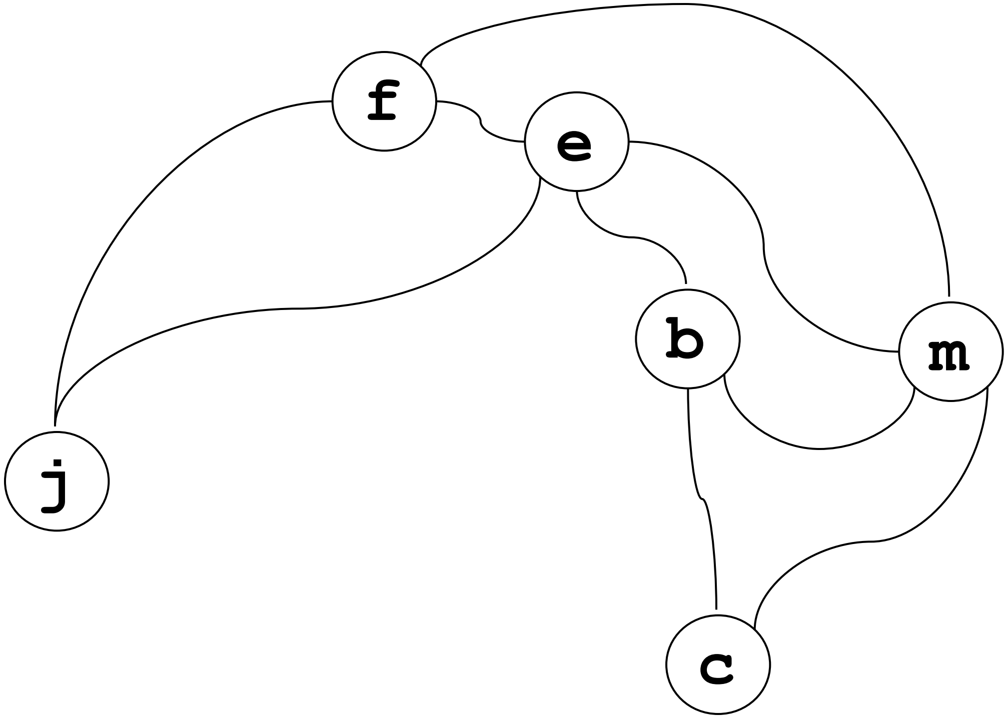
Stack:

g

h

k

d



Simplification

Stack:

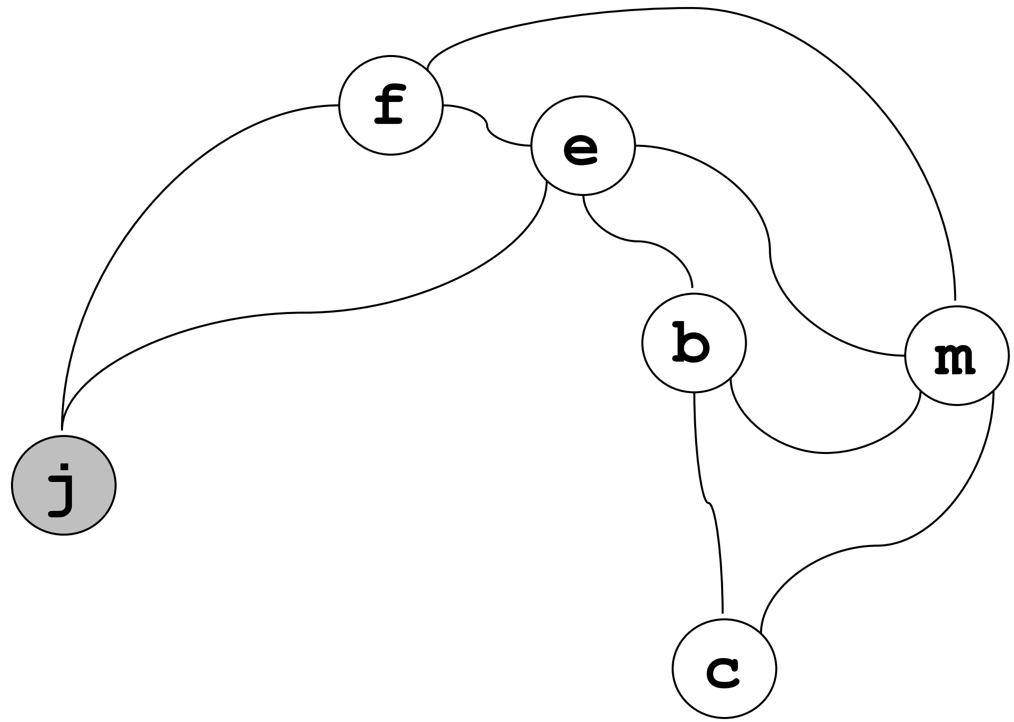
g

h

k

d

j



Simplification

Stack:

g

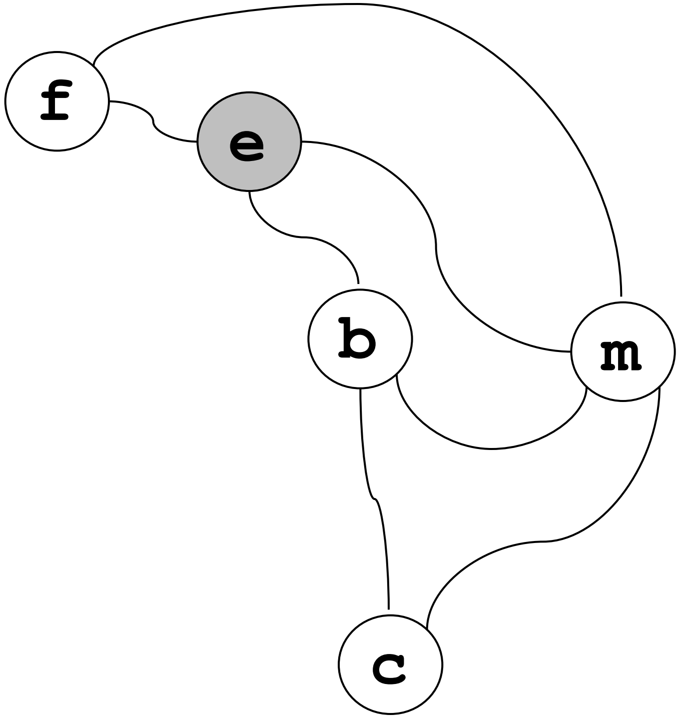
h

k

d

j

e



Simplification

Stack:

g

h

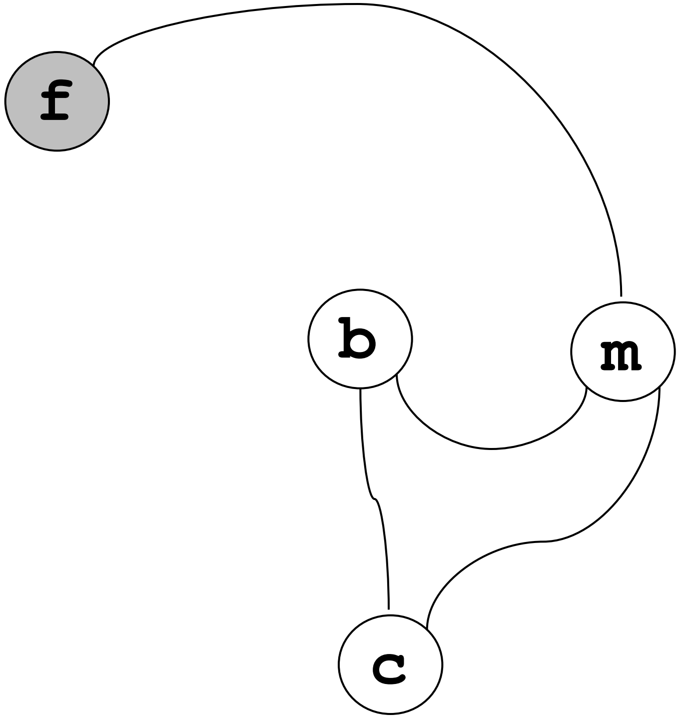
k

d

j

e

f



Simplification

Stack:

g

h

k

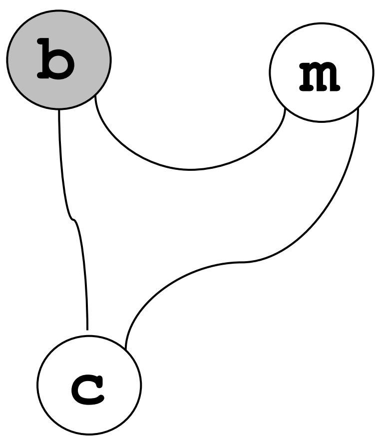
d

j

e

f

b



Simplification

Stack:

g

h

k

d

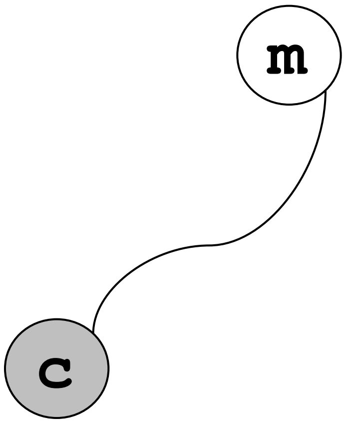
j

e

f

b

c



Simplification

Stack:

`g`

`h`

`k`

`d`

`j`

`e`

`f`

`b`

`c`

`m`

Select:

Stack:

g

h

k

d

j

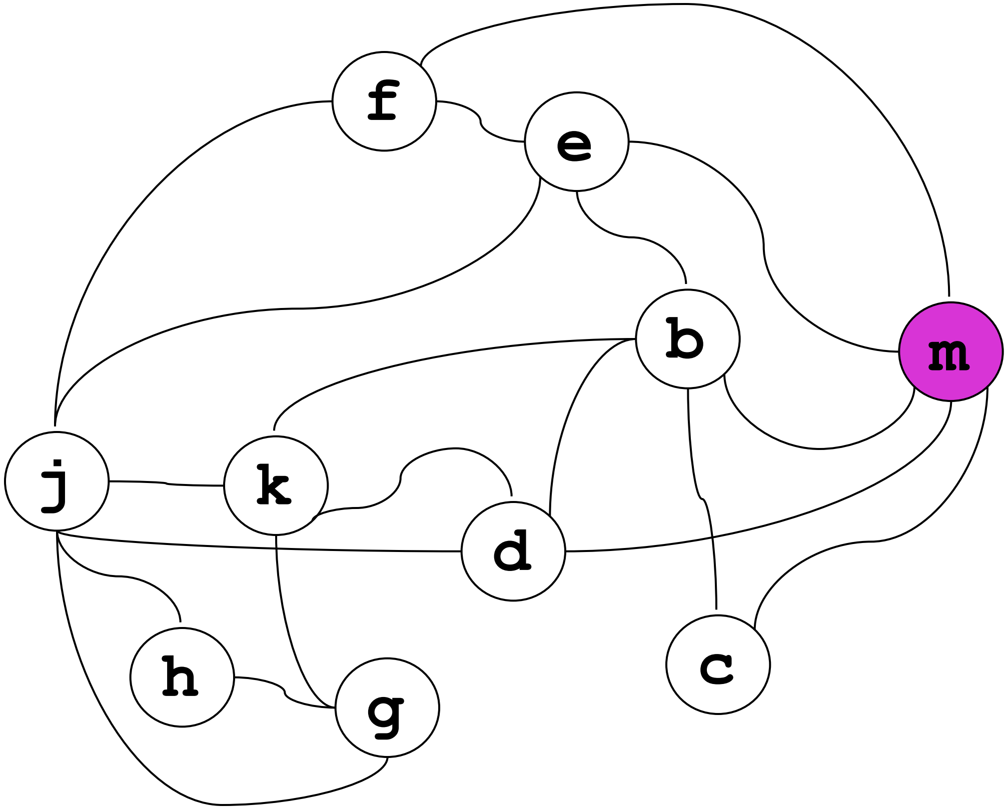
e

f

b

c

m



Select:

Stack:

g

h

k

d

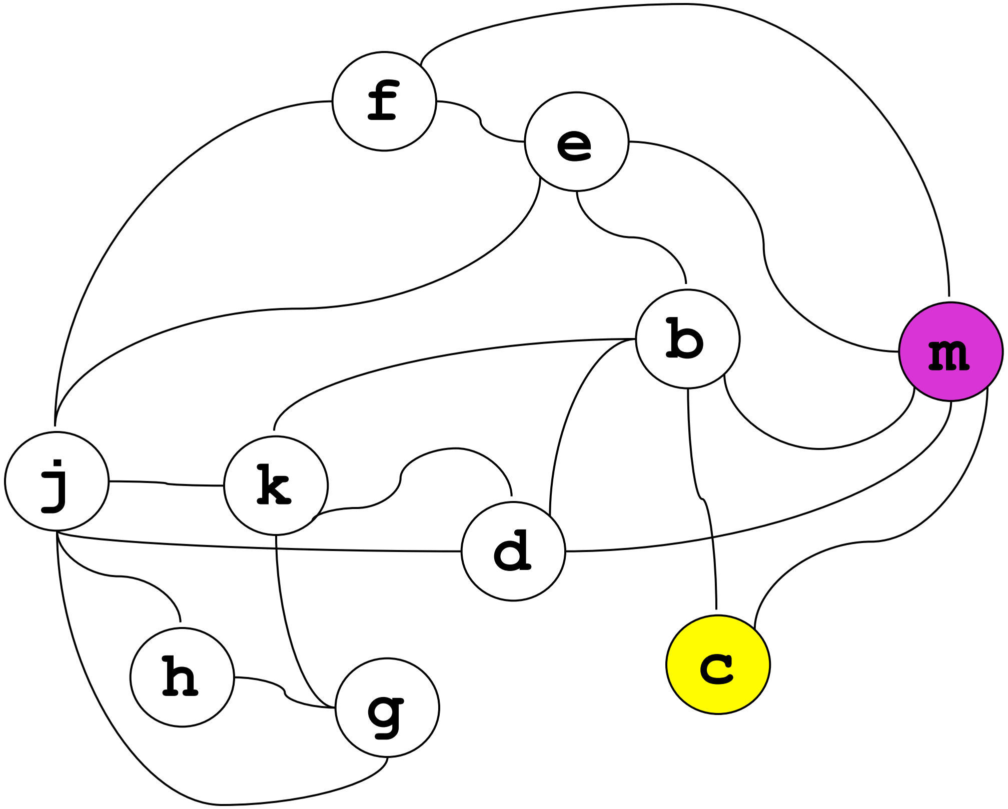
j

e

f

b

c



Select:

Stack:

g

h

k

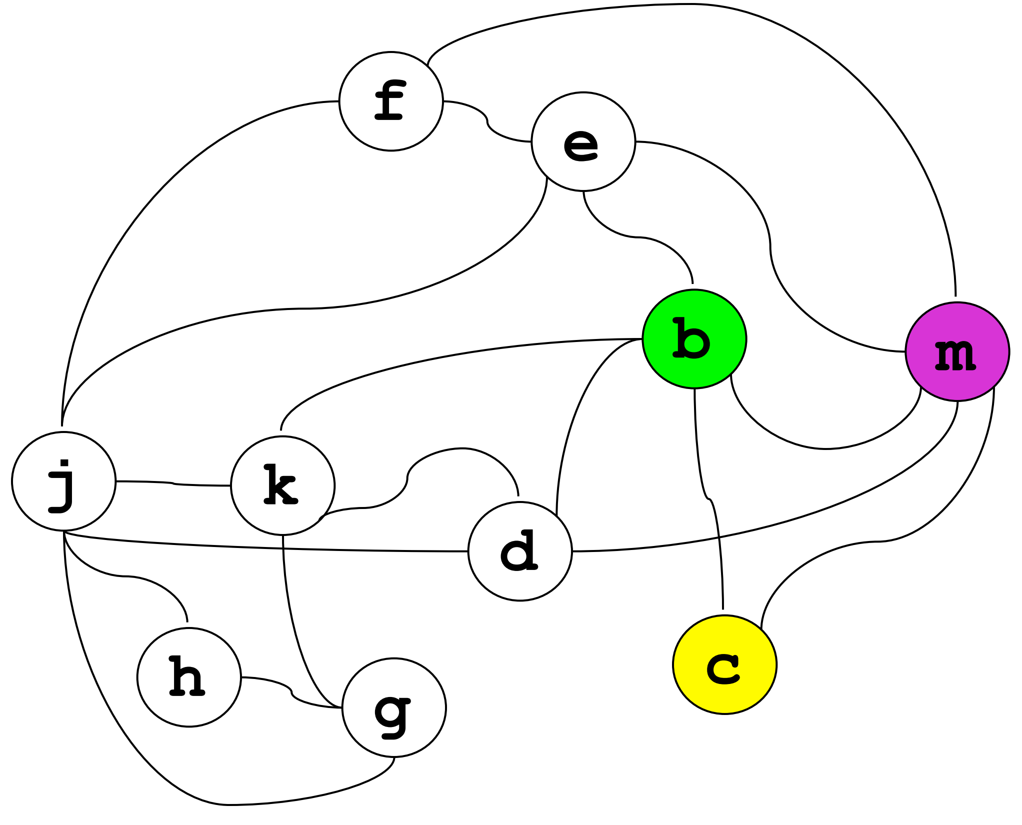
d

j

e

f

b



Select:

Stack:

g

h

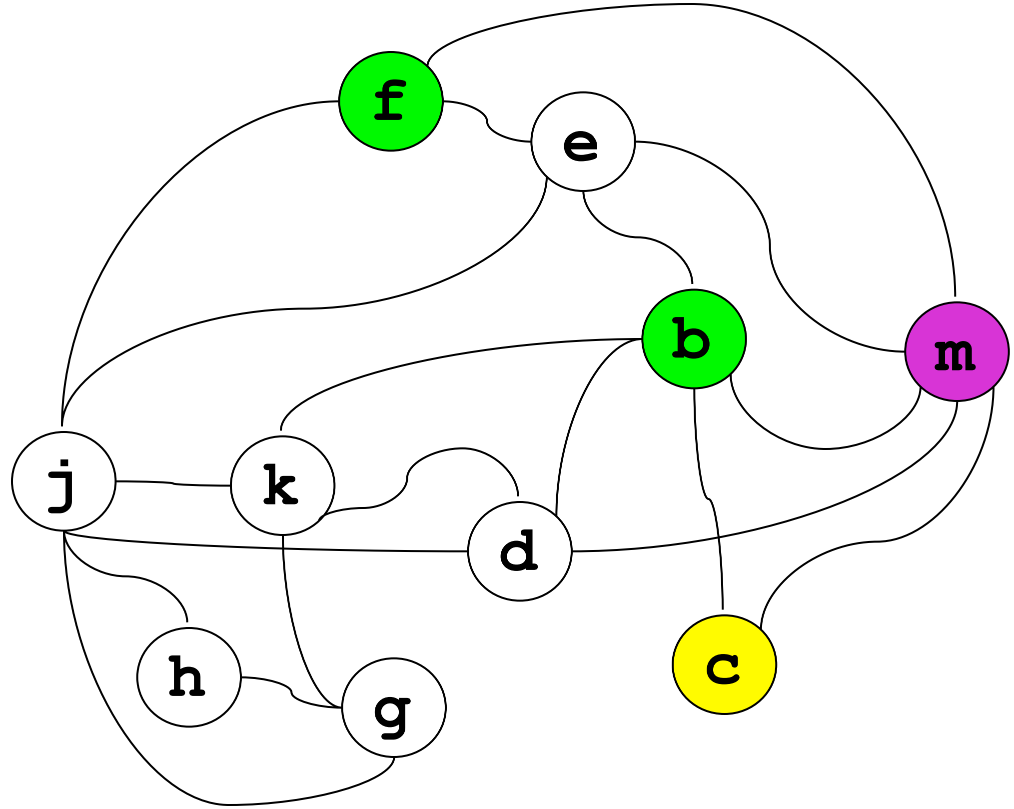
k

d

j

e

f



Select:

Stack:

g

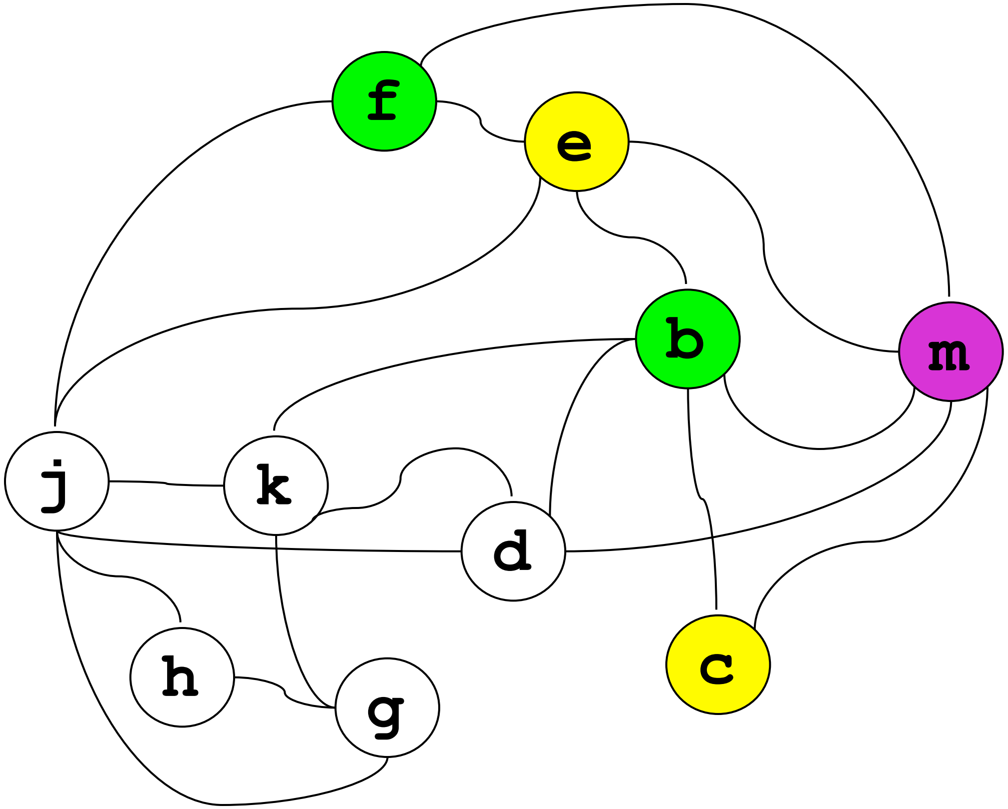
h

k

d

j

e



Select:

Stack:

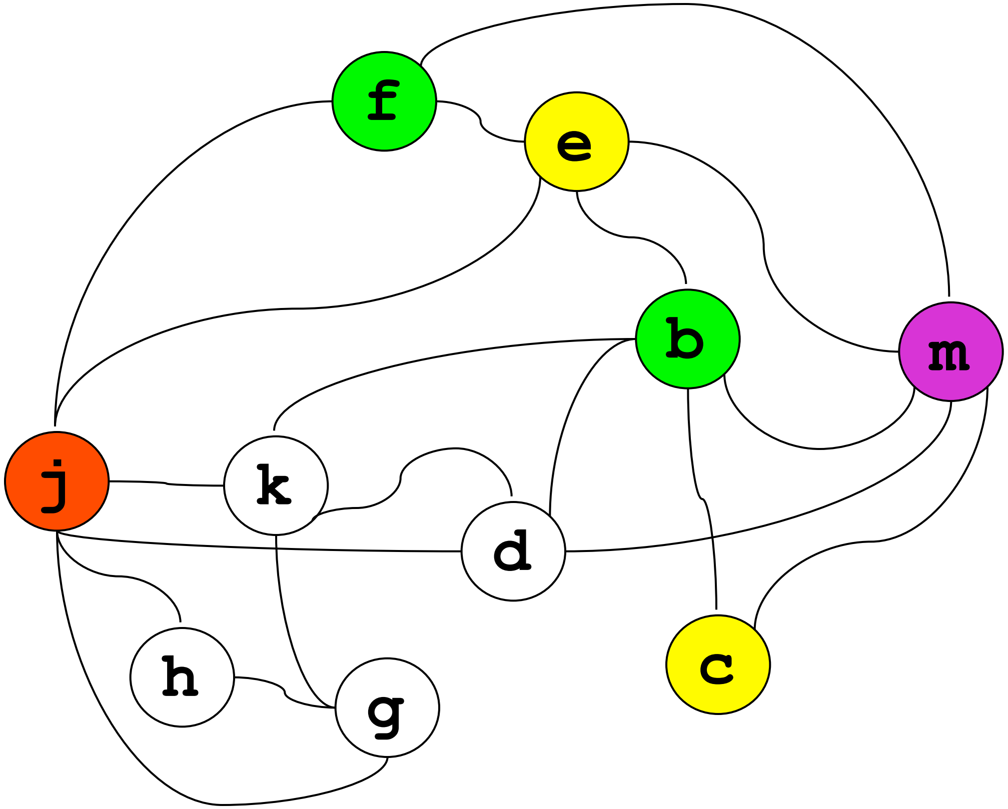
g

h

k

d

j



Select:

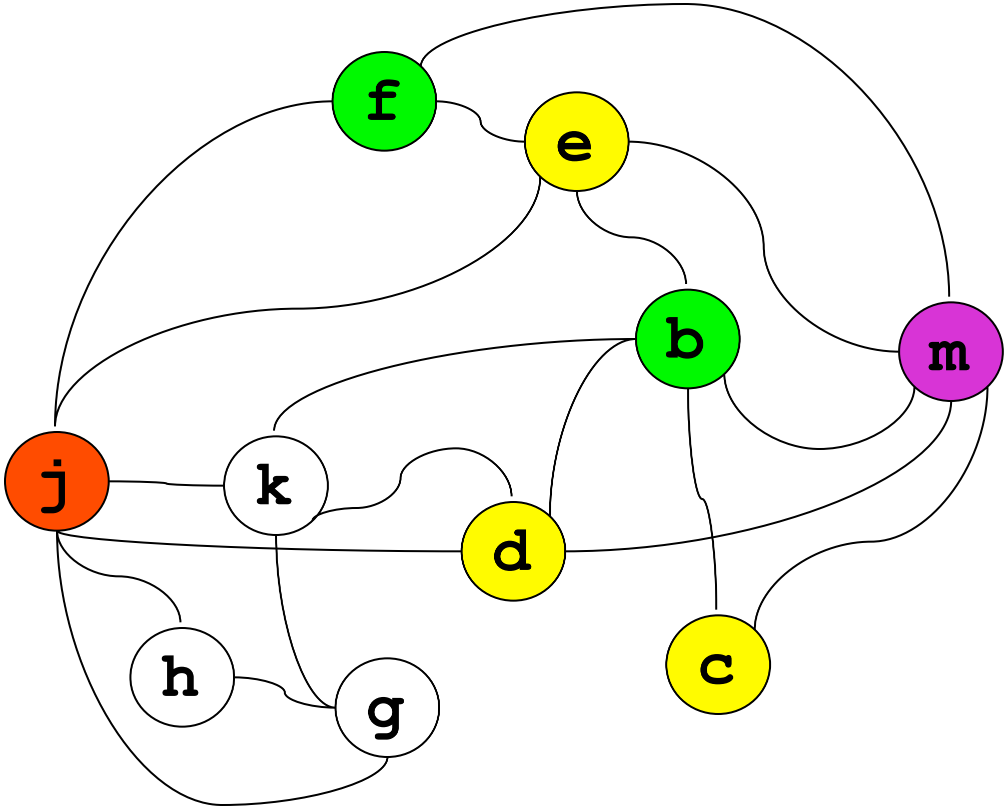
Stack:

g

h

k

d



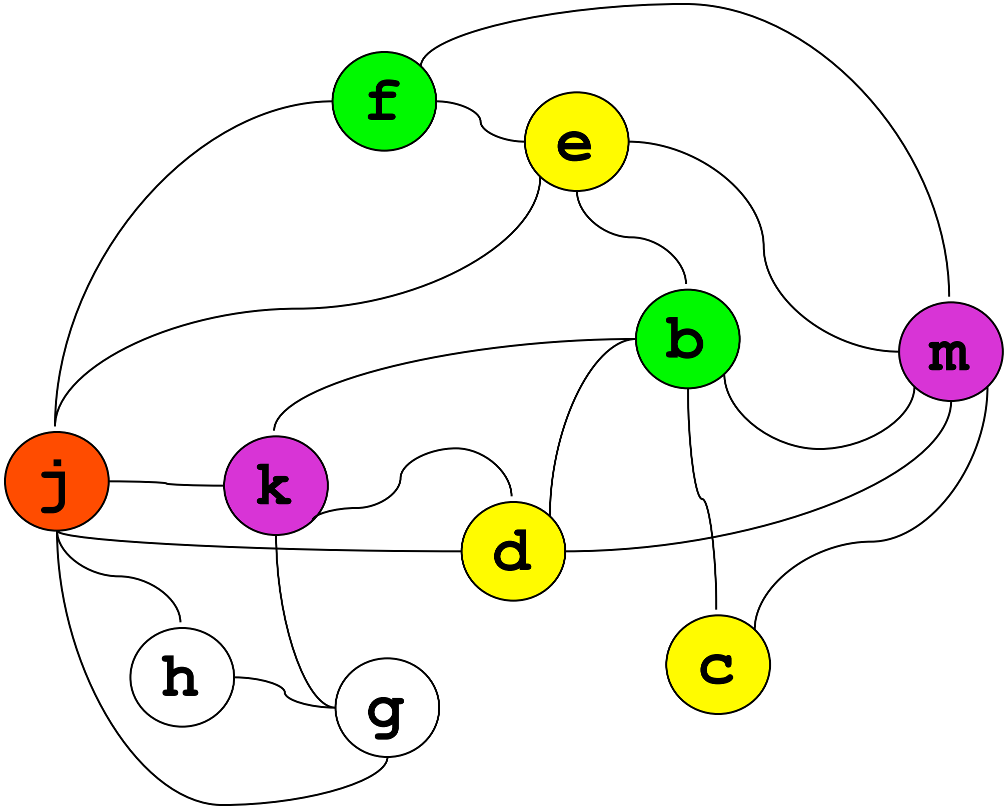
Select:

Stack:

g

h

k

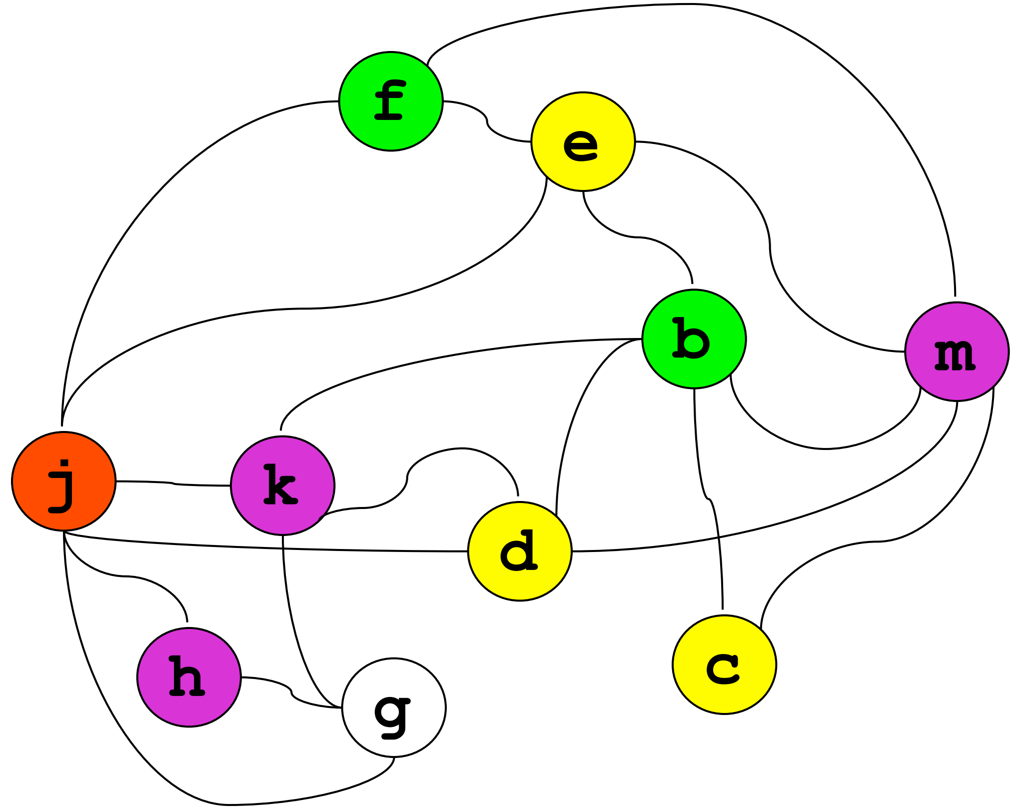


Select:

Stack:

g

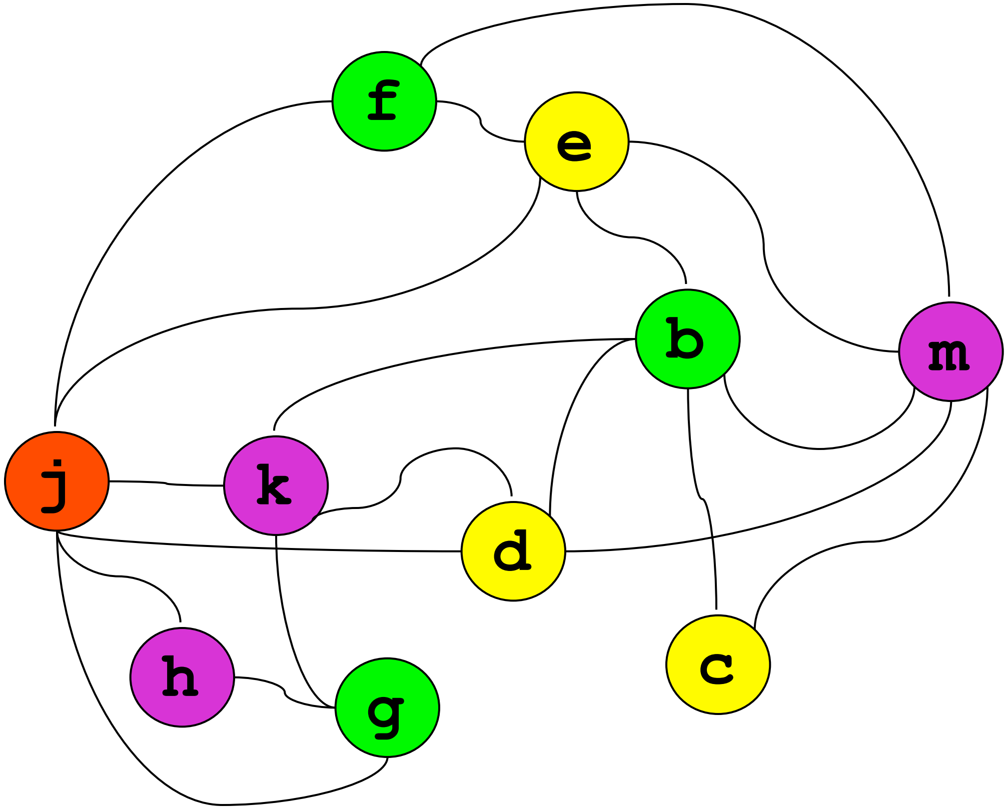
h



Select:

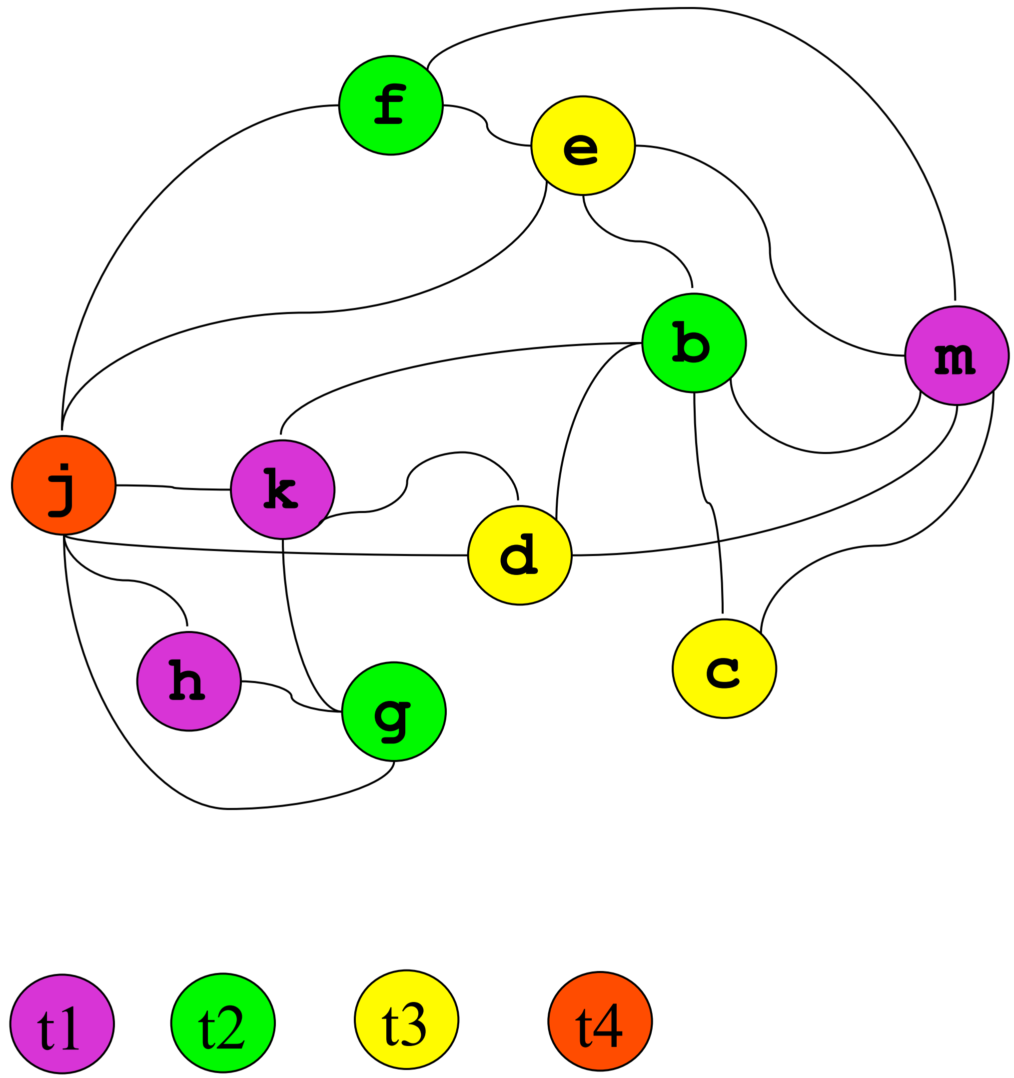
Stack:

g



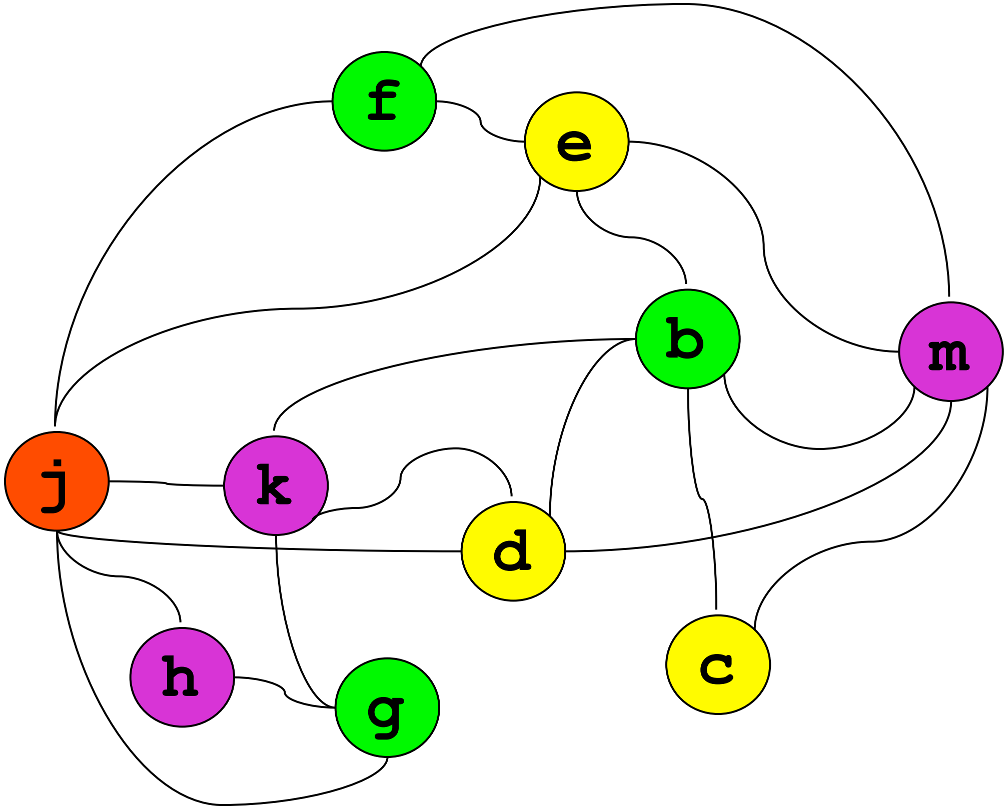
Use coloring to codegen:

```
g := *(j+12)
h := k - 1
f := g * h
e := *(j+8)
m := *(j+16)
b := *(f+0)
c := e + 8
d := c
k := m + 4
j := b
```



Use coloring to codegen:

```
t2 := *(t4+12)
t1 := t1 - 1
t2 := t2 * t1
t3 := *(t4+8)
t1 := *(t4+16)
t2 := *(t2+0)
t3 := t3 + 8
t3 := t3
t1 := t1 + 4
t4 := t2
```



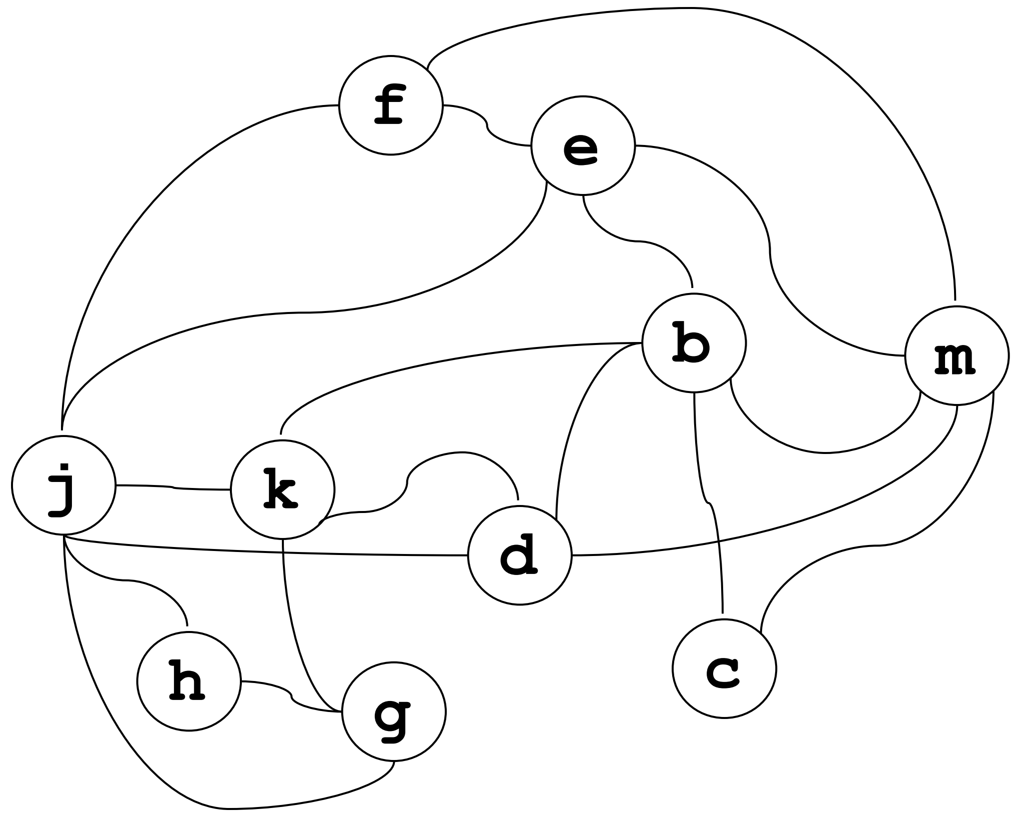
Notice that we can simplify away moves such as this one…

Spilling…

- Suppose all of the nodes in the graph have degree >= k.
- Pick one of the nodes to spill.
  - Picking a high-degree temp will make it more likely that we can color the rest of the graph.
  - Picking a temp that is used infrequently will likely generate better code.
    - e.g., spilling a register used in a loop when we could spill one accessed outside the loop is a bad idea…
- Rewrite the code:
  - after definition of temp, write it into memory.
  - before use of temp, load it into another temp.

Try it with 3 registers…

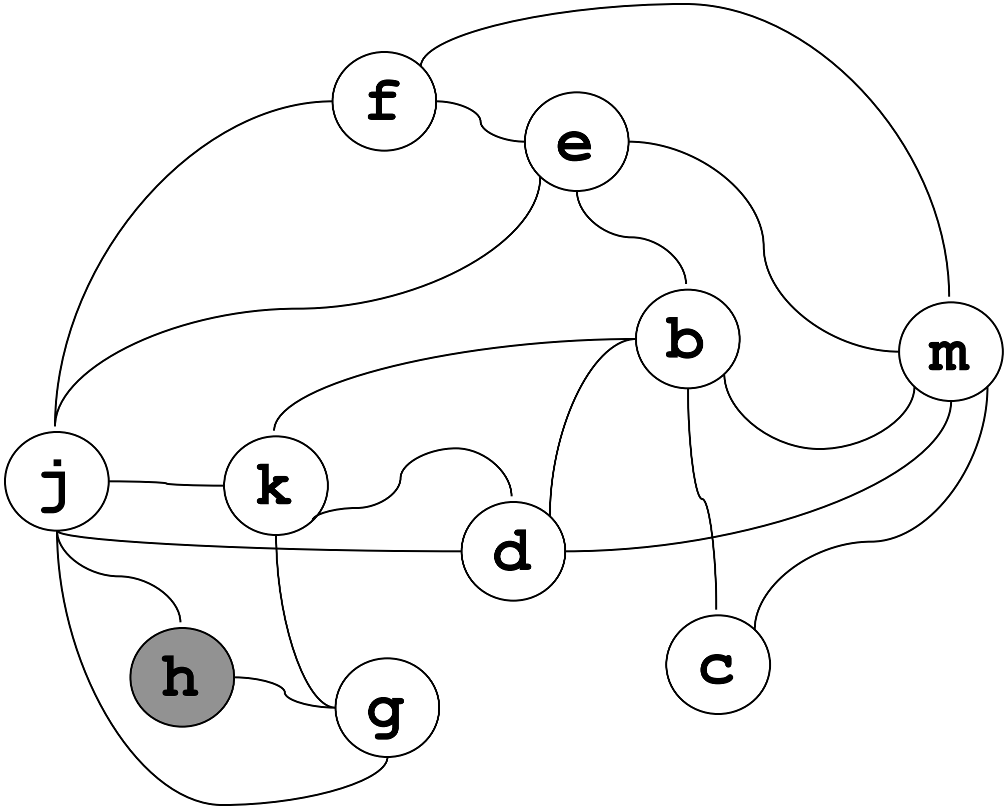
```
{live-in: j, k}
g := *(j+12)
h := k - 1
f := g * h
e := *(j+8)
m := *(j+16)
b := *(f+0)
c := e + 8
d := c
k := m + 4
j := b
{live-out: d,j,k}
```



Simplify:

Stack:

h

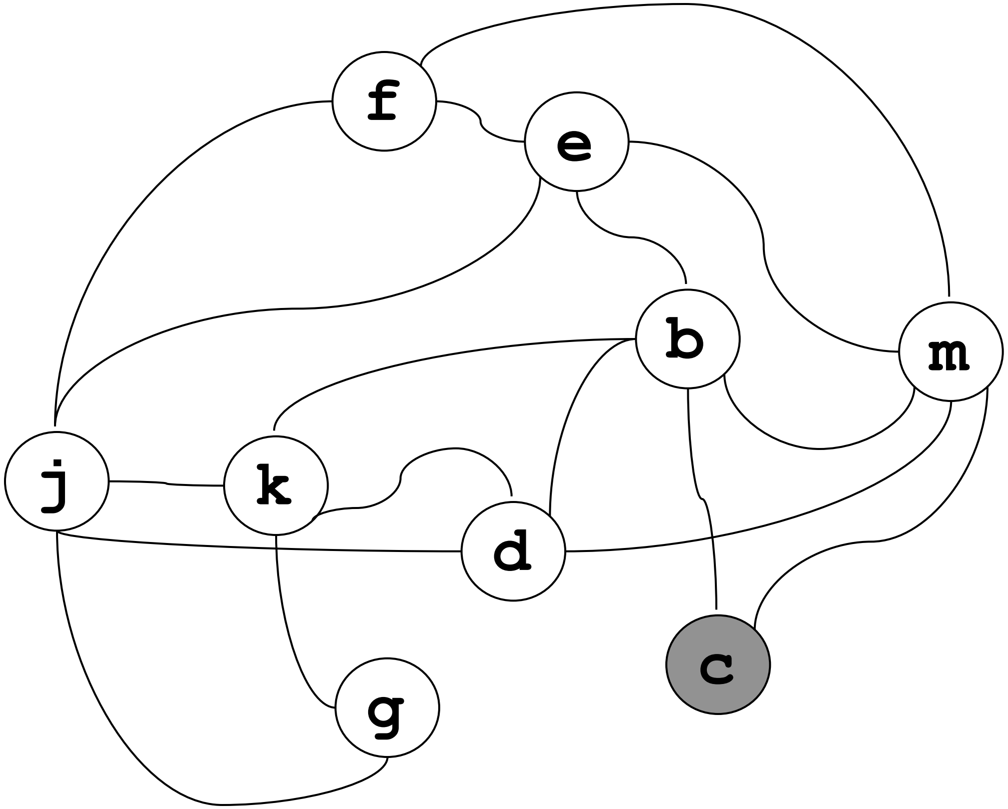


Simplify:

Stack:

h

c



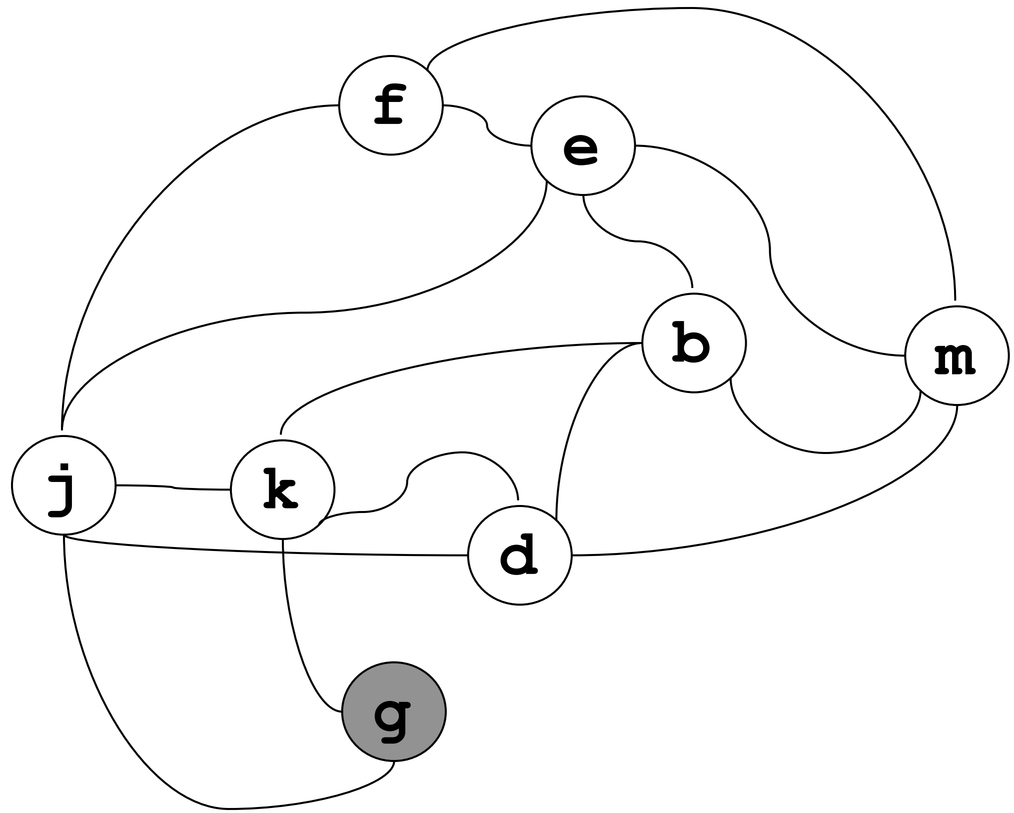
Simplify:

Stack:

h

c

g



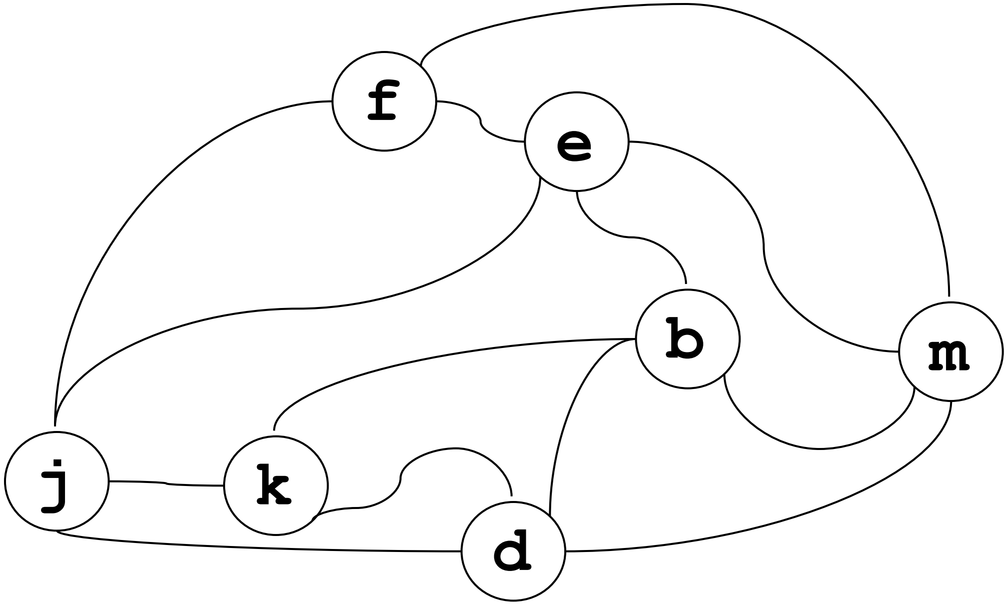
Simplify:

Stack:

h

c

g



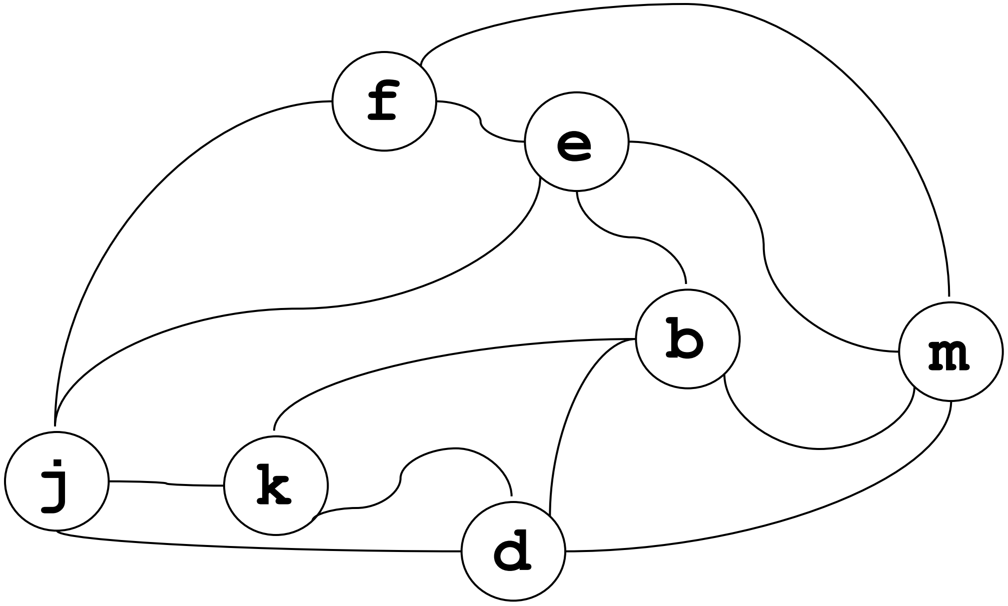
3 Regs

Stack:

h

c

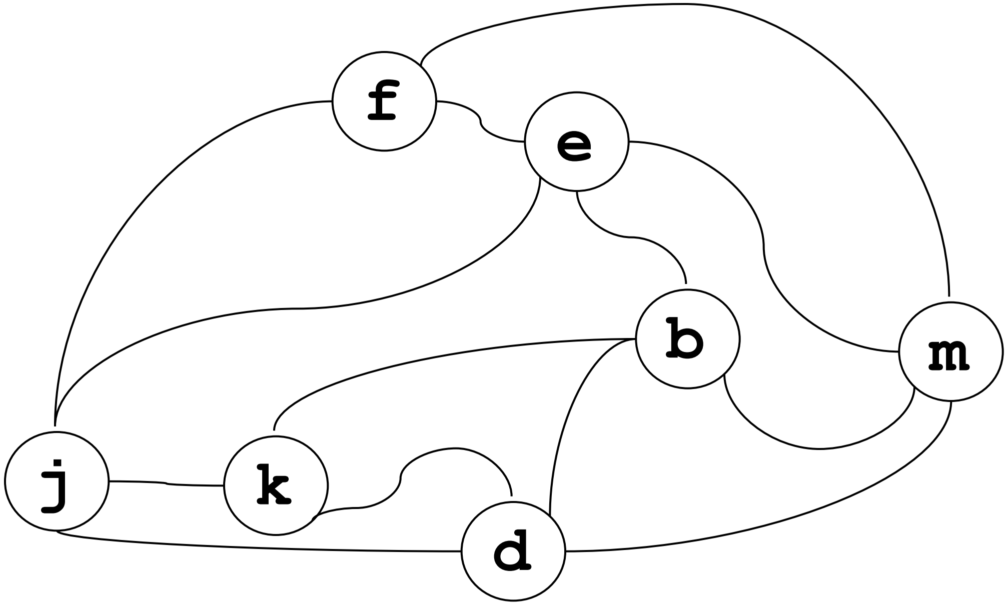
g



We're stuck…

3 Regs

```
{live-in: j, k}
g := *(j+12)
h := k - 1
f := g * h
e := *(j+8)
m := *(j+16)
b := *(f+0)
c := e + 8
d := c
k := m + 4
j := b
{live-out: d,j,k}
```



Don't want to spill j, it's used a lot.
Don't want to spill f or k, they have relatively low degree.
So let's pick m…

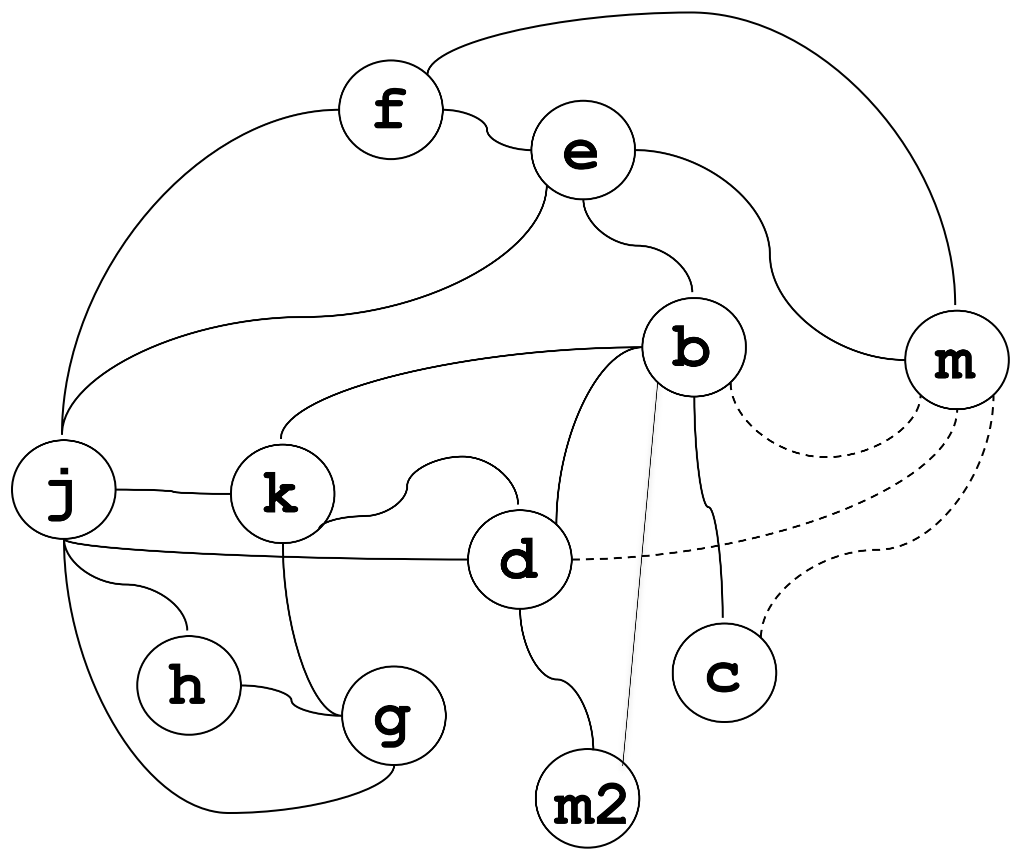
Rewrite:

```
{live-in: j, k}
g := *(j+12)
h := k - 1
f := g * h
e := *(j+8)
m := *(j+16)
*(fp+<moff>) := m
b := *(f+0)
c := e + 8
d := c
m2 := *(fp+<moff>)
k := m2 + 4
j := b
{live-out: d,j,k}
```

Eliminated this chunk of code from m's live range…

New Interference Graph

```
{live-in: j, k}
g := *(j+12)
h := k - 1
f := g * h
e := *(j+8)
m := *(j+16)
*(fp+<moff>) := m
b := *(f+0)
c := e + 8
d := c
m2 := *(fp+<moff>)
k := m2 + 4
j := b
{live-out: d,j,k}
```

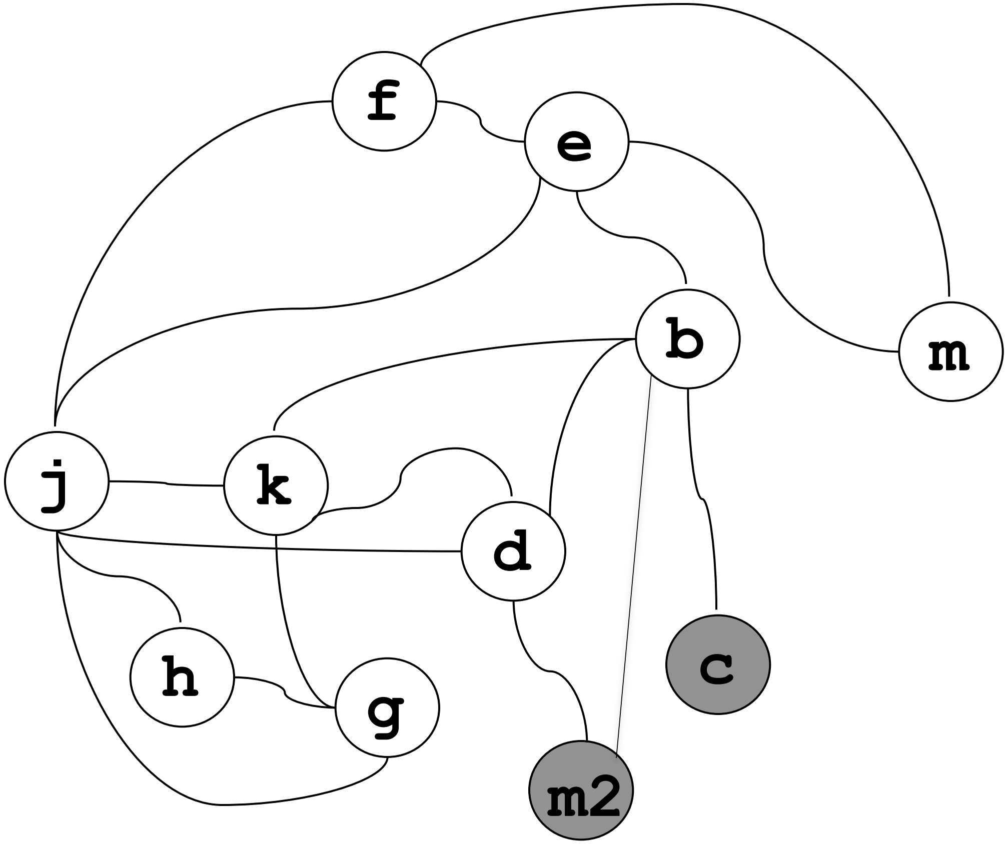


Simplify:

Stack:

m2

c



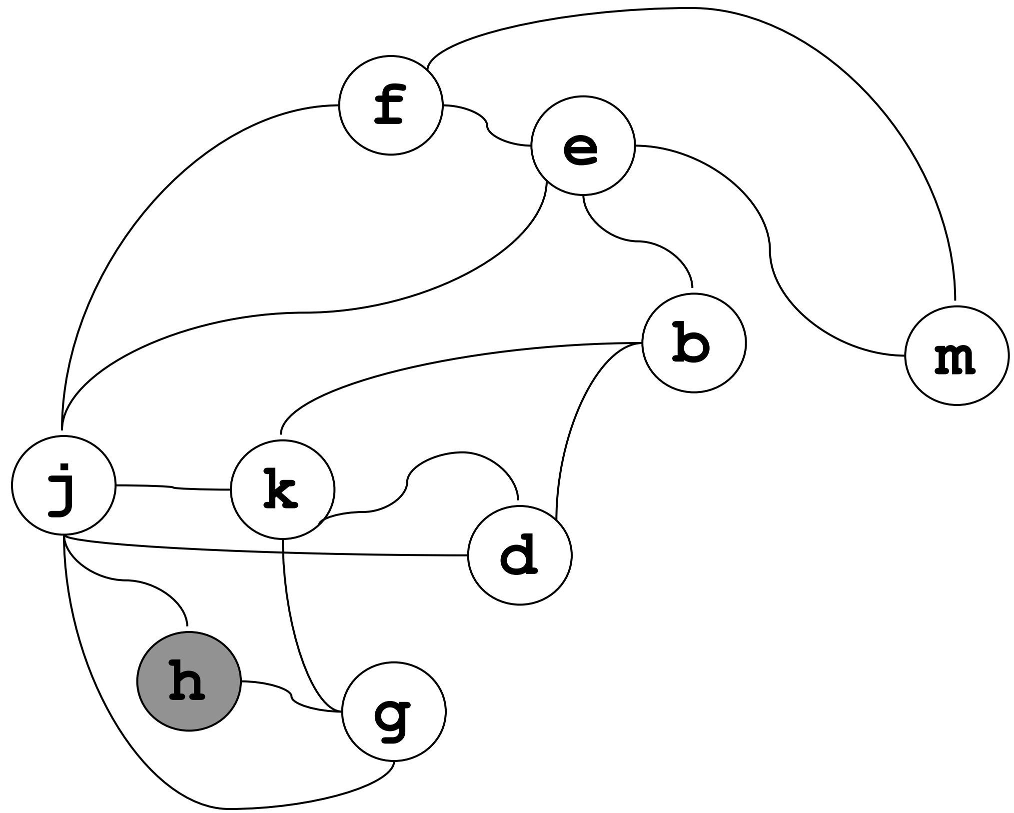
Simplify:

Stack:

m2

c

h



Back to simplification…

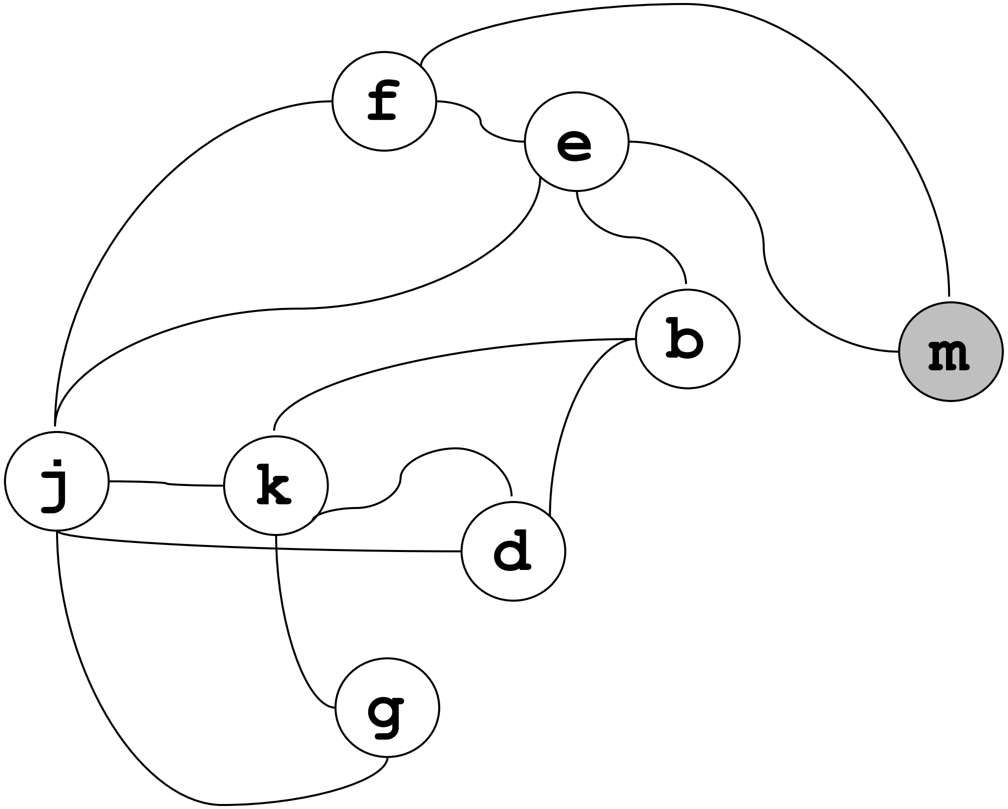
Stack:

m2

c

h

m



Back to simplification…

Stack:

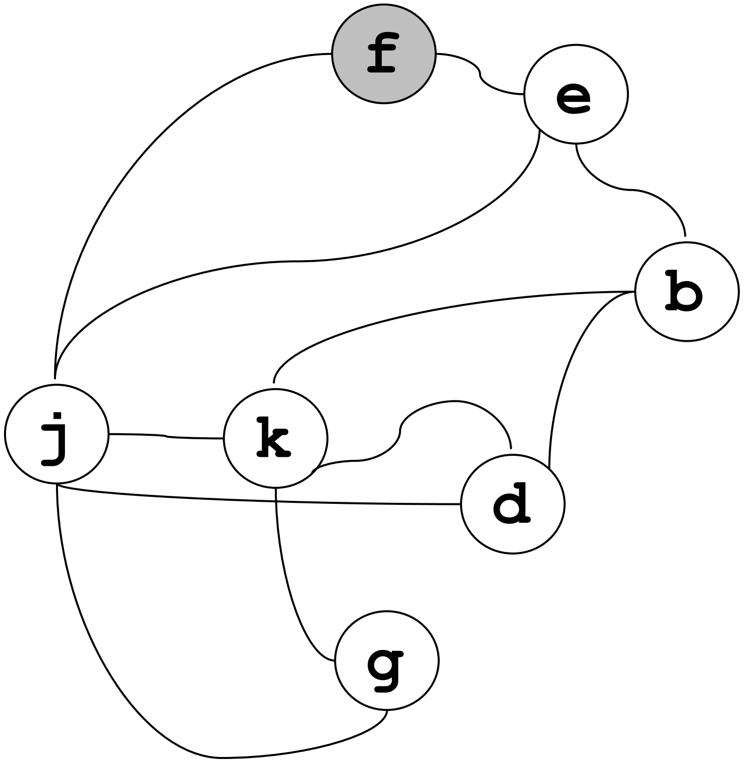
m2

c

h

m

f



Back to simplification…

Stack:

m2

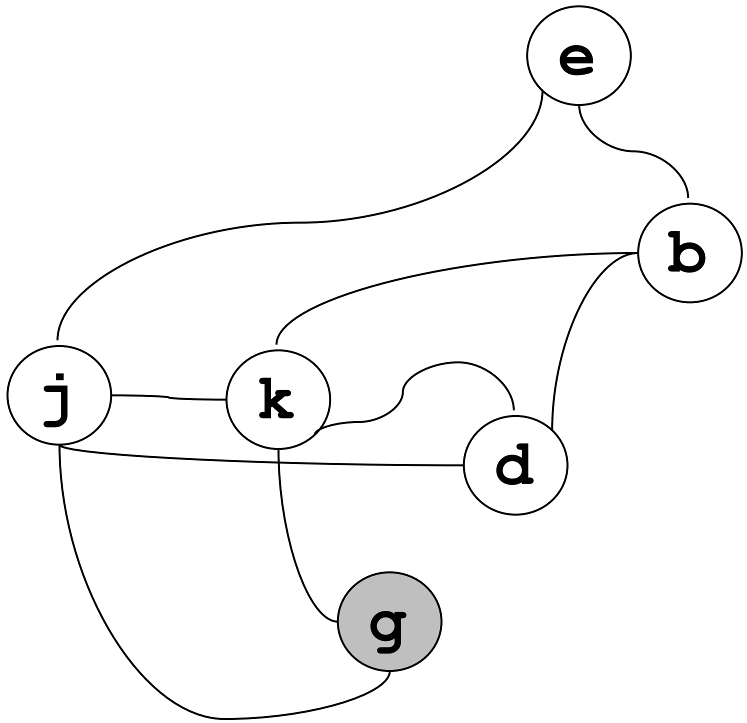
c

h

m

f

g



Back to simplification…

Stack:

m2

c

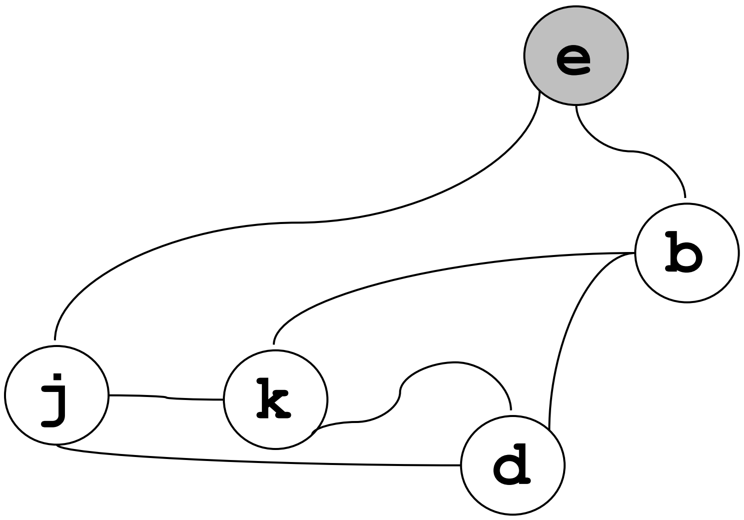
h

m

f

g

e



Back to simplification…

Stack:

m2

c

h

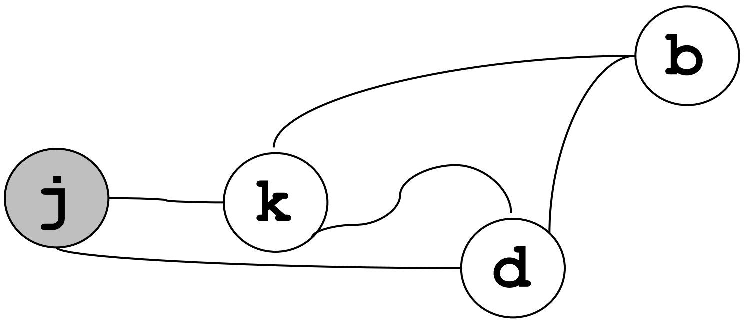
m

f

g

e

j



Back to simplification…

Stack:

m2

c

h

m

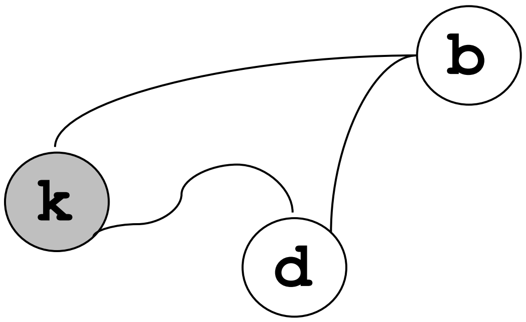
f

g

e

j

k



Back to simplification…

Stack:

m2

c

h

m

f

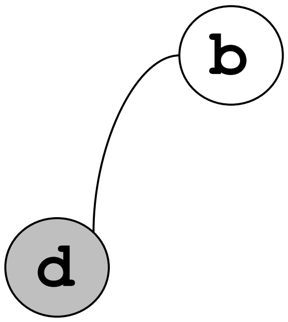
g

e

j

k

d



Back to simplification…

Stack:

m2

c

h

m

f

g

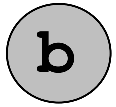
e

j

k

d

b



Then Color

Stack:

m2

c

h

m

f

g

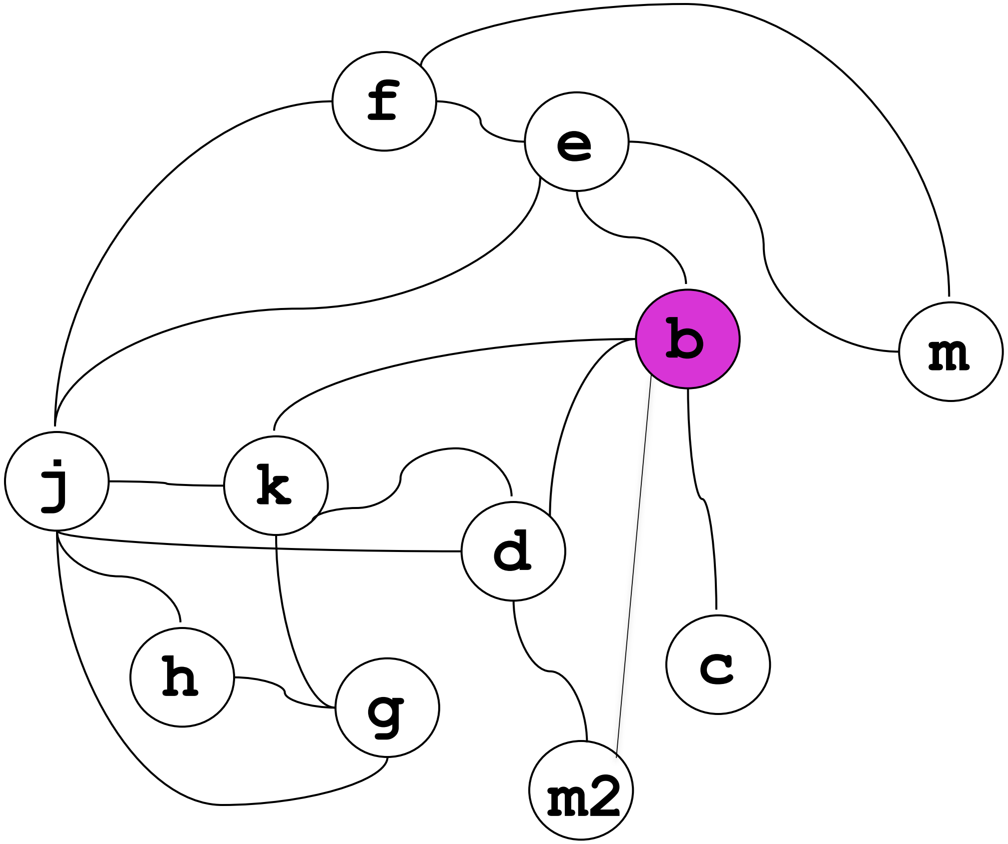
e

j

k

d

b



Then Color

Stack:

m2

c

h

m

f

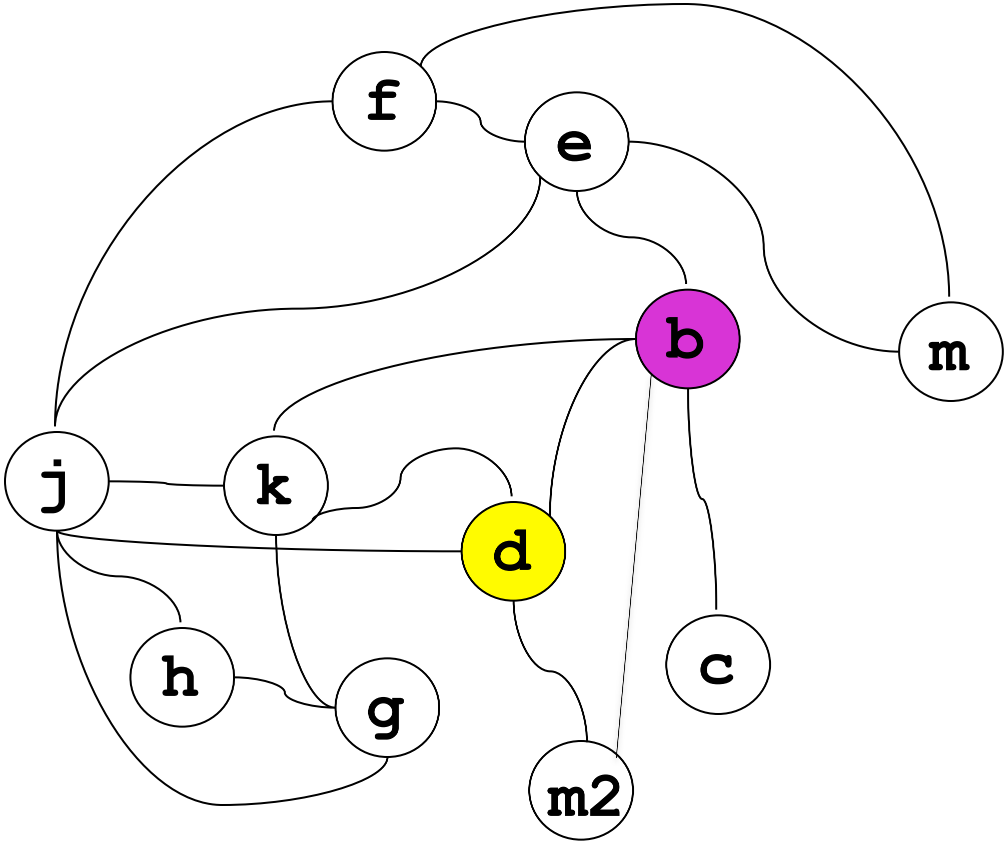
g

e

j

k

d



Then Color

Stack:

m2

c

h

m

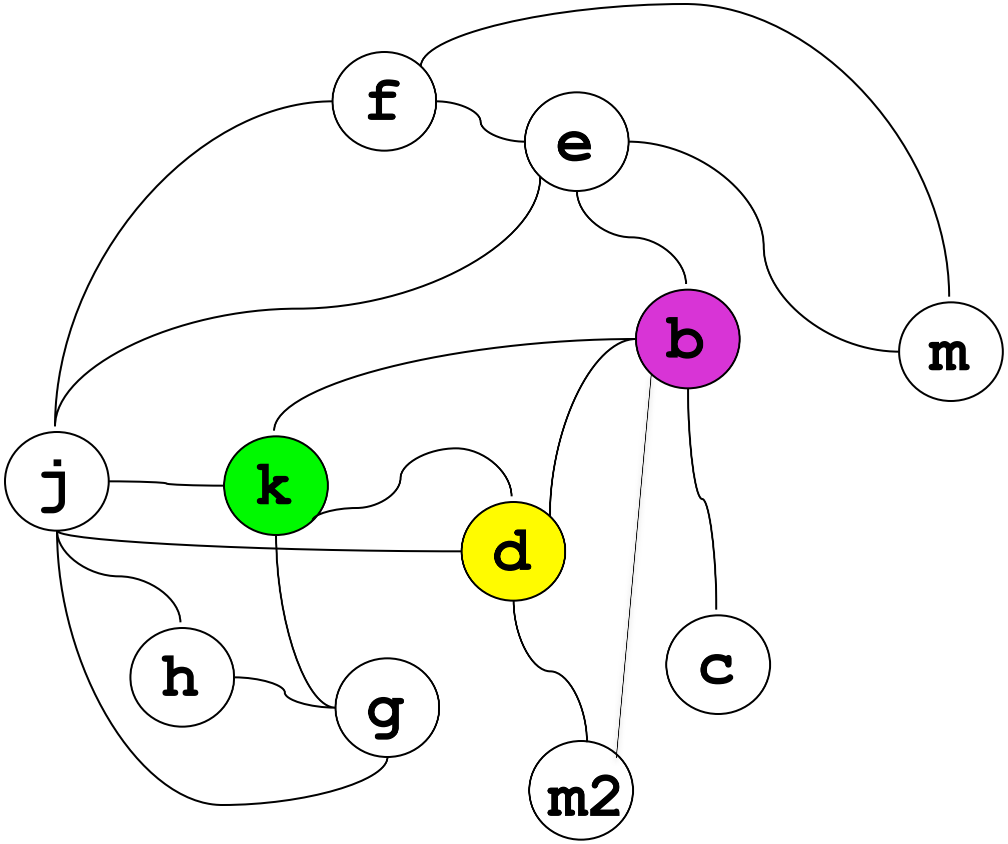
f

g

e

j

k



Then Color

Stack:

m2

c

h

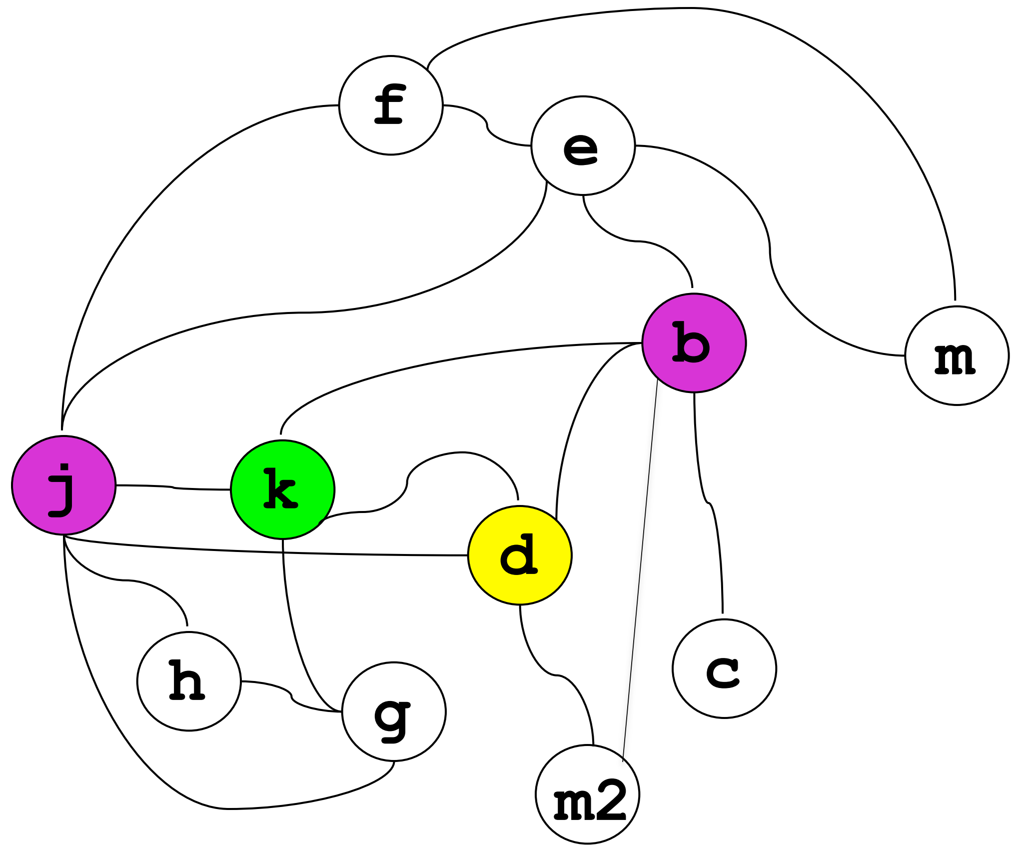
m

f

g

e

j



Then Color

Stack:

m2

c

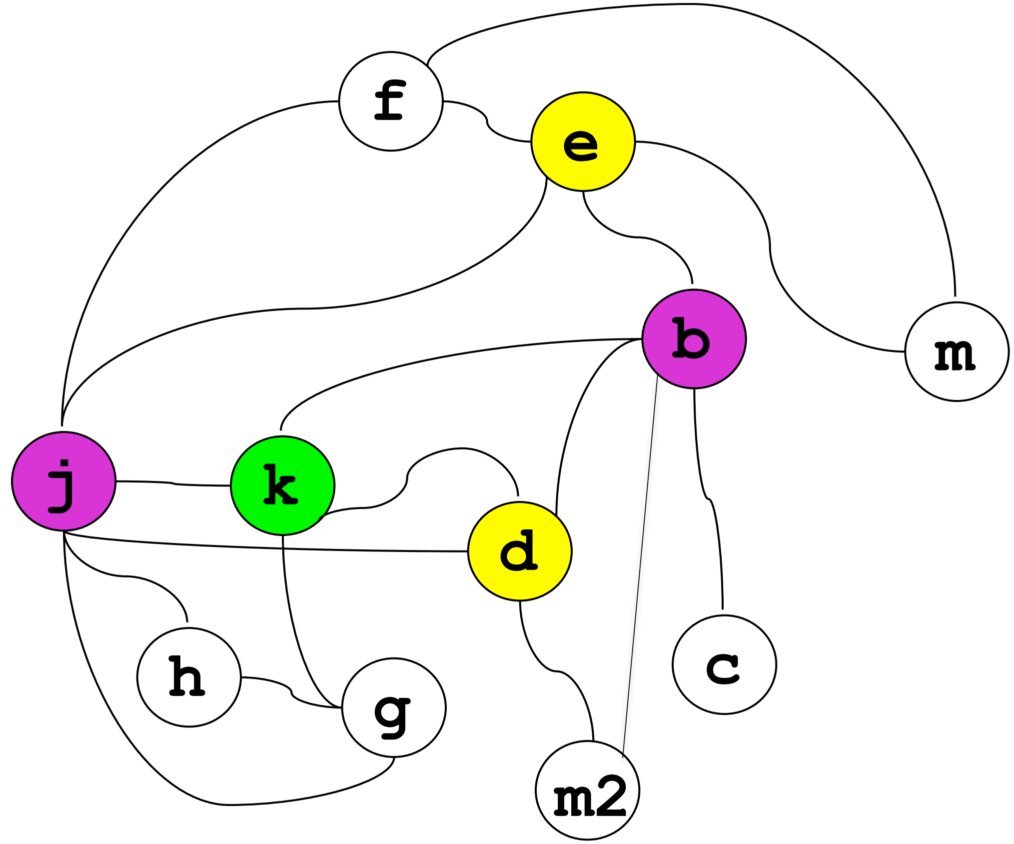
h

m

f

g

e



Then Color

Stack:

m2

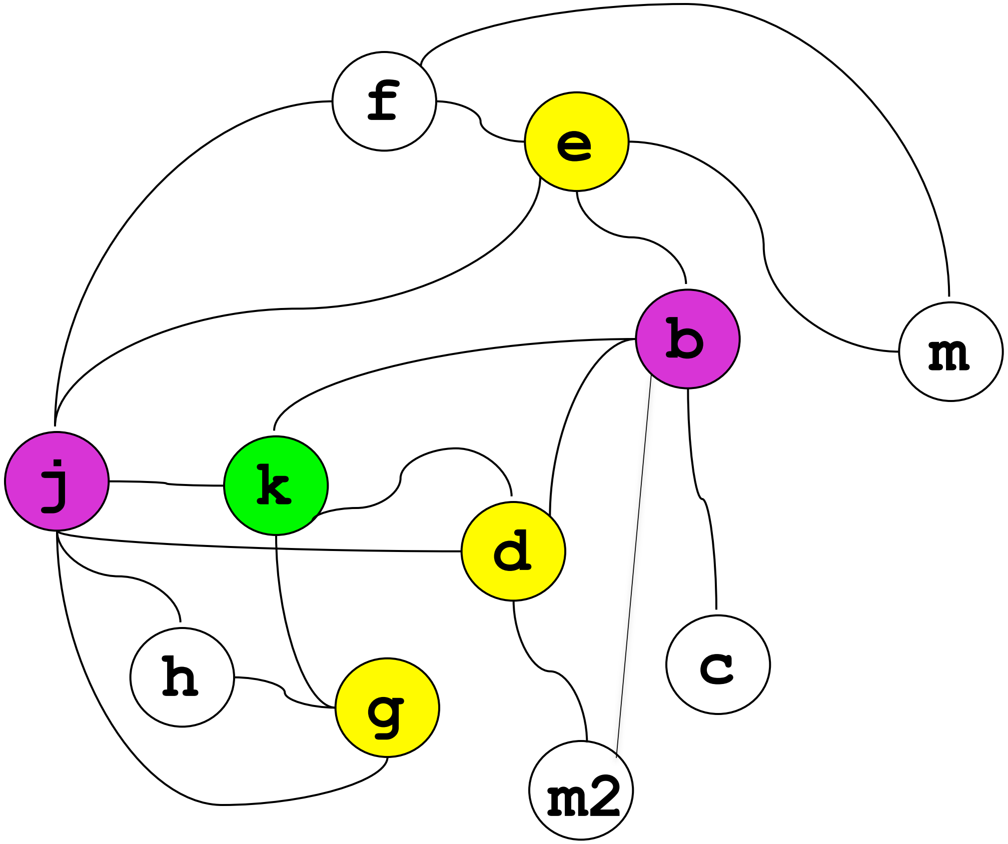
c

h

m

f

g



Then Color

Stack:

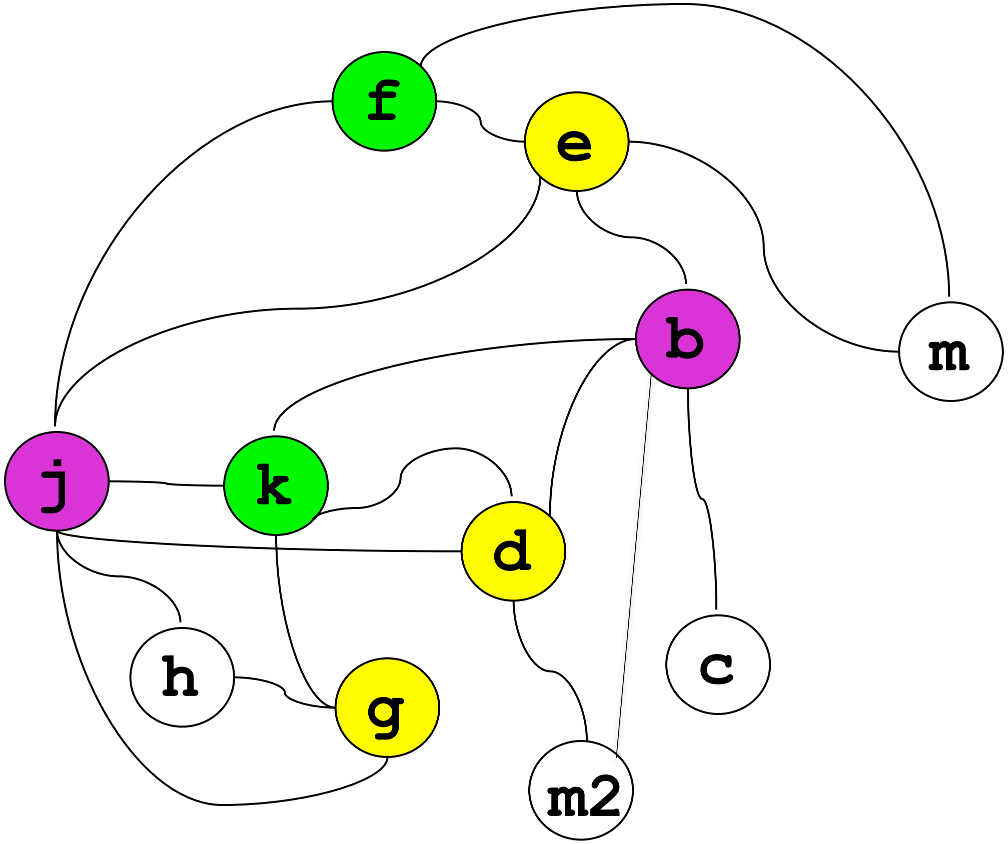
m2

c

h

m

f



Then Color

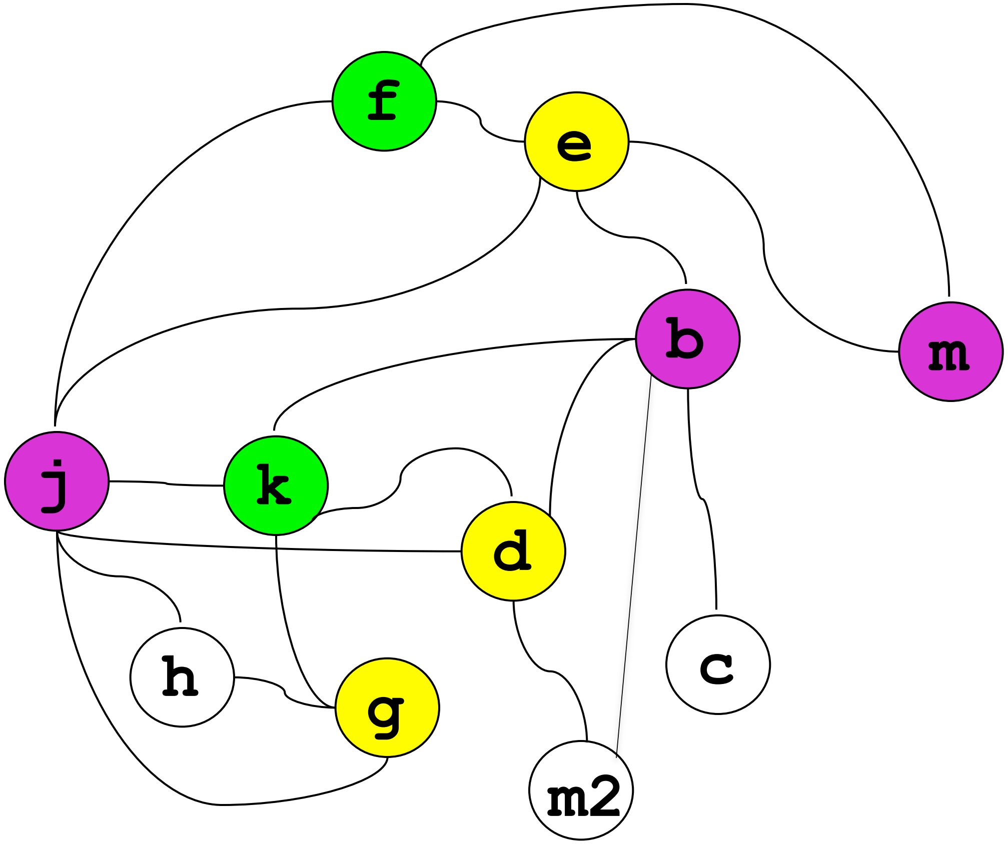
Stack:

m2

c

h

m



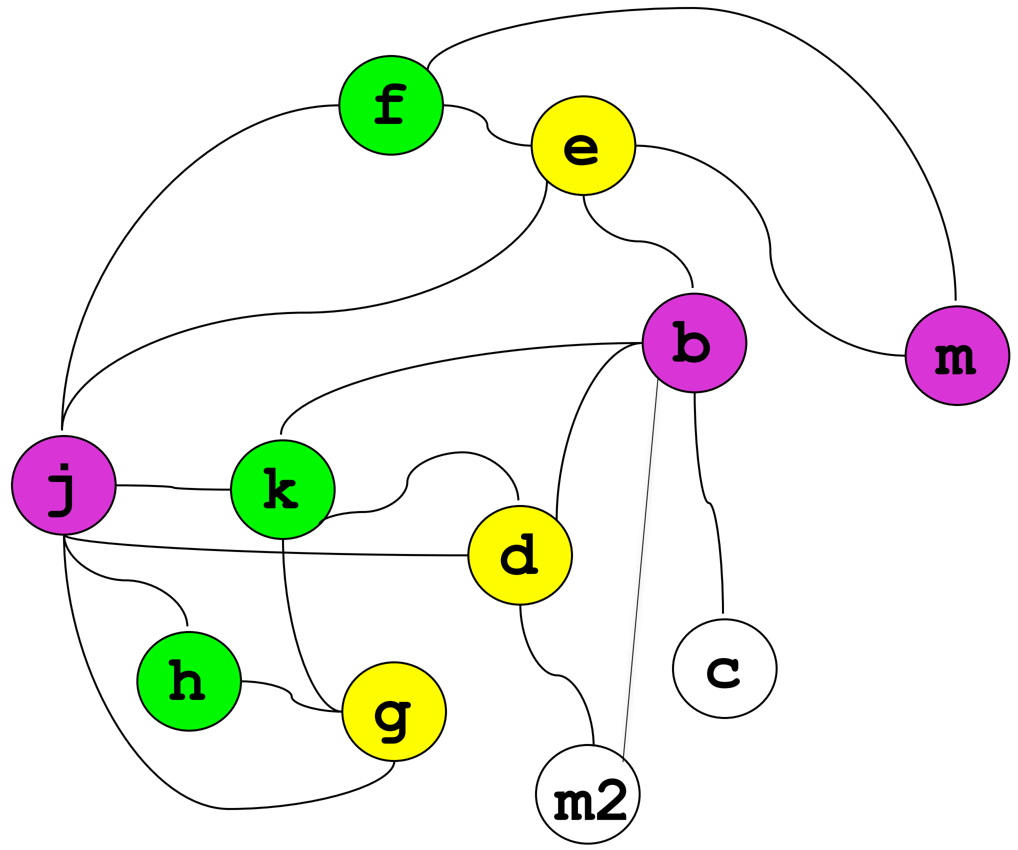
Then Color

Stack:

m2

c

h

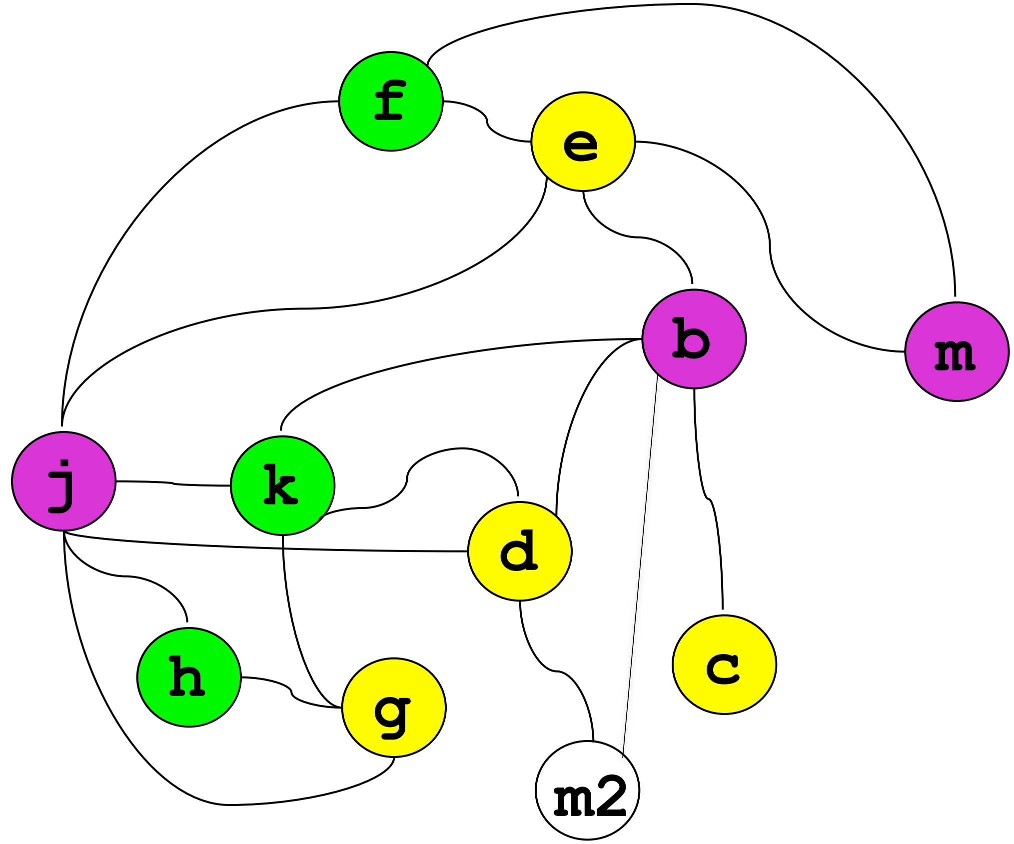


Then Color

Stack:

m2

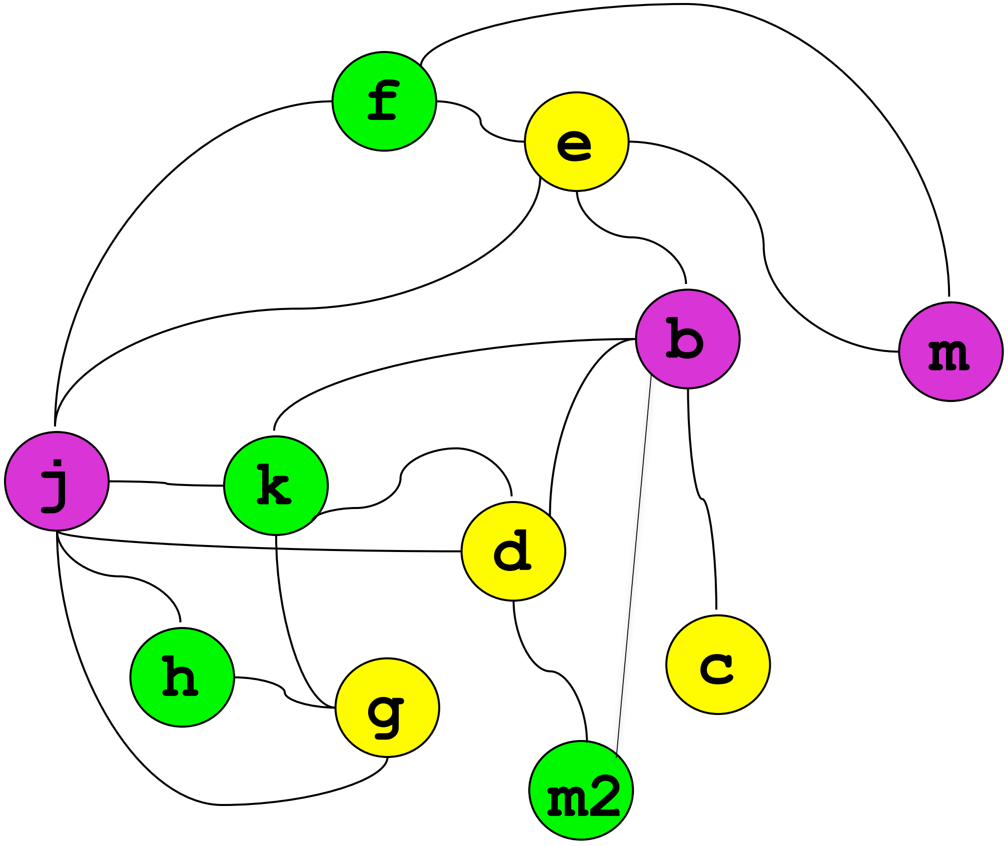
c



Then Color

Stack:

m2



Register Pressure

Some optimizations increase live-ranges:

- Copy propagation
- Common sub-expression elimination
- Loop invariant removal

In turn, that can cause the allocator to spill.

Copy propagation isn't that useful anyway:

- Let register allocator figure out if it can assign the same register to two temps!
- Then the copy can go away.
- And we don't have to worry about register pressure.

Coming Up:

- How to do *coalescing* register allocation.
- An optimistic spilling strategy.
- Some real-world issues:
  - caller/callee-saves registers
  - fixed resources (e.g., mflo, mfhi)
  - allocation of stack slots