Compiladores

JEAN LUCA BEZ

LUCAS MELLO SCHNORR

Geração de Código

INF01147

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

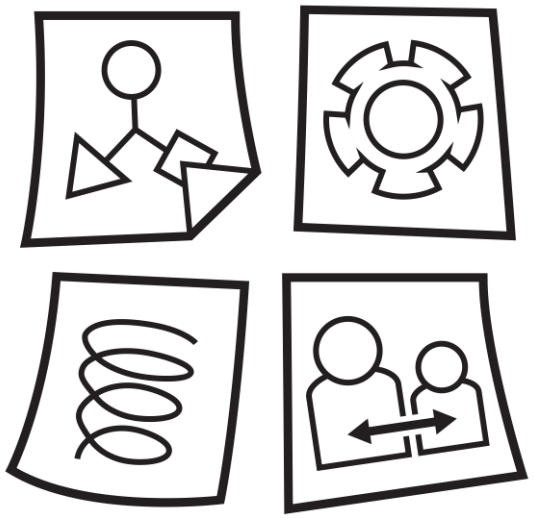
JEAN LUCA BEZ
LUCAS MELLO SCHNORR

- Última fase do compilador
- Recebe como entrada uma representação intermediária
- Produz um programa-alvo equivalente

- Requisitos:
  - Código deve preservar o **significado** semântico do fonte
  - Código deve ser de alta **qualidade**
  - Usar eficientemente os **recursos** disponíveis

Teoria vs. Prática

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

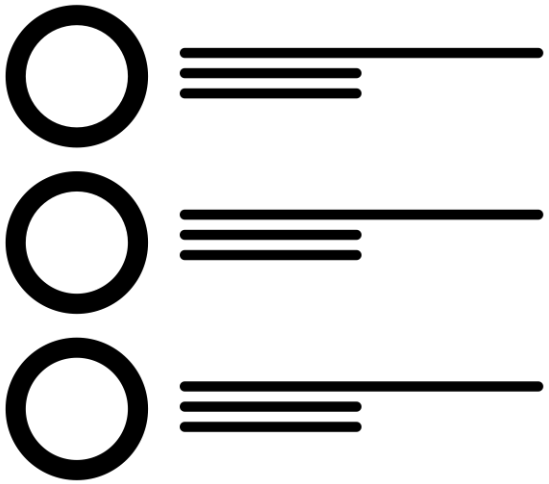


- Na prática gerar código objeto ótimo não pode ser solucionável
- Problemas como alocação de registradores são computacionalmente intratáveis
- Na prática usamos técnicas heurísticas
- Geram código bom, mas não necessariamente ótimo

- O **gerador de código** é composto por **três** tarefas principais:
  - Seleção de instrução
  - Alocação e atribuição de registrador
  - Escalonamento de instruções

Tarefas Principais

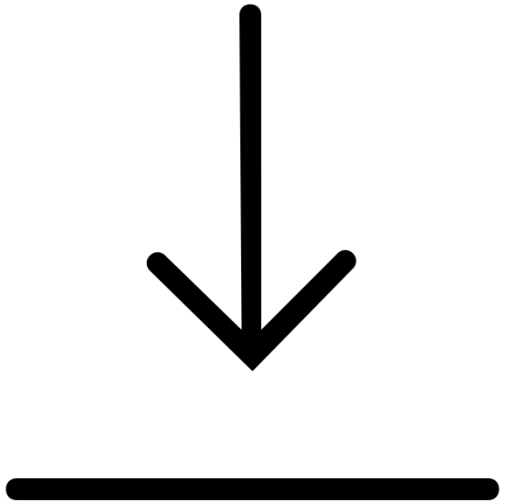
JEAN LUCA BEZ
LUCAS MELLO SCHNORR



- **Seleção de instrução**
  - Escolha das instruções apropriadas da arquitetura alvo
  - Representar os comando da representação intermediária (RI)
- **Alocação e distribuição de registrador**
  - Decide que valores devem ser mantidos em registradores
  - Quais registradores utilizar
- **Escalonamento de instruções**
  - Decisão sobre a ordem em que a execução das instruções deve ser escalonada

Entrada

JEAN LUCA BEZ
LUCAS MELLO SCHNORR



- Representação **intermediária** (RI) do fonte:
  - Três endereços
  - Máquina virtual (*bytecode* e códigos de máquina de pilha)
  - Lineares (notação posfixa)
  - Gráficas (árvores de sintaxe, DAGs)
- Todos erros sintáticos e semânticos foram detectados
- Verificação de tipos foi feita
- Operadores de conversão de tipos foram adicionados onde necessários
- Assume que a entrada está **livre** destes **erros**

Programas-Alvo

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

- Saída do gerador de código é o **programa alvo**
- Pode assumir diversas formas:
  - Linguagem **absoluta** de máquina (uso acadêmico)
    - Pode ser carregado em um posição fixa da memória
    - Executado imediatamente
  - Linguagem **relocável** de máquina (módulo objeto)
    - Permite que subprogramas sejam compilados separadamente
    - Custo adicional para ligação e carga dos módulos objeto relocáveis
    - Flexibilidade
  - Linguagem de **montagem**
    - Assembly
    - Processo de geração é bem mais fácil
    - Gera instruções simbólicas e utiliza Montador para auxiliar na geração o código
    - Etapa adicional (*assembly*) após a geração do código

Seleção de Instrução

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

- Gerador de código precisa mapear o programa da RI
- Sequência de código que possa ser executada pela arquitetura
- Tradução ingênua pode produzir código correto mas ineficiente

COMANDOS DE TRÊS ENDEREÇOS

```
// x = y + z

LD          R0, y                   // R0 = y          (carrega y)
ADD         R0, R0, z               // R0 = R0 + z     (soma z a R0)
ST          x, R0                   // x = R0          (armazena em x)
```

- Instrução de incremento (`INC`) ou operação `a=a+1`

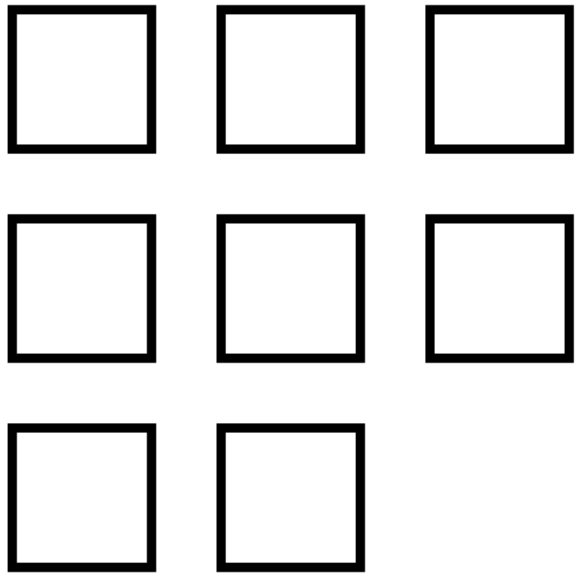
CARGAS E ARMAZENAMENTOS REDUNDANTES

```
// a = b + c
// d = a + e

LD          R0, b                   // R0 = b
ADD         R0, R0, c               // R0 = R0 + c
ST          a, R0                   // a = R0       (SE NÃO USA a)
LD          R0, a                   // R0 = a       (REDUDANTE)
ADD         R0, R0, e               // R0 = R0 + e
ST          d, R0                   // d = R0
```

Alocação de Registradores

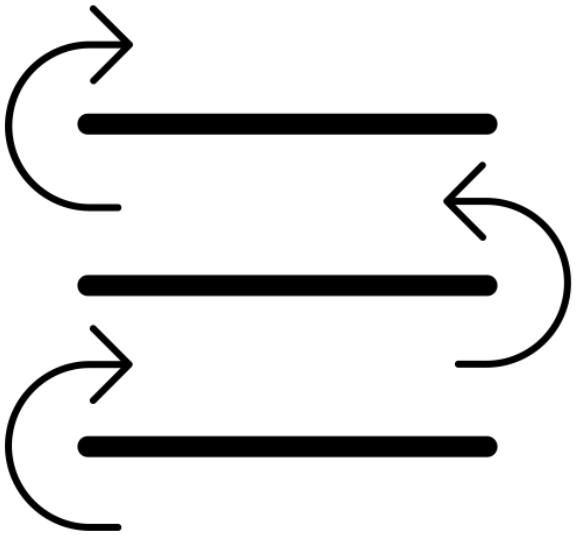
JEAN LUCA BEZ
LUCAS MELLO SCHNORR



- **Registrador**
  - Unidade computacional mais rápida da máquina alvo
  - Número limitado
- Devemos decidir:
  - **Alocação de Registradores:** quais variáveis ficarão em registradores
  - **Atribuição de Registradores:** associar um registrador a uma variável
- Atribuição ótima é **NP completo**
- Podem haver restrições no uso (dependendo da máquina)
  - Multiplicação
    - Registrador par recebe x e registrador ímpar recebe y, resultado ocupa a dupla
  - Divisão
    - Registrador par recebe x (dividendo), ímpar recebe y (divisor)
    - Resultado no registrador par, resto no registrador ímpar

Ordem de Avaliação

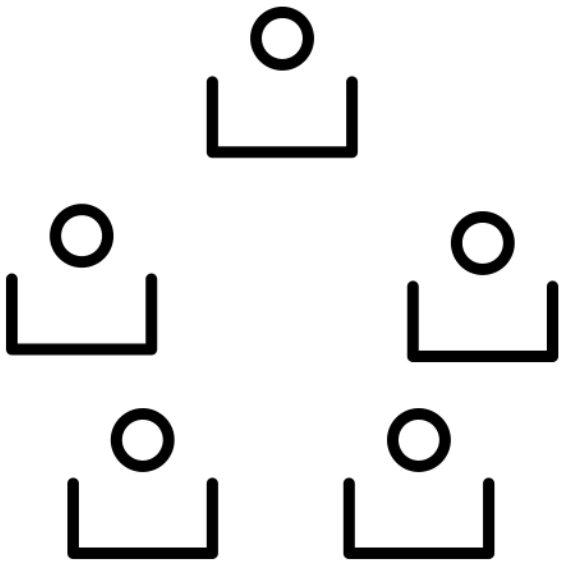
JEAN LUCA BEZ
LUCAS MELLO SCHNORR



- Ordem em que as computações são efetuadas podem afetar:
  - Eficiência do código
  - Quantidade de registradores necessários
- Encontrar a melhor ordem é **NP completo**
- O que fazer?
  - Evitamos o problema
  - Utilizamos a ordem produzida pelo gerado de código intermediário (RI)

Introdução

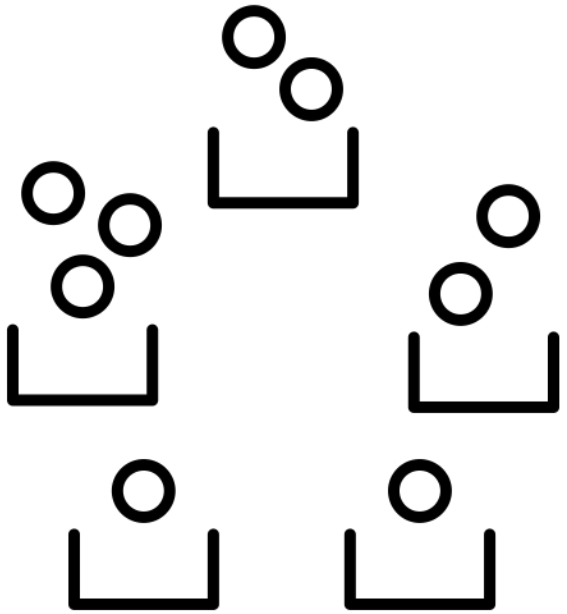
JEAN LUCA BEZ
LUCAS MELLO SCHNORR



- Operações com operandos em registradores é **rápida**
- Abordagem simples:
  - Atribuir valores específicos a certos registradores
  - Endereços de base (inicial a um RA) a um grupo de registradores
  - Cálculos aritméticos a outro grupo de registradores
  - Topo da pilha a um registrador fixo
  - ...
- **Vantagem**
  - Simplificaria o projeto do gerador de código
- **Desvantagem**
  - Uso ineficiente dos registradores
  - Alguns registradores podem ficar sem uso por grandes porções do código
  - Outros podem ter carga excessiva
- É usual manter um para os endereços base e topo da pilha

Introdução

JEAN LUCA BEZ
LUCAS MELLO SCHNORR



- Código intermediário utiliza número ilimitado de temporários
  - Simplifica a geração e otimização
  - Complica a tradução final para *assembly*
- **Problema:**
  - Reescrever o código intermediário para não utilizar mais valores temporários do que registradores disponíveis
- **Método:**
  - Atribuir múltiplas variáveis para cada registrador
  - Sem alterar o comportamento do programa

Introdução

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

- Considere o programa abaixo
- Precisaria de 6 registradores (`a`, `b`, `c`, `d`, `e`, `f`)?
- Considere que `a` e `e` não são usados mais a frente

PROGRAMA (VARIÁVEIS)

```
a := c + d
e := a + b
f := e - 1
```

REGISTRADORES UTILIZADOS

```
r1 := r3 + r4
r5 := r1 + r2
r6 := r5 - 1
```

Introdução

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

- Considere o programa abaixo
- Precisaria de 6 registradores (`a`, `b`, `c`, `d`, `e`, `f`)?
- Considere que `a` e `e` não são usados mais a frente

- Podemos alocar `a`, `e` e `f` para um único registrador
- *Many-to-one*

PROGRAMA (VARIÁVEIS)

```
a := c + d
e := a + b
f := e - 1
```

REGISTRADORES UTILIZADOS

```
r1 := r2 + r3
r1 := r1 + r4
r1 := r1 - 1
```

- Temporários $t1$ e $t2$ podem compartilhar um mesmo registrador se em qualquer ponto do programa no **máximo** um deles ($t1$ ou $t2$) estiver **vivo**
- i.e. Se $t1$ e $t2$ estão **vivos** ao **mesmo tempo**, eles **não** podem **compartilhar** um registrador

Vida e Morte de Variáveis

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

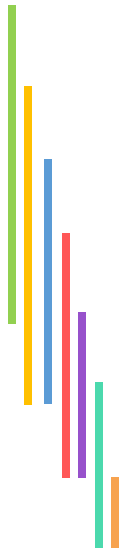

```
a: s1 = ld(x)
b: s2 = s1 + 4
c: s3 = s1 * 8
d: s4 = s1 - 4
e: s5 = s1/2
f: s6 = s2 * s3
g: s7 = s4 - s5
h: s8 = s6 * s7
```

- Variável v está **viva** entre:
  - p_i após a sua definição
  - p_j após seu último uso
- Intervalo de Vida $[p_i, p_j]$
- `s1` e `s2` tem intervalo de vida de 4 instruções
- Quantas variáveis estão vivas em `e` e `f`?

Análise de Vida

JEAN LUCA BEZ
LUCAS MELLO SCHNORR


```
a: s1 = ld(x)
b: s2 = s1 + 4
c: s3 = s1 * 8
d: s4 = s1 - 4
e: s5 = s1/2
f: s6 = s2 * s3
g: s7 = s4 - s5
h: s8 = s6 * s7
```



- Qual o número de registradores necessários?
- Calcular intervalos
- Encontrar o mais **largo**
- Quem tem o maior número de variáveis vivas simultaneamente?

Análise de Vida

JEAN LUCA BEZ
LUCAS MELLO SCHNORR


```
a: s1 = ld(x)
b: s2 = s1 + 4
c: s3 = s1 * 8
d: s4 = s1 - 4
e: s5 = s1/2
f: s6 = s2 * s3
g: s7 = s4 - s5
h: s8 = s6 * s7
```

- Qual o número de registradores necessários?
- Calcular intervalos
- Encontrar o mais **largo**
- Quem tem o maior número de variáveis vivas simultaneamente?

Análise de Vida

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

- **live-in(x):** conjunto de variáveis que estão vivas no ponto imediatamente anterior ao comando x
- **live-out(x):** conjunto de variáveis que estão vivas no ponto imediatamente após o comando x

```
a: s1 = ld(x)
b: s2 = s1 + 4
c: s3 = s1 * 8
d: s4 = s1 - 4
e: s5 = s1/2
f: s6 = s2 * s3
g: s7 = s4 - s5
h: s8 = s6 * s7
```

EXERCÍCIO 1

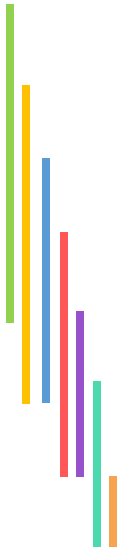
Qual é valor do live-in(e) e do live-out(e)?

Análise de Vida

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

- **live-in(x):** conjunto de variáveis que estão vivas no ponto imediatamente anterior ao comando x
- **live-out(x):** conjunto de variáveis que estão vivas no ponto imediatamente após o comando x

```
a: s1 = ld(x)
b: s2 = s1 + 4
c: s3 = s1 * 8
d: s4 = s1 - 4
e: s5 = s1/2
f: s6 = s2 * s3
g: s7 = s4 - s5
h: s8 = s6 * s7
```



EXERCÍCIO 1

Qual é valor do live-in(e) e do live-out(e)?

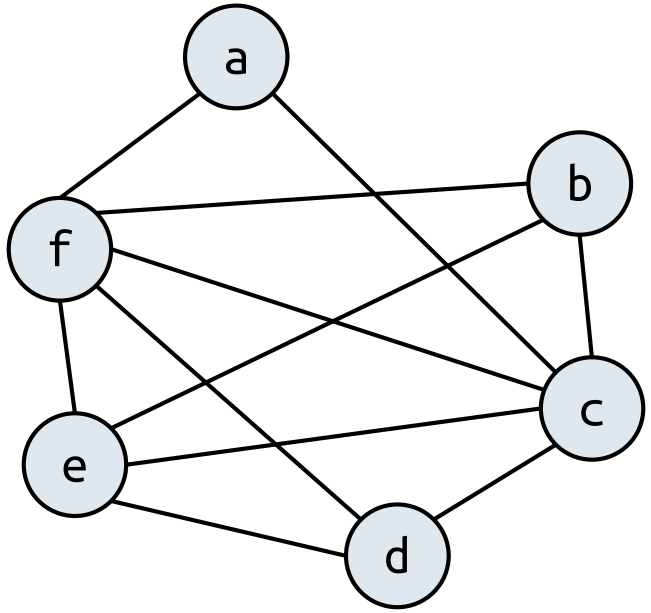
RESPOSTA

$$live_{in}(e) = \{s1, s2, s3, s4\}$$
$$live_{out}(e) = \{s2, s3, s4, s5\}$$

Grafo de Interferência de Registradores (GIR)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

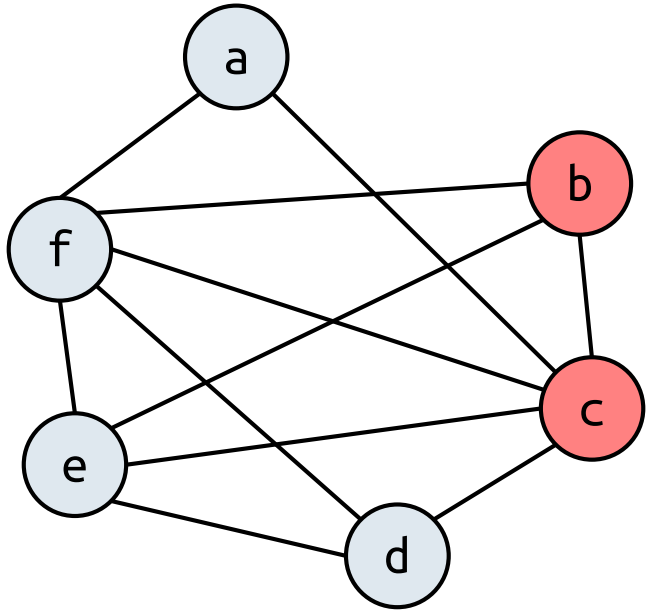
- Grafo não direcionado
- Resume a **análise de vida** das variáveis
- Cada **nó** representa uma **variável**
  - Candidata a ser alocada em um registrador
- Um **aresta** interconecta duas variáveis v_1 e v_2
  - Se existe instrução onde ambas estão vivas
  - v_1 e v_2 interferem uma na outra



Grafo de Interferência de Registradores (GIR)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

- Grafo não direcionado
- Resume a **análise de vida** das variáveis
- Cada **nó** representa uma **variável**
  - Candidata a ser alocada em um registrador
- Um **aresta** interconecta duas variáveis v_1 e v_2
  - Se existe instrução onde ambas estão vivas
  - v_1 e v_2 interferem uma na outra

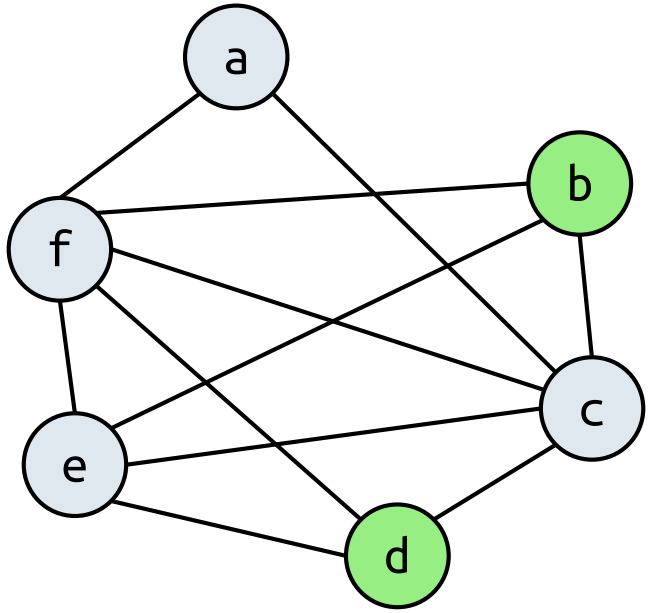


b e c **não podem** estar em um mesmo registrador

Grafo de Interferência de Registradores (GIR)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

- Grafo não direcionado
- Resume a **análise de vida** das variáveis
- Cada **nó** representa uma **variável**
  - Candidata a ser alocada em um registrador
- Um **aresta** interconecta duas variáveis v_1 e v_2
  - Se existe instrução onde ambas estão vivas
  - v_1 e v_2 interferem uma na outra

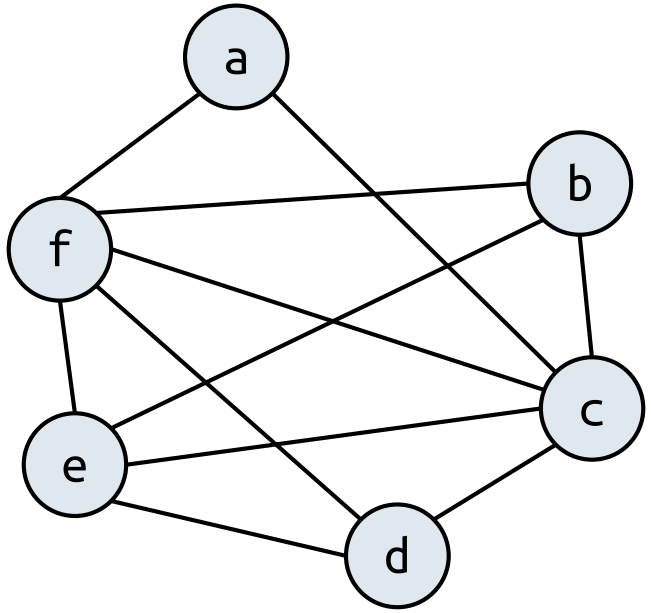


b e c **podem** estar em um mesmo registrador

Grafo de Interferência de Registradores (GIR)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

- Informação necessária para caracterizar uma distribuição válida de registradores
- Noção geral da alocação de registradores
- Leva em consideração todo o grafo de fluxo
- Após a construção do GIR
  - Alocação é independente da arquitetura
  - Depende somente do número de registradores



Atribuição por Coloração de Grafos

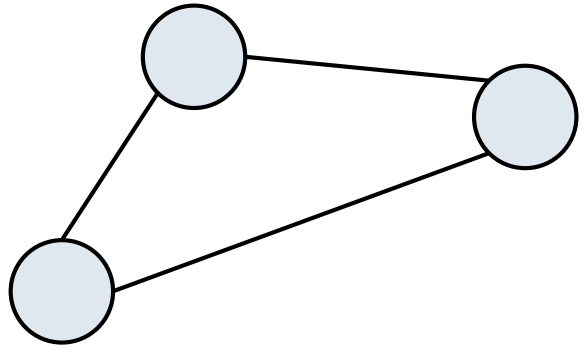
JEAN LUCA BEZ
LUCAS MELLO SCHNORR

Coloração de Grafos

- Atribuir cores aos nós
- Nós conectados por uma aresta tenham cores diferentes
- Um grafo é dito k-colorível se é possível colorir com k cores

Atribuição de Registradores

- Cores equivalem a registradores
- Precisamos atribuir cores (registradores) aos nodos (variáveis)
- $k = número\ de\ registradores$
- Se um GIR é k-colorível, existe distribuição de registradores que não usa mais que k registradores

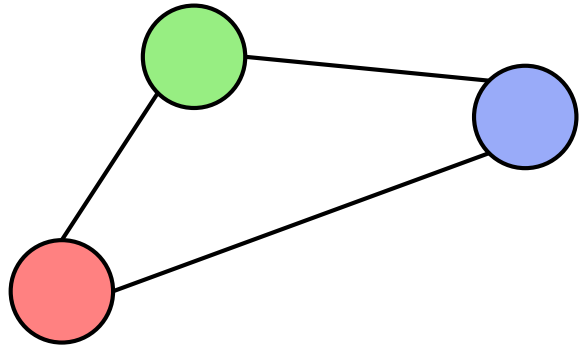


Atribuição por Coloração de Grafos

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

Coloração de Grafos

- Atribuir cores aos nós
- Nós conectados por uma aresta tenham cores diferentes
- Um grafo é dito k-colorível se é possível colorir com k cores



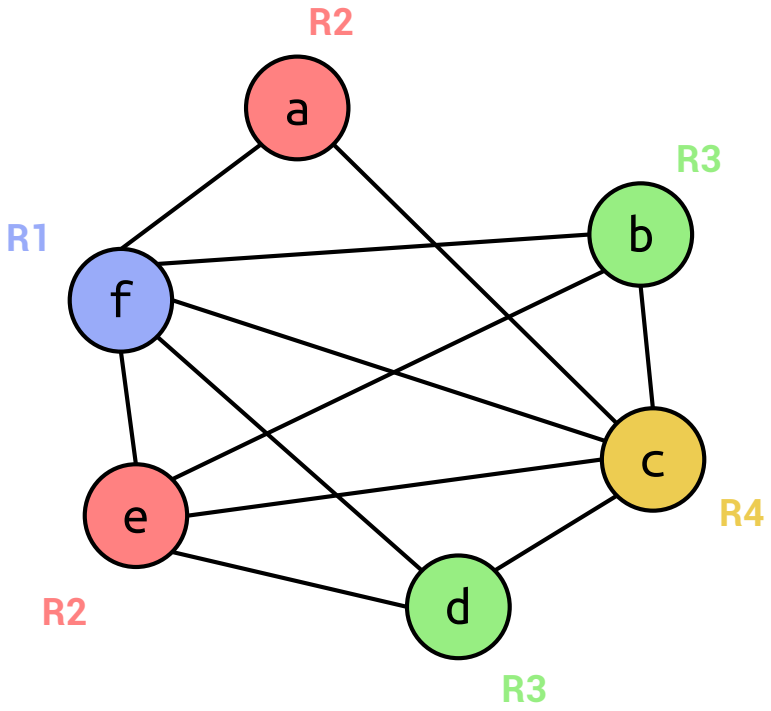
Atribuição de Registradores

- Cores equivalem a registradores
- Precisamos atribuir cores (registradores) aos nodos (variáveis)
- $k = número\ de\ registradores$
- Se um GIR é k-colorível, existe distribuição de registradores que não usa mais que k registradores

Atribuição por Coloração de Grafos

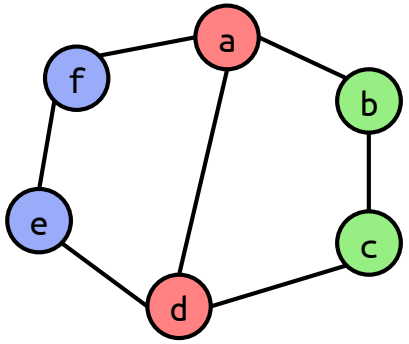
JEAN LUCA BEZ
LUCAS MELLO SCHNORR

- Exemplo de GIR
- Há mais que 4 variáveis
- Não existe coloração com menos de 4 cores
- Necessário 4 registradores (4 cores)

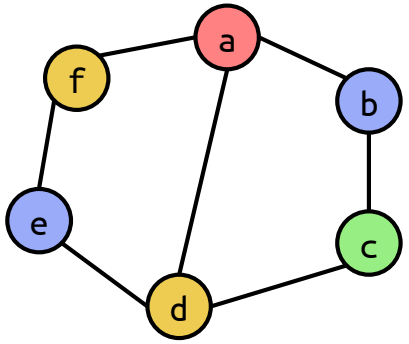


EXERCÍCIO 2

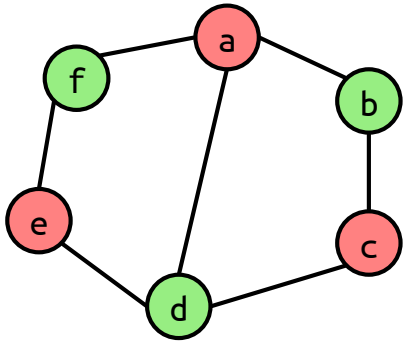
Qual das colorações abaixo representa uma coloração mínima válida para o GIR?



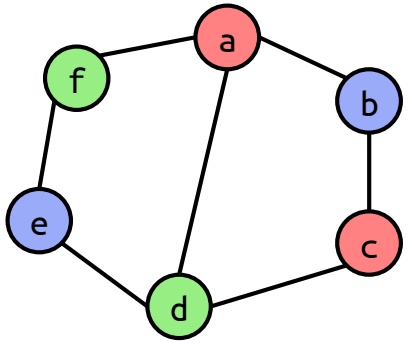
OPÇÃO A



OPÇÃO B



OPÇÃO C

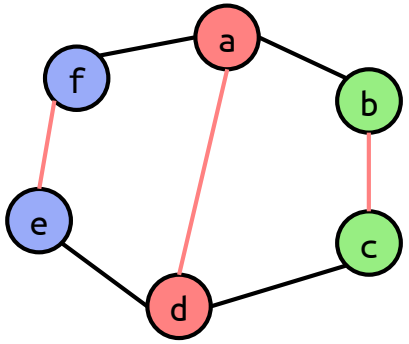


OPÇÃO D

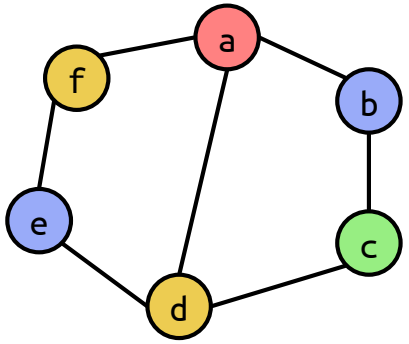
JEAN LUCA BEZ
LUCAS MELLO SCHNORR

EXERCÍCIO 2

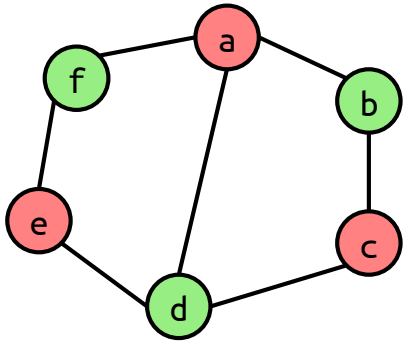
Qual das colorações abaixo representa uma coloração mínima válida para o GIR?



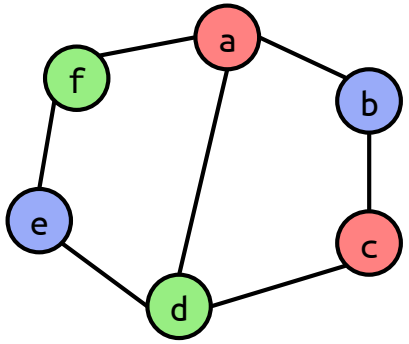
OPÇÃO A



OPÇÃO B



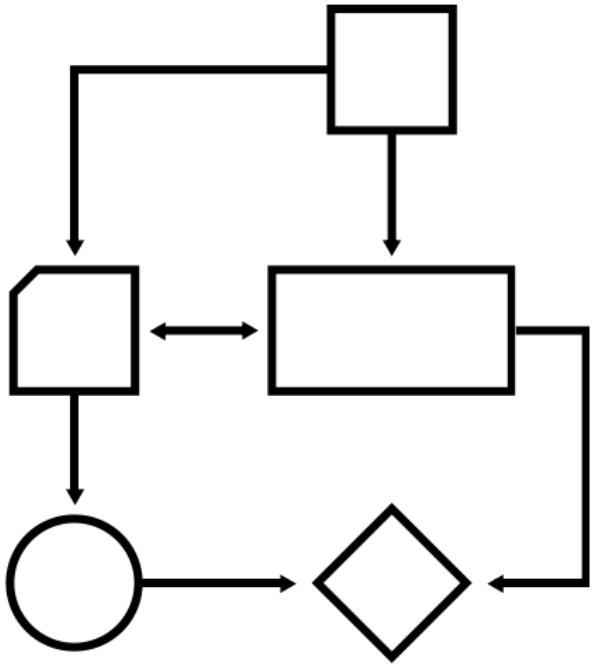
OPÇÃO C



OPÇÃO D

Como Computar a Coloração

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

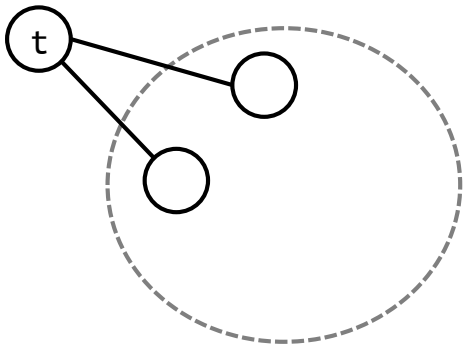


- Como computar a coloração de um grafo?
- Problema é difícil (**NP-hard**), não se conhece nenhum algoritmo eficiente
  - **Solução:** utilizar heurísticas
- Um coloração pode não existir para um número k de registradores
  - **Solução:** (vamos ver na sequência)

Heurística

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

- Heurística mais popular para coloração de grafos
- **Ideia**:
  - Selecionar um (qualquer) nó t com menos que k vizinhos no GIR
  - Eliminar t e suas arestas
  - Se o grafo resultante for k-colorível, então o grafo original também é

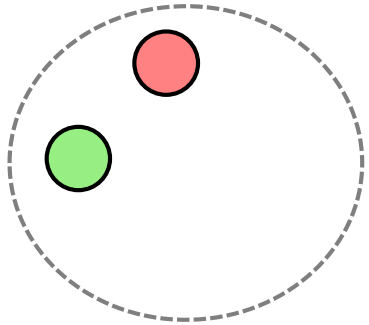


- **Porque funciona?**
  - Seja $c_1, \ldots, c_n$ as cores atribuídas ao vizinhos de t no grafo reduzido
  - Como $n < k$ nós podemos selecionar uma cor para t que seja diferente das dos seus vizinhos

Heurística

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

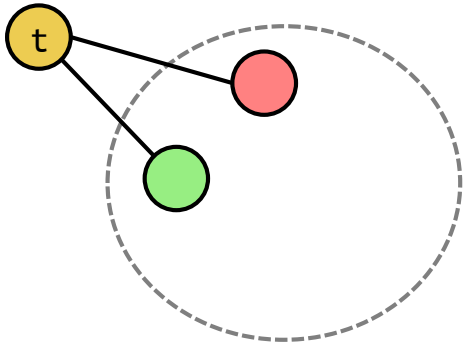
- Heurística mais popular para coloração de grafos
- **Ideia**:
  - Selecionar um (qualquer) nó t com menos que k vizinhos no GIR
  - Eliminar t e suas arestas
  - Se o grafo resultante for k-colorível, então o grafo original também é
- **Porque funciona?**
  - Seja $c_1, \ldots, c_n$ as cores atribuídas ao vizinhos de t no grafo reduzido
  - Como $n < k$ nós podemos selecionar uma cor para t que seja diferente das dos seus vizinhos



Heurística

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

- Heurística mais popular para coloração de grafos
- **Ideia**:
  - Selecionar um (qualquer) nó t com menos que k vizinhos no GIR
  - Eliminar t e suas arestas
  - Se o grafo resultante for k-colorível, então o grafo original também é
- **Porque funciona?**
  - Seja $c_1, \ldots, c_n$ as cores atribuídas ao vizinhos de t no grafo reduzido
  - Como $n < k$ nós podemos selecionar uma cor para t que seja diferente das dos seus vizinhos



Heurística

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

- Heurística divida em dois passos

PASSO 1

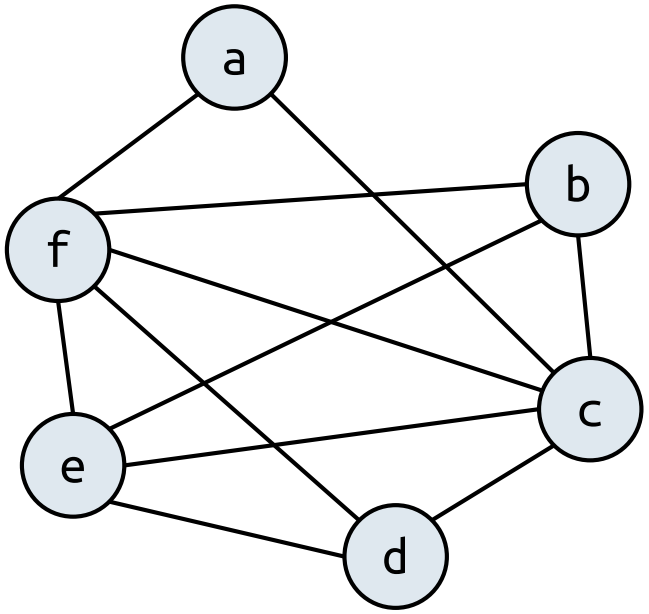
- Selecionar um (qualquer) nó t com menos que k vizinhos
- Colocar t em uma pilha e remover t do GIR
- Repetir até que o grafo esteja vazio

PASSO 2

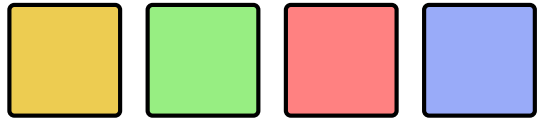
- Atribuir cores ao nós que estão na pilha
- Começamos com o último nó adicionado
- A cada passo selecionamos uma cor diferente das que foram atribuídas ao vizinhos já coloridos

Exemplo (Heurística)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR



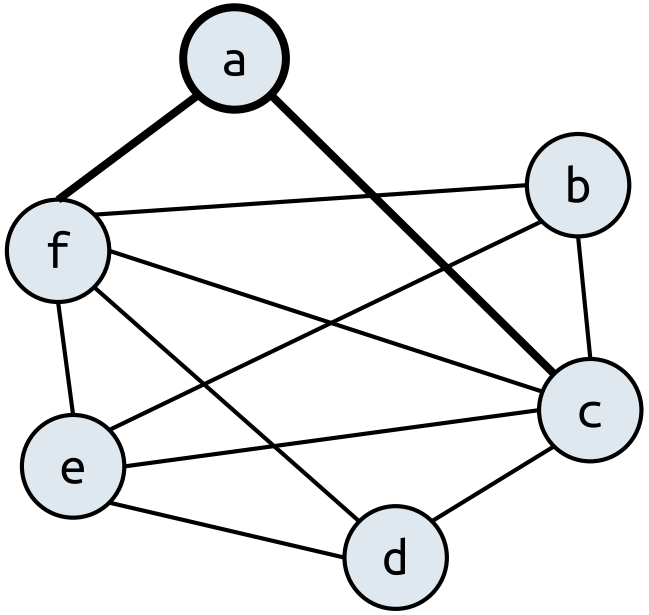
NÚMERO DE REGISTRADORES (K)



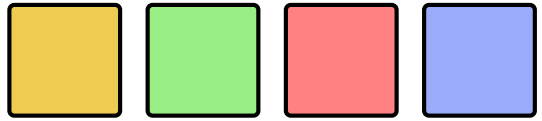
PILHA

Exemplo (Heurística)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR



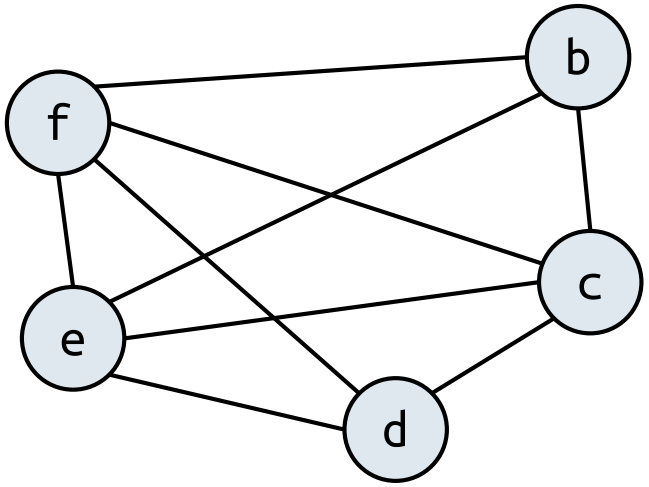
NÚMERO DE REGISTRADORES (K)



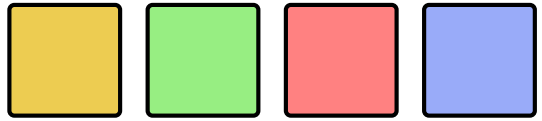
PILHA

Exemplo (Heurística)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR



NÚMERO DE REGISTRADORES (K)

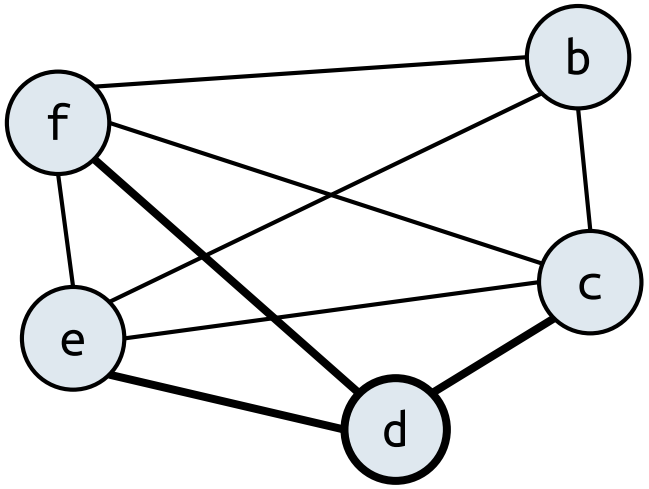


PILHA

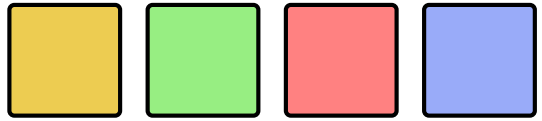
a

Exemplo (Heurística)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR



NÚMERO DE REGISTRADORES (K)

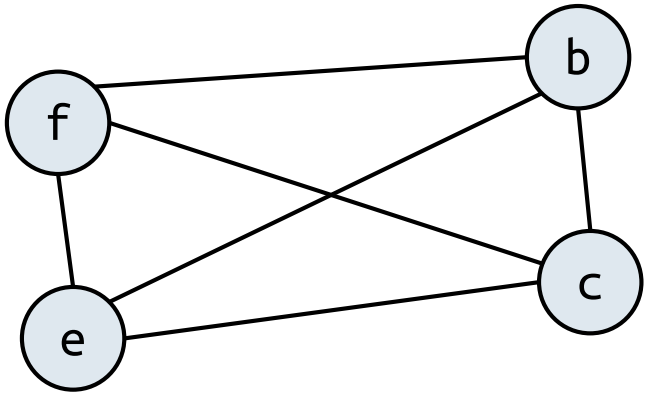


PILHA

a

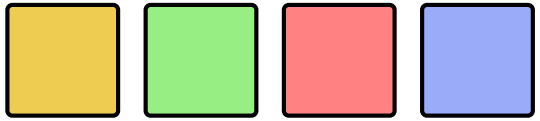
Exemplo (Heurística)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR



Todos os nós agora tem menos que 4 vizinhos

NÚMERO DE REGISTRADORES (K)

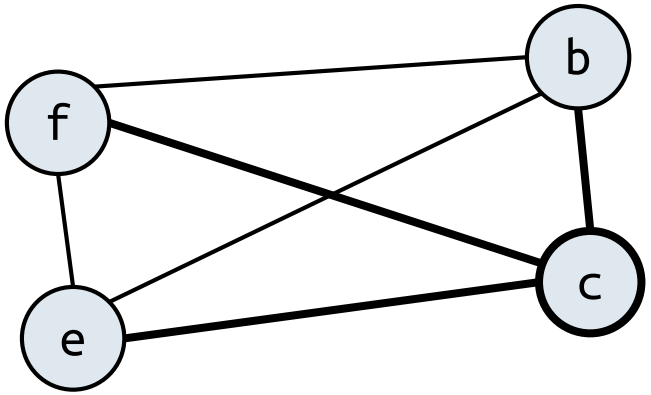


PILHA

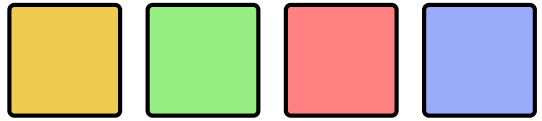
d
a

Exemplo (Heurística)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR



NÚMERO DE REGISTRADORES (K)

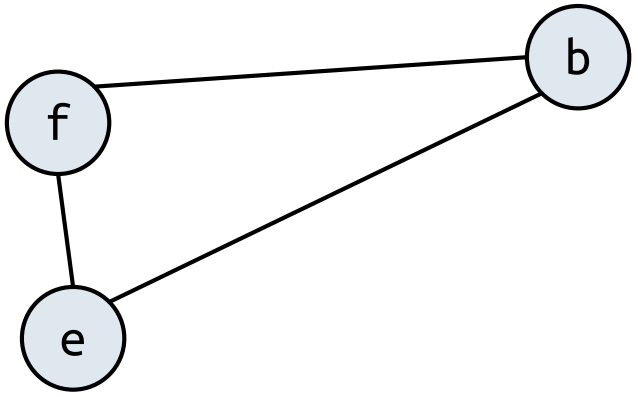


PILHA

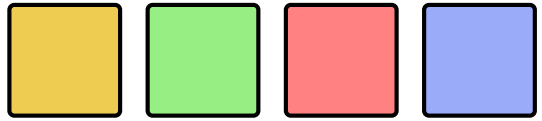
d
a

Exemplo (Heurística)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR



NÚMERO DE REGISTRADORES (K)

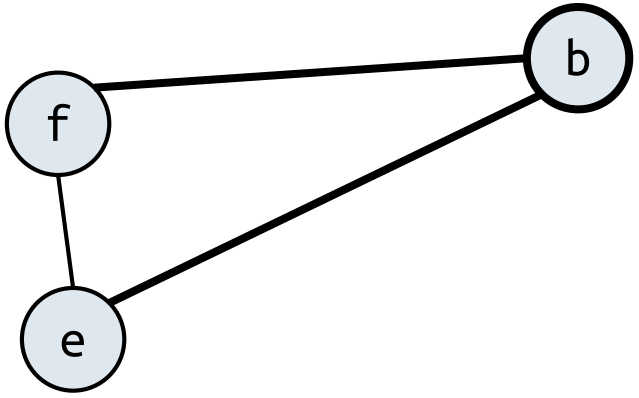


PILHA

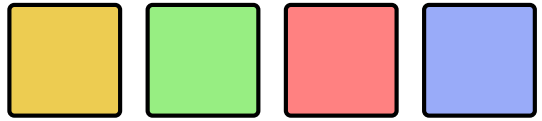
c
d
a

Exemplo (Heurística)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR



NÚMERO DE REGISTRADORES (K)

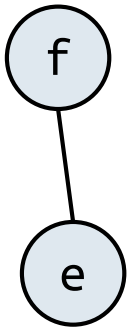


PILHA

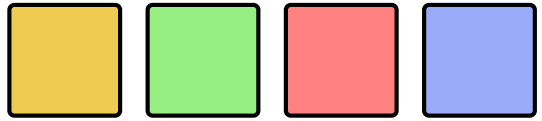
c
d
a

Exemplo (Heurística)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR



NÚMERO DE REGISTRADORES (K)

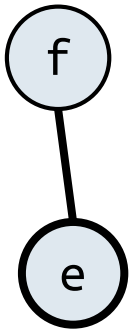


PILHA

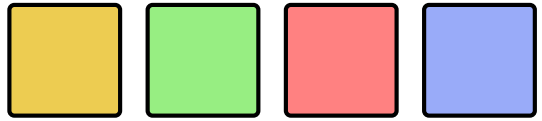
b
c
d
a

Exemplo (Heurística)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR



NÚMERO DE REGISTRADORES (K)



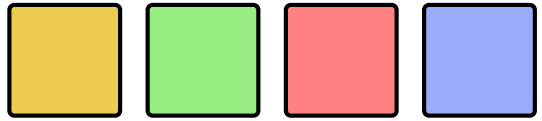
PILHA

b
c
d
a

Exemplo (Heurística)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

NÚMERO DE REGISTRADORES (K)



PILHA

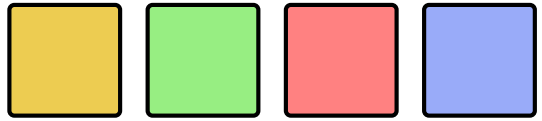
f

e
b
c
d
a

Exemplo (Heurística)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

NÚMERO DE REGISTRADORES (K)



PILHA

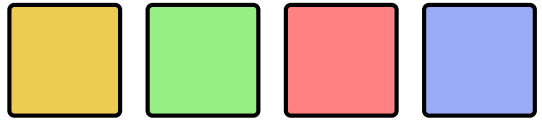
f

e
b
c
d
a

Exemplo (Heurística)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

NÚMERO DE REGISTRADORES (K)



PILHA

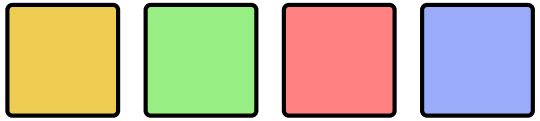
f
e
b
c
d
a

Determinamos a ordem que devemos colorir os nós do GIR

Exemplo (Heurística)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

NÚMERO DE REGISTRADORES (K)



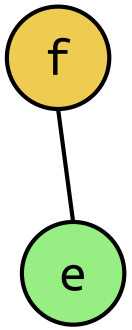
PILHA

f

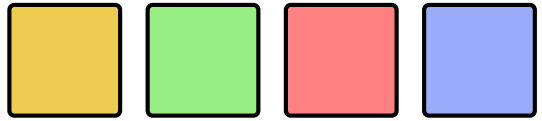
e
b
c
d
a

Exemplo (Heurística)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR



NÚMERO DE REGISTRADORES (K)

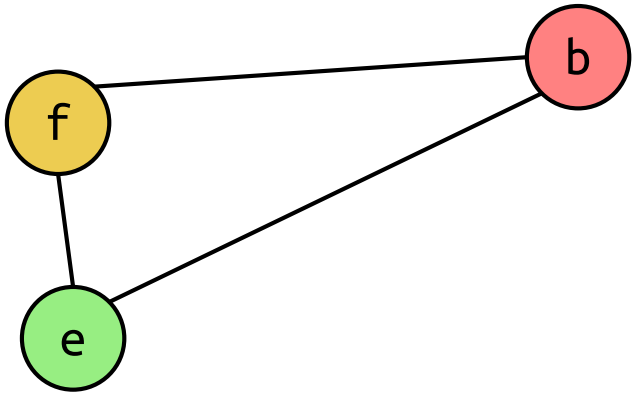


PILHA

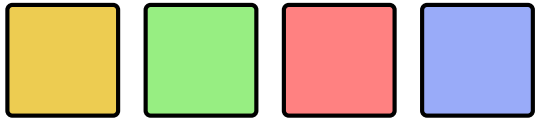
b
c
d
a

Exemplo (Heurística)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR



NÚMERO DE REGISTRADORES (K)

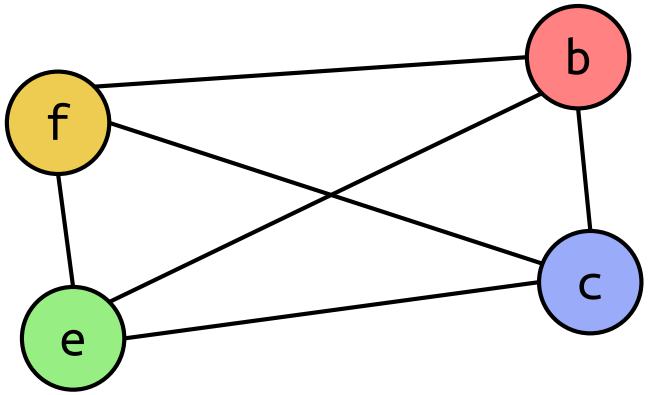


PILHA

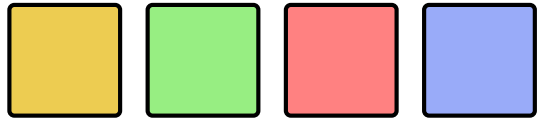
c
d
a

Exemplo (Heurística)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR



NÚMERO DE REGISTRADORES (K)

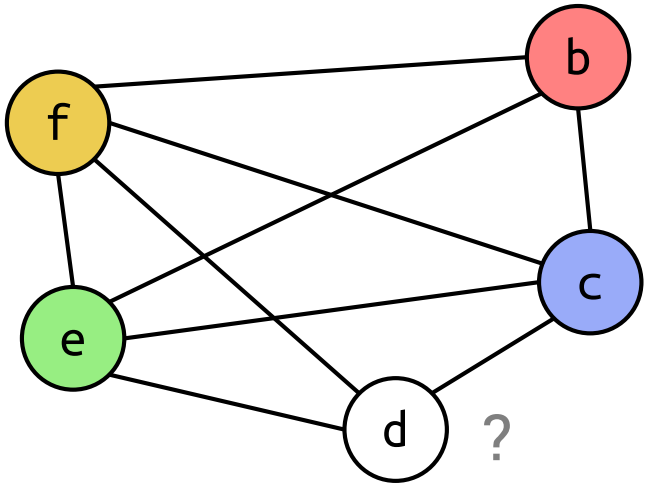


PILHA

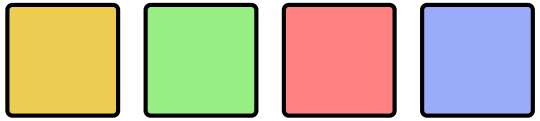
d
a

Exemplo (Heurística)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR



NÚMERO DE REGISTRADORES (K)

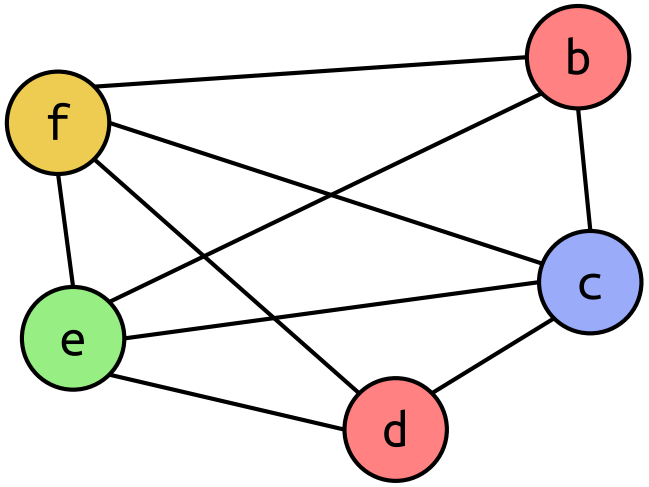


PILHA

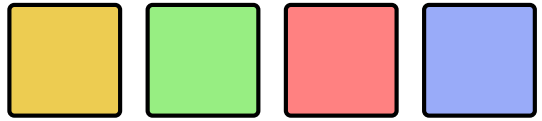
a

Exemplo (Heurística)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR



NÚMERO DE REGISTRADORES (K)

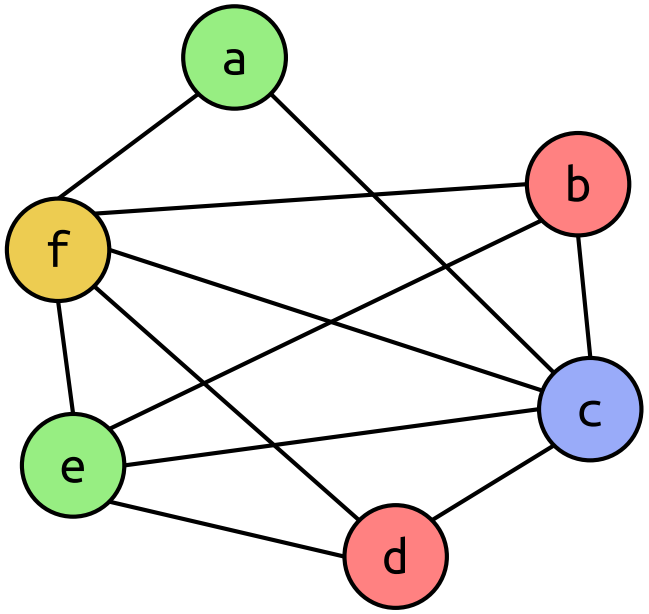


PILHA

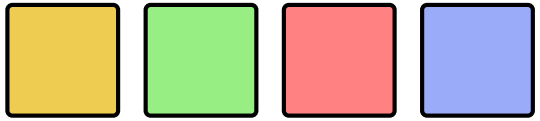
a

Exemplo (Heurística)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR



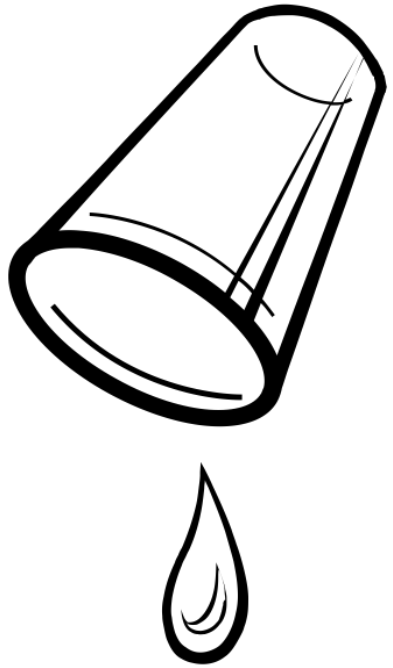
NÚMERO DE REGISTRADORES (K)



PILHA

Derramamento (Spilling)

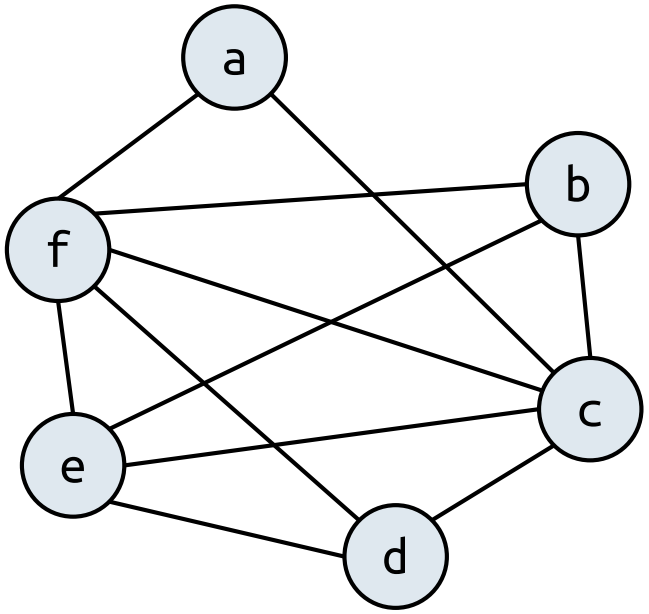
JEAN LUCA BEZ
LUCAS MELLO SCHNORR



- O que acontece se a heurística **falhar** em encontrar uma colocação?
- Não podemos manter todos os valores em registradores
- Alguns valores precisam ser “derramados”, i.e. armazenado em **memória**

Exemplo (Spilling)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

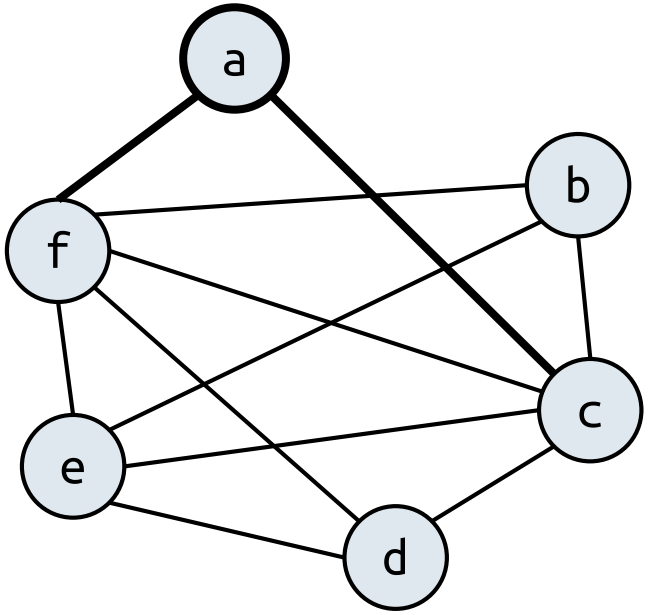


NÚMERO DE REGISTRADORES (K)

PILHA

Exemplo (Spilling)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

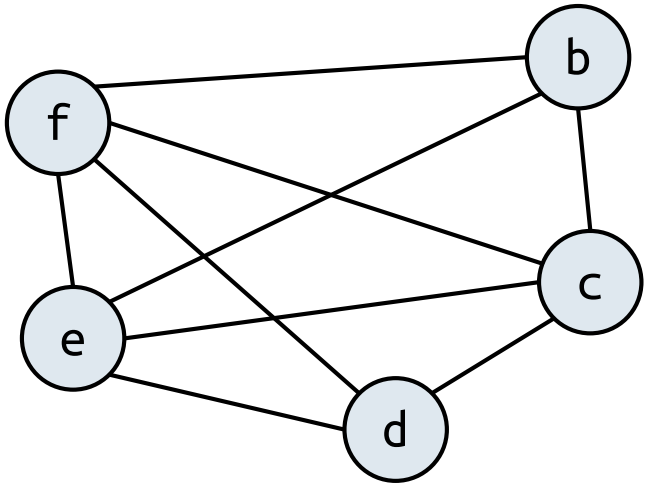


NÚMERO DE REGISTRADORES (K)

PILHA

Exemplo (Spilling)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR



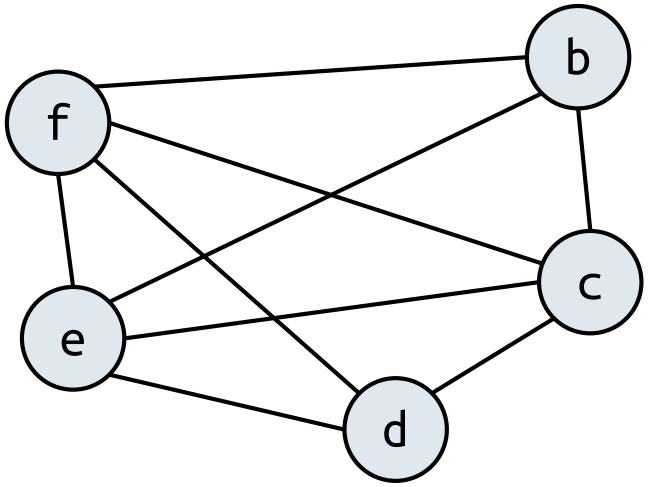
NÚMERO DE REGISTRADORES (K)

PILHA

a

Exemplo (Spilling)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR



NÚMERO DE REGISTRADORES (K)

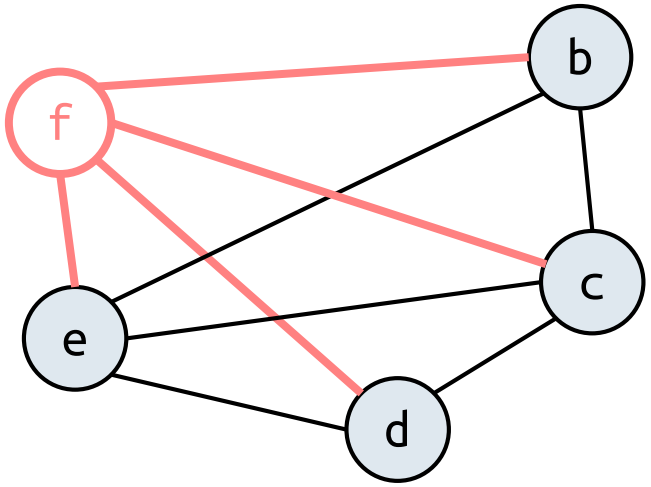
PILHA

- Não há o que fazer, todos os nós tem ≥ 3 (k) vizinhos
- Temos que selecionar um nó para *spilling*

a

Exemplo (Spilling)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR



NÚMERO DE REGISTRADORES (K)

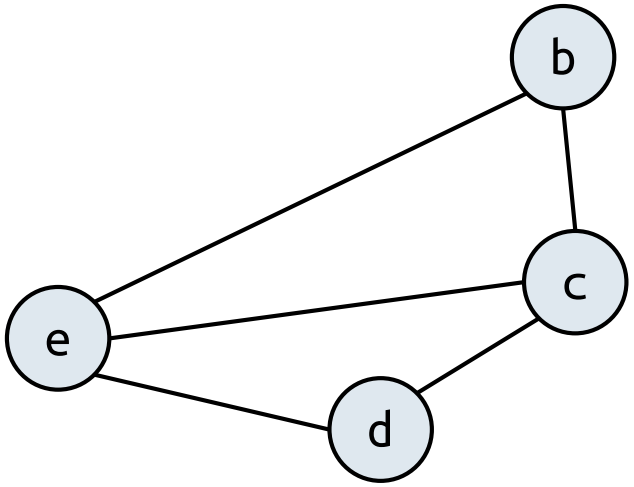
PILHA

- Não há o que fazer, todos os nós tem ≥ 3 (k) vizinhos
- Temos que selecionar um nó para *spilling*

a

Exemplo (Spilling)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR



NÚMERO DE REGISTRADORES (K)

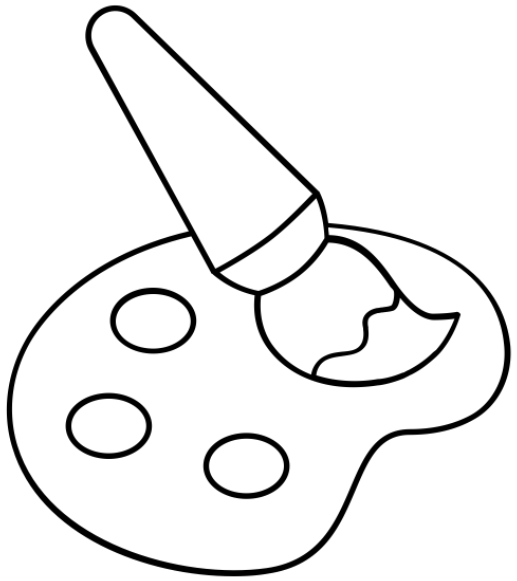
PILHA

- Simplificação continua
- Selecionamos **b**, **d**, **e**, **c**

a

Derramamento (Spilling)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR



- Eventualmente vamos ter que determinar uma cor para `f`
- Alocamos um **espaço** para `f` na memória
- Geralmente na **pilha**
- Ao gerar o código:
  - Antes de cada operação que lê `f` precisamos inserir:

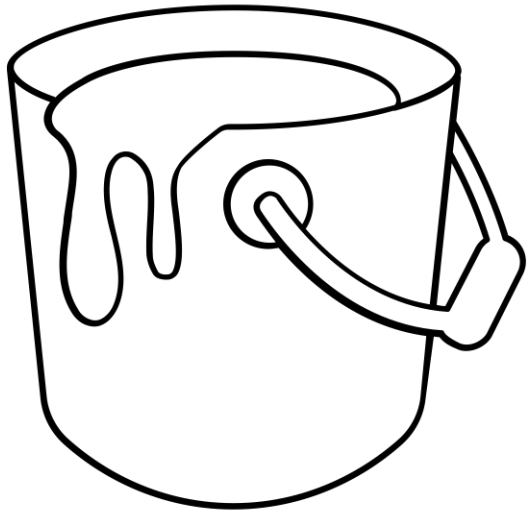
```
f := load fa
```

  - Após cada operação que esceve em `f`, precisamos inserir:

```
store f, fa
```

Derramamento (Spilling)

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

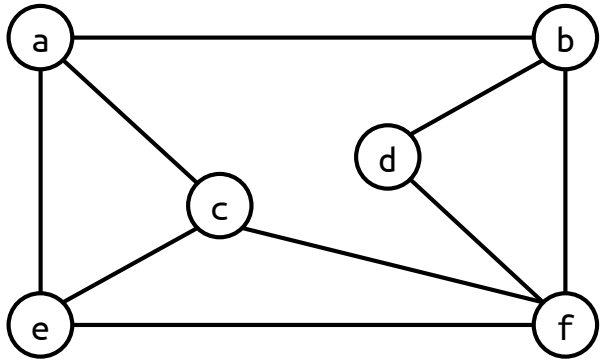


- Pode ser necessário vários “derramamentos” para encontrar uma coloração
- Como escolher **quem** deve ser derramado?
  - Qualquer escolha é correta
- Heurísticas:
  - Derramar temporários com **mais conflitos**
  - Derramar temporários com **pouca** definições e **usos**
  - Evitar derramamentos dentro de **loops**

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

EXERCÍCIO 3

Tente colorir o seguinte grafo supondo que temos somente 3 registradores disponíveis. Lembre de montar a pilha conforme os exemplos. No final diga quais variáveis ocupariam cada registrador.



Como podem existir mais de uma alocação e atribuição válida, vamos determinar que sempre vamos remover os labels em ordem alfabética e para colorir vamos utilizar o primeiro registrador disponível.