MO910 – Construção de Compiladores

Prof. Guido Araujo
www.ic.unicamp.br/~guido

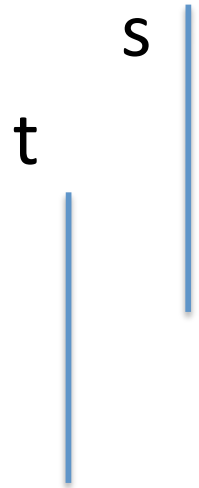
Alocação Global de Registradores

Grafo de Interferência

- Informa quais variáveis estão vivas em um mesmo ponto do programa.
- Construído a partir da análise de longevidade.
- Utilizado para alocação de registradores.
- Interferência: ocorre quando a e b não podem ocupar o mesmo registrador
- Outras formas de interferência:
  - Ex: Quando a é gerado por uma instrução que não pode escrever no registrador r1. Neste caso a interfere com r1.

Grafo de Interferência

- Tratamento especial de cópias: é importante não criar falsas interferências entre a fonte e destino




```
t := s        ; cópia
...
x := op s     ; uso de s
...
y := op t     ; uso de t
```

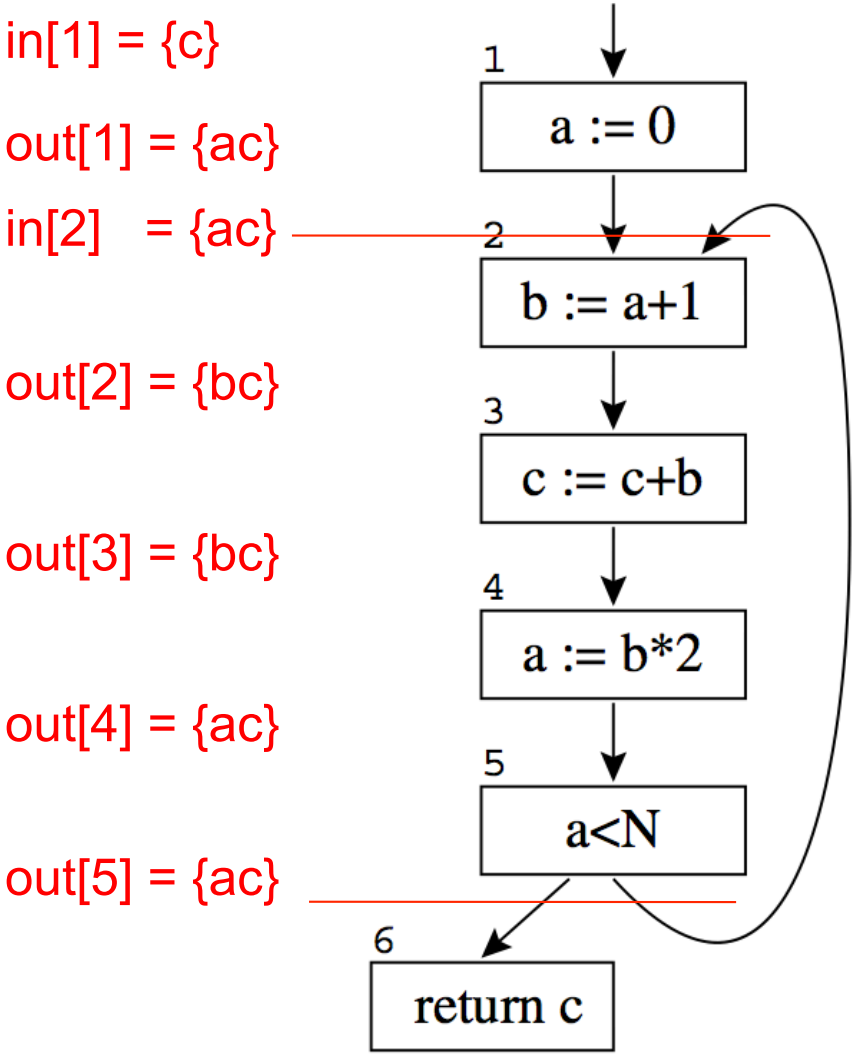
- s e t estiverem vivas após a instrução de cópia
- Podemos aproveitar o mesmo registrador: Não criar interferência entre as duas.

Grafo de Interferência

1. Definição de *a* que não seja move:
   - *Live-out* = $b_1, \ldots, b_j$
     - Adicione as arestas $(a, b_1), \ldots, (a, b_j)$.

2. Moves a ← c:
   - *Live-out* = $b_1, \ldots, b_j$
     - Adicione as arestas $(a, b_1), \ldots, (a, b_j)$ para os b_i's diferentes de c.

Liveness Analysis



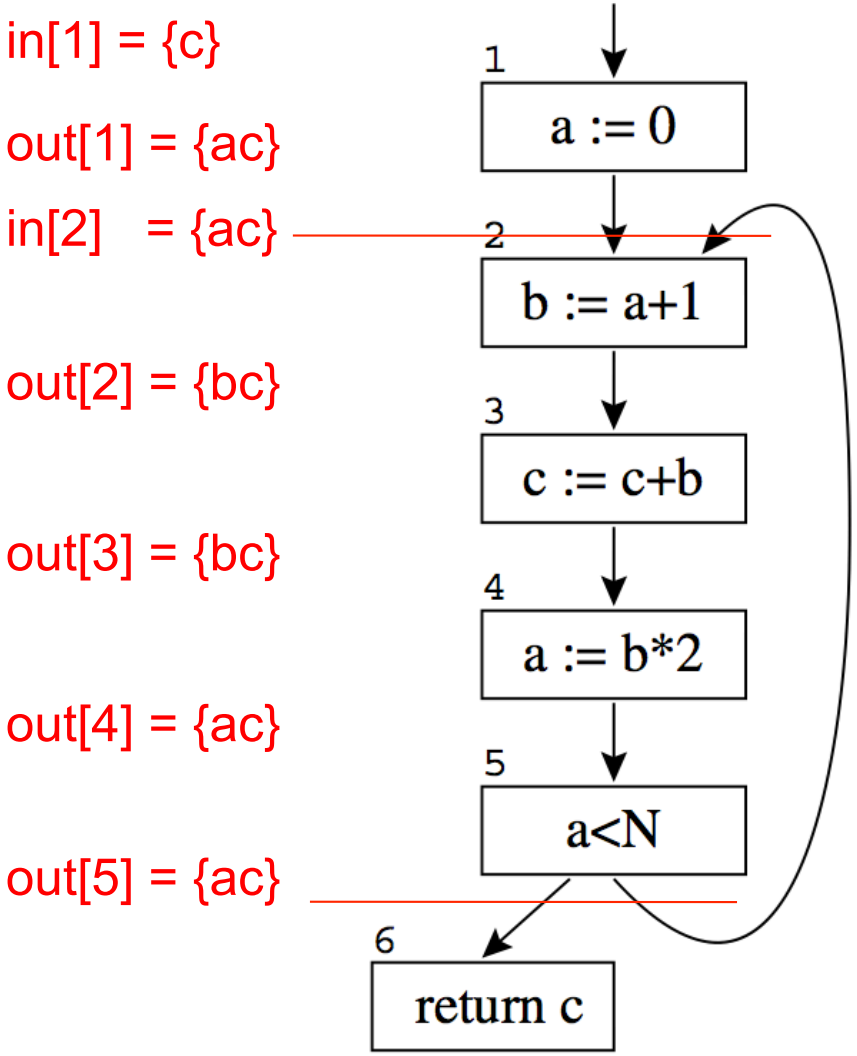
Sentenças a := .....

Live-out = b1,..., bj

Adicione as arestas

(a, b1),...,(a, bj).

Liveness Analysis

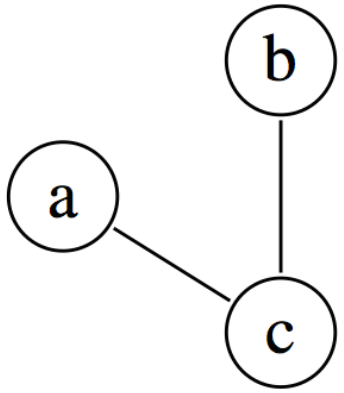


Sentenças a := .....

Live-out = b1,..., bj

Adicione as arestas

(a, b1),...,(a, bj).



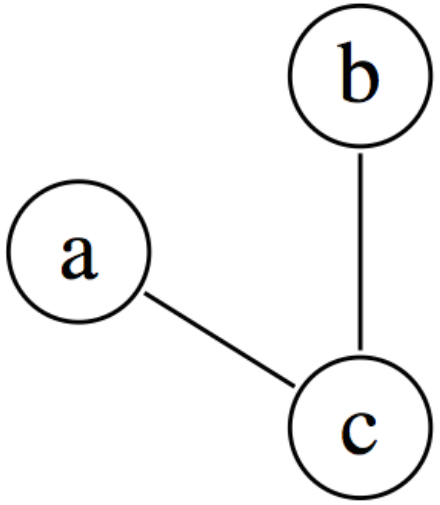
(b) Graph

Grafo de Interferência

- Representação:

Variáveis	a	b	c
a			x
b			x
c	x	x	

Matriz



(b) Graph

Alocação de Registradores

- A IR e a seleção de instruções assumiram que o número de registradores era infinito
- Objetivo:
  - Atribuir registradores físicos (da máquina) para os temporários usados nas instruções.
  - Se possível, atribuir a fonte e o destino de operações de cópia (MOVES) para o mesmo registrador
    - Elimina operações inúteis!

Alocação de Registradores

- Grafo de Interferência (IG):
  - Temos arestas entre t1 e t2 se eles não podem ocupar o mesmo registrador.
  - Estão vivas no mesmo ponto; ou
  - Restrições da arquitetura;
    - P. Ex: a = a + b não pode ser atribuido ao r12
- O problema se transforma em um problema de coloração de grafos.

Coloração do IG

- Queremos colorir o IG com o mínimo de cores possíveis, de maneira que nenhum par de nós conectados por uma aresta tenham a mesma cor.
  - Coloração de vértices
  - As cores representam os registradores
  - Se nossa máquina tem k registradores e encontrarmos uma k-coloração para o IG a coloração é uma alocação válida dos registradores.

Coloração do IG

- E se não existir uma k-coloração?
  - Então teremos que colocar alguns dos temporários ou variáveis na memória.
  - Operação conhecida como *spilling.*
- Coloração de vértices é um problema NP-Completo.
  - Logo, alocação de registradores por coloração também é.
- Existe uma aproximação linear que traz bons resultados

Coloração por Simplificação

- Principais fases

1. *Build*
2. *Simplify*
3. *Spill*
4. *Select*

Coloração por Simplificação

1. *Build*:
    - Construir o IG
    - Usa a análise de longevidade

Coloração por Simplificação

2. *Simplify*:
   - Heurística
   - Suponha que o grafo G tenha um nó *m* com menos de k vizinhos
   - K é o número de registradores
   - Faça G’ = G – {m}
   - Se G’ pode ser colorido com k cores, G também pode

Coloração por Simplificação

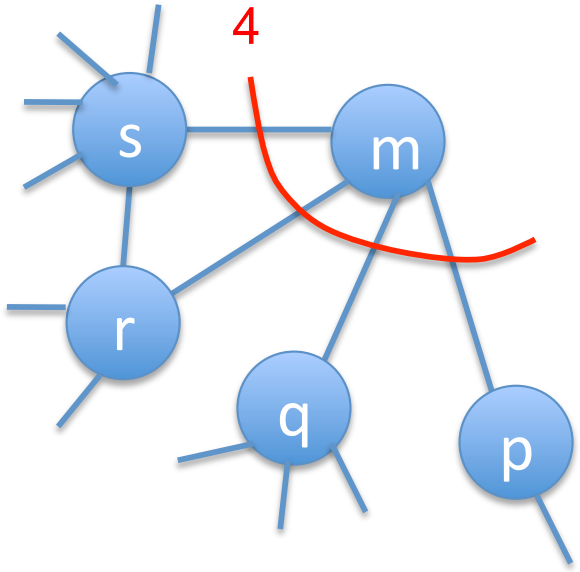
2. Simplify:
   - Leva a um algoritmo recursivo (pilha)
     - Repetidamente:
       - Remova nós de grau menor que K
       - Coloque na pilha
     - Cada remoção diminui o grau dos nós em G, dando oportunidades para novas remoções

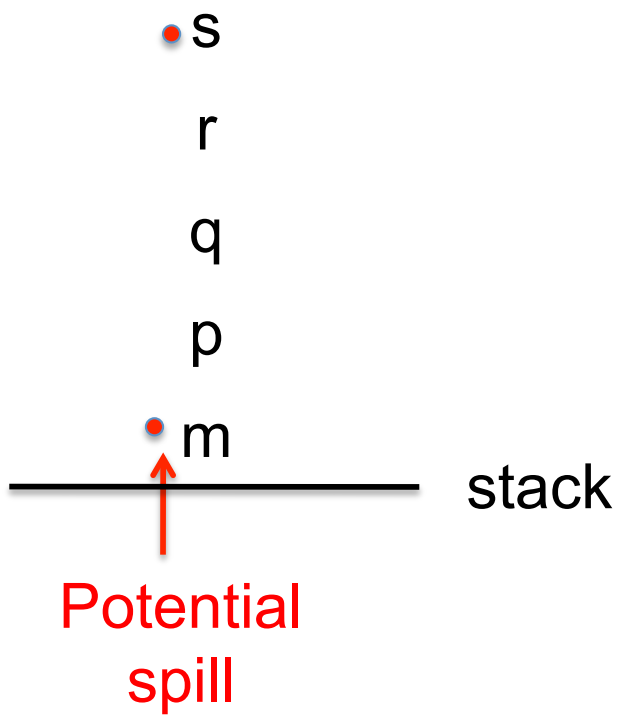
Coloração por Simplificação

3. Spill:
   - Em algum momento não teremos um nó com grau < k
   - A heurística falha
   - Temos que marcar algum nó para *spill*
   - A escolha desse nó é também uma heurística
     - Nó que reduza o grau do maior número de outros nós
     - Nó com menor custo relacionado as operações de memória

Spill em Potencial

- K = 4

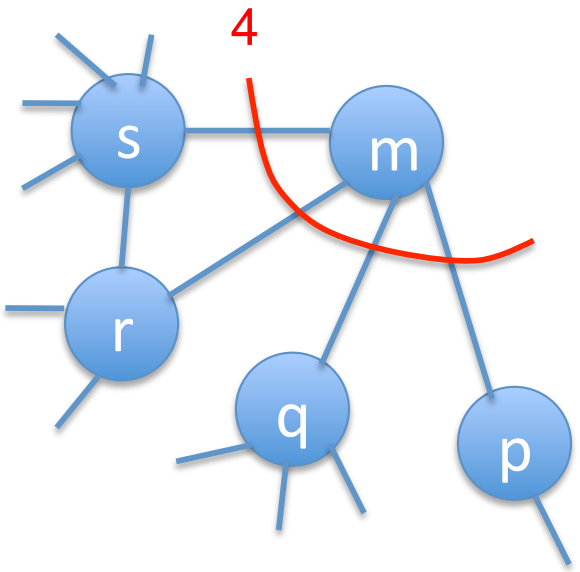




Custo de Spill em Potencial

- K = 4
- Program

```
        :
m = m +1;
        :
for (i=0; i < N; i++)
    k +=  a[i] + m;
        :
```



s
r
q
p
m

$$\text{cost}(m) = \frac{\sum \text{uses}(m)}{\text{degree}(m)} = \frac{2 + N}{4}$$

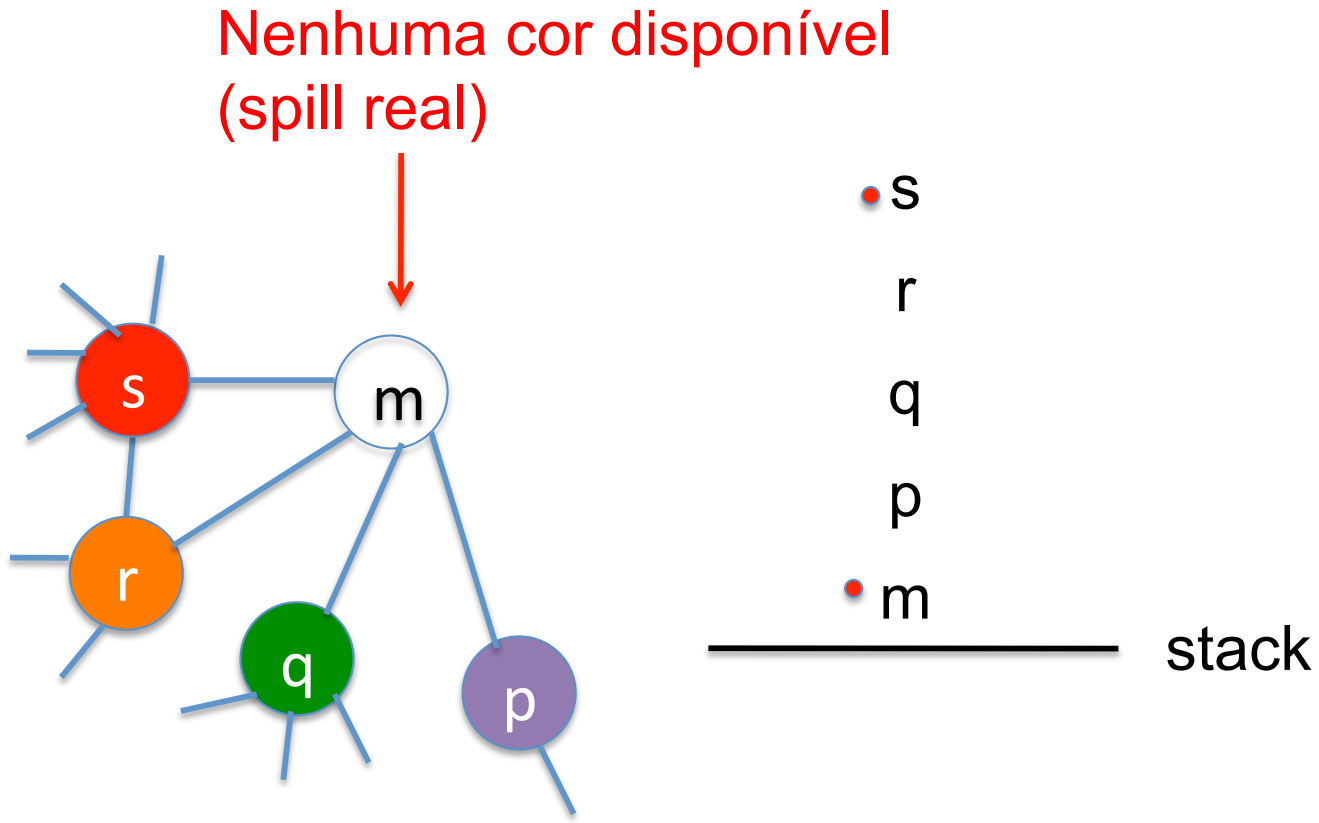
M foi a melhor escolha para spill?

Coloração por Simplificação

4. Select:
   - Atribui as cores
   - Reconstrói o grafo G adicionando os nós na ordem determinada pela pilha
   - Quando adicionamos um nó, devemos ter uma cor para ele dado o critério de seleção usado para remover
   - Isso não vale para os nós empilhados marcados como spill
     - Se todos os vizinhos já usarem k cores, não adicionamos no grafo
     - Continua o processo

Select color

- K = 4

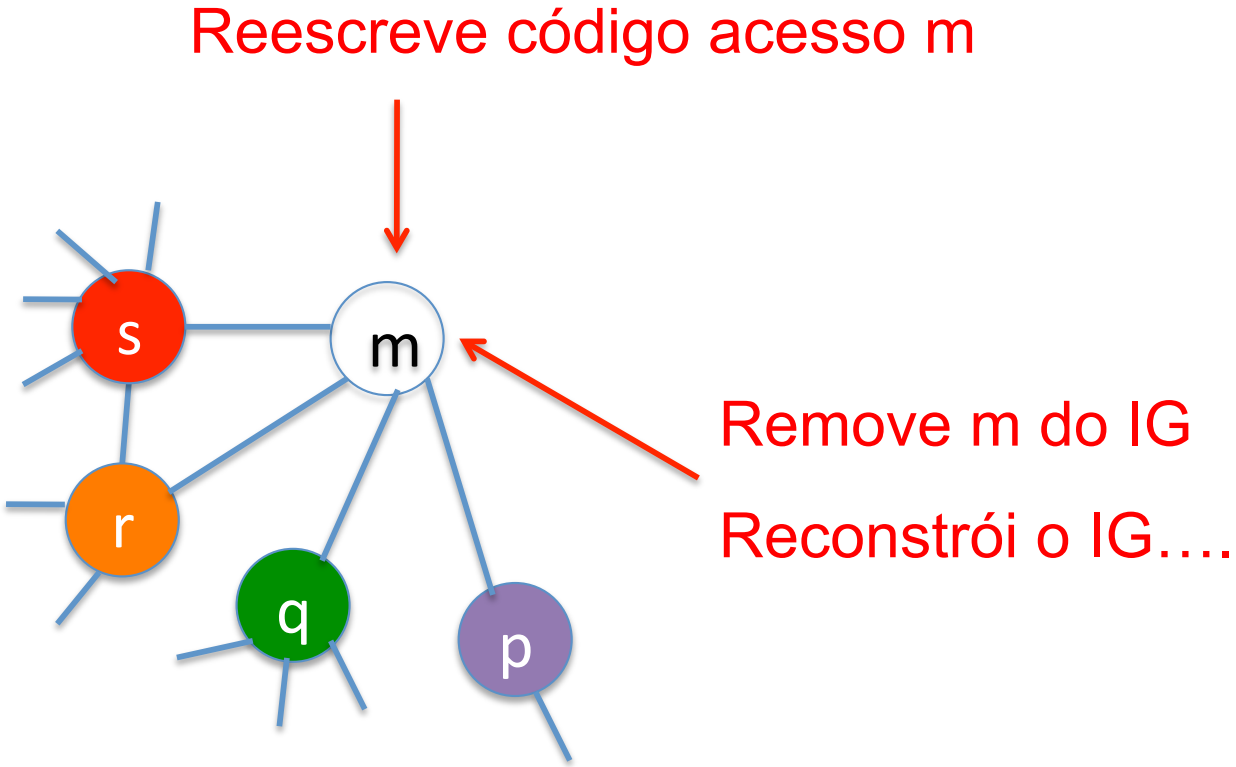


Fazendo spill no código

- K = 4

m = m +1;

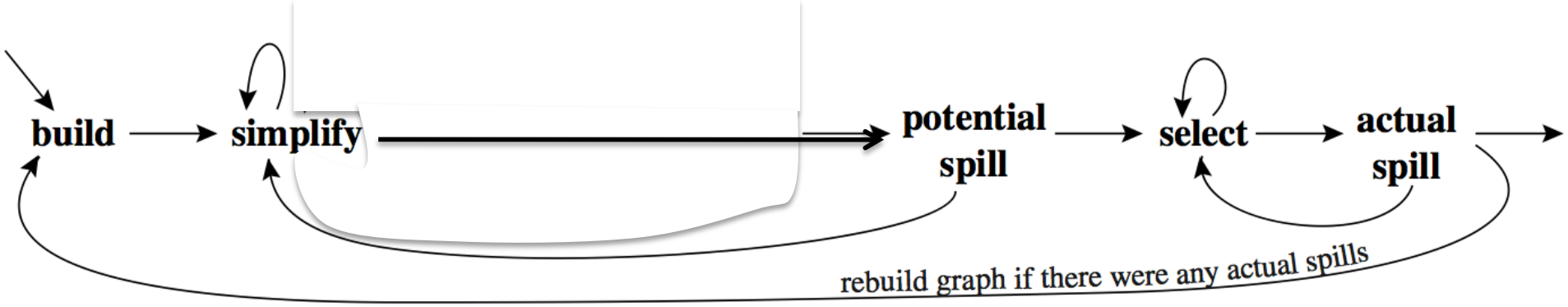
t3 := Mem[m]
t3 := t3 + 1
Mem[m] := t3



Coloração por Simplificação

5. Start Over:
   - Pode ser que o *Select* não consiga atribuir uma cor a algum nó
   - Reescrever o código para gravar o valor na memória após cada definição e ler da memória antes de cada uso.
   - Isso gera novos temporários
     - Com *live ranges* mais curtas
   - O algoritmo é repetido desde a construção do IG
   - O processo acaba quando *Select* tiver sucesso para todos os vértices

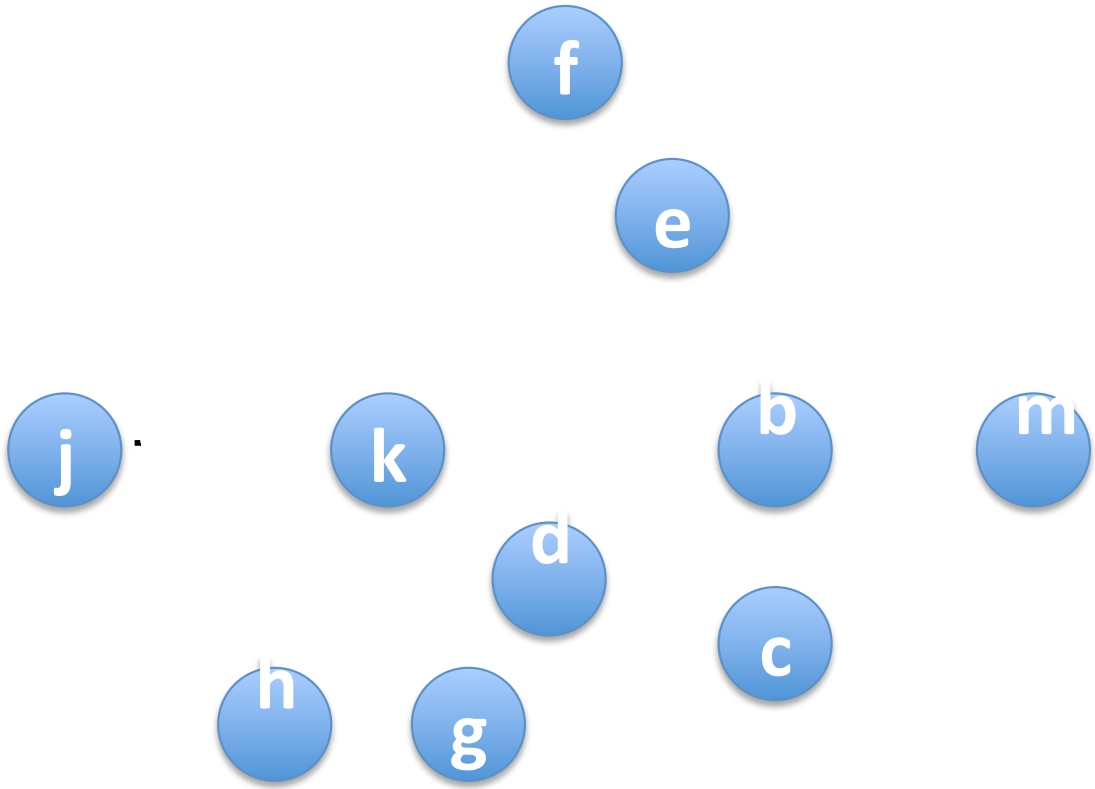
Coloração por Simplificação



Exemplo

- Construa o IG

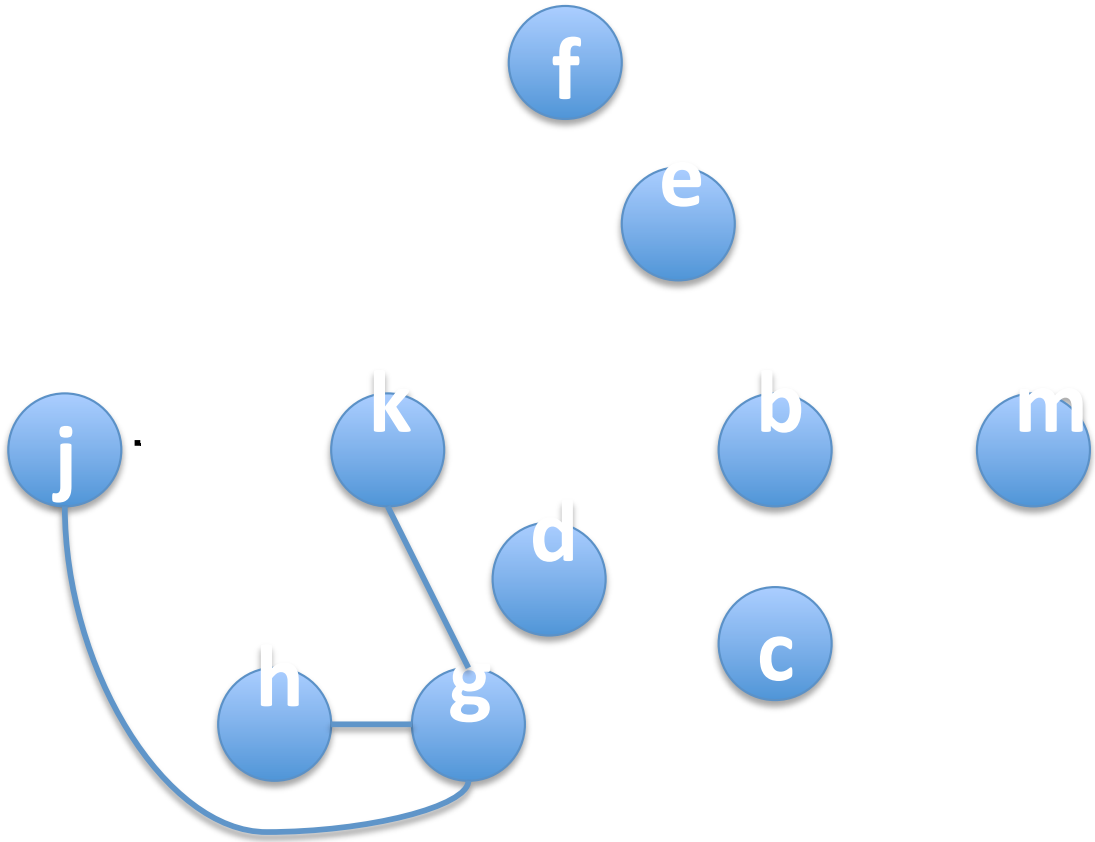
```
live-in: k j
   g := mem[j+12]
   h := k – 1
   f := g * h
   e := mem[j+8]
   m := mem[j+16]
   b := mem[f]
   c := e + 8
   d := c
   k := m + 4
   j := b
live-out: d k j
```



Exemplo

- Construa o IG

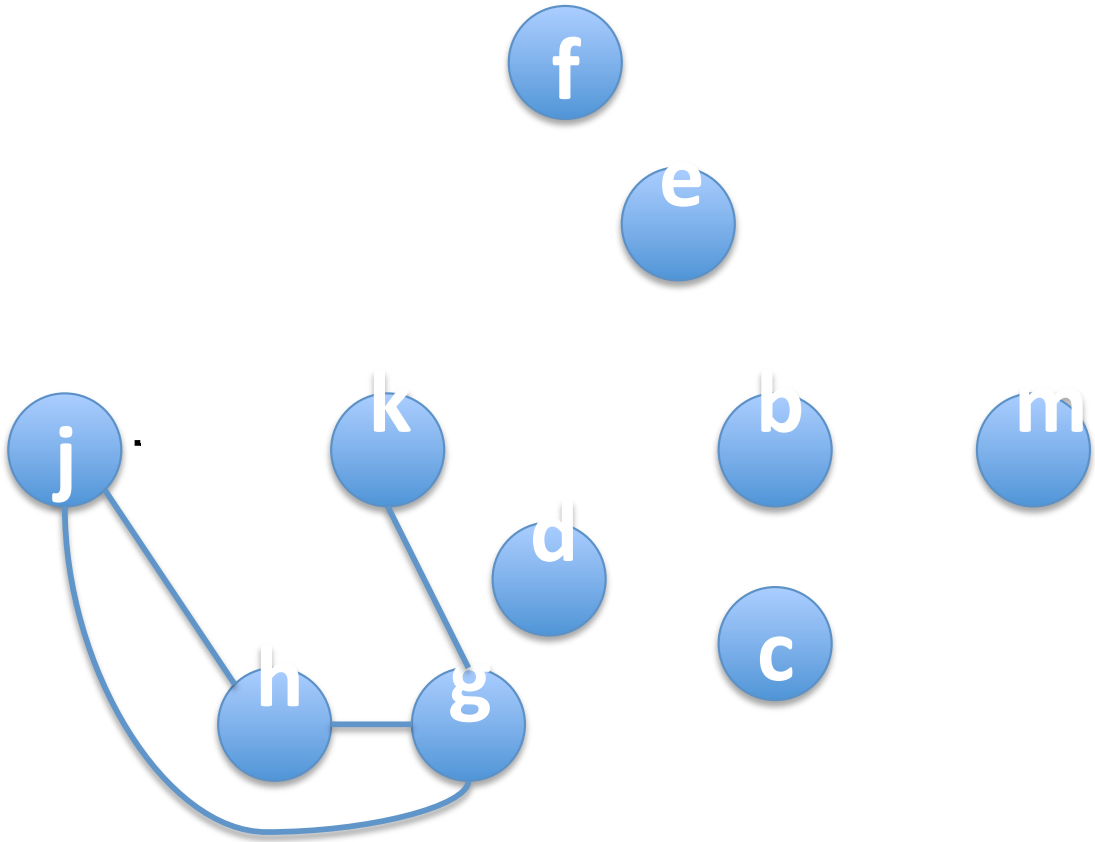
```
live-in: k j
   g := mem[j+12]
   h := k – 1
   f := g * h
   e := mem[j+8]
   m := mem[j+16]
   b := mem[f]
   c := e + 8
   d := c
   k := m + 4
   j := b
live-out: d k j
```



Exemplo

- Construa o IG

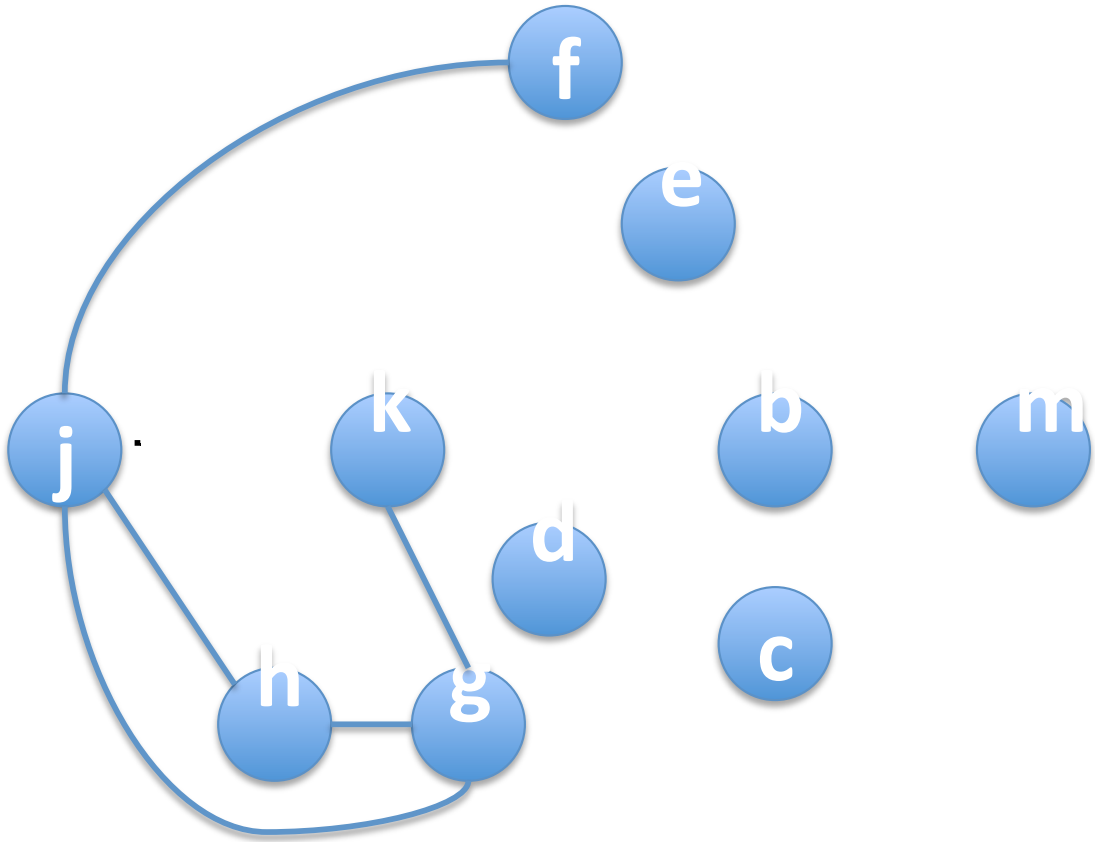
```
live-in: k j
   g := mem[j+12]
   h := k – 1
   f := g * h
   e := mem[j+8]
   m := mem[j+16]
   b := mem[f]
   c := e + 8
   d := c
   k := m + 4
   j := b
live-out: d k j
```



Exemplo

- Suponha que temos 4 registradores

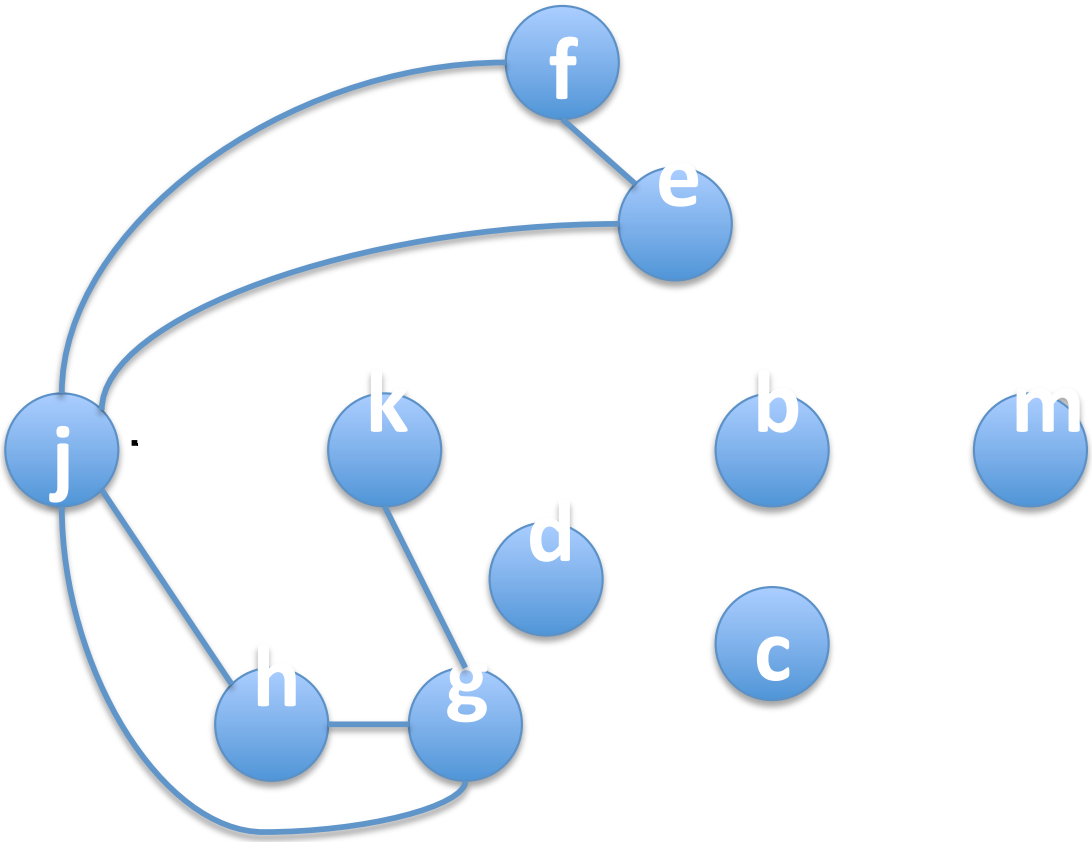
```
live-in: k j
   g := mem[j+12]
   h := k - 1
   f := g * h
   e := mem[j+8]
   m := mem[j+16]
   b := mem[f]
   c := e + 8
   d := c
   k := m + 4
   j := b
live-out: d k j
```



Exemplo

- Suponha que temos 4 registradores

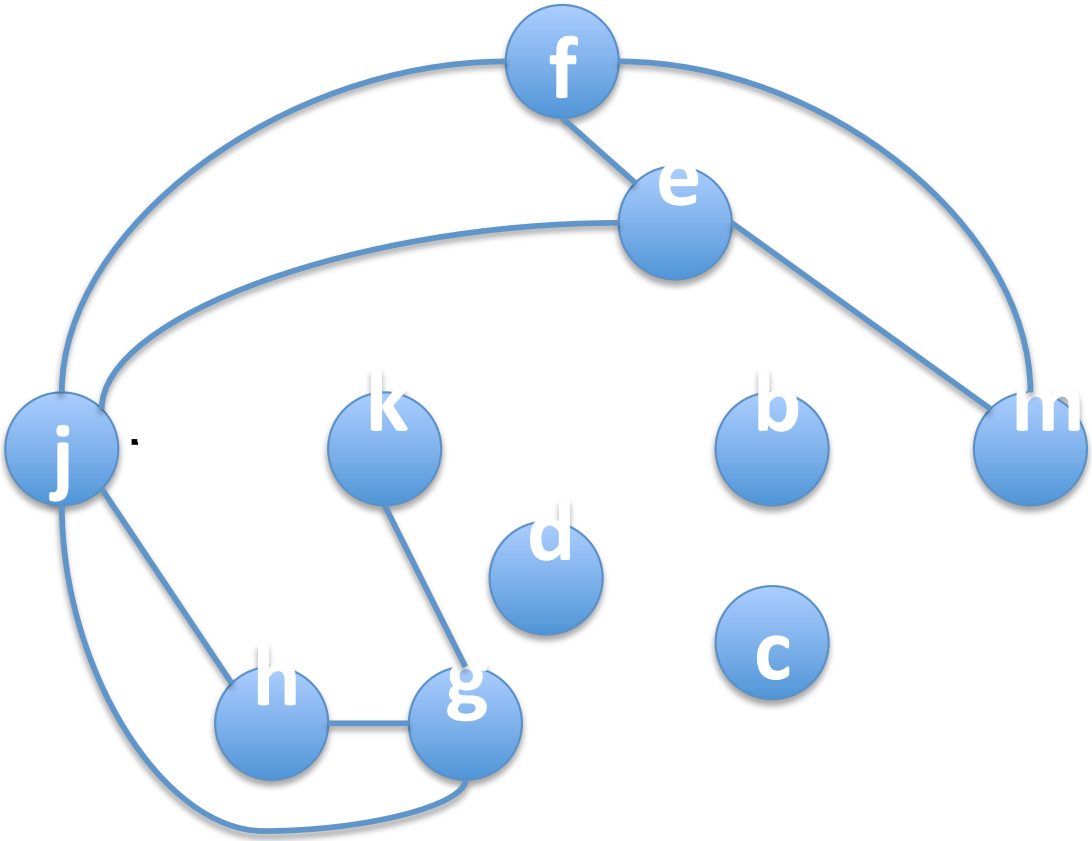
```
live-in: k j
   g := mem[j+12]
   h := k - 1
   f := g * h
   e := mem[j+8]
   m := mem[j+16]
   b := mem[f]
   c := e + 8
   d := c
   k := m + 4
   j := b
live-out: d k j
```



Exemplo

- Suponha que temos 4 registradores

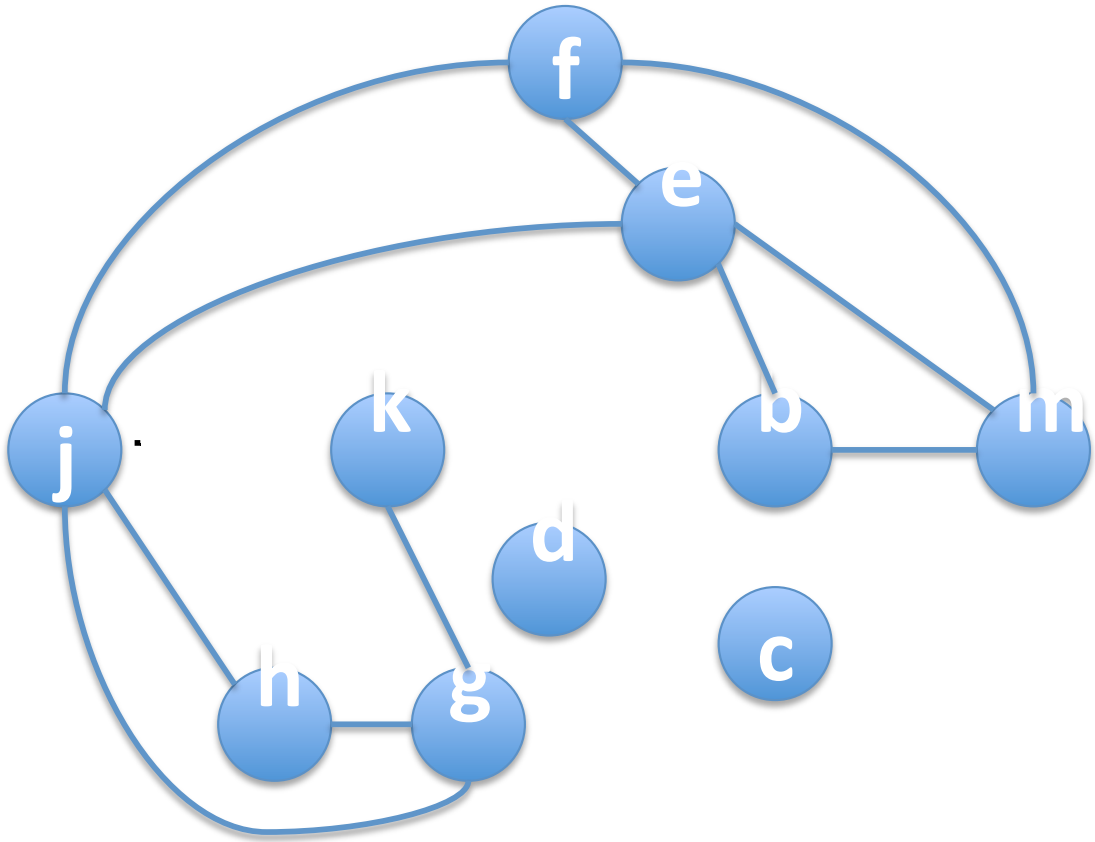
```
live-in: k j
   g := mem[j+12]
   h := k – 1
   f := g * h
   e := mem[j+8]
   m := mem[j+16]
   b := mem[f]
   c := e + 8
   d := c
   k := m + 4
   j := b
live-out: d k j
```



Exemplo

- Suponha que temos 4 registradores

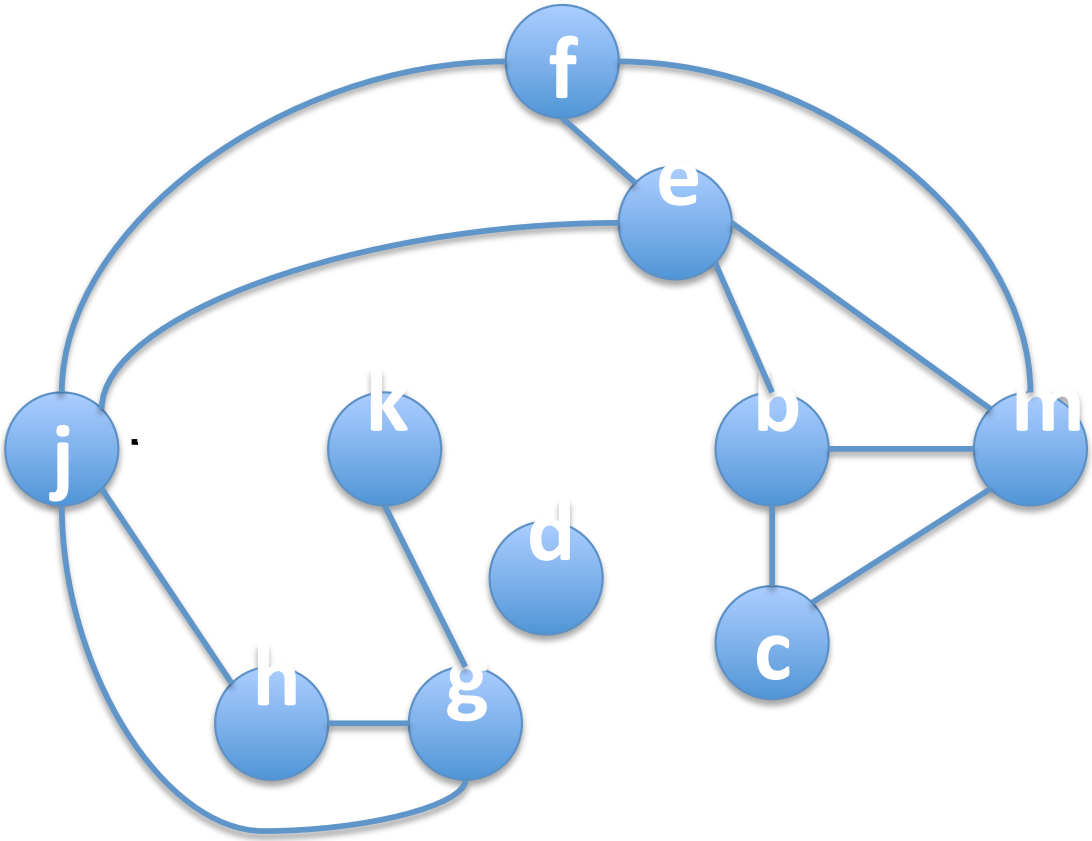
```
live-in: k j
   g := mem[j+12]
   h := k - 1
   f := g * h
   e := mem[j+8]
   m := mem[j+16]
   b := mem[f]
   c := e + 8
   d := c
   k := m + 4
   j := b
live-out: d k j
```



Exemplo

- Construa o IG

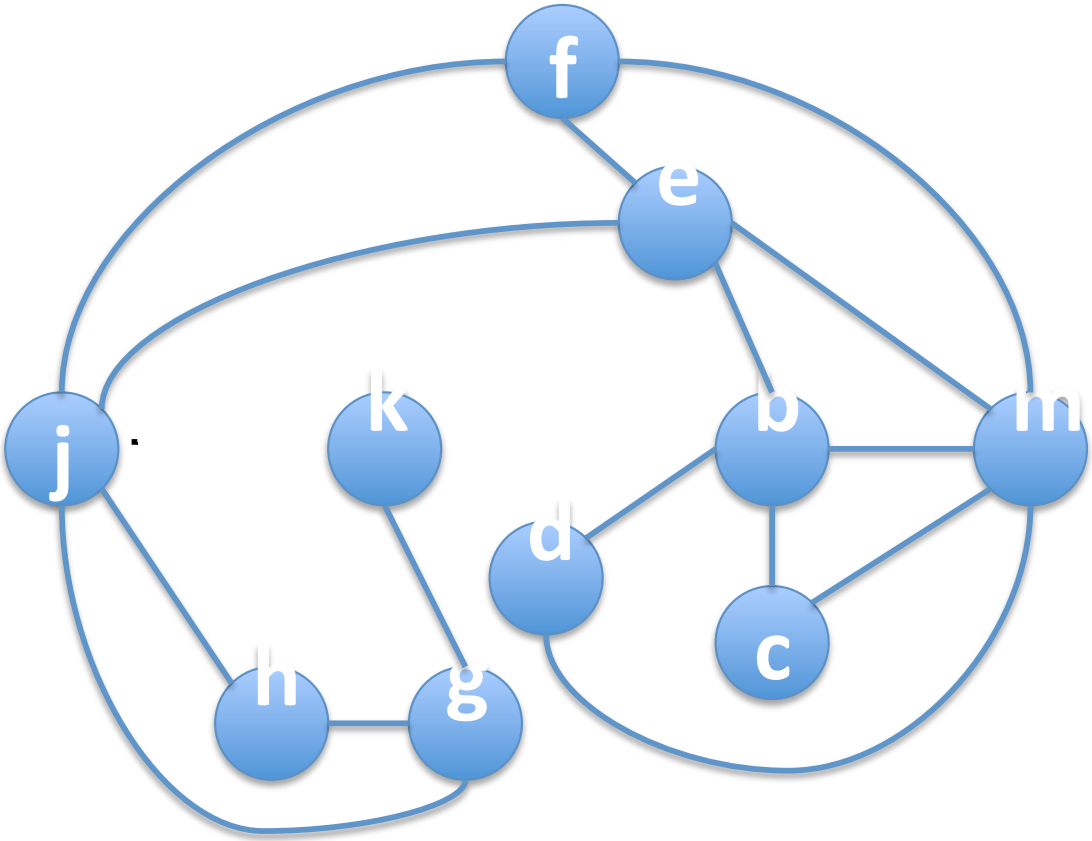
```
live-in: k j
   g := mem[j+12]
   h := k – 1
   f := g * h
   e := mem[j+8]
   m := mem[j+16]
   b := mem[f]
   c := e + 8
   d := c
   k := m + 4
   j := b
live-out: d k j
```



Exemplo

- Construa o IG

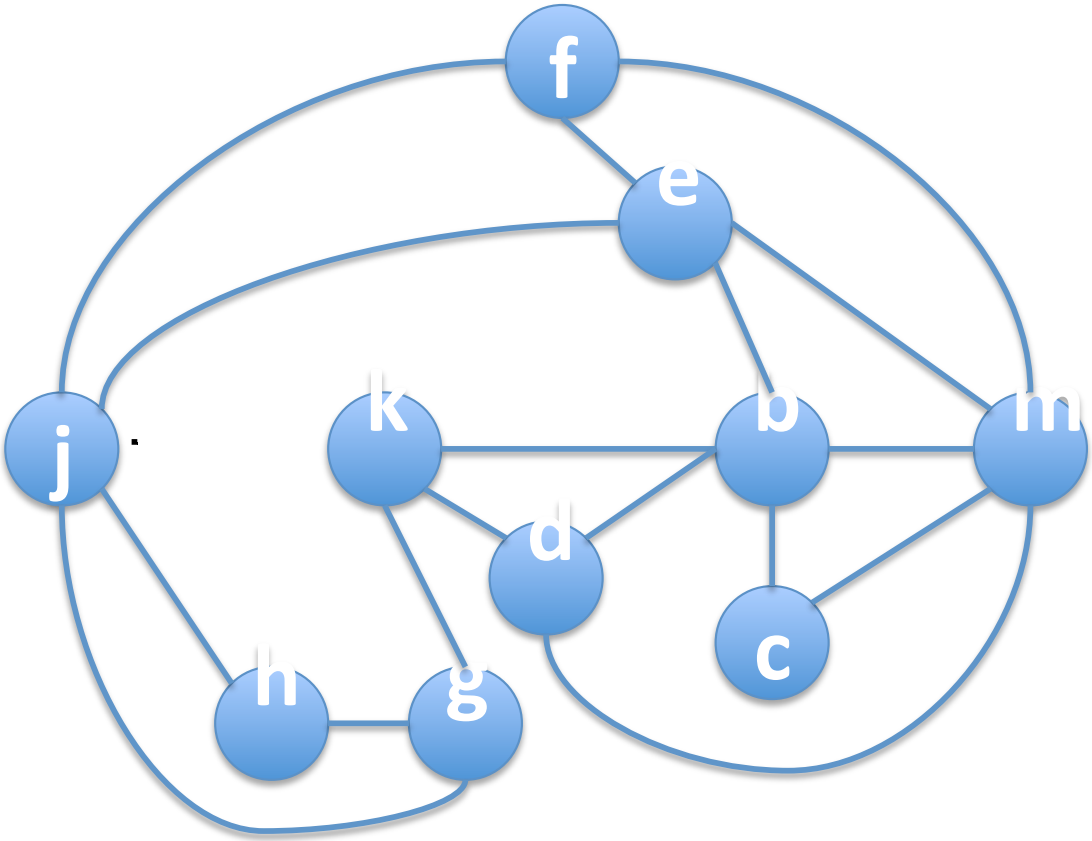
```
live-in: k j
   g := mem[j+12]
   h := k – 1
   f := g * h
   e := mem[j+8]
   m := mem[j+16]
   b := mem[f]
   c := e + 8
   d := c
   k := m + 4
   j := b
live-out: d k j
```



Exemplo

- Construa o IG

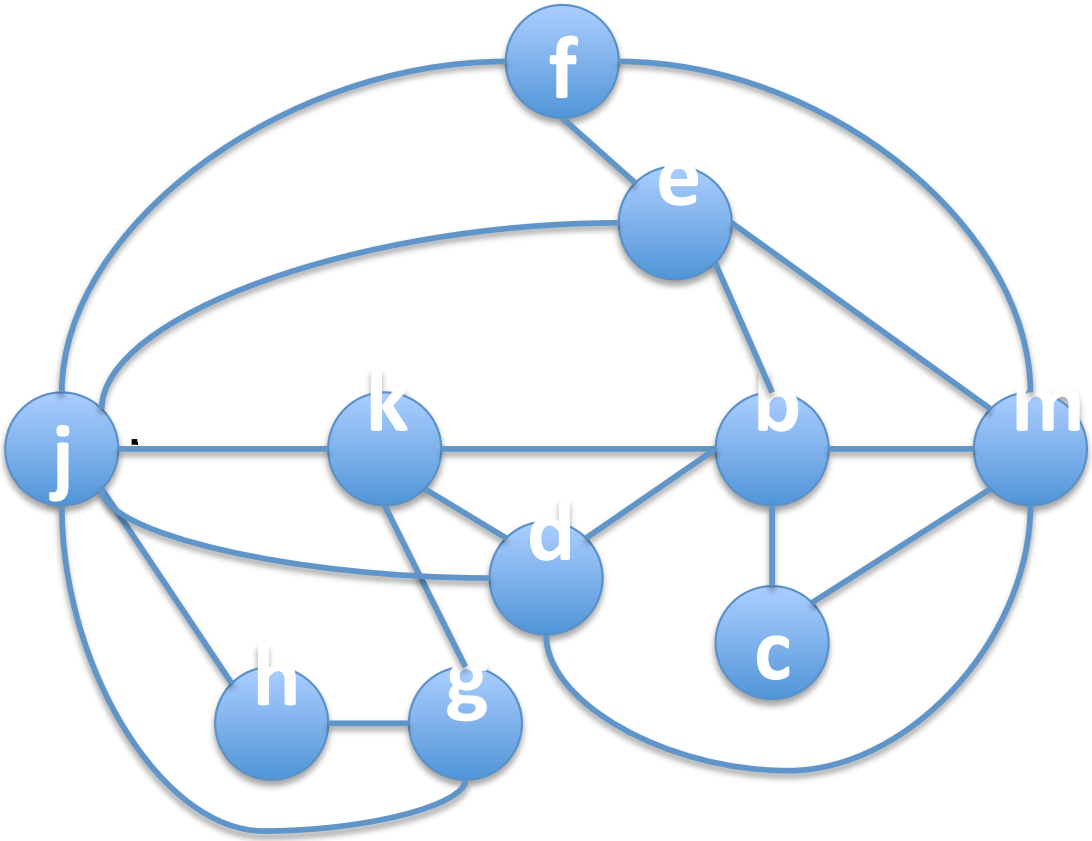
```
live-in: k j
   g := mem[j+12]
   h := k – 1
   f := g * h
   e := mem[j+8]
   m := mem[j+16]
   b := mem[f]
   c := e + 8
   d := c
   k := m + 4
   j := b
live-out: d k j
```



Exemplo

- Construa o IG

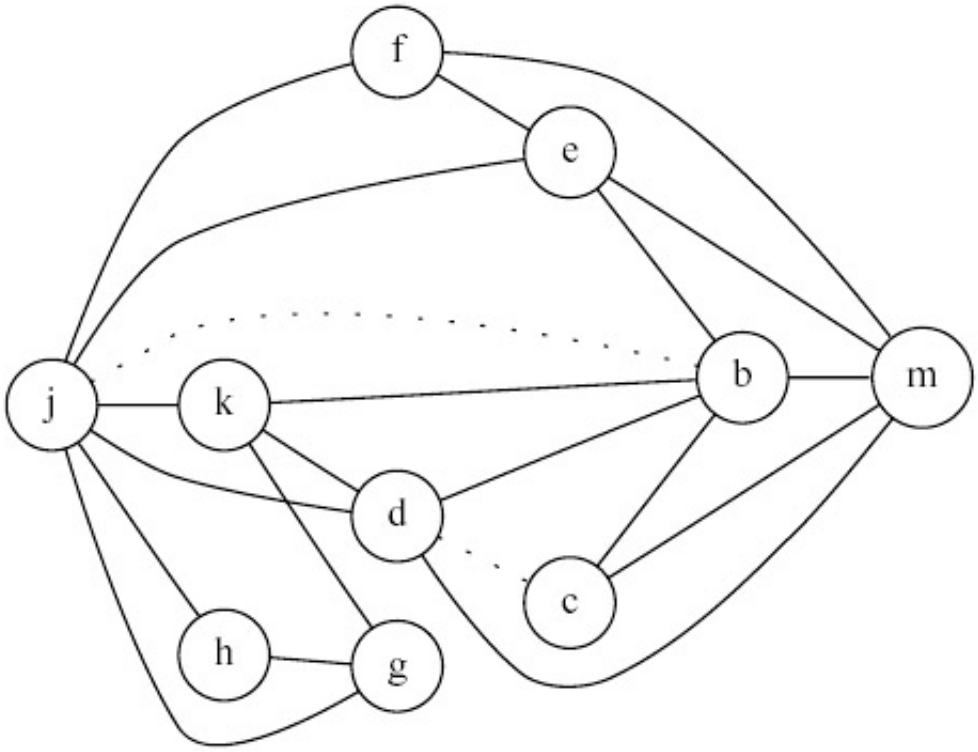
```
live-in: k j
   g := mem[j+12]
   h := k – 1
   f := g * h
   e := mem[j+8]
   m := mem[j+16]
   b := mem[f]
   c := e + 8
   d := c
   k := m + 4
   j := b
live-out: d k j
```



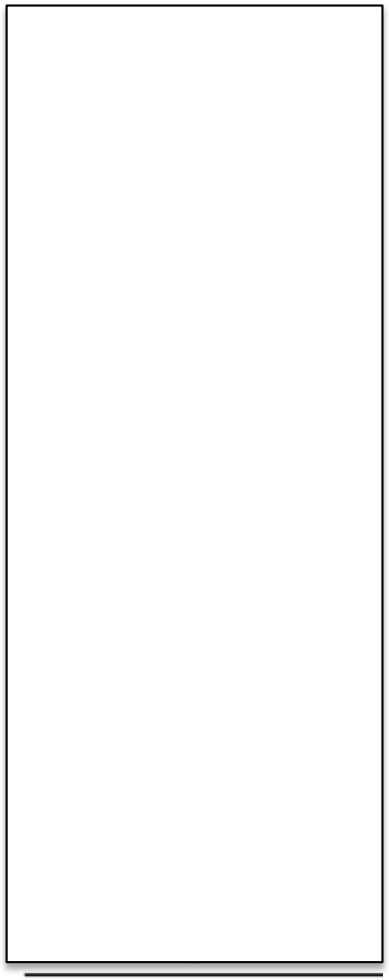
Exemplo

- Suponha que temos 4 registradores

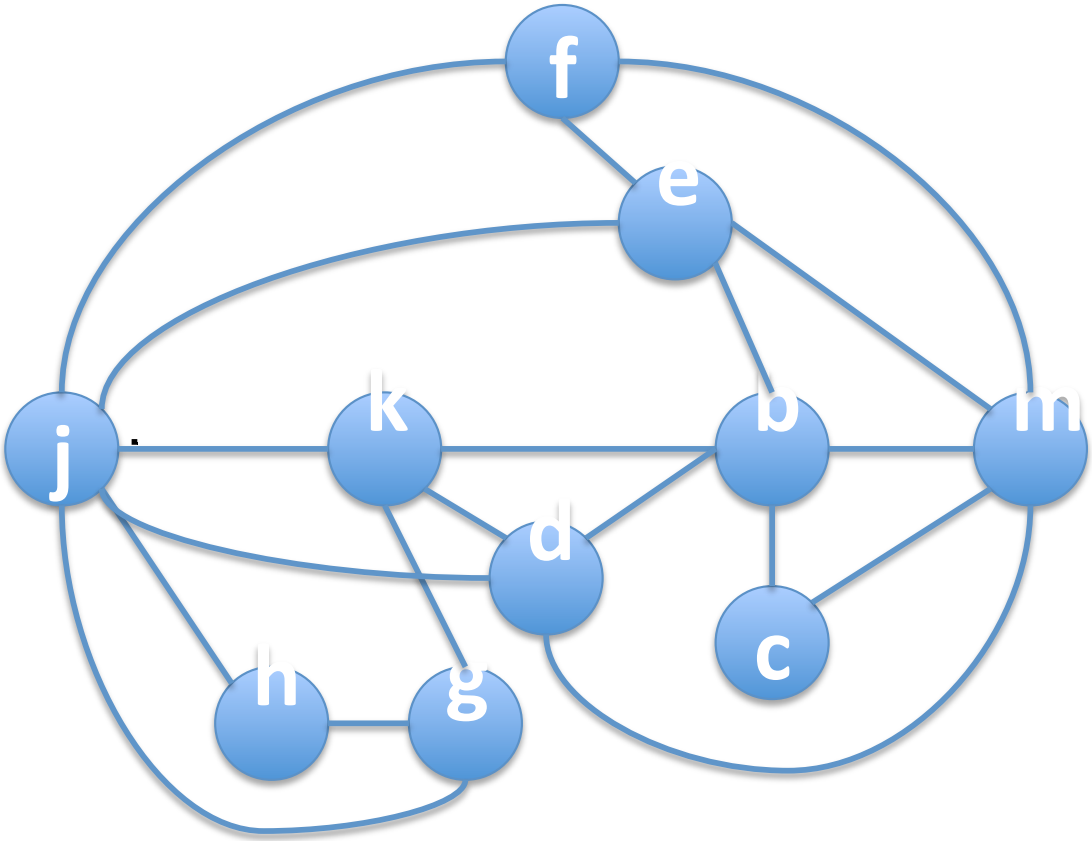
```
live-in: k j
         g := mem[j+12]
         h := k - 1
         f := g * h
         e := mem[j+8]
         m := mem[j+16]
         b := mem[f]
         c := e + 8
         d := c
         k := m + 4
         j := b
live-out: d k j
```



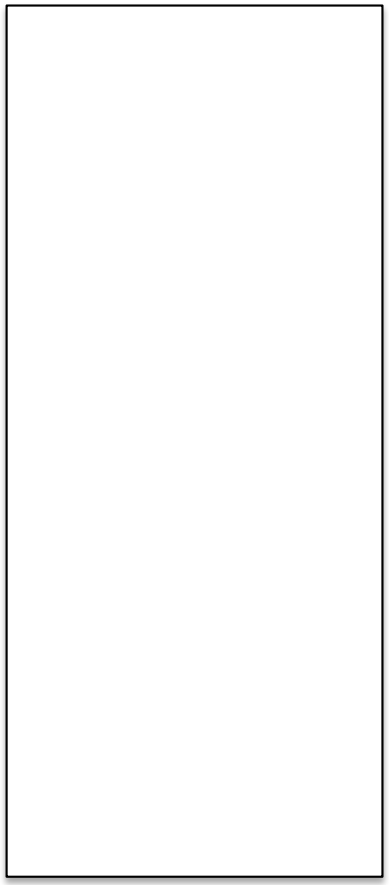
Simplify g



(a) stack

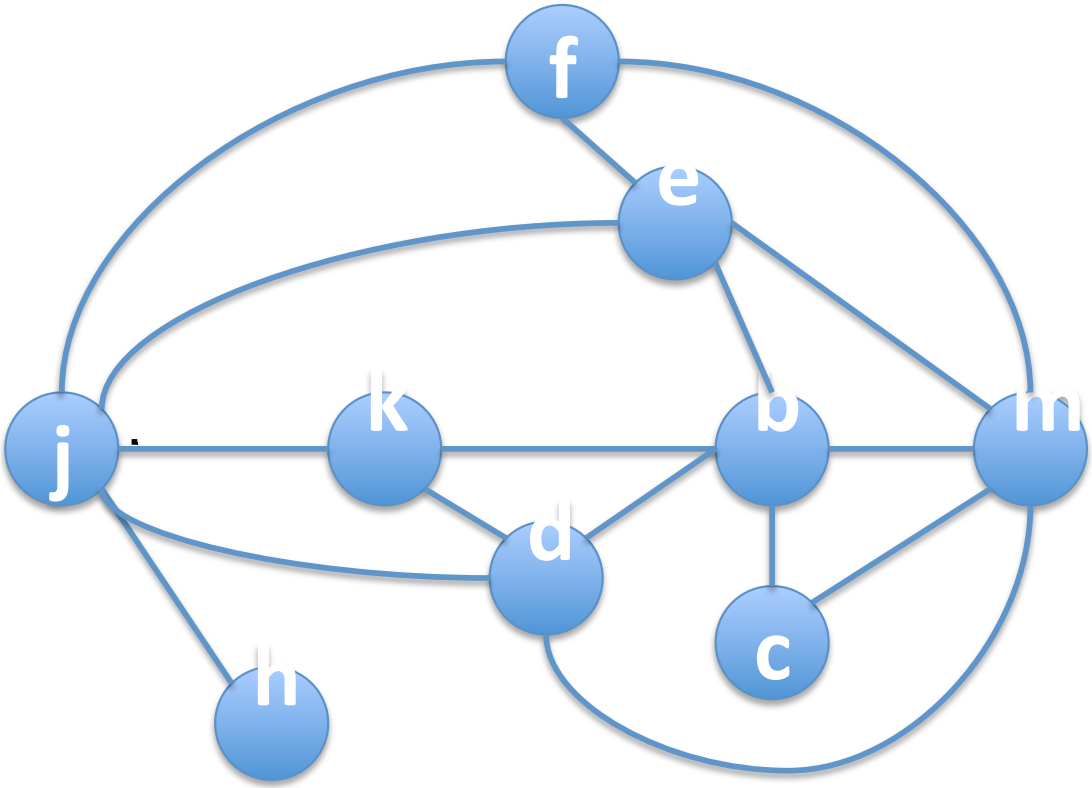


Simplify g

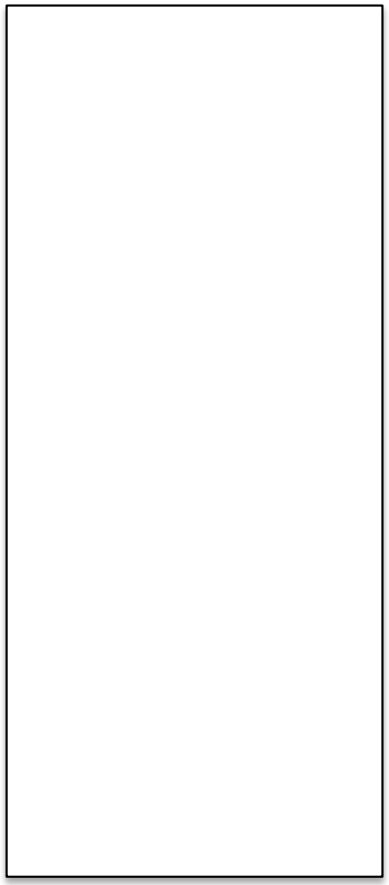


g

(a) stack

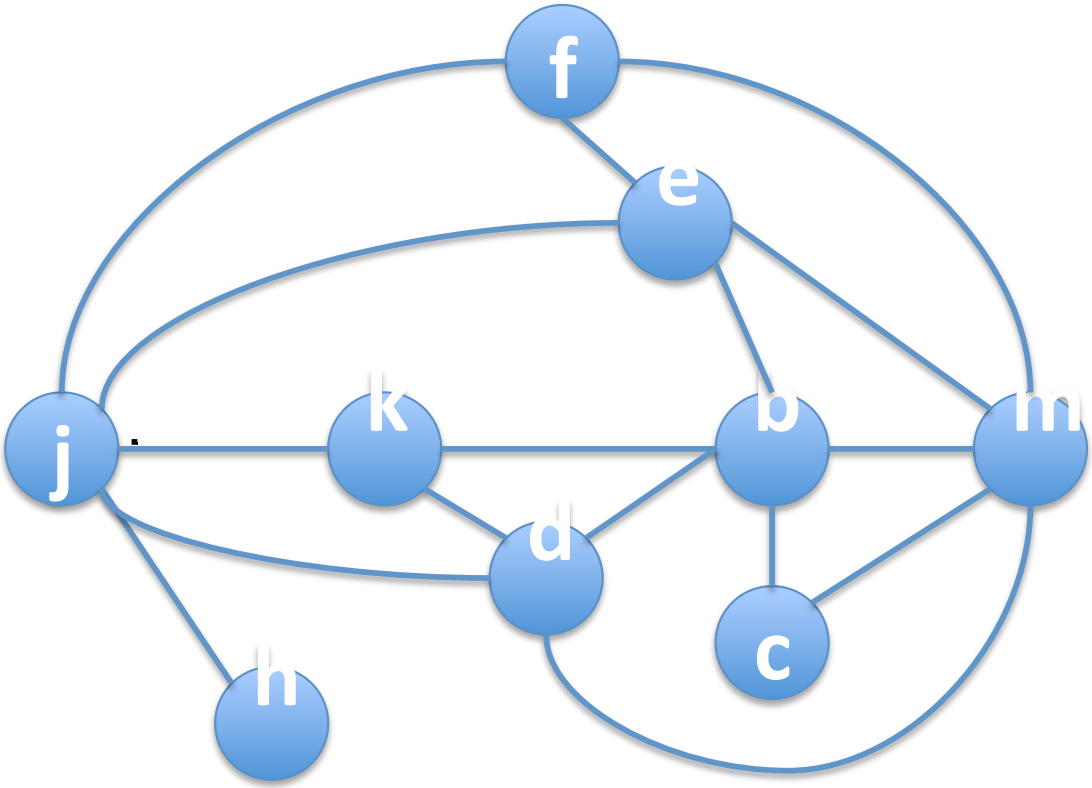


Simplify h

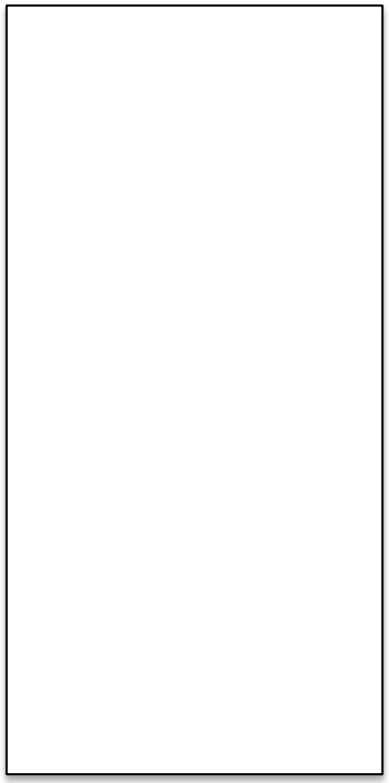


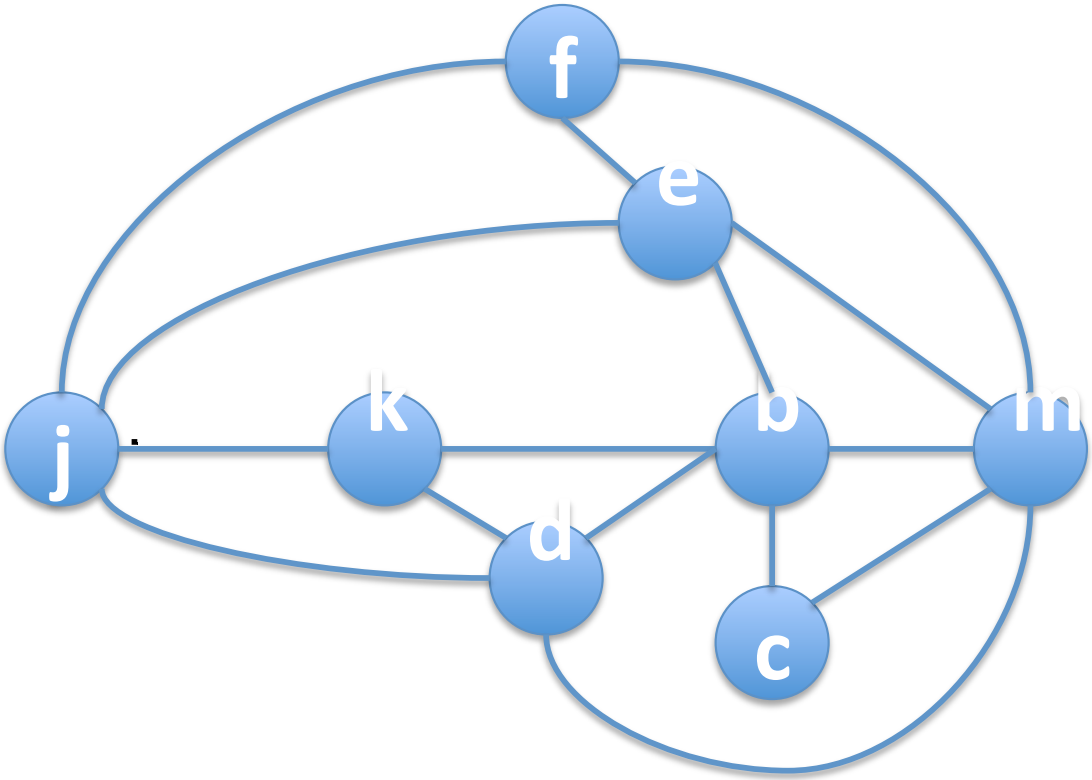
g

(a) stack

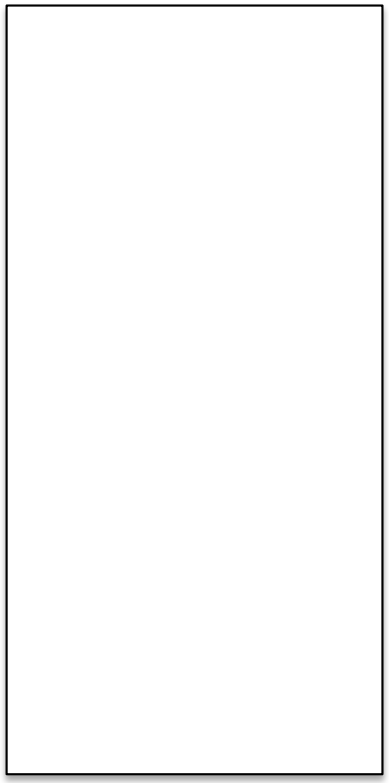


Simplify h

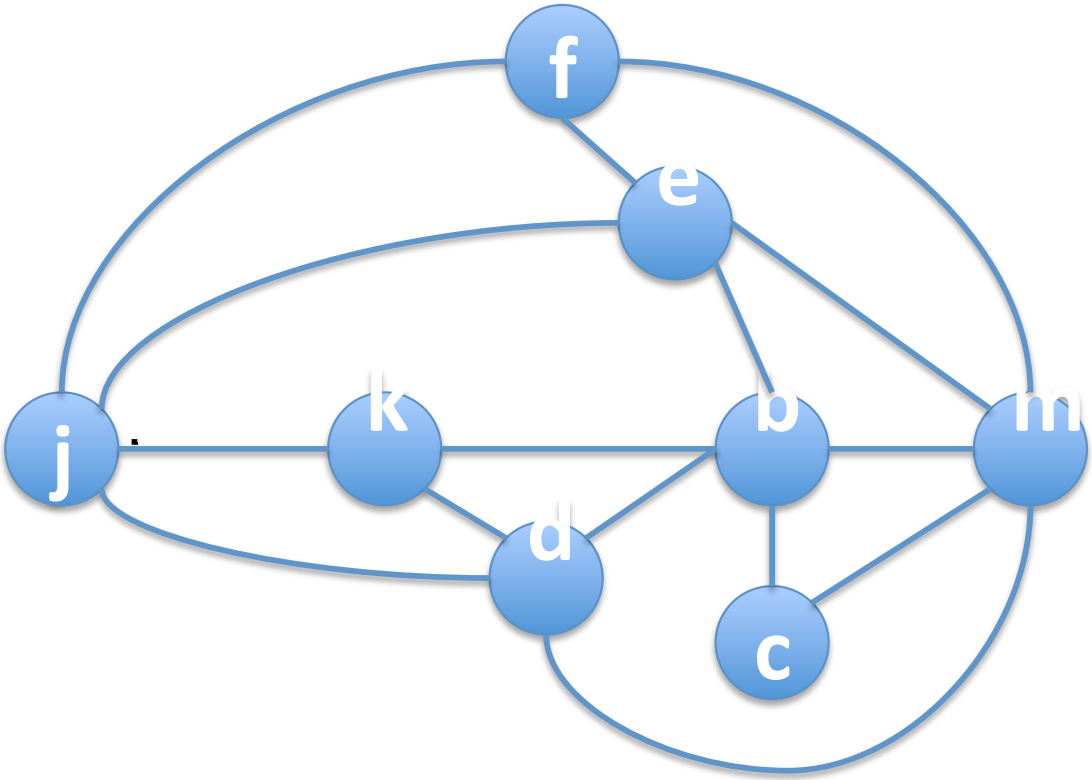




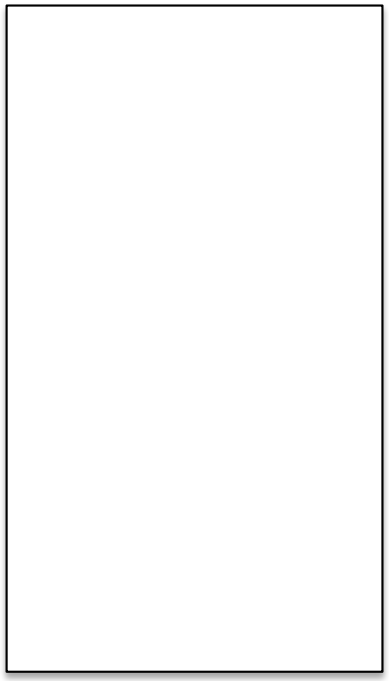
Simplify k



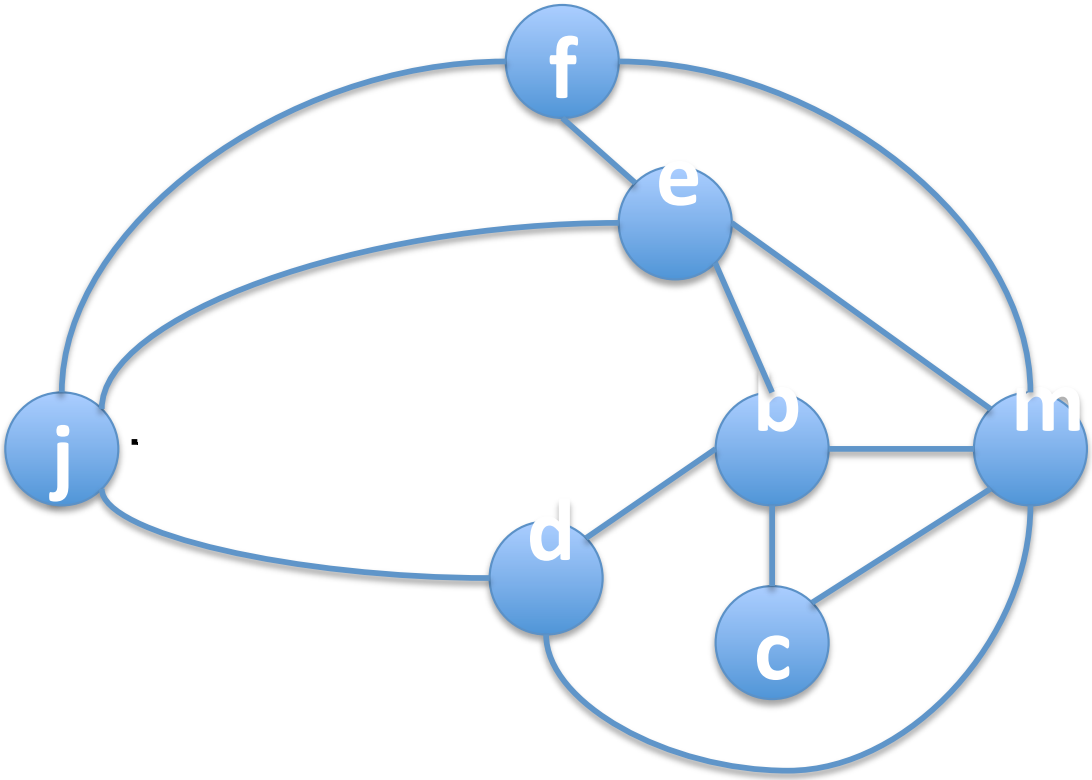
(a) stack



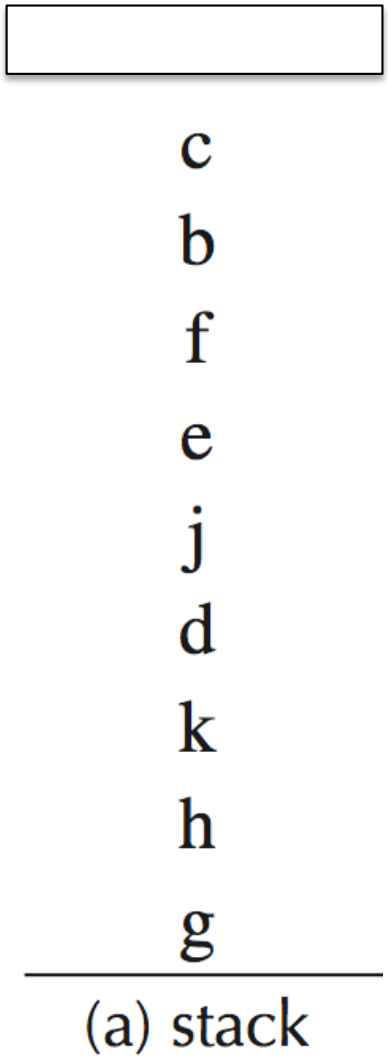
Simplify k



(a) stack

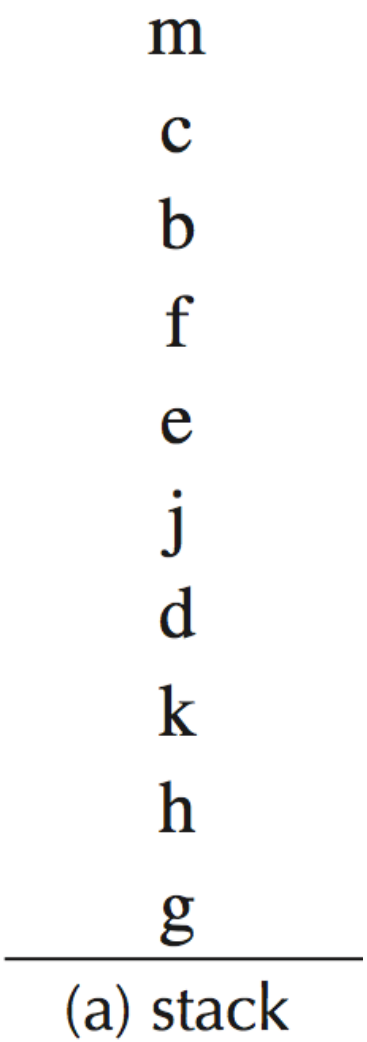


Continue Simplifying...



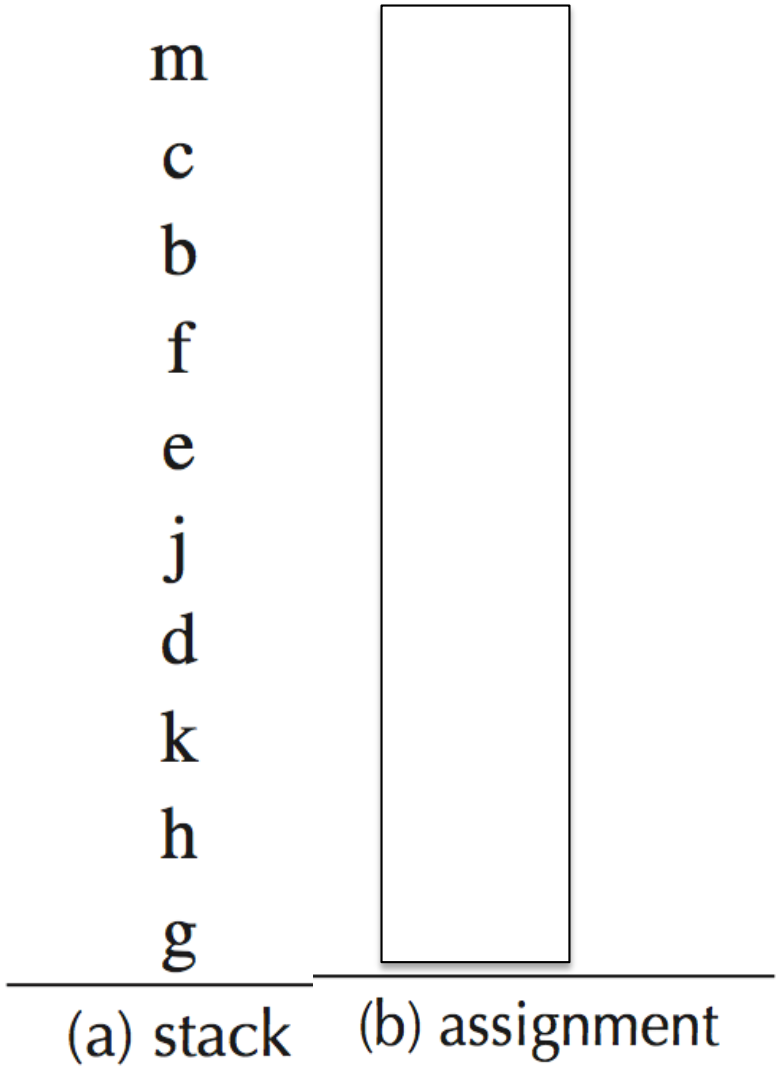
(a) stack

Continue Simplifying...

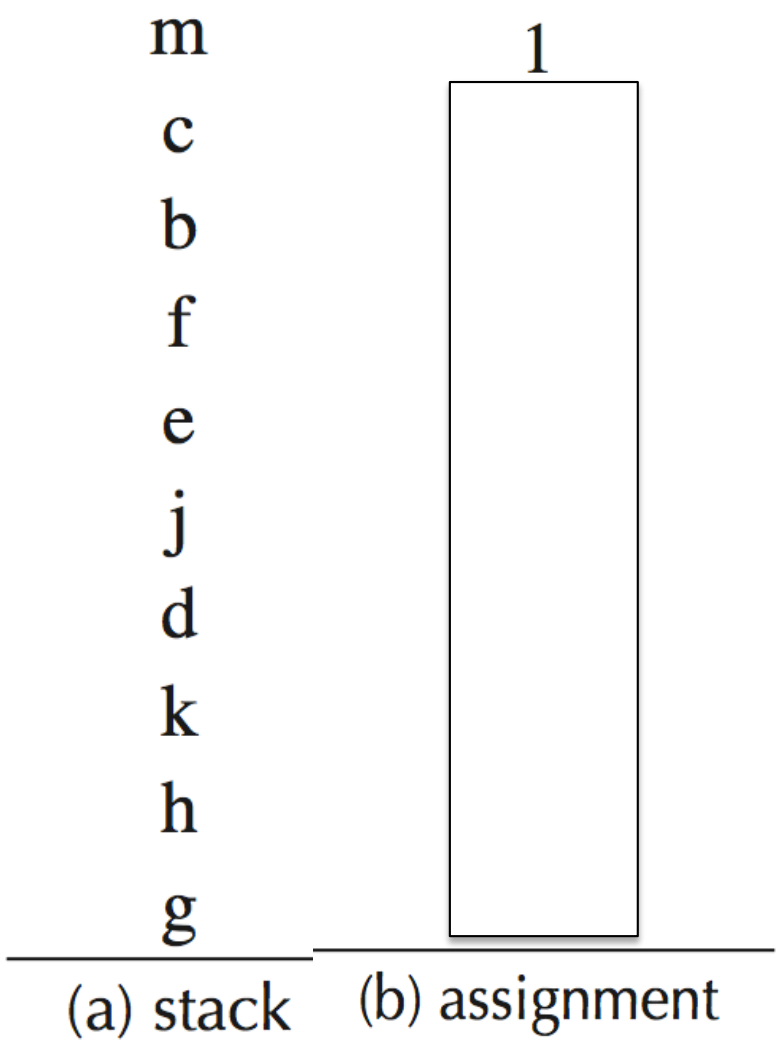


(a) stack

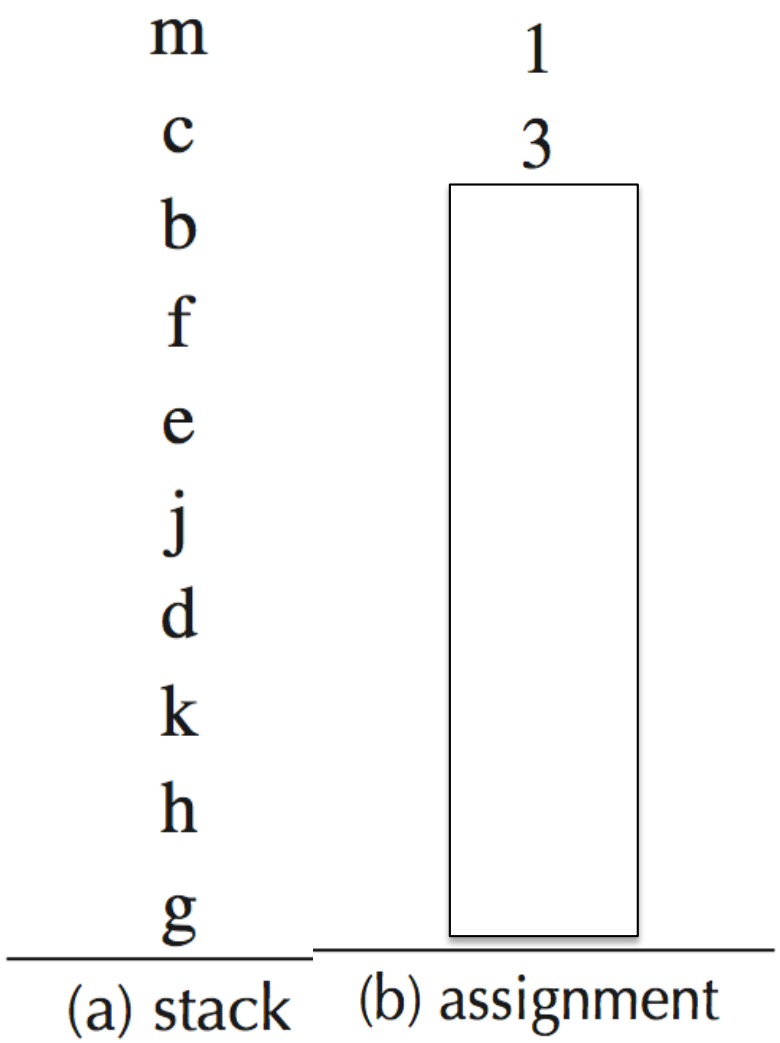
Select

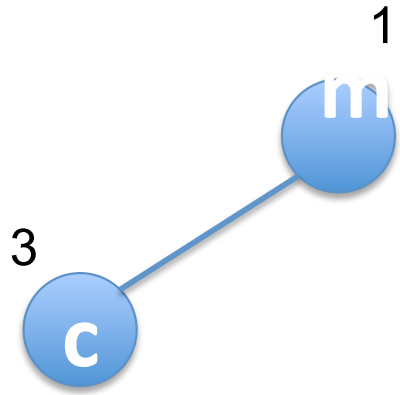


Select m

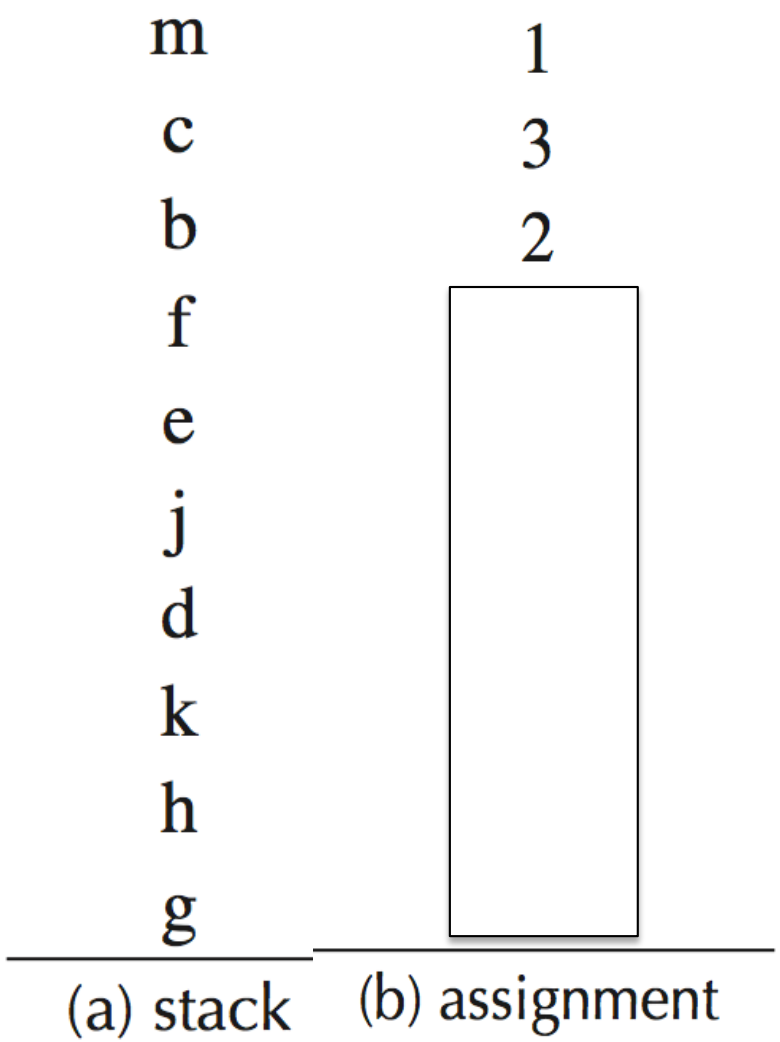


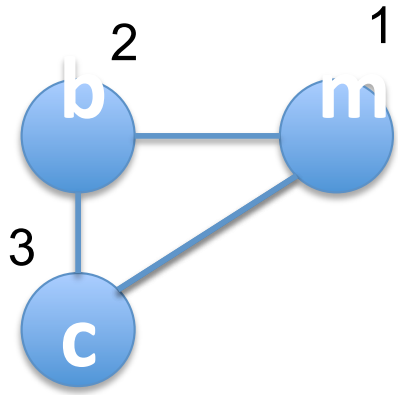
Select c



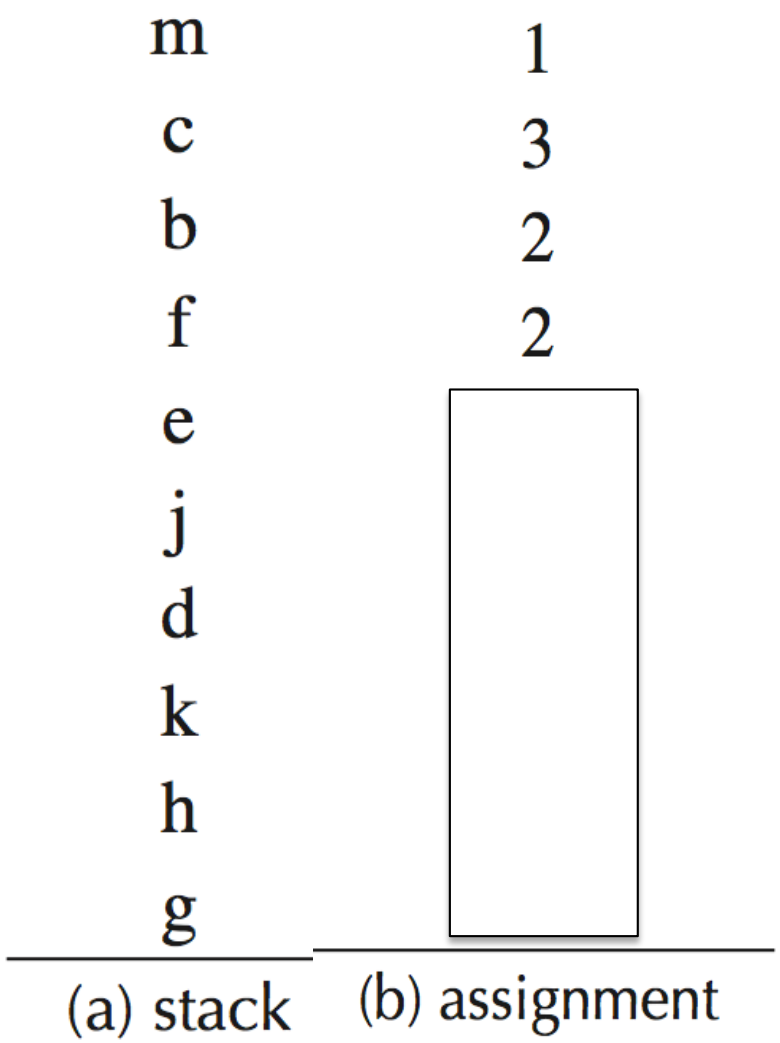


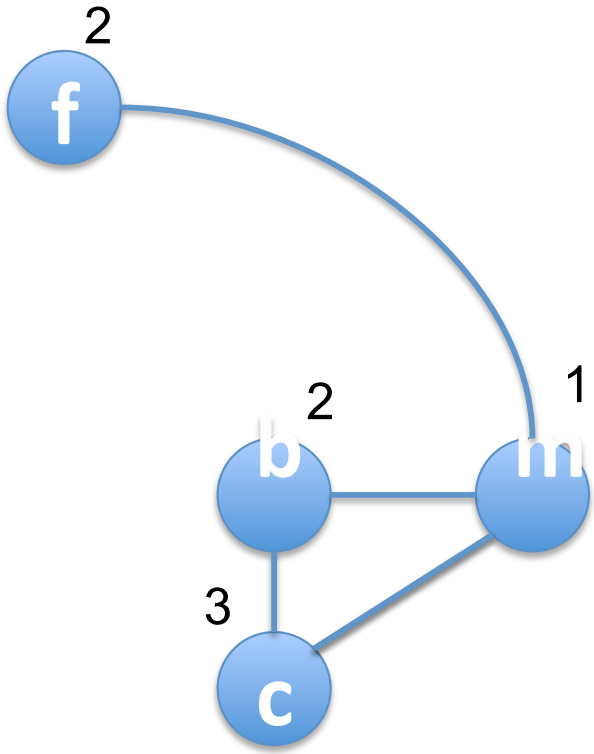
Select b





Select f





Continue selecting….

(a) stack	(b) assignment
m	1
c	3
b	2
f	2
e	4
j	3
d	4
k	1
h	2
g	4