Compiler Construction

Naive Register Allocation

Goals of register Allocation

How to assign variables to finitely many registers?

What to do when it can't be done?

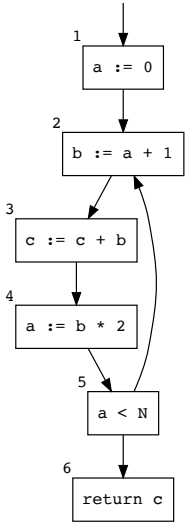
How to efficienty do so?

Goals of register Allocation

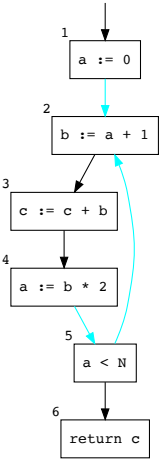
How to assign variables to finitely many registers?

Example: flowgraph

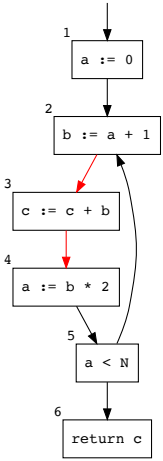
```
     a := 0
L1:  b := a + 1
     c := c + b
     a := b * 2
     if a < N goto L1
     return c
```



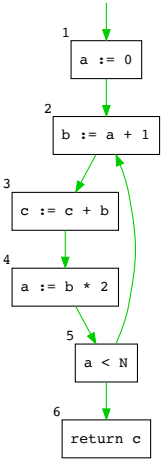
Example: liveness analysis



Liveness for a



Liveness for b



Liveness for c

Interference graph

Observation

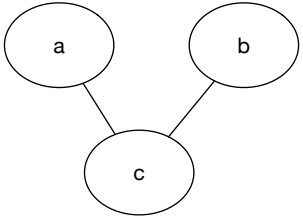
Two temporaries interfere if at some point in the program they cannot both occupy the same register.

An **interference graph (IG)** is an undirected graph where:

- nodes are temporaries
- there is an edge between two nodes if their live ranges overlap

Interference graph representations

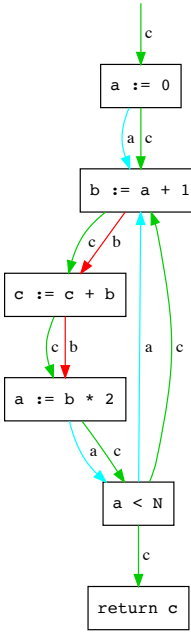
- Graph representation:



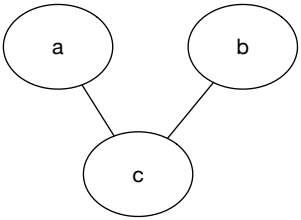
- Matrix representation:

	a	b	c
a			x
b			x
c	x	x	

How to compute interference graph?

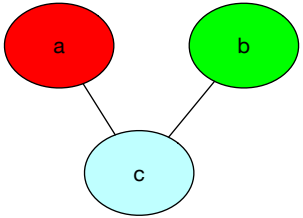


⇒

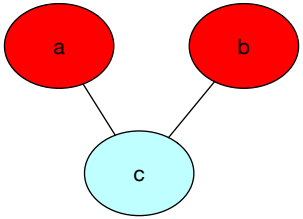


Relation between IG & register allocation

- 3 registers available R_1, R_2, R_3



- 2 registers available R_1, R_3



Register allocation & graph coloring

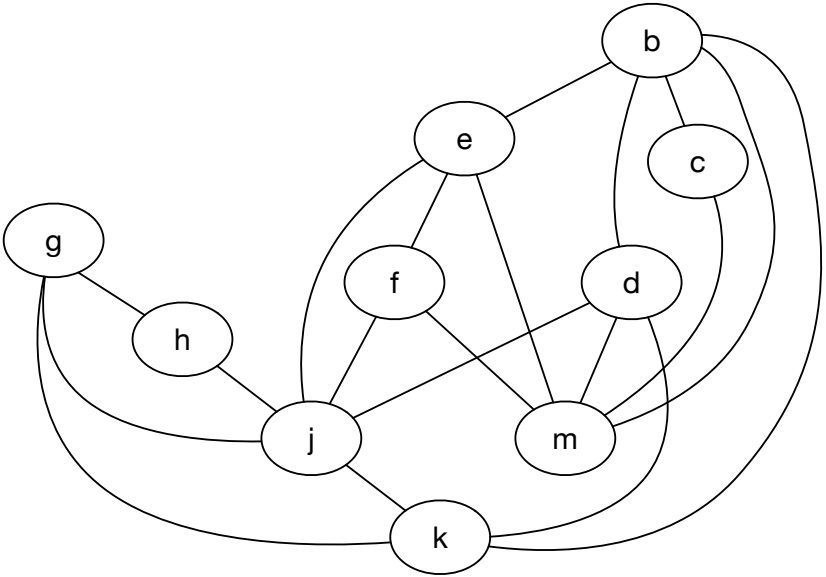
Register allocation can be reduced to graph coloring.

A map can always be colored with 4 colors...but for graph coloring, there is no reason for!

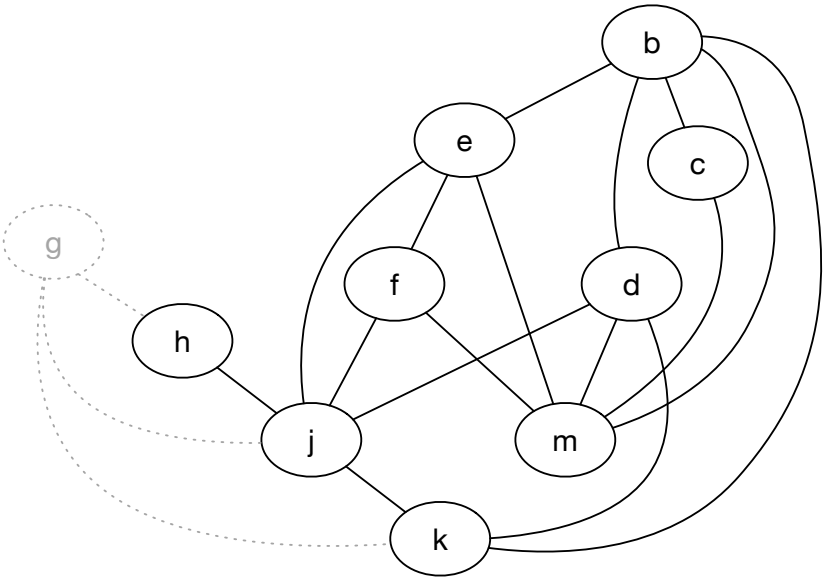
Full example

```
# live in: k j
    g := [j + 12]
    h := k - 1
    f := g * h
    e := [j + 8]
    m := [j + 16]
    b := [f]
    c := e + 8
    d := c
    k := m + 4
    j := b
# live out: d k j
```

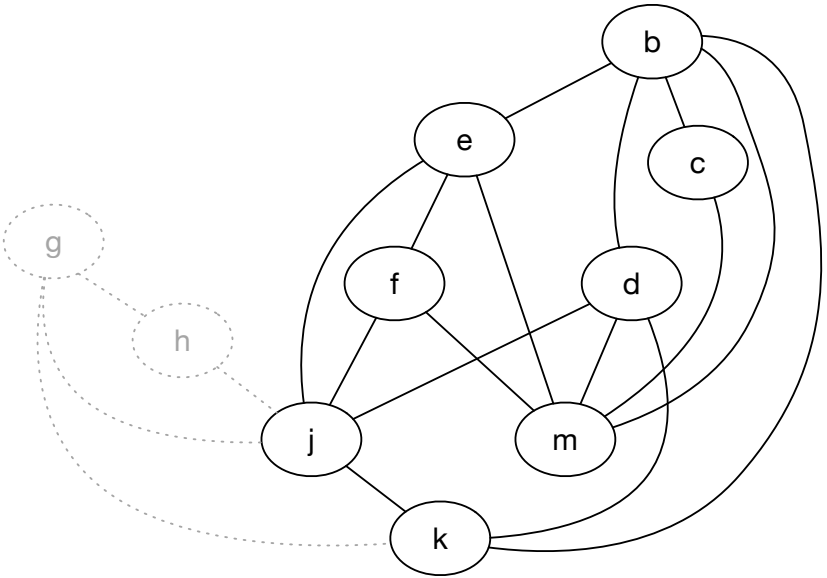
Full example



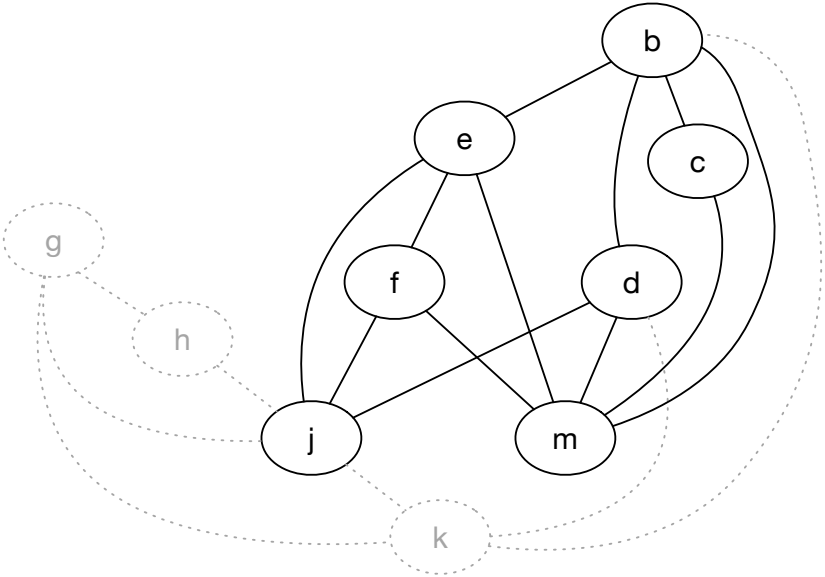
Full example



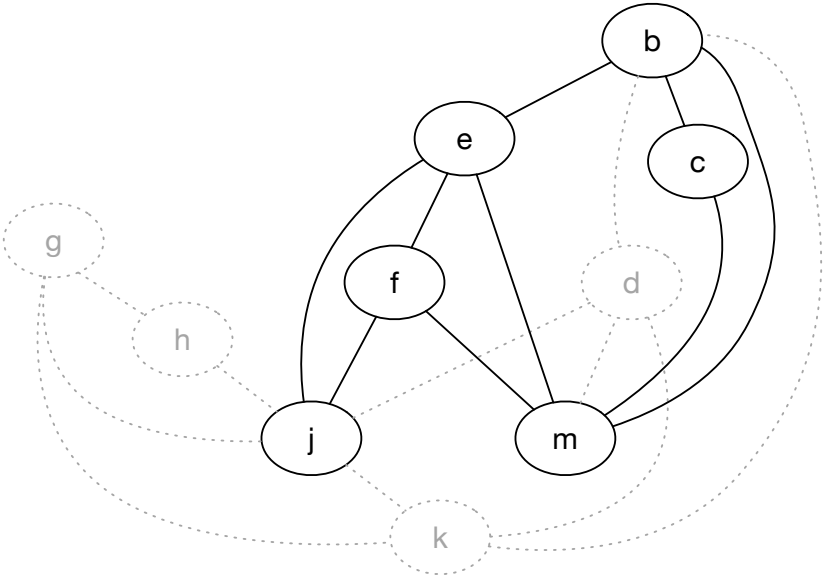
Full example



Full example

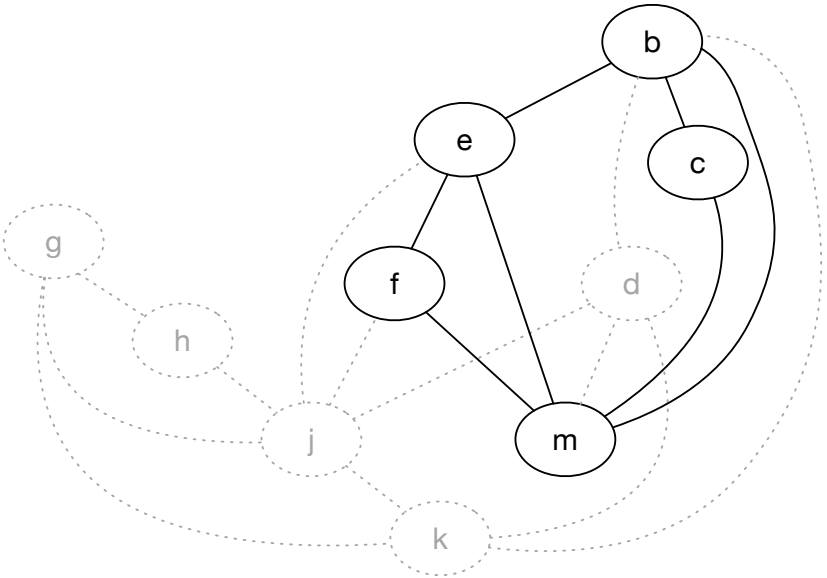


Full example



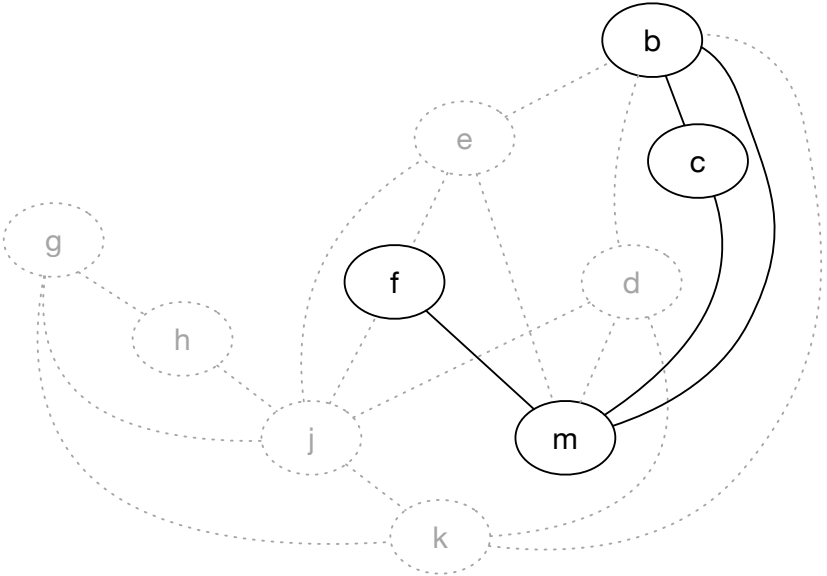
Remove d
Remove k
Remove h
Remove g

Full example



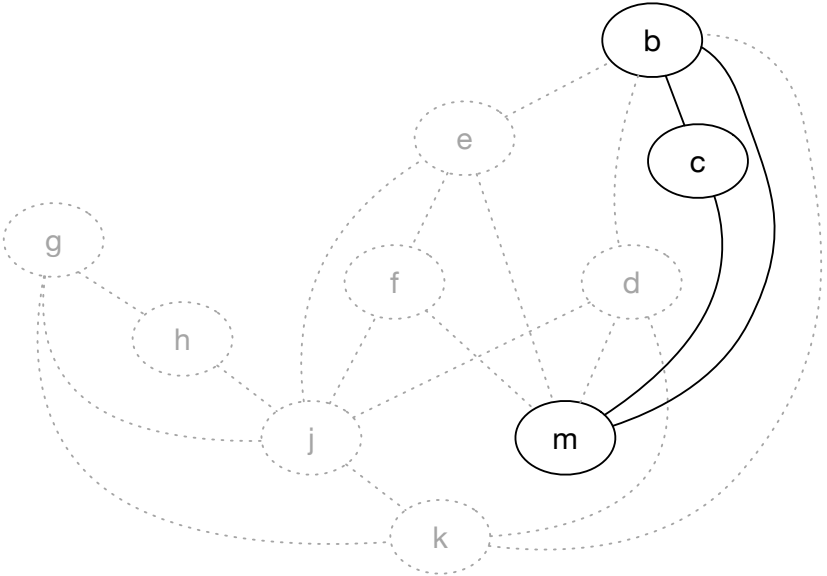
Remove j
Remove d
Remove k
Remove h
Remove g

Full example



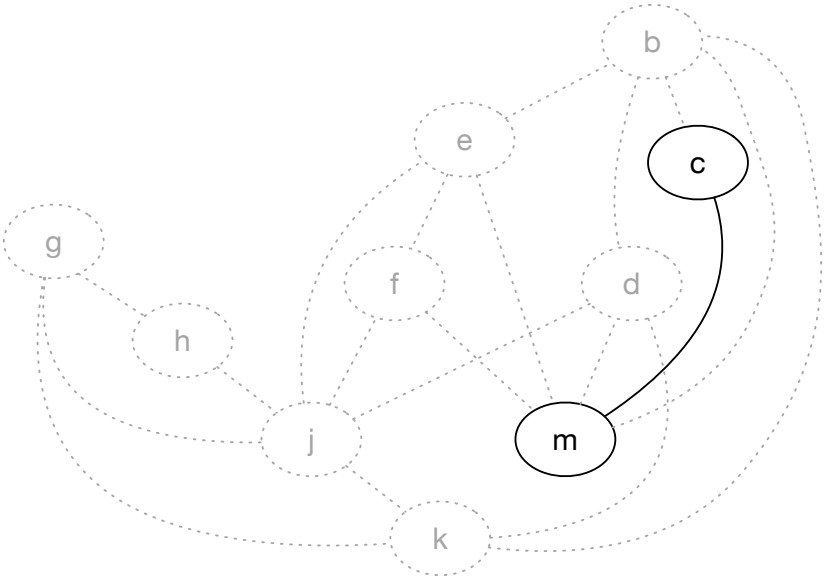
Remove e
Remove j
Remove d
Remove k
Remove h
Remove g

Full example



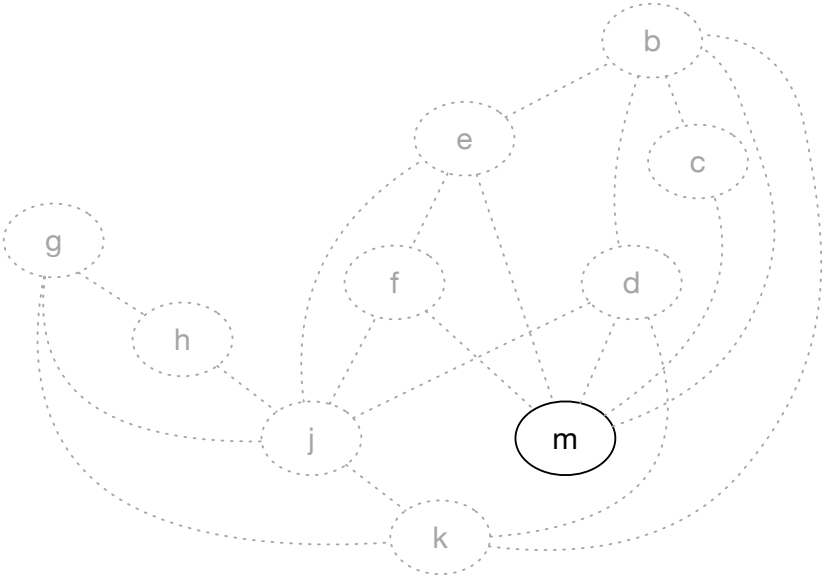
Remove f
Remove e
Remove j
Remove d
Remove k
Remove h
Remove g

Full example



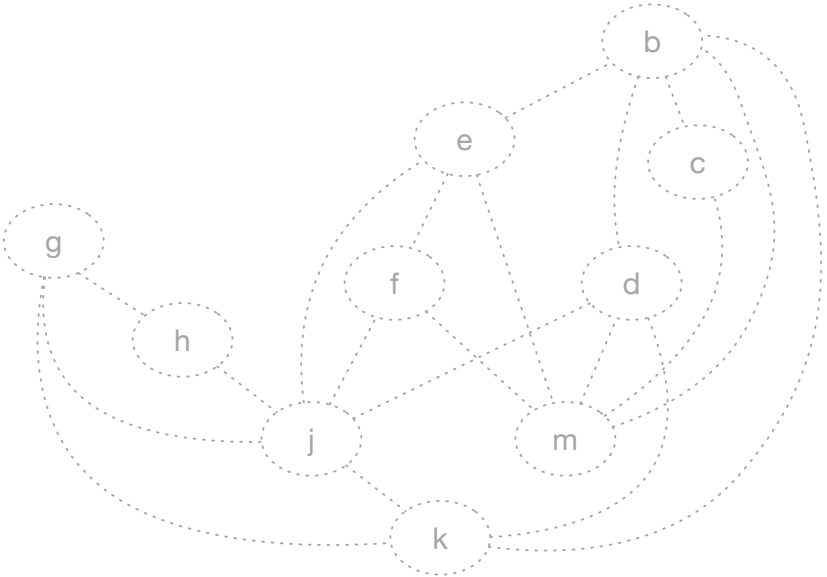
Remove b
Remove f
Remove e
Remove j
Remove d
Remove k
Remove h
Remove g

Full example



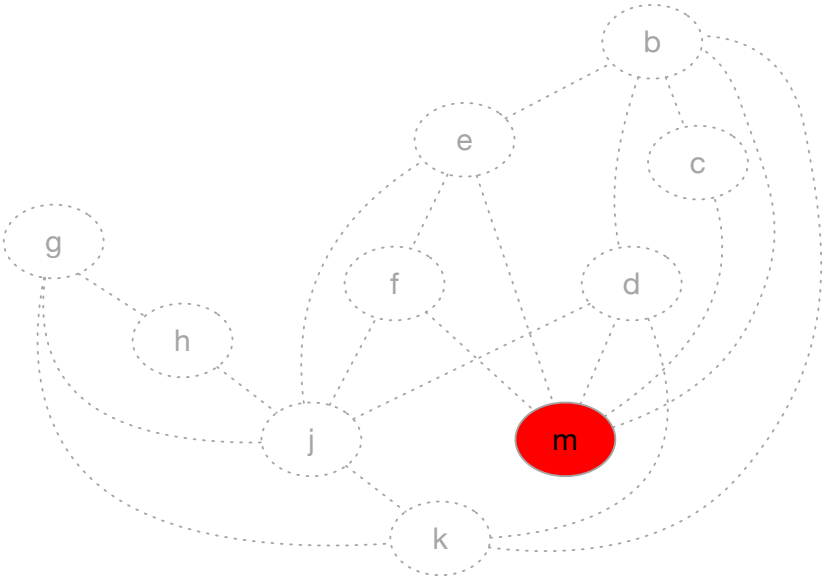
Remove c
Remove b
Remove f
Remove e
Remove j
Remove d
Remove k
Remove h
Remove g

Full example



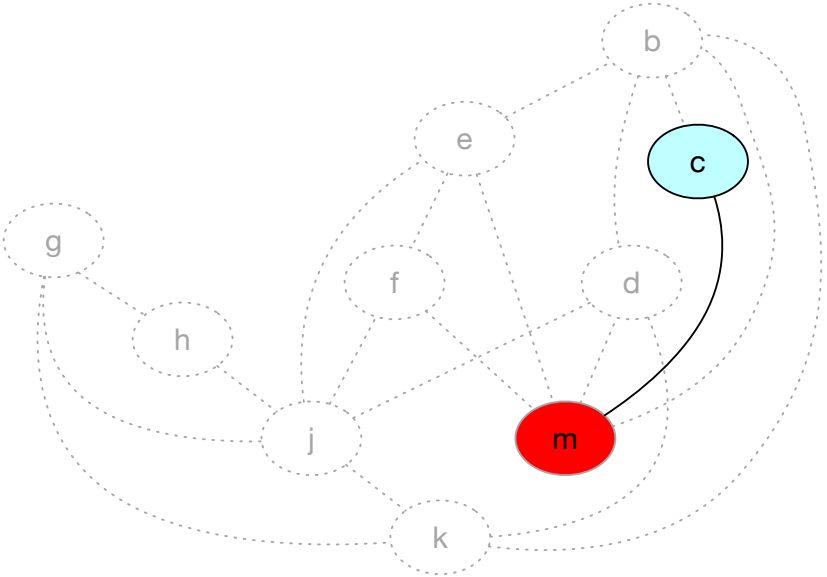
Remove m
Remove c
Remove b
Remove f
Remove e
Remove j
Remove d
Remove k
Remove h
Remove g

Full example



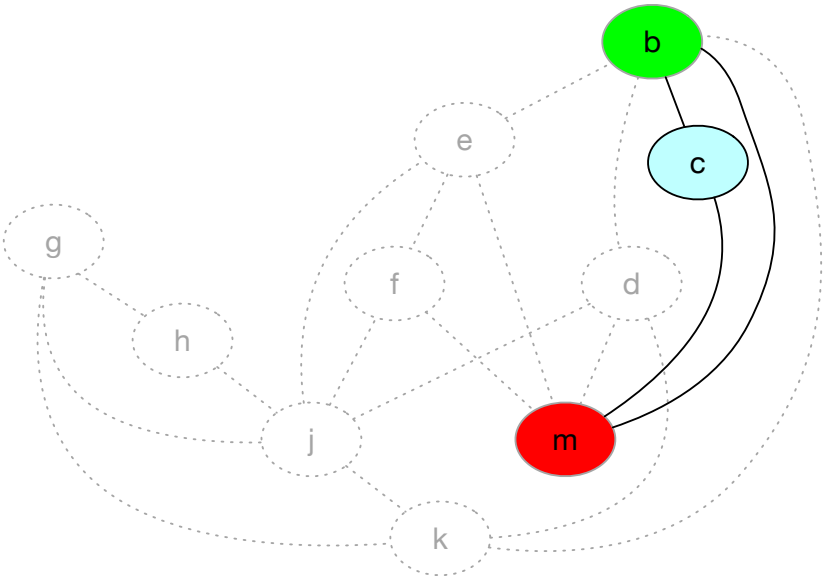
Remove c
Remove b
Remove f
Remove e
Remove j
Remove d
Remove k
Remove h
Remove g

Full example



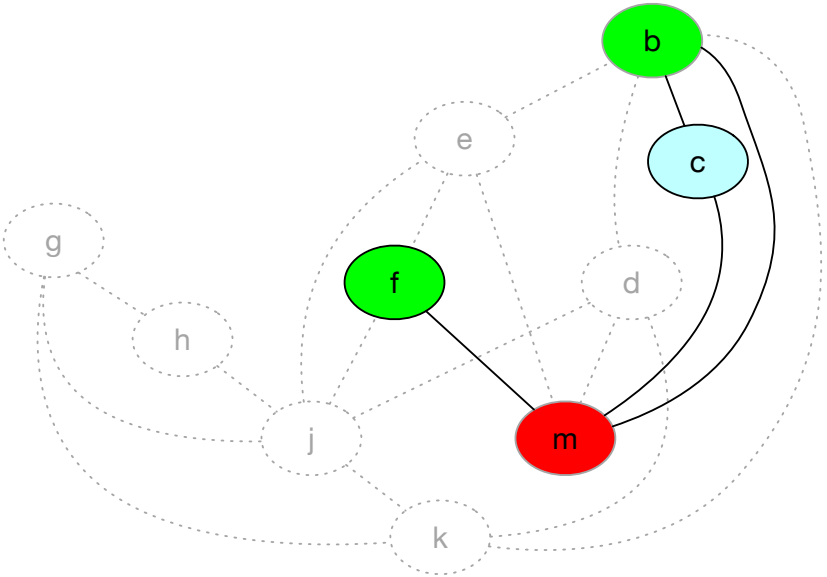
Remove b
Remove f
Remove e
Remove j
Remove d
Remove k
Remove h
Remove g

Full example



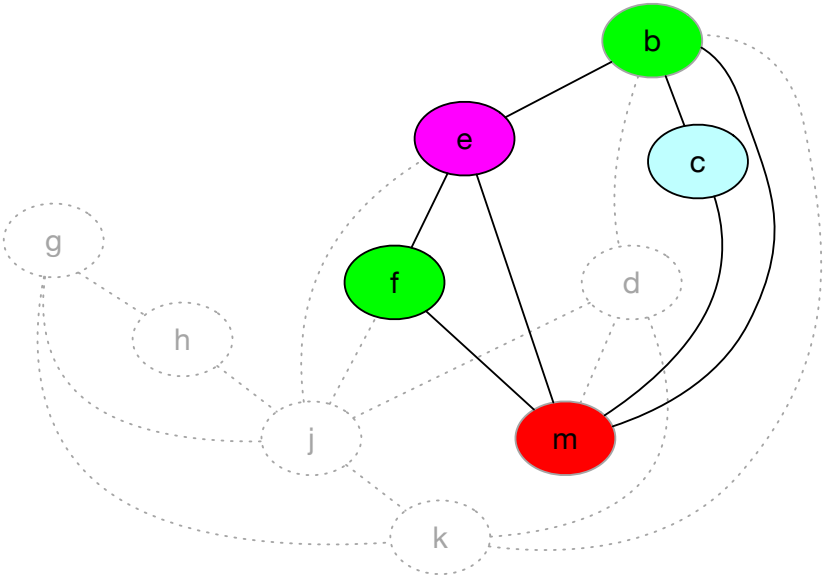
Remove f
Remove e
Remove j
Remove d
Remove k
Remove h
Remove g

Full example



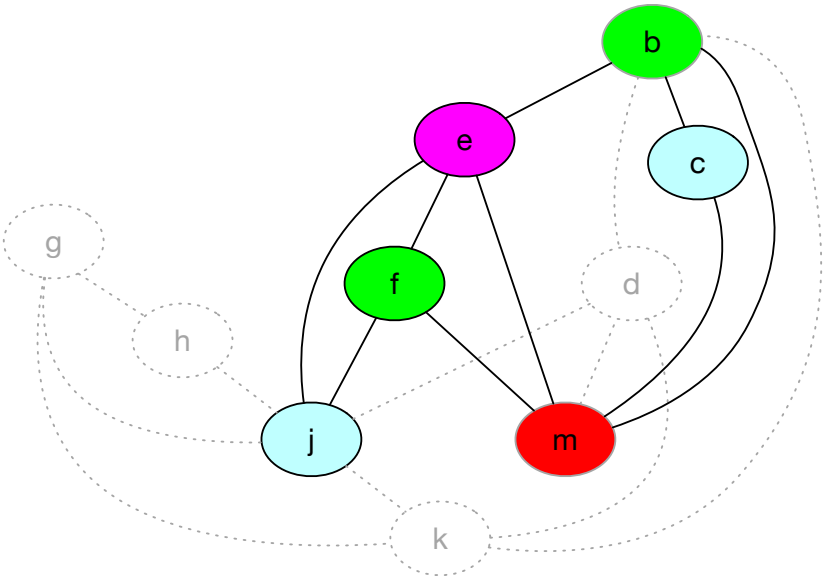
Remove e
Remove j
Remove d
Remove k
Remove h
Remove g

Full example



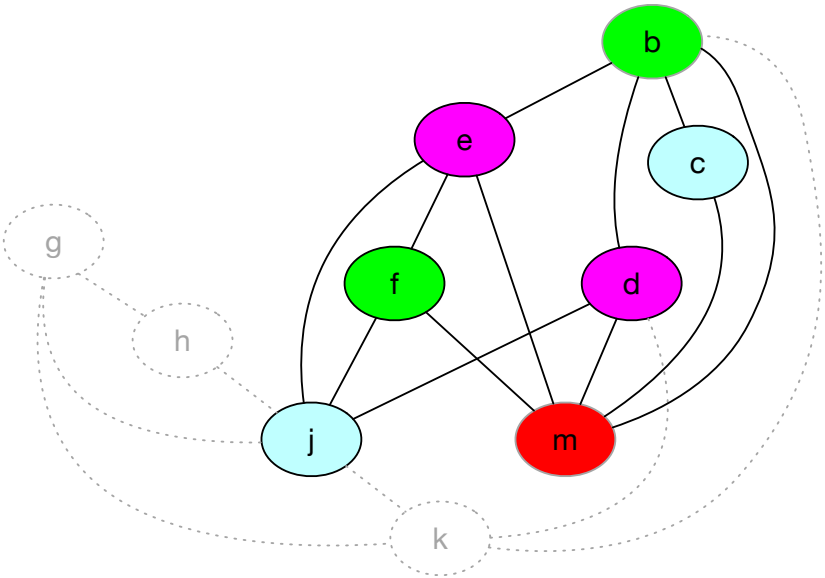
Remove j
Remove d
Remove k
Remove h
Remove g

Full example

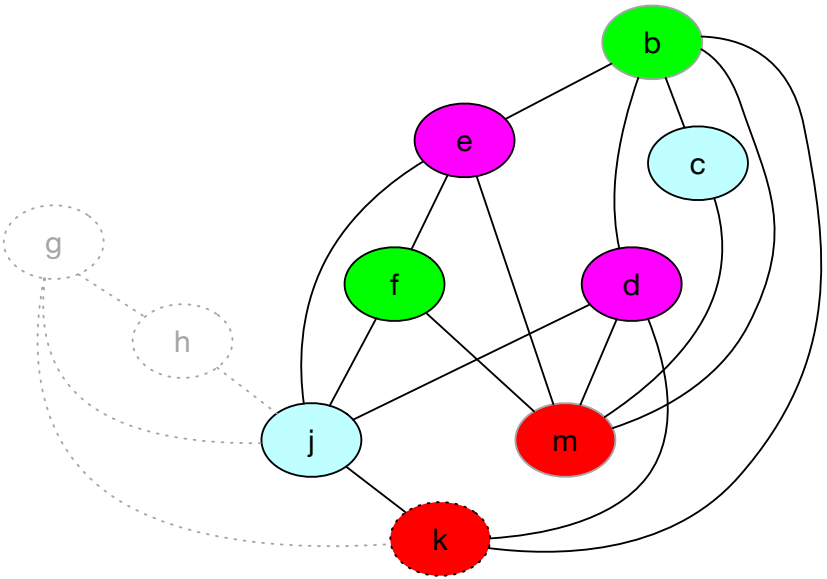


Remove d
Remove k
Remove h
Remove g

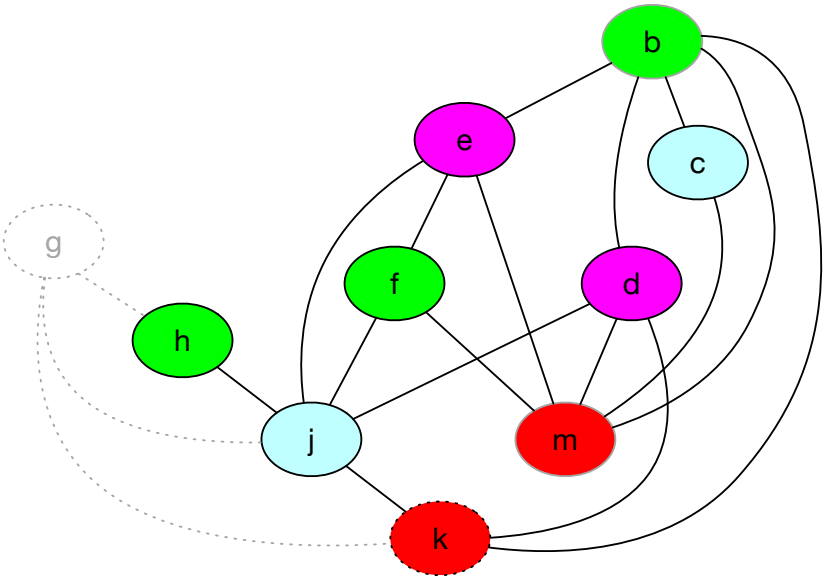
Full example



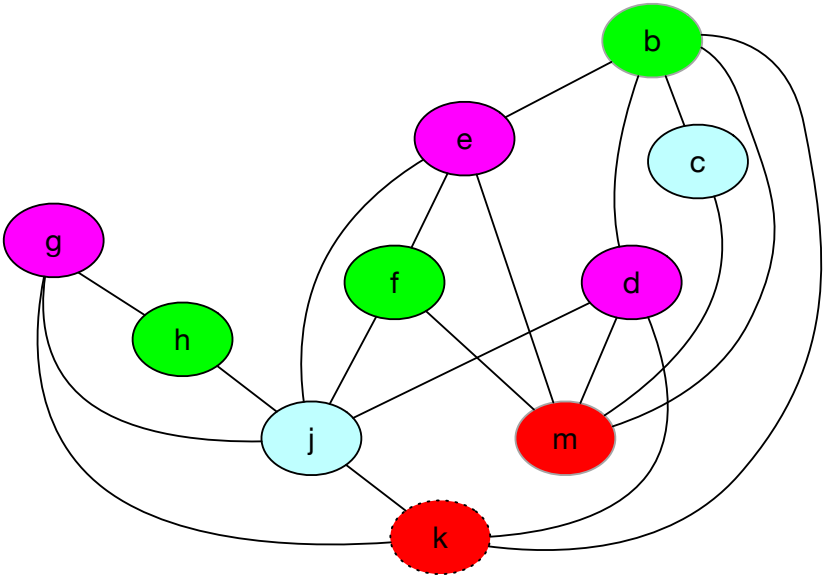
Full example



Full example



Full example



Result

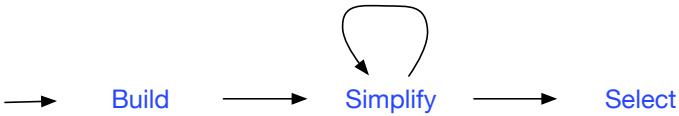
```
live in: k j
  g := [j + 12]
  h := k - 1
  f := g * h
  e := [j + 8]
  m := [j + 16]
  b := [f]
  c := e + 8
  d := c
  k := m + 4
  j := b
live out: d k j
```

```
live in: r1 r3
  r4 := [r3 + 12]
  r2 := r1 - 1
  r2 := r4 * r2
  r4 := [r3 + 8]
  r1 := [r3 + 16]
  r2 := [r2]
  r3 := r4 + 8
  r4 := r3
  r1 := r1 + 4
  r3 := r2
live out: r4 r1 r3
```

Yes, but What Color?

- Usually, first-fit (registers are ordered).
- Trying caller save first helps.
- **Biased Coloring.**
  Use a color already unavailable to our neighbors.

Naive workflow



build the conflict graph from the program

simplify the nodes with insignificant degree

select (or color) while rebuilding the graph.

Summary