COMP130014.02 编译

第十二讲：寄存器分配

徐 辉
xuh@fudan.edu.cn

大纲

- 一、寄存器分配问题
- 二、着色问题和解法
- 三、预分配和溢出

一、寄存器分配问题

Review: IR

```
define i32 @fac(i32 %0) {
bb0:
    %n = alloca i32
    %r = alloca i32
    store i32 %0, i32* %n
    store i32 1, i32* %r
    br label %bb1
bb1:
    %t1 = load i32, i32* %n
    %t2 = icmp sgt i32 %t1, 0
    br i1 %t2, label %bb2, label %bb3
bb2:
    %t3 = load i32, i32* %r
    %t4 = load i32, i32* %n
    %t5 = mul i32 %t3, %t4
    store i32 %t5, i32* %r
    %t6 = load i32, i32* %n
    %t7 = sub i32 %t6, 1
    store i32 %t7, i32* %n
    br label %bb1
bb3:
    %t8 = load i32, i32* %r
    ret i32 %t8
}
```

```
fn fac(n: int) -> int {
    let r = 1;
    while (n>0) {
        r = r * n;
        n = n-1;
    }
    ret r;
}
```

=> 汇编?

Review: SSA

```
define i32 @fac(i32 %0) {
bb0:
    br label %bb1
bb1:
    %n0 = phi i32 [ %0, %bb0 ], [ %t7, %bb2 ]
    %r0 = phi i32 [ 1, %bb0 ], [ %t5, %bb2 ]
    %t2 = icmp sgt i32 %n0, 0
    br i1 %t2, label %bb2, label %bb3

bb2:
    %t5 = mul i32 %r0, %n0
    %t7 = sub i32 %n0, 1
    br label %bb1
bb3:
    ret i32 %r0
}
```

=> 汇编？

Review: SSA=>deSSA

```
define i32 @fac(i32 %0) {
bb0:
    %n = alloca i32
    %r = alloca i32
    store i32 %0, i32* %n
    store i32 1, i32* %r
    br label %bb1
bb1:
    %n0 = load i32, i32* %n
    %r0 = load i32, i32* %r
    %t2 = icmp sgt i32 %n0, 0
    br i1 %t2, label %bb2, label %bb3

bb2:
    %t5 = mul i32 %r0, %n0
    %t7 = sub i32 %n0, 1
    store i32 %t5, i32* %r
    store i32 %t7, i32* %n
    br label %bb1
bb3:
    ret i32 %r0
}
```

=>

```
_fac:
    sub sp, sp, #16
    str w0, [sp, #12]
    mov w1, #1
    str w1, [sp, #8]
    b LBB0_1
LBB0_1:
    ldr w2, [sp, #12]
    ldr w3, [sp, #8]
    cmp w2, #0
    b.le LBB0_3
LBB0_2:
    mul w4, w3, w2
    sub w5, w2, #1
    str w4, [sp, #8]
    str w5, [sp, #12]
    b LBB0_1
LBB0_3:
    ldr w0, [sp, #8]
    add sp, sp, #16
    ret
```

deSSA IR指令翻译结果

- 虚拟寄存器：单赋值，编号递增
- 同一虚拟寄存器在多个代码块有效

```
_main:
    sub sp, sp, #32
    str x30, [sp, #16]
    mov w0, #10
    bl _fac
    ldr x30, [sp, #16]
    add sp, sp, #32
    ret
```

```
_fac:
    sub sp, sp, #16
    str w0, [sp, #12]
    mov w1, #1
    str w1, [sp, #8]
LBB0_1:
    ldr w2, [sp, #12]
    ldr w3, [sp, #8]
    cmp w2, #0
    b.le LBB0_3
LBB0_2:
    mul w4, w3, w2
    sub w5, w2, #1
    str w4, [sp, #8]
    str w5, [sp, #12]
    b LBB0_1
LBB0_3:
    ldr w0, [sp, #8]
    add sp, sp, #16
    ret
```

寄存器分配问题

- 将不限数量的虚拟寄存器翻译为有限的物理寄存器
- 寄存器使用需遵循寄存器使用规约
- 物理寄存器不足则将数据写入内存（spill），使用时再读取

aarch64寄存器	调用规约	注释
X0-X7	参数1-8	
X0-X1	返回值	
X8	特殊用途：间接调用返回地址	
X9-X15	**临时寄存器**	**Caller-saved**
X16-X17	特殊用途：Intra-Procedure-Call	
X18	特殊用途：平台寄存器	
X19-X28	**普通寄存器**	**Callee-saved**
X29	栈帧基指针	Callee-saved
X30	返回地址	Caller-saved
SP	栈顶指针	Callee-saved

活跃性分析（SSA）

- 逆向遍历控制流图

```
sub sp, sp, #16
str w0, [sp, #12]
mov w1, #1
str w1, [sp, #8]
```

.LBB0_1

```
ldr w2, [sp, #12]
ldr w3, [sp, #8]
cmp w2, #0
b.le LBB0_3
```

.LBB0_2

```
mul w4, w3, w2
sub w5, w2, #1
str w4, [sp, #8]
str w5, [sp, #12]
b LBB0_1
```

.LBB0_3

```
mov w0, w1
add sp, sp, #16
ret
```

- 如遇到指令：`ldr x, [addr]`
  - $\mathrm{KILL(n)} = \{\mathrm{x}\}$
- 如遇到指令：`str x, [addr]`
  - $\mathrm{Gen(n)} = \{\mathrm{x}\}$
- 如遇到指令：`add x1, x2, x3`
  - $\mathrm{Gen(n)} = \{\mathrm{x2}, \mathrm{x3}\}$
  - $\mathrm{KILL(n)} = \{\mathrm{x1}\}$

$$\mathrm{OUT}(n) = \bigcup_{n' \in \mathrm{successor}(n)} \mathrm{IN}(n')$$

活跃性分析（SSA）

```
sub sp, sp, #16
str w0, [sp, #12]
mov w1, #1
str w1, [sp, #8]
```

.LBB0_1

```
ldr w2, [sp, #12]
ldr w3, [sp, #8]
cmp w2, #0
b.le LBB0_3
```

∅
w2
w2
∅
∅

.LBB0_2

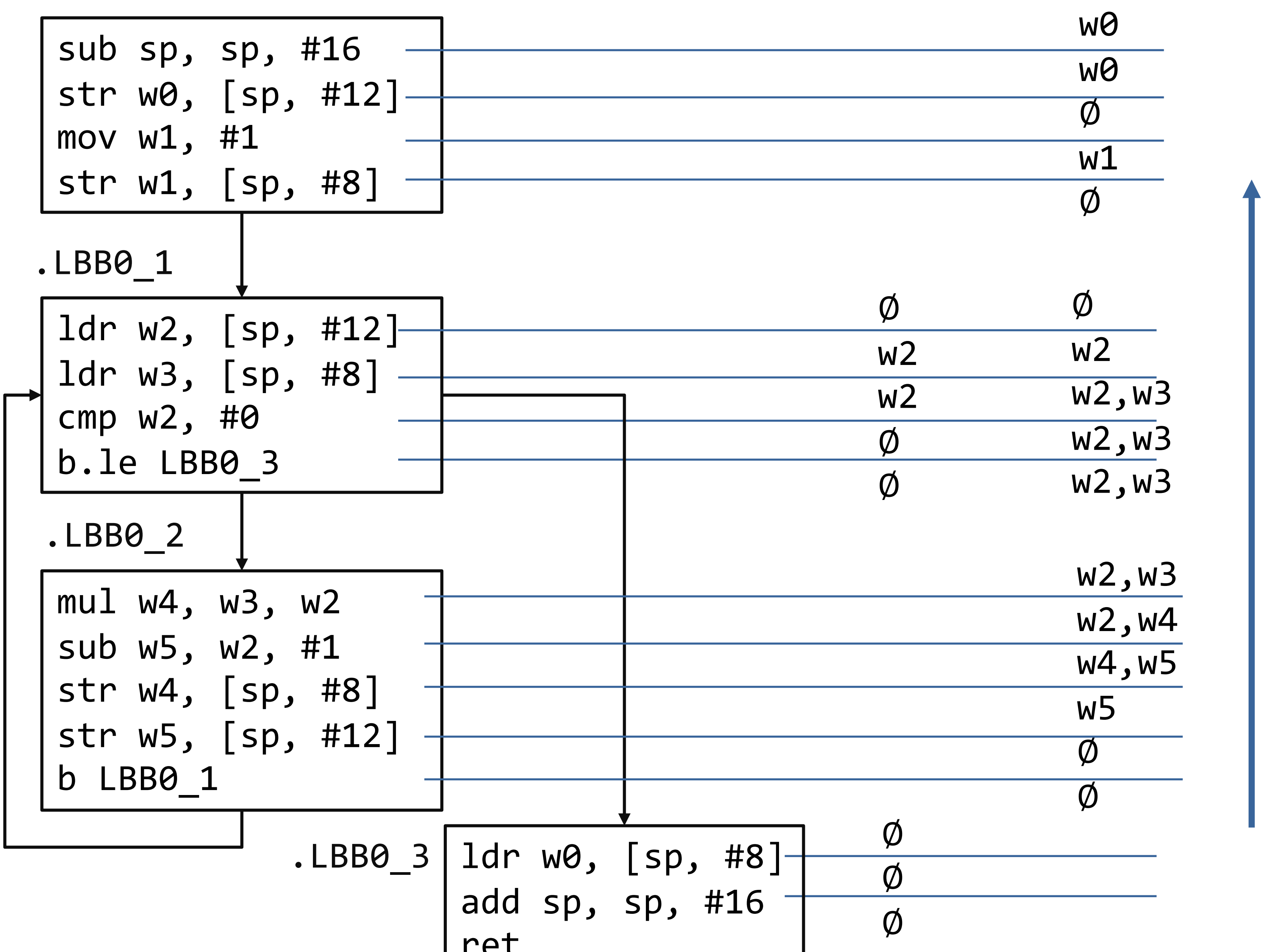
```
mul w4, w3, w2
sub w5, w2, #1
str w4, [sp, #8]
str w5, [sp, #12]
b LBB0_1
```

.LBB0_3

```
ldr w0, [sp, #8]
add sp, sp, #16
ret
```

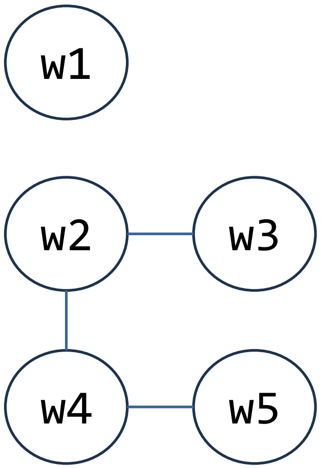
∅
∅
∅

活跃性分析（SSA）


sub sp, sp, #16
str w0, [sp, #12]
mov w1, #1
str w1, [sp, #8]
w0
w0
Ø
w1
Ø
.LBB0_1
ldr w2, [sp, #12]
ldr w3, [sp, #8]
cmp w2, #0
b.le LBB0_3
Ø
w2
w2
Ø
Ø
Ø
w2
w2,w3
w2,w3
w2,w3
.LBB0_2
mul w4, w3, w2
sub w5, w2, #1
str w4, [sp, #8]
str w5, [sp, #12]
b LBB0_1
w2,w3
w2,w4
w4,w5
w5
Ø
Ø
.LBB0_3
ldr w0, [sp, #8]
add sp, sp, #16
ret
Ø
Ø
Ø

干扰图（Interference Graph）

- 干扰：两个同时活跃的寄存器存在干扰关系
- 干扰图：连接所有存在干扰关系的寄存器节点
- 含义：存在干扰关系的节点应分配不同的物理寄存器



分配结果：
- w1, w2, w5 => w9
- w3, w4 => w10

翻译结果

```
_fac:
    sub sp, sp, #16
    str w0, [sp, #12]
    mov w1, #1
    str w1, [sp, #8]
LBB0_1:
    ldr w2, [sp, #12]
    ldr w3, [sp, #8]
    cmp w2, #0
    b.le LBB0_3
LBB0_2:
    mul w4, w3, w2
    sub w5, w2, #1
    str w4, [sp, #8]
    str w5, [sp, #12]
    b LBB0_1
LBB0_3:
    ldr w0, [sp, #8]
    add sp, sp, #16
    ret
```

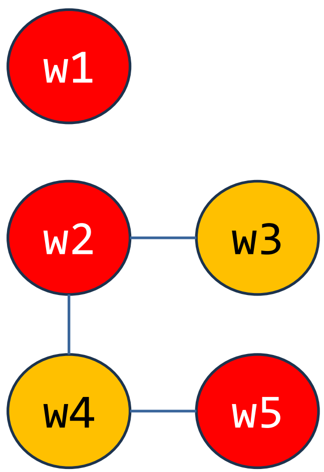
⇨

```
_fac:
    sub sp, sp, #16
    str w0, [sp, #12]
    mov w9, #1
    str w9, [sp, #8]
LBB0_1:
    ldr w9, [sp, #12]
    ldr w10, [sp, #8]
    cmp w9, #0
    b.le LBB0_3
LBB0_2:
    mul w10, w10, w9
    sub w9, w9, #1
    str w10, [sp, #8]
    str w9, [sp, #12]
    b LBB0_1
LBB0_3:
    ldr w0, [sp, #8]
    add sp, sp, #16
    ret
```

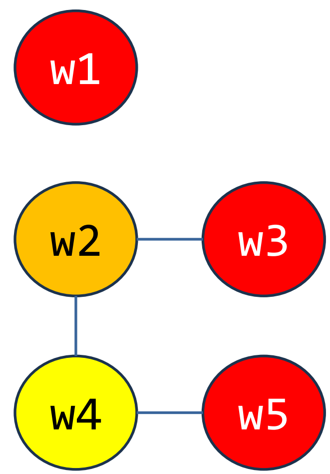
二、着色问题和解法

寄存器分配=>着色问题（Graph Coloring）

- 使用不超过K种（X9-X15）颜色，要求相邻节点颜色均不同
- 当K≥3时，该问题是NP完全问题（Chaitin的证明）

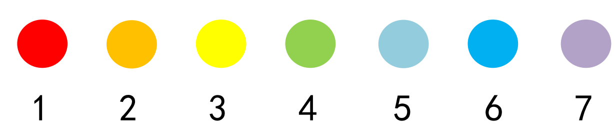


着色顺序：1-2-3-4-5



着色顺序：1-3-2-5-4

颜色顺序：



基于SAT问题证明

- k-SAT：CNF的每个Clause有不超过k个literals
  - 3SAT是NP-Complete问题
  - 2SAT是多项式复杂度可解
- 如果所有SAT问题可以多项式时间reduce到目标问题，则说明目标问题的难度至少与SAT相当

Literal: $x_1, \overline{x_1}, x_2, \overline{x_2}, x_3, \ldots$

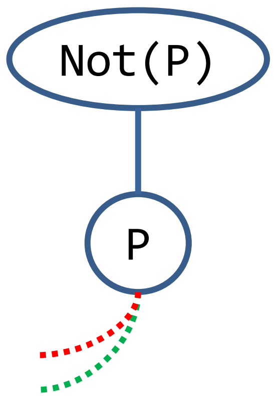
Clause: $l_1 \vee l_2 \vee l_3$

Conjunctive Normal Form: $C_1 \wedge C_2 \wedge \cdots$

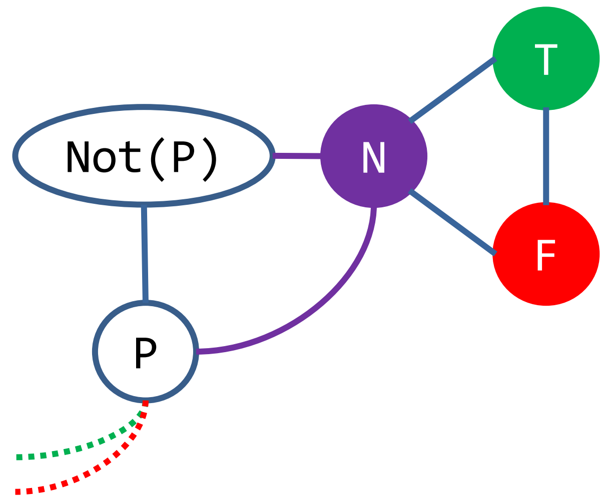
举例: $(x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2 \vee \overline{x_3} \vee x_4) \wedge \cdots$

3SAT可以reduce到着色问题

- 构造not和or
- and可以用not和or表示：
  - $C_1 \land C_2 = \neg(\neg C_1 \lor \neg C_2) \ldots$



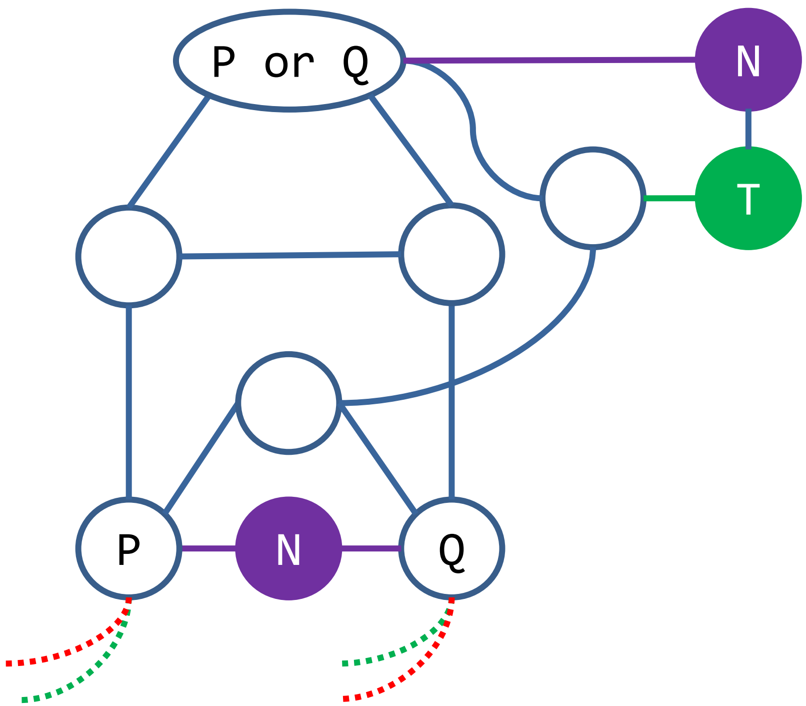
构造not



构造not

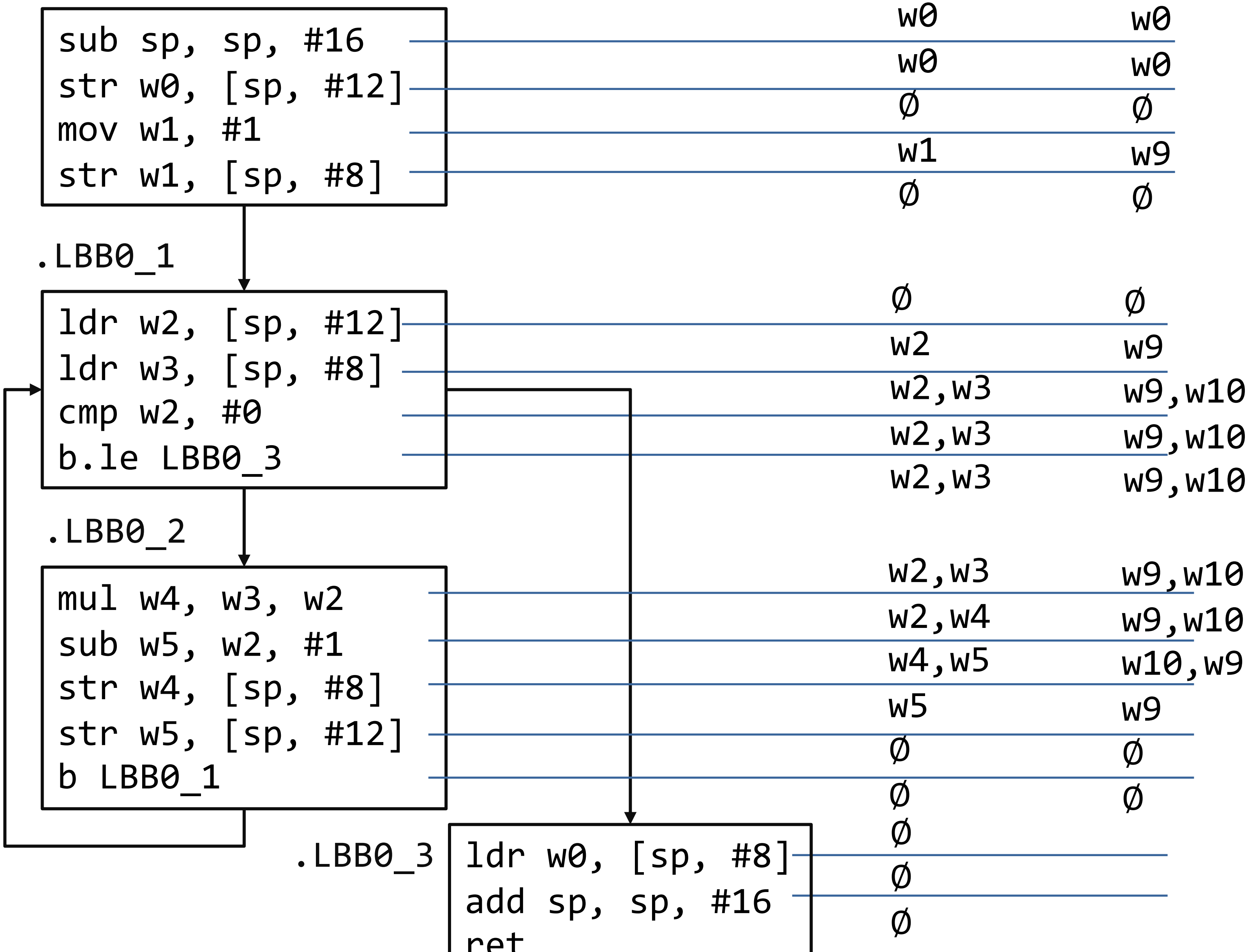
Albert R. Meyer和他的学生，MIT

3SAT可以reduce到着色问题



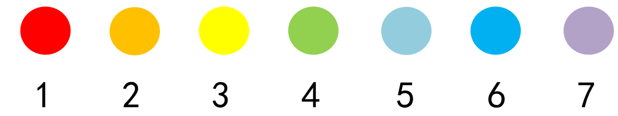
构造or

线性扫描算法：先到先得


sub sp, sp, #16
str w0, [sp, #12]
mov w1, #1
str w1, [sp, #8]
.LBB0_1
ldr w2, [sp, #12]
ldr w3, [sp, #8]
cmp w2, #0
b.le LBB0_3
.LBB0_2
mul w4, w3, w2
sub w5, w2, #1
str w4, [sp, #8]
str w5, [sp, #12]
b LBB0_1
.LBB0_3
ldr w0, [sp, #8]
add sp, sp, #16
ret
w0
w0
Ø
w1
Ø
Ø
w2
w2,w3
w2,w3
w2,w3
w2,w3
w2,w4
w4,w5
w5
Ø
Ø
Ø
Ø
Ø
w0
w0
Ø
w9
Ø
Ø
w9
w9,w10
w9,w10
w9,w10
w9,w10
w9,w10
w10,w9
w9
Ø
Ø

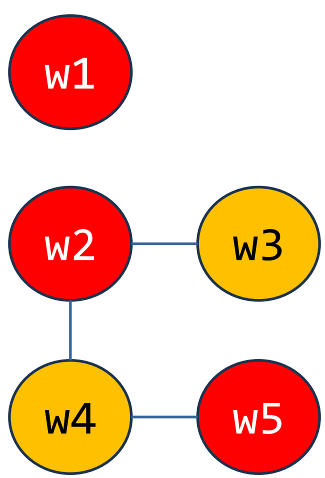
贪心法着色

颜色顺序：

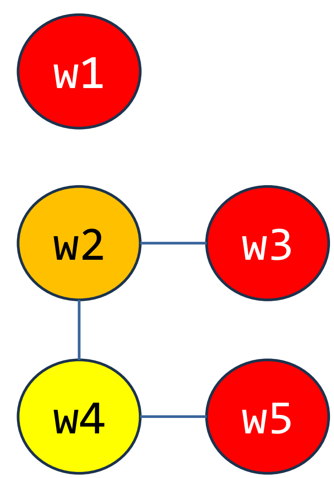


- 根据邻居节点颜色，为当前节点选取编号最小的可用颜色

```
颜色选取方法
Input: G=(V,E)
Output: Assignment of colors
For i = 1..n do
  Let c be the lowest color not used in Neighbor(vi)
  Set Col(vi) = c
```



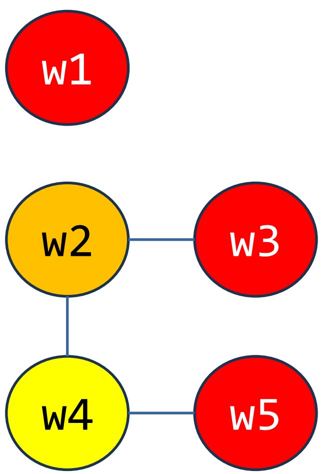
着色顺序：1-2-3-4-5

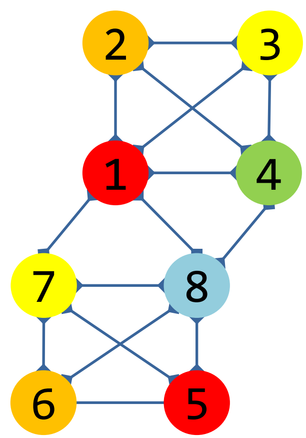


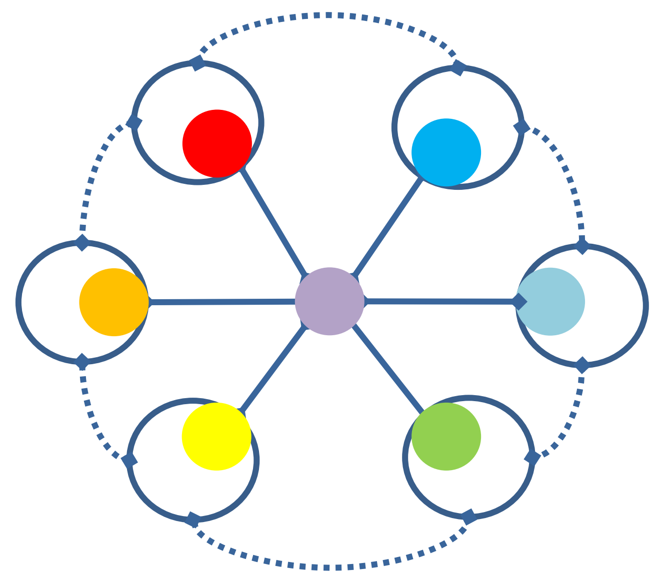
着色顺序：1-3-2-5-4

如何选择着色顺序：启发式

- 在图上搜索团(clique)：所有节点两两连接
- 团着色所需颜色数与团的大小一致
- 找最大团也是np-hard问题
- 不能保证最优解

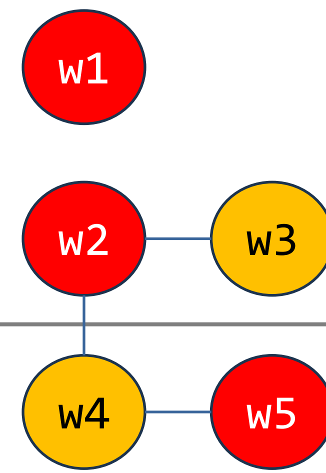






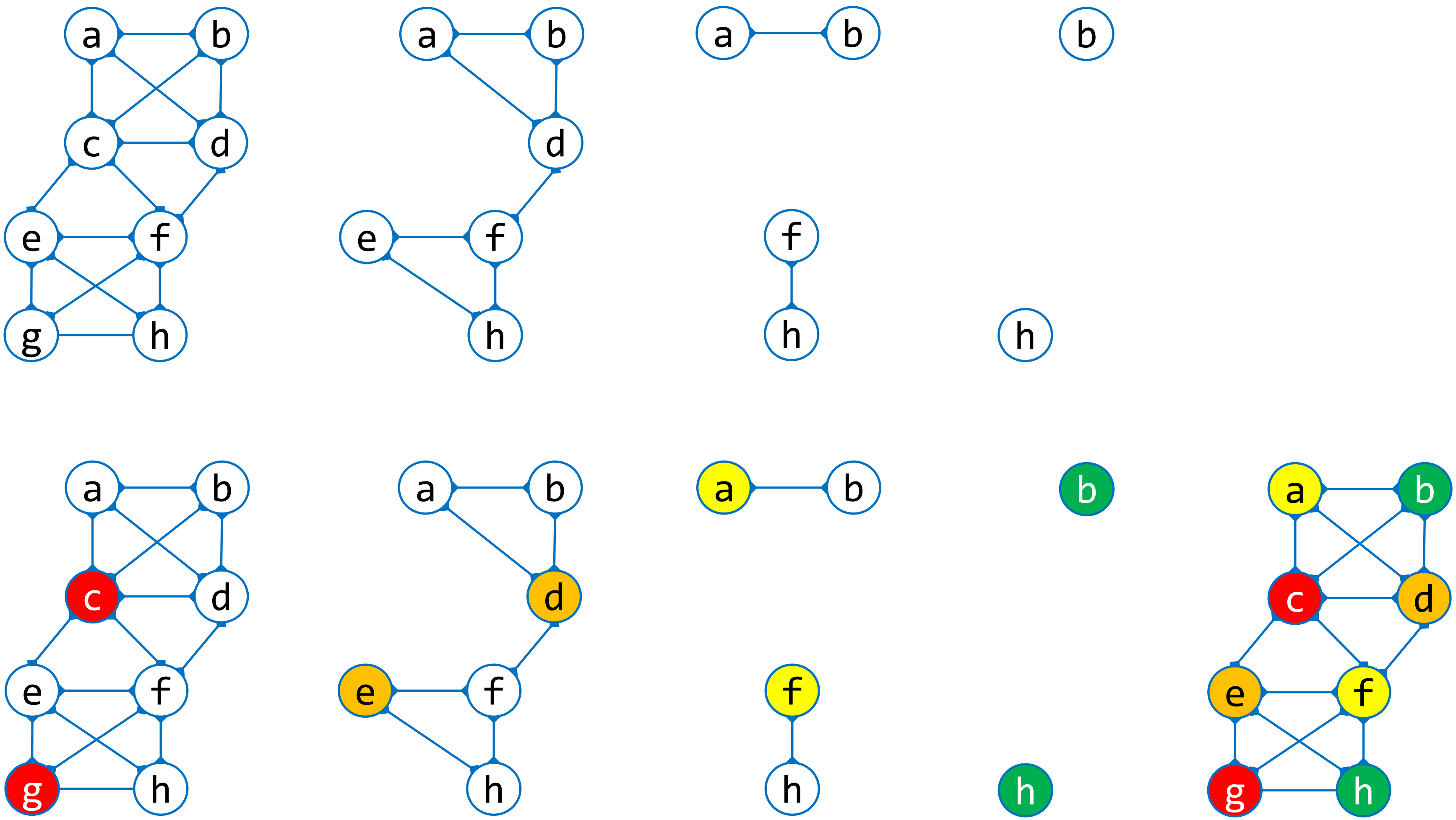
启发式方法：Recursive Largest First算法

```
RLF(G):
    Find v_i ∈ G with the max degree
    Add v_i to S
    Let T be the rest nodes in G non-adjacent to any node in S
    Repeat until T is NULL:
        Find v_j ∈ T with the max degree
        Add v_j to S
        Update T
    color(S)
    G = G - S
    S = NULL
    RLF(G)
```



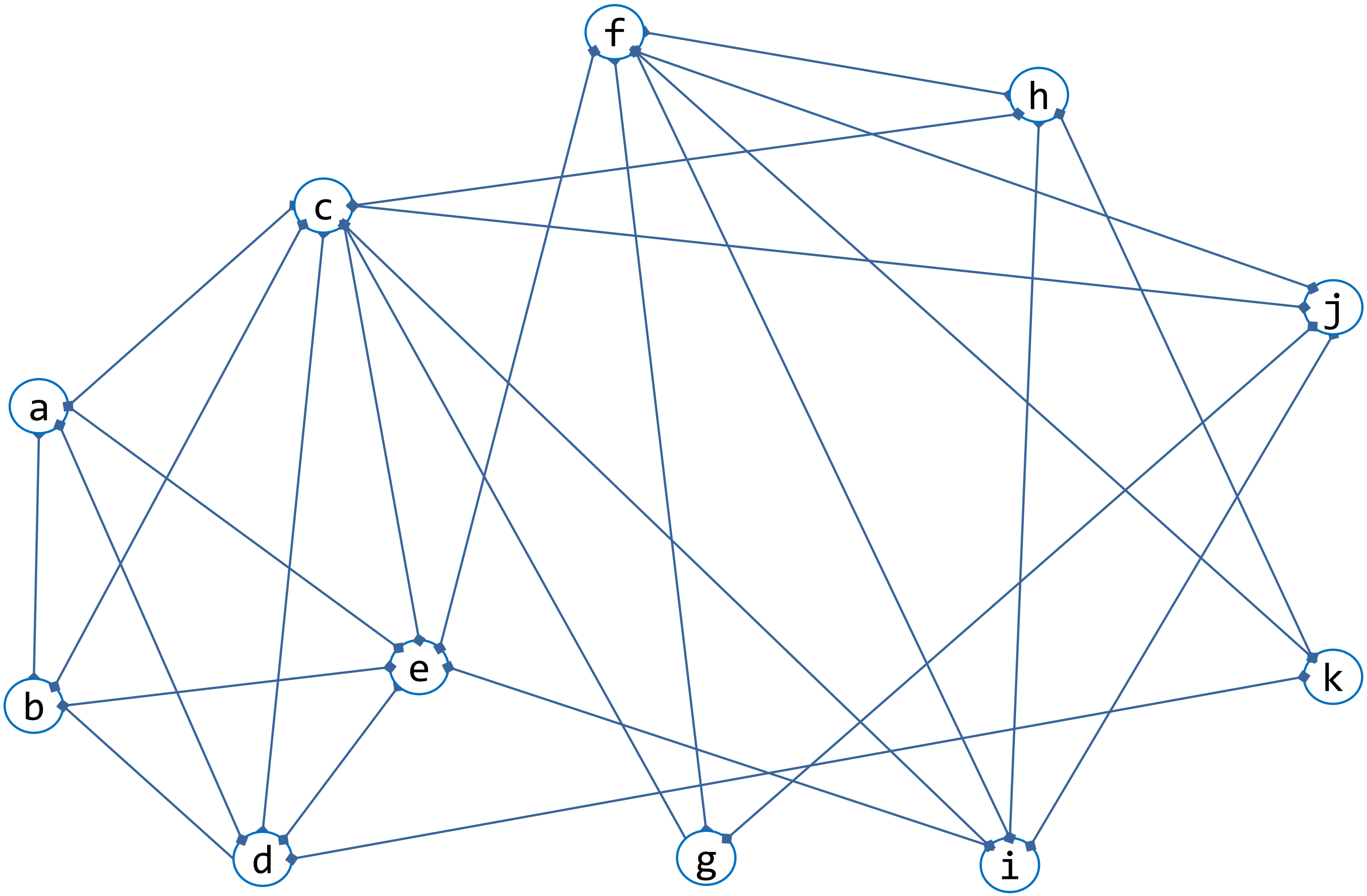
着色顺序：2-5-3-4-1

RLF示例



练习

- 找出下图的最佳着色顺序，共需几种颜色？

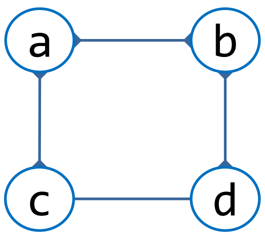


更多启发式思路

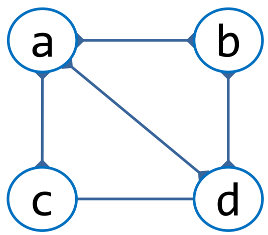
- 虎书中介绍的方法：每次从图G中选取邻居数小于K的节点v，如果图G-v可以使用K种颜色着色，则G也可以

一类特殊的着色问题：弦图（Chordal Graph）

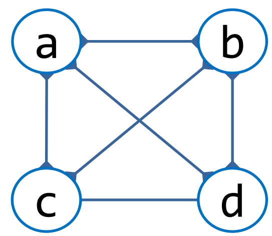
- 任意长度大于3的环都有弦（chord）
- 多项式时间可解
- SSA的干扰图都是弦图



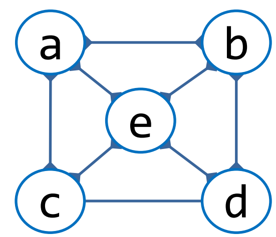
非弦图



弦图

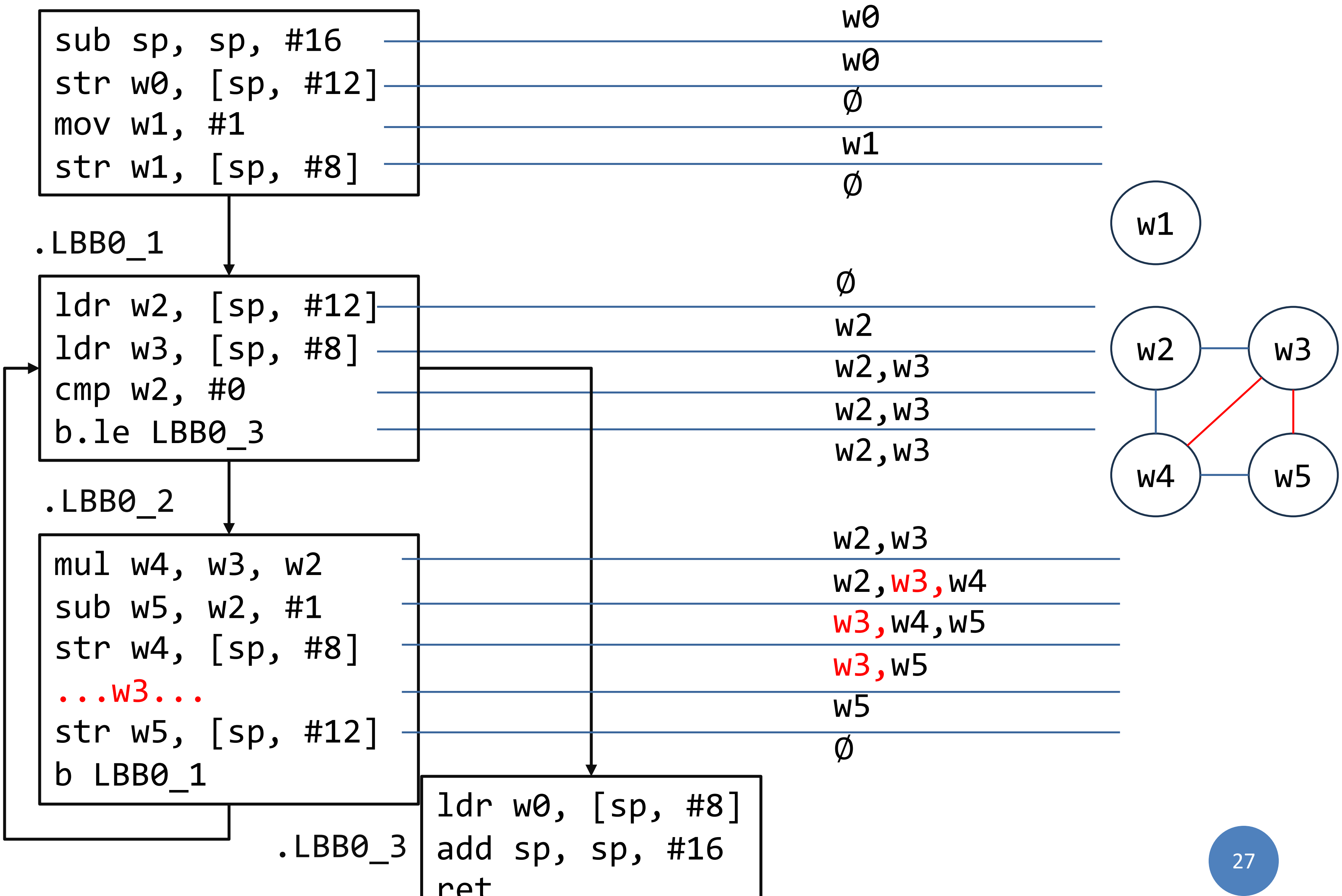


弦图



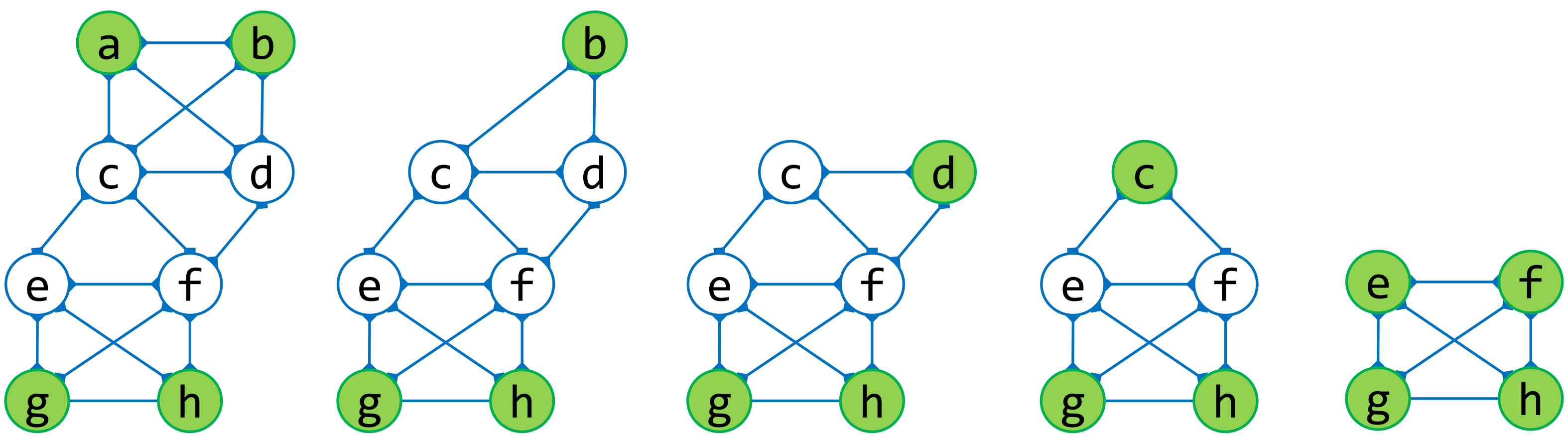
非弦图

尝试为SSA构造非弦图？



单纯消除序列（Simplicial Elimination Ordering）

- **单纯点**（simplicial）：所有邻居组成一个团
- **完美消除序列：**按照该序列消除的每一个点都是单纯点
- **单纯消除序列：**完美消除序列的逆序
- 如果一个图是弦图，则该图存在完美消除序列

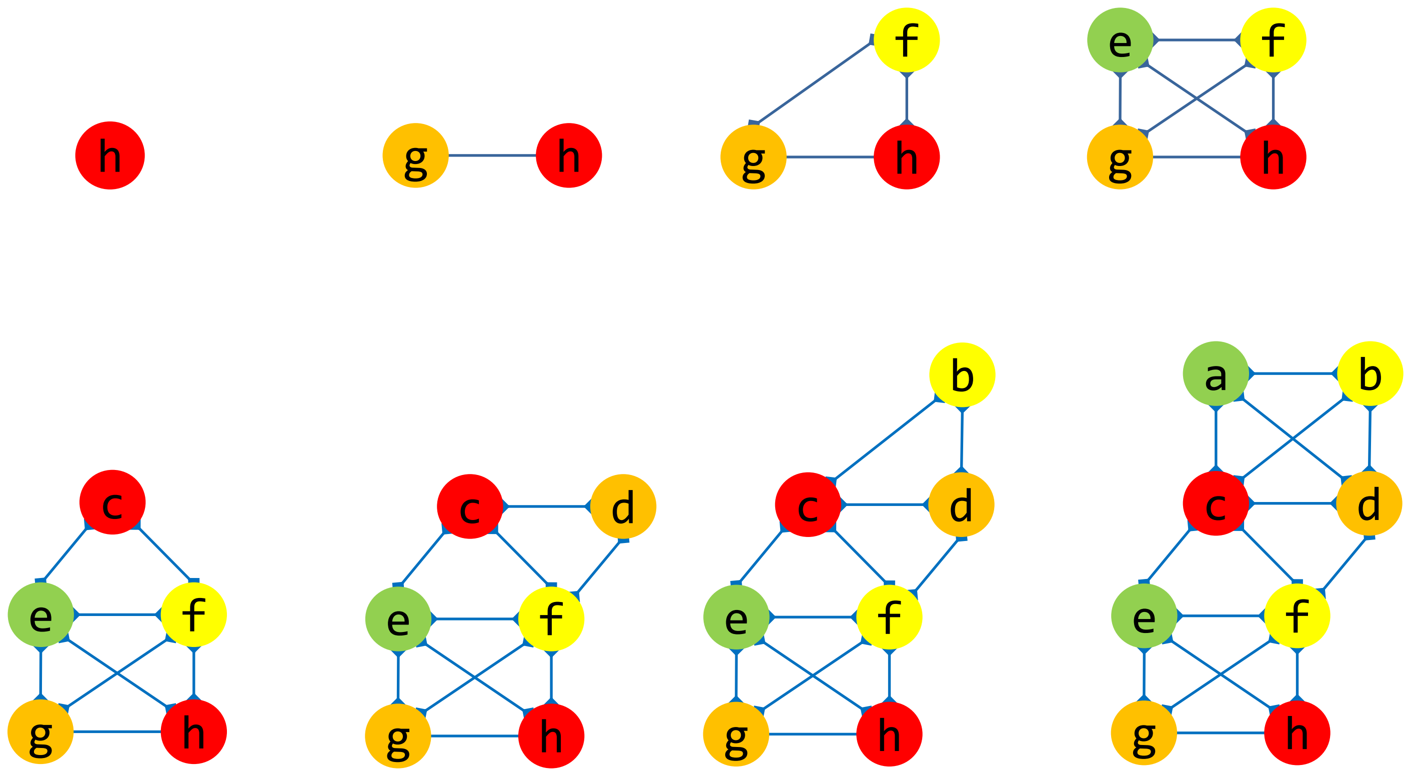


消除a　消除b　消除d　消除c　消除e　消除f…

单纯点　非单纯

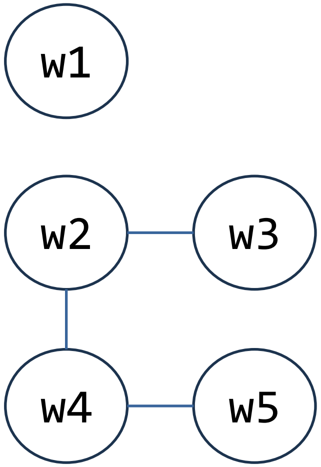
基于单纯消除序列着色

- 每次在已着色团的基础上新增一个点，连接该团的所有点



最大势算法求单纯消除序列

- Maximum Cardinality Search
- 思路：搜索与已着色节点邻居最多的点
  - 维护一个所有点的向量，每次选取值最大的点；
  - 选取一个点后，则其邻居计数加1。



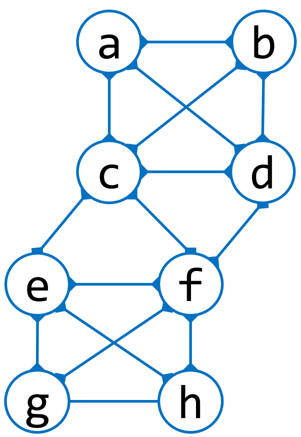
步骤	选取	w1	w2	w3	w4	25
1	w1		0	0	0	0
2	w2			1	1	0
3	w3				1	1
4	w4					1
5	w5					

算法参考

```
Maximum Cardinality Search
Input: G = (V, E)
Output: Simplicial elimination ordering v_1, …, v_n
For all v_i ∈ V
    w(v_i) = 0
Let W = V
For i = 1,..,n do
    Let v be a node with max weight in W
    Set v_i = v
    For all u ∈ W ∩ N(v)
        w(u) = w(u) + 1
    W = W\{v}
```

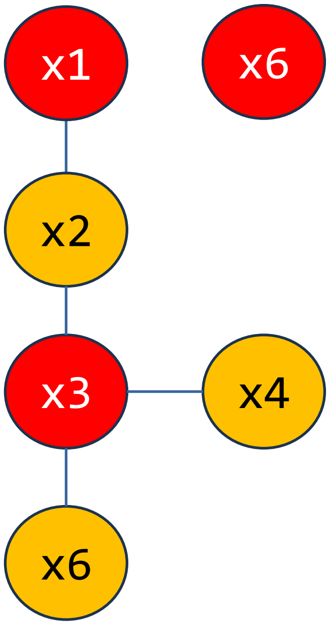
练习

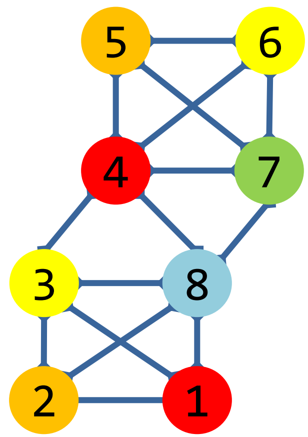
- 求下列冲突图的单纯消除序列

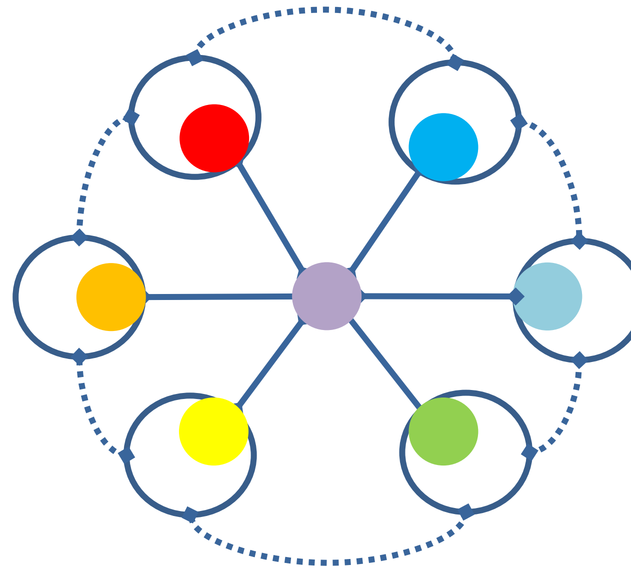


步骤	选取	a	b	c	d	e	f	g	h
		0	0	0	0	0	0	0	0
1	a								
2	b								
3	c								
4	d								
5	f								
6	e								
7	g								
8	h								

思考：为何最大势算法能得到单纯消除序列?







三、预分配和溢出

寄存器分配问题

- 将不限数量的虚拟寄存器翻译为有限的物理寄存器
- 寄存器使用需遵循寄存器使用规约
- 物理寄存器不足则将数据写入内存（spill），使用时再读取

aarch64寄存器	调用规约	注释
X0-X7	参数1-8	
X0-X1	返回值	
X8	特殊用途：间接调用返回地址	
X9-X15	**临时寄存器**	**Caller-saved**
X16-X17	特殊用途：Intra-Procedure-Call	
X18	特殊用途：平台寄存器	
X19-X28	**普通寄存器**	**Callee-saved**
X29	栈帧基指针	Callee-saved
X30	返回地址	Caller-saved
SP	栈顶指针	Callee-saved

函数调用

```
_fac:
    sub sp, sp, #16
    str w0, [sp, #12]
    mov w1, #1
    str w1, [sp, #8]
LBB0_1:
    ldr w2, [sp, #12]
    ldr w3, [sp, #8]
    cmp w2, #0
    b.le LBB0_3
LBB0_2:
    mul w4, w3, w2
    sub w5, w2, #1
    str w4, [sp, #8]
    str w5, [sp, #12]
    b LBB0_1
LBB0_3:
    ldr w0, [sp, #8]
    add sp, sp, #16
    ret
```

```
_main:
    sub sp, sp, #32
    str x30, [sp, #16]      → 保存返回地址
    mov w0, #10
    bl _fac
    ldr x30, [sp, #16]      → 还原返回地址
    add sp, sp, #32
    ret
```

使用x29的情况

- 不便使用sp作为栈帧参照

```
str x1, [sp, #-8]!
str x2, [sp, #-8]!
str x3, [sp, #-8]!
str x4, [sp, #-8]!
...
ldr x4, [sp], #8
ldr x3, [sp], #8
ldr x2, [sp], #8
ldr x1, [sp], #8
```

```
sub sp, sp, #32
stp x29, x30, [sp, #16]
add x29, sp, #16
str w0, [sp, #12]
```

```
stp x1, x2, [sp, #-16]!
```

```
ldp w0, [x29, #-4]
ldp x29, x30, [x29]
```

函数调用需要Spill所有Caller-saved寄存器

```
ldr     w9, [sp, #8]
ldr     w10, [sp, #12]
add     w0, w10, w9
mov     w1, w9
str     w9, [sp, #4]
str     w10, [sp]
bl      fnA
ldr     w10, [sp]
ldr     w9, [sp, #4]
add     w10, w8, w9
```

将临时寄存器入栈

函数调用

将临时寄存器还原

使用x19-x28的情况

- 如无函数调用：使用x9-x15无需spill
- 函数调用频繁：使用x9-x15会频繁spill；使用x19-x28更优

LBB0_1:

```
add     w11, w10, w9
cmp     w11, #0
b.le    LBB0_3
```

LBB0_3:

LBB0_2:

```
sub     w0, w10, w9
mov     w1, w9
str     w9, [sp, #4]
str     w10, [sp]
bl      _foo
ldr     w10, [sp]
ldr     w9, [sp, #4]
b LBB0_1
```

参数过多的情况

```
%b2 = call i32 @foo(
      i32 %a2, i32 %b1,
      i32 %a2, i32 %b1,
      i32 %a2, i32 %b1,
      i32 %a2, i32 %b1,
      i32 %a2, i32 %b1
      )
```

⇨

```
mov     w0, w8
mov     w1, w9
mov     w2, w8
mov     w3, w9
mov     w4, w8
mov     w5, w9
mov     w6, w8
mov     w7, w9
str     w8, [sp]
str     w9, [sp, #8]
bl      _foo
```

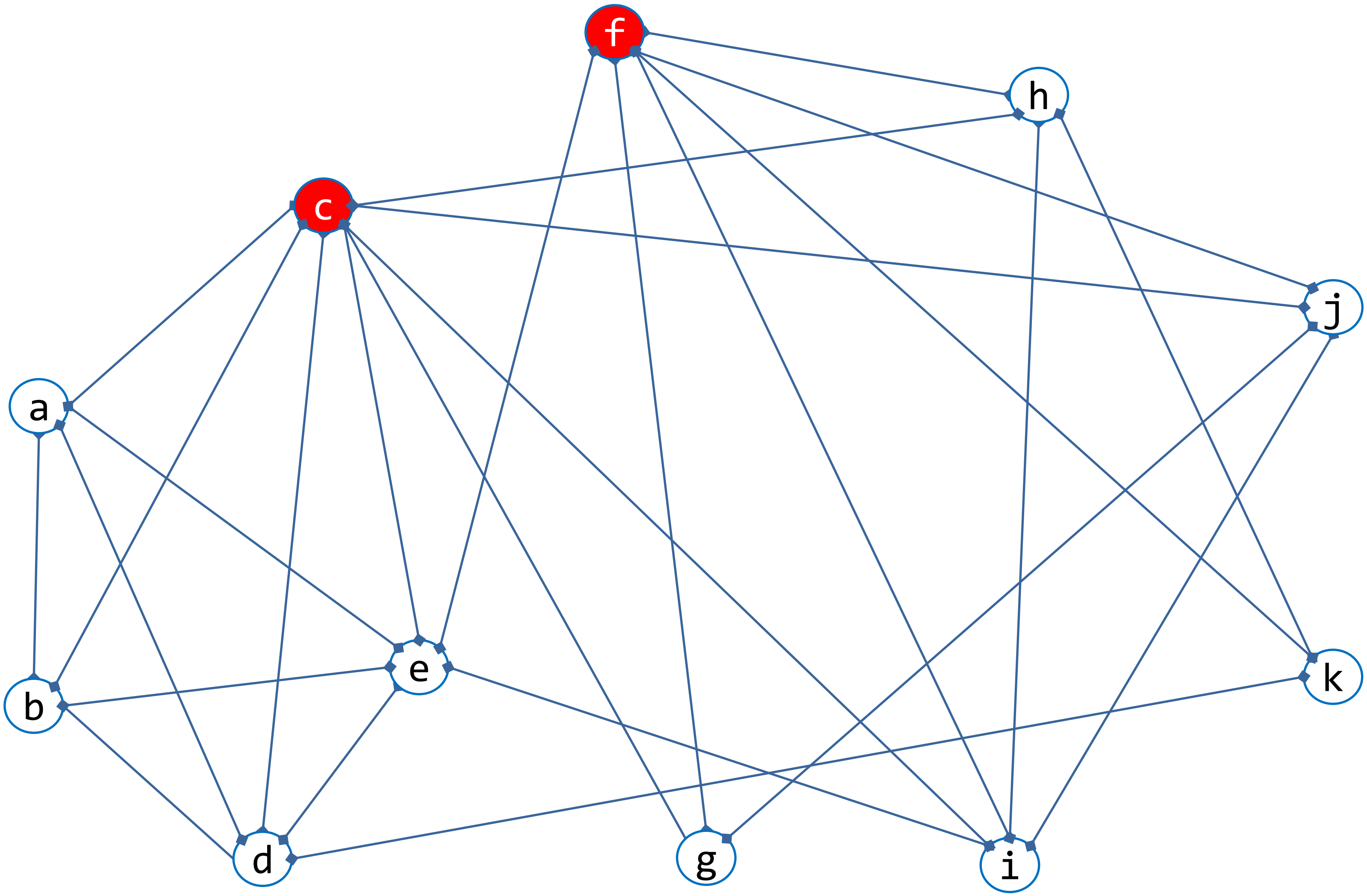
物理寄存器不足导致的溢出

- ARM架构通用寄存器数量较多，一般不易存在溢出情况
- X86架构通用寄存器数量较少，更容易出现溢出情况
- 寄存器不足时应优先溢出哪个虚拟寄存器？
  - 策略1：使用最少的颜色
  - 策略2：度数最高的寄存器，重新着色
- 目标：最少的溢出次数
  - 线性统计：代码中出现次数最少的
  - 考虑控制流：代码运行次数最少的

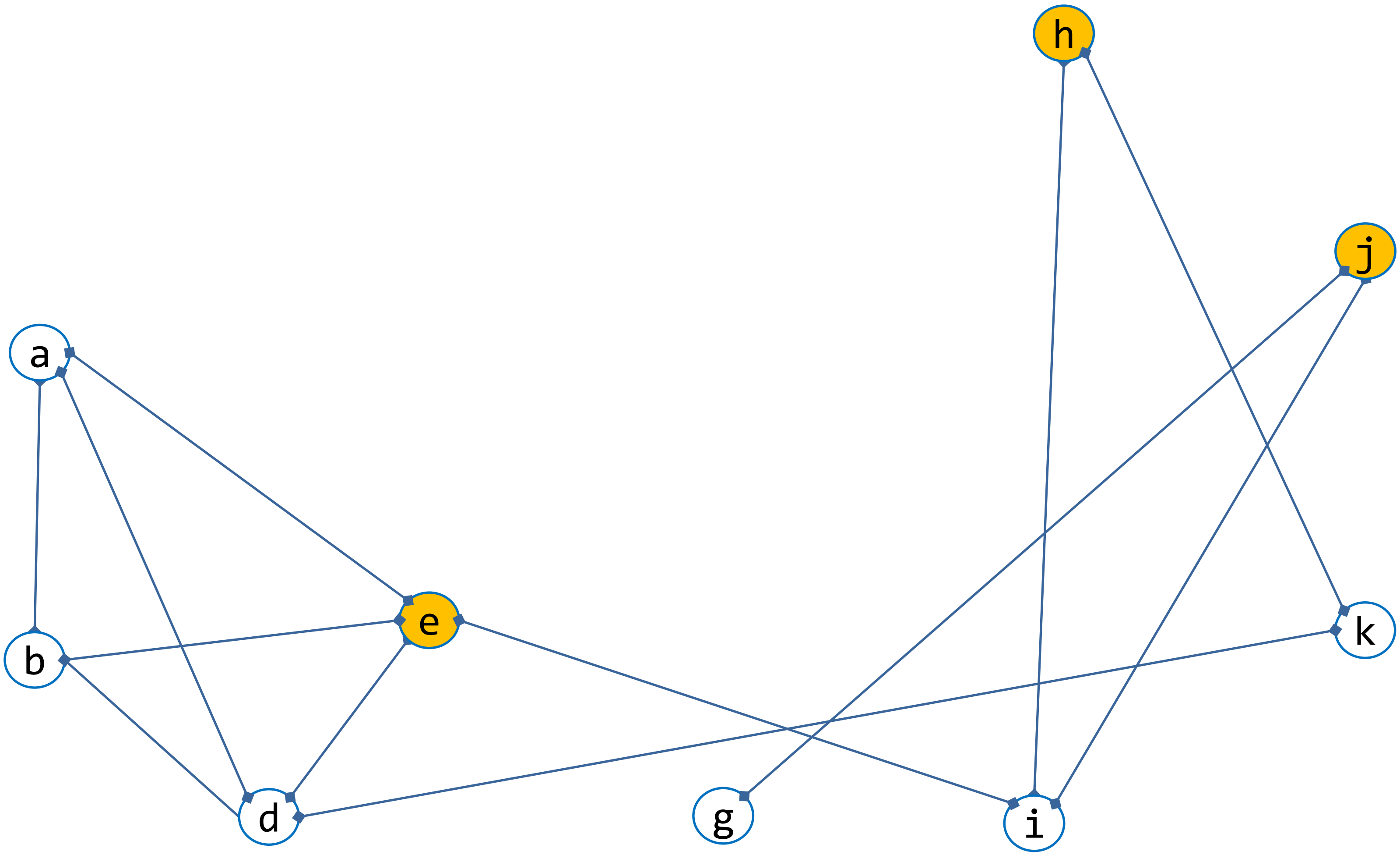
Backup Slides

练习

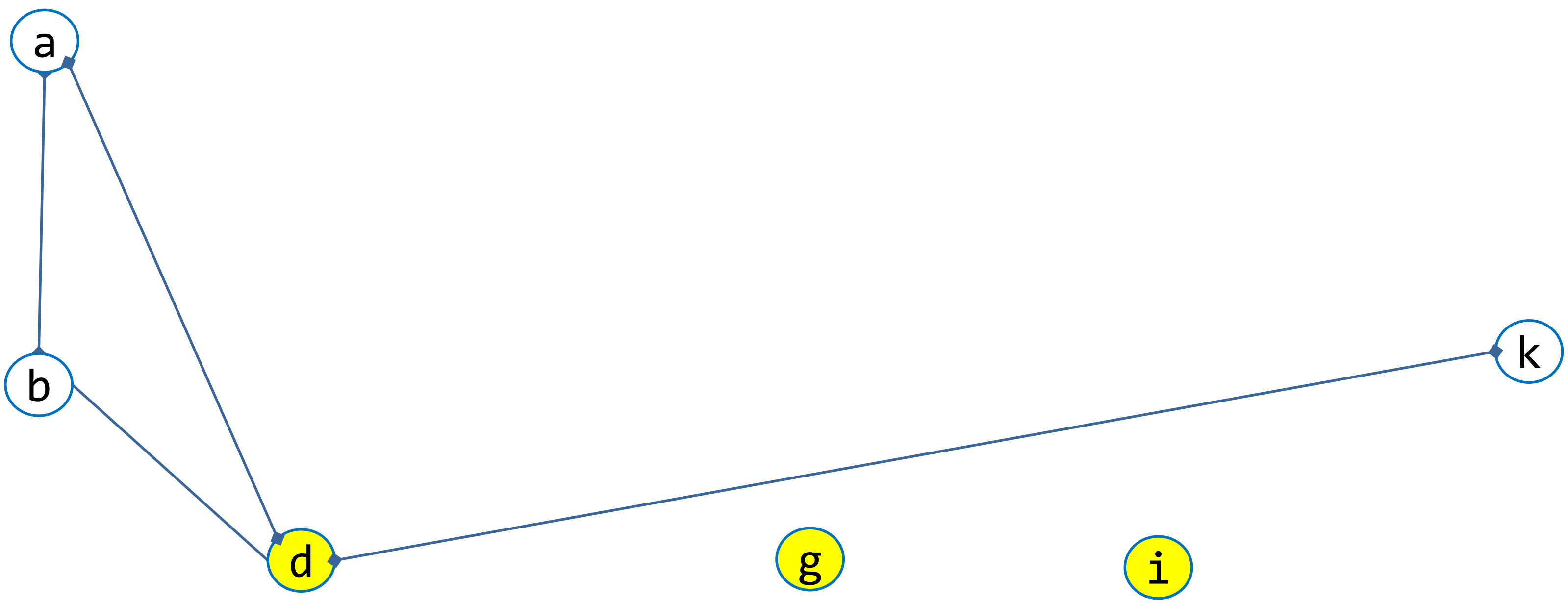
- 找出下图的最佳着色顺序，共需几种颜色？



练习

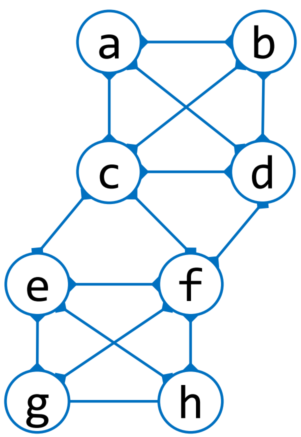


练习



练习

- 求下列冲突图的单纯消除序列



步骤	选取	a	b	c	d	e	f	g	h
		0	0	0	0	0	0	0	0
1	a		1	1	1	0	0	0	0
2	b			2	2	0	0	0	0
3	c				3	1	1	0	0
4	d					1	2	0	0
5	f					2		1	1
6	e							2	2
7	g								3
8	h								

- Chaitin, Gregory J. "Register allocation & spilling via graph coloring." *ACM Sigplan Notices* 17, no. 6 (1982): 98-101.
- Bouchez, Florent, Alain Darte, Christophe Guillon, and Fabrice Rastello. "Register allocation and spill complexity under SSA." PhD diss., Laboratoire de l'informatique du parallélisme, 2005
- Pereira, Fernando Magno Quintao, and Jens Palsberg. "Register allocation via coloring of chordal graphs." In *Asian Symposium on Programming Languages and Systems*, pp. 315-329. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005
- Appel, Andrew W., and Lal George. "Optimal spilling for CISC machines with few registers." *ACM SIGPLAN Notices* 36, no. 5 (2001): 243-253.