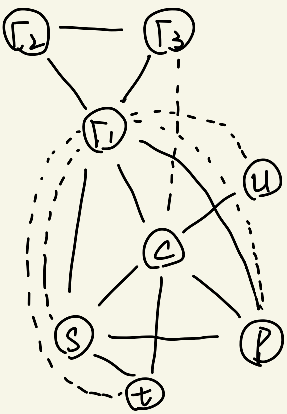
11.1 The following program has been compiled for a machine with three registers r_1, r_2, r_3; r_1 and r_2 are (caller-save) argument registers and r_3 is a callee-save register. Construct the interference graph and show the steps of the register allocation process in detail, as on pages 244–248. When you coalesce two nodes, say whether you are using the Briggs or George criterion.

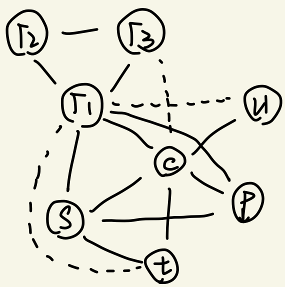
Hint: When two nodes are connected by an interference edge *and* a move edge, you may delete the move edge; this is called *constrain* and is accomplished by the first **else if** clause of procedure *Coalesce*.

```
f :   c  ← r3
      p  ← r1
      if p = 0 goto L1
      r1 ← M[p]
      call f            (uses r1, defines r1, r2)
      s  ← r1
      r1 ← M[p + 4]
      call f            (uses r1, defines r1, r2)
      t  ← r1
      u  ← s + t
      goto L2
L1 :  u  ← 1
L2 :  r1 ← u
      r3 ← c
      return            (uses r1, r3)
```

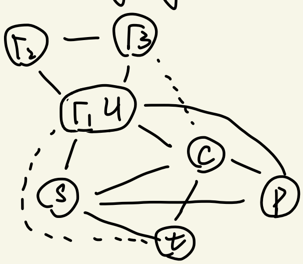
1. interference graph



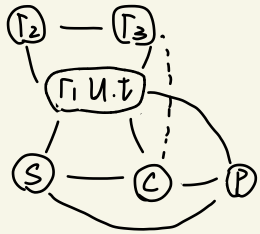
2. 删除 constrain 边



3. 根据 george. 合并. r1.u



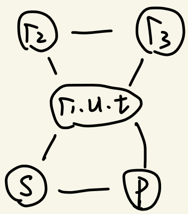
4. 根据 george. 合并. r1.u与t



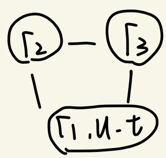
将 r3,c合并后.该图依然需要借助 spill 操作.因而直接图4的基础上 spill .c

此时

⇒ 5.



6. 化简. s.p

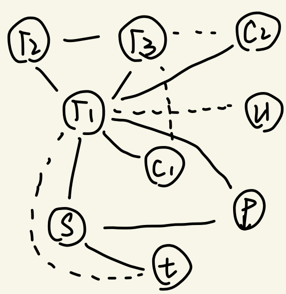


∵ c spill. ∴ 修改原程序如下

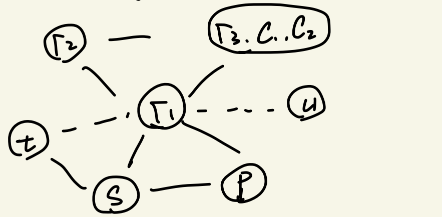
```
f:  c1 ← r3
    M[ci] ← c1
    p ← r1
    if p = 0 goto L1
    r1 ← M[p]
    call f
    s ← r1
    r1 ← M[p+4]
    call f
    t ← r1
    u ← s+t
    goto L2
L1: u ← 1
L2: r1 ← u
    c2 ← M[ci]
    r3 ← c2
    return
```

∴ 冲突图转化为

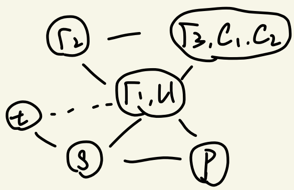
1.



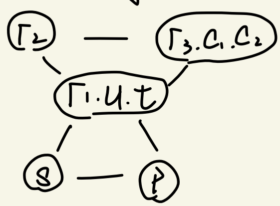
2. 根据 george. 合并. r_3, c_1, c_2



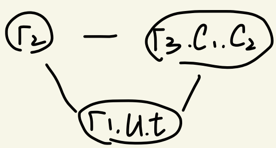
3. 合并. r_1, u



4. 根据 george 合并 r_1u 与 t



5. 依次化简 p. s



s
p

∵ 只剩 预着色寄存器 ∴ 着色成功

Node	color
c_1	r_3
c_2	r_3
u	r_1
t	r_1
p	r_2
s	r_3

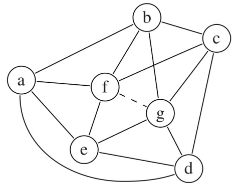
程序化简为

```
f: r3 <- r3
   M[ci] <- r3
   r2 <- r1
   if r2 = 0 goto L1
   r1 <- M[r2]
   call f
   r3 <- r1
   r1 <- M[r2+4]
   call f
   r1 <- r1
   r1 <- r1 + r3
   goto L2
L1: r1 <- 1
L2: r1 <- r1
   r3 <- M[ci]
   r3 <- r3
   return
```

最终可得

```
f: M[ci] <- r3
   r2 <- r1
   if r2 = 0 goto L1
   r1 <- M[r2]
   call f
   r3 <- r1
   r1 <- M[r2+4]
   call f
   r1 <- r1 + r3
   goto L2
L1: r1 <- 1
L2: r3 <- M[ci]
   return
```

11.3 *Conservative coalescing* is so called because it will not introduce any (potential) spills. But can it avoid spills? Consider this graph, where the solid edges represent interferences and the dashed edge represents a MOVE:



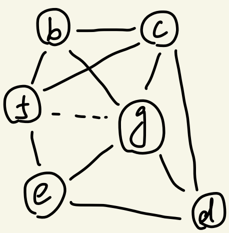
a. 4-color the graph without coalescing. Show the *select*-stack, indicating the order in which you removed nodes. Is there a potential spill? Is there an actual spill?

b. 4-color the graph with conservative coalescing. Did you use the Briggs or George criterion? Is there a potential spill? Is there an actual spill?

a 度均 ≥4 ∴存在 spill

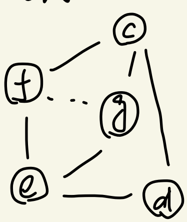
先spill a

⇒ 1.



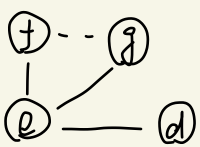
a

2. 化简 b



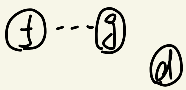
b
a

3. 化简 c



c
b
a

4. 化简. e



e
c
b
a

5. 可得栈

f
g
d
e
c
b
a

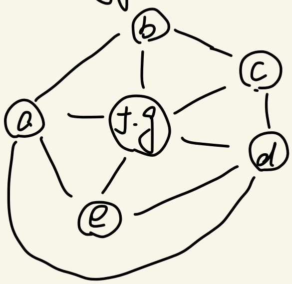
可分配A.B.C.D. 颜色

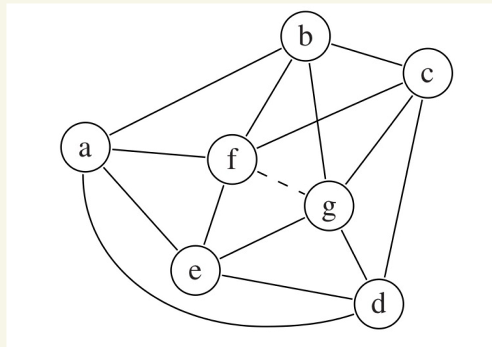
Node	color
f	A
g	A
d	A
e	B
c	B
b	C
a	D

∴仅存在 potential spill

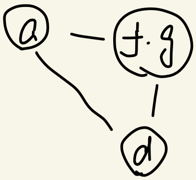
b

1. 使用 Briggs 准则，合并 f.g





2. 依次简化，e. b. c



c
b
e

3. 再依次化简 fg. a. d

得

d
a
fg
c
b
e

，图不剩任何结点

∴

Node	color
f.&g	A
d	B
a	C
c	C
b	B
e	D

∴ 不存在任何 spill，潜在的与实际的