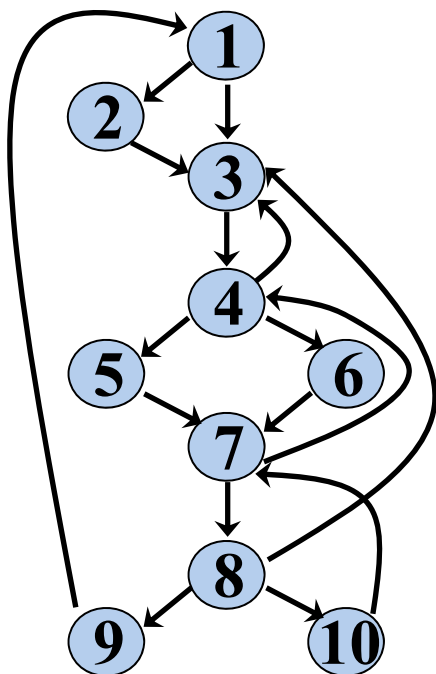


## 支配结点 (*Dominators*)

- 如果从流图的入口结点到结点 $n$ 的每条路径都经过结点 $d$ ，则称结点 $d$ 支配(*dominate*)结点 $n$ ，记为 $d \text{ dom } n$

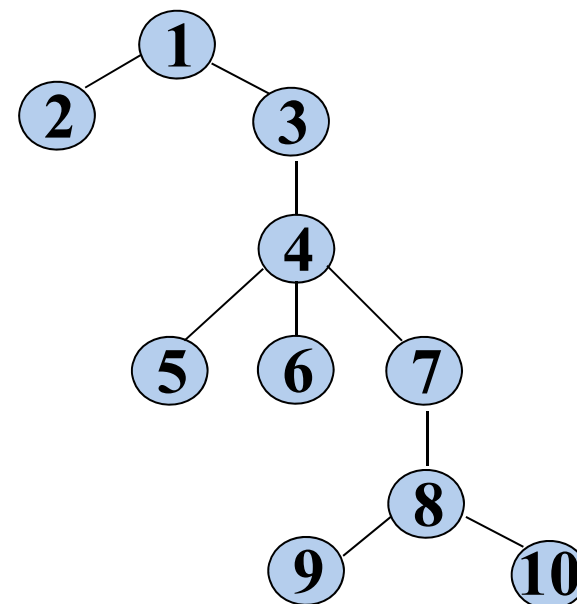
每个结点都支配它自己

## 例



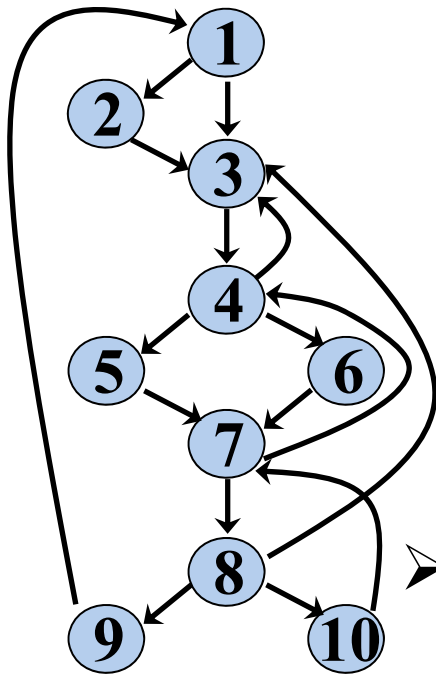
支配结点	支配对象
1	1~10
2	2
3	3~10
4	4~10
5	5
6	6
7	7~10
8	8~10
9	9
10	10

### ➤ 支配结点树 (Dominator Tree)



每个结点只支配它和它的后代结点

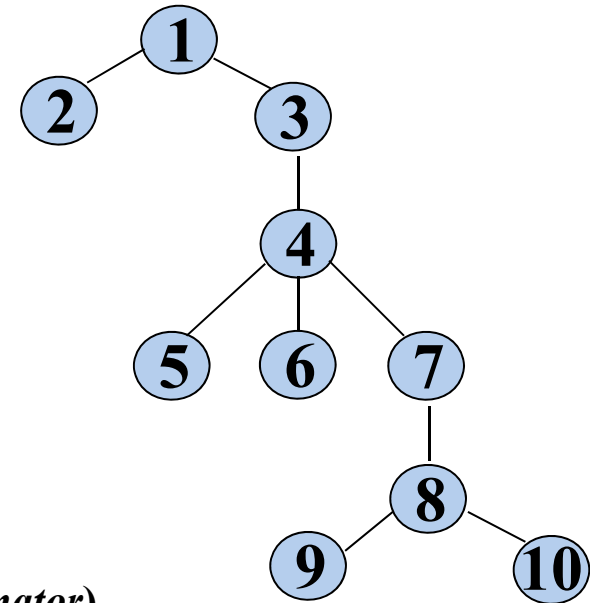
## 例



➤ 直接支配结点 (*Immediate Dominator*)

➤ 从入口结点到达 $n$ 的所有路径上，结点 $n$ 的最后一个支配结点称为直接支配结点

➤ 支配结点树 (*Dominator Tree*)



## 寻找支配结点

### ➤ 支配结点的数据流方程

➤  $IN[B]$ : 在基本块 $B$ 入口处的支配结点集合

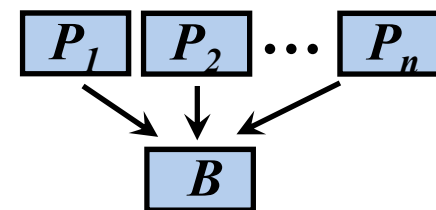
$OUT[B]$ : 在基本块 $B$ 出口处的支配结点集合

### ➤ 方程

➤  $OUT[ENTRY] = \{ ENTRY \}$

➤  $OUT[B] = IN[B] \cup \{ B \}$  ( $B \neq ENTRY$ )

➤  $IN[B] = \bigcap_{P \text{ 是 } B \text{ 的一个前驱}} OUT[P]$  ( $B \neq ENTRY$ )



## 计算支配结点的迭代算法

- 输入：流图 $G$ ， $G$ 的结点集是 $N$ ，边集是 $E$ ，入口结点是 $ENTRY$
- 输出：对于 $N$ 中的各个结点 $n$ ，给出 $D(n)$ ，即支配 $n$ 的所有结点的集合

- 方法：

```
 $OUT[ENTRY] = \{ENTRY\}$   
for(除 $ENTRY$ 之外的每个基本块 $B$ )  
     $OUT[B] = N$   
while(某个 $OUT$ 值发生了改变)  
    for(除 $ENTRY$ 之外的每个基本块 $B$ )  
        {  $IN[B] = \bigcap_{P \text{ 是 } B \text{ 的一个前驱}} OUT[P]$   
           $OUT[B] = IN[B] \cup \{B\}$   
        }
```



例

$OUT[ENTRY] = \{ENTRY\}$

for(除ENTRY之外的每个基本块B)  $OUT[B] = N$

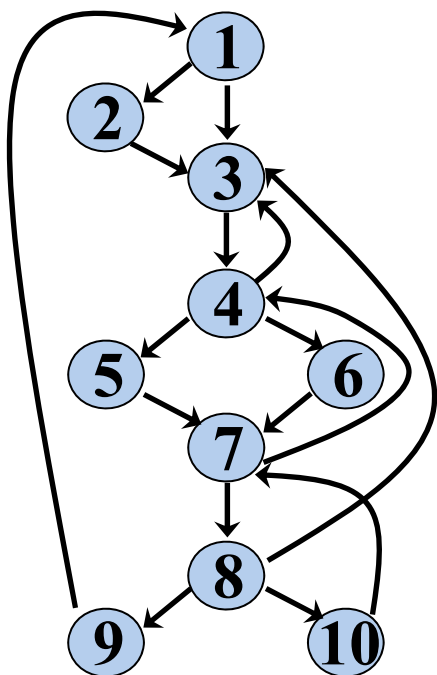
while(某个OUT值发生了改变)

for(除ENTRY之外的每个基本块B)

$\{ IN[B] = \bigcap_{P \text{ 是 } B \text{ 的一个前驱}} OUT[P]$

$OUT[B] = IN[B] \cup \{B\}$

}

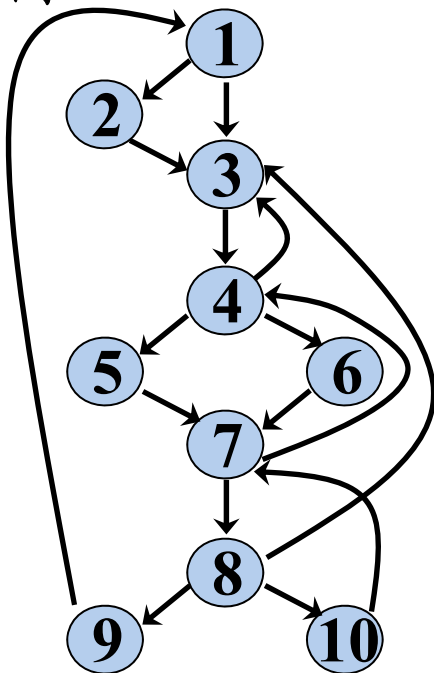


	$OUT^0[B]$	$IN^1[B]$	$OUT^1[B]$
$E$	$\{E\}$		
①	$N$	$\{E\}$	$\{E \text{ ①}\}$
②	$N$	$\{E \text{ ①}\}$	$\{E \text{ ① ②}\}$
③	$N$	$\{E \text{ ①}\}$	$\{E \text{ ① ③}\}$
④	$N$	$\{E \text{ ① ③}\}$	$\{E \text{ ① ③ ④}\}$
⑤	$N$	$\{E \text{ ① ③ ④}\}$	$\{E \text{ ① ③ ④ ⑤}\}$
⑥	$N$	$\{E \text{ ① ③ ④}\}$	$\{E \text{ ① ③ ④ ⑥}\}$
⑦	$N$	$\{E \text{ ① ③ ④}\}$	$\{E \text{ ① ③ ④ ⑦}\}$
⑧	$N$	$\{E \text{ ① ③ ④ ⑦}\}$	$\{E \text{ ① ③ ④ ⑦ ⑧}\}$
⑨	$N$	$\{E \text{ ① ③ ④ ⑦ ⑧}\}$	$\{E \text{ ① ③ ④ ⑦ ⑧ ⑨}\}$
⑩	$N$	$\{E \text{ ① ③ ④ ⑦ ⑧}\}$	$\{E \text{ ① ③ ④ ⑦ ⑧ ⑩}\}$

## 回边 (Back Edges)

➤ 假定流图中存在两个结点  $d$  和  $n$  满足  $d \text{ dom } n$ 。如果存在从结点  $n$  到  $d$  的有向边  $n \rightarrow d$ ，那么这条边称为 **回边**

➤ 例



	$OUT^0[B]$	$IN^I[B]$	$OUT^I[B]$
$E$	$E$		
①	$N$	$E$	$E$ ①
②	$N$	$E$ ①	$E$ ① ②
③	$N$	$E$ ①	$E$ ① ③
④	$N$	$E$ ① ③	$E$ ① ③ ④
⑤	$N$	$E$ ① ③ ④	$E$ ① ③ ④ ⑤
⑥	$N$	$E$ ① ③ ④	$E$ ① ③ ④ ⑥
⑦	$N$	$E$ ① ③ ④	$E$ ① ③ ④ ⑦
⑧	$N$	$E$ ① ③ ④ ⑦	$E$ ① ③ ④ ⑦ ⑧
⑨	$N$	$E$ ① ③ ④ ⑦ ⑧	$E$ ① ③ ④ ⑦ ⑧ ⑨
⑩	$N$	$E$ ① ③ ④ ⑦ ⑧	$E$ ① ③ ④ ⑦ ⑧ ⑩

回边:

4→3

7→4

8→3

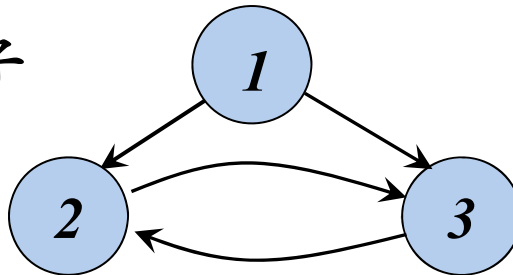
9→1

10→7

## 自然循环 (Natural Loops)

自然循环是一种适合于优化的循环

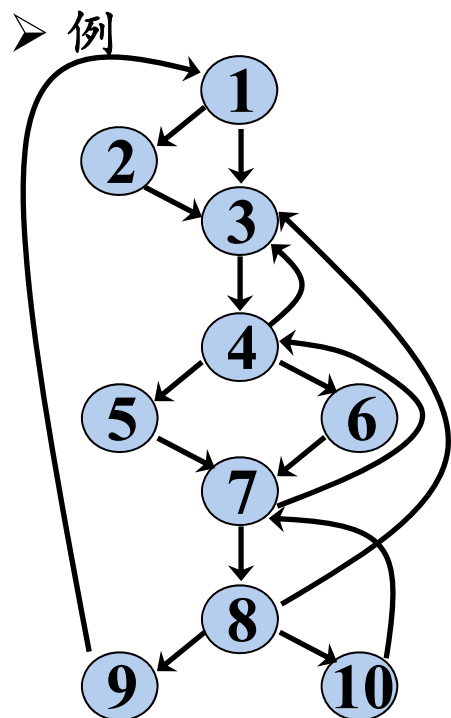
- 从程序分析的角度来看，循环在代码中以什么形式出现并不重要，重要的是它是否具有易于优化的性质
- 自然循环是满足以下性质的循环
  - 有唯一的入口结点，称为首结点(header)。首结点支配循环中的所有结点，否则，它就不会成为循环的唯一入口
  - 循环中至少有一条返回首结点的路径，否则，控制就不可能从“循环”中直接回到循环头，也就无法构成循环
- 非自然循环的例子



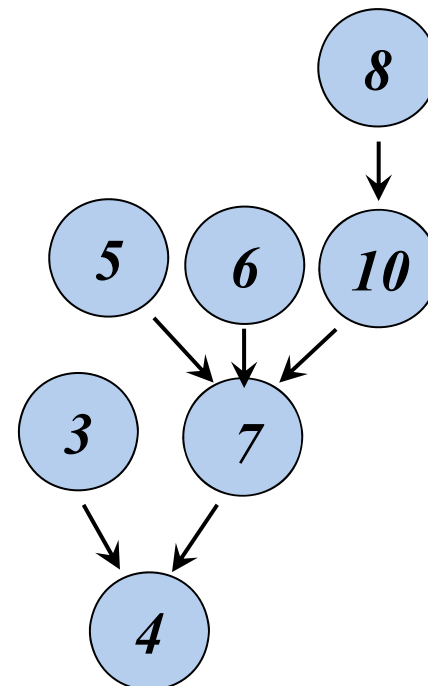


## 自然循环的识别

- 给定一个回边  $n \rightarrow d$ ，该回边的自然循环为： $d$ ，以及所有可以不经过  $d$  而到达  $n$  的结点。 $d$  为该循环的首结点。



回边	自然循环
$4 \rightarrow 3$	③④⑤⑥⑦⑧⑩
$7 \rightarrow 4$	④⑤⑥⑦⑧⑩
$8 \rightarrow 3$	③④⑤⑥⑦⑧⑩
$9 \rightarrow 1$	① ~ ⑩
$10 \rightarrow 7$	⑦⑧⑩



## 算法：构造一条回边的自然循环

- 输入：流图  $G$  和回边  $n \rightarrow d$
- 输出：由回边  $n \rightarrow d$  的自然循环中的所有结点组成的集合
- 方法：

```
 $stack = \Phi; loop = \{ n, d \}; push(stack, n);$   
while  $stack$  不空 do  
  {  $m = top(stack); pop(stack);$   
    for  $m$  的每个前驱  $p$   
      { if  $p$  不在  $loop$  中 then  
        {  $loop = loop \cup \{ p \}; push(stack, p);$  }  
      }  
  }
```

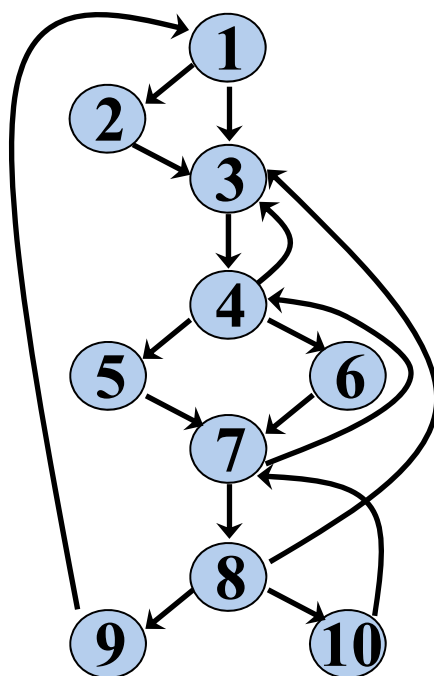
结点  $d$  在初始时刻已经在  $loop$  中，不会去考虑它的前驱。因此，找出的都是不经过  $d$  而能到达  $n$  的结点。



➤ 自然循环的一个重要性质

- 除非两个自然循环的首结点相同，否则，它们或者互不相交，或者一个完全包含(嵌入)在另外一个里面

➤ 例

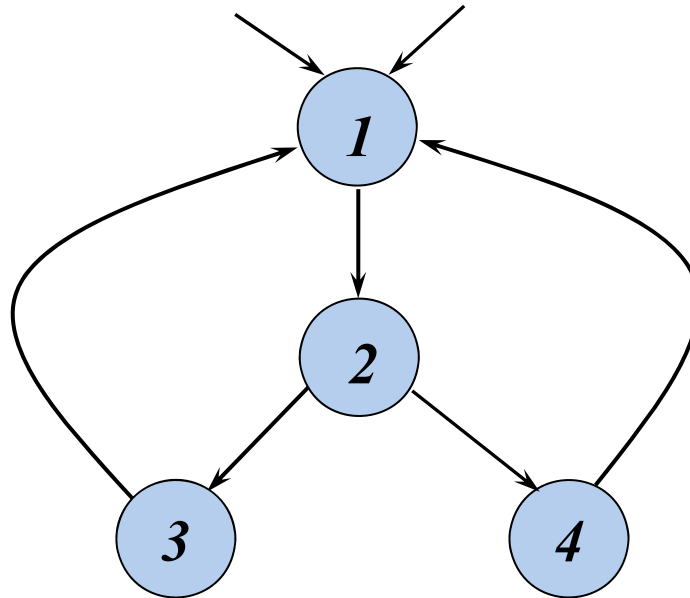


回边	自然循环
4->3	③④⑤⑥⑦⑧⑩
7->4	④⑤⑥⑦⑧⑩
8->3	③④⑤⑥⑦⑧⑩
9->1	① ~ ⑩
10->7	⑦⑧⑩

**最内循环 (Innermost Loops):**  
不包含其它循环的循环



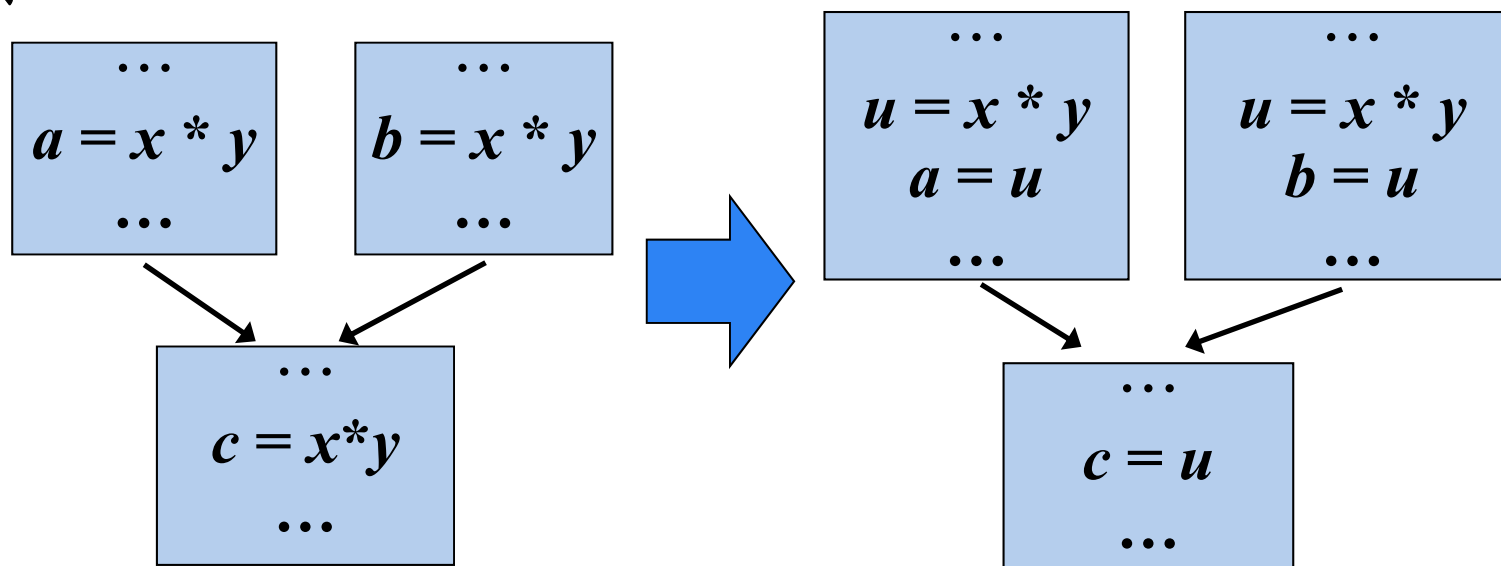
➤ 如果两个循环具有相同的首结点，那么很难说哪个是最内循环。此时把两个循环合并



## 删除全局公共子表达式

- 可用表达式的数据流问题可以帮助确定位于流图中 $p$ 点的表达式是否为全局公共子表达式

➤ 例



## 全局公共子表达式删除算法

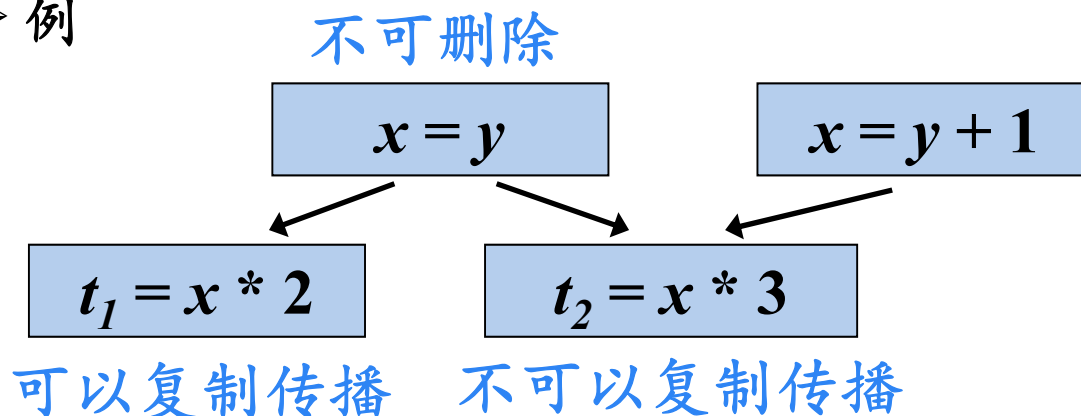
- 输入：带有可用表达式信息的流图
- 输出：修正后的流图
- 方法：
  - 对于语句  $s: z = x \text{ op } y$ ，如果  $x \text{ op } y$  在  $s$  之前可用，那么执行如下步骤：
    - ① 从  $s$  开始逆向搜索，但不穿过任何计算了  $x \text{ op } y$  的块，找到所有离  $s$  最近的计算了  $x \text{ op } y$  的语句
    - ② 建立新的临时变量  $u$
    - ③ 把步骤①中找到的语句  $w = x \text{ op } y$  用下列语句代替：

$u = x \text{ op } y$   
 $w = u$
    - ④ 用  $z = u$  替代  $s$

## 删除复制语句

- 对于复制语句  $s: x=y$ , 如果在  $x$  的所有引用点都可以用对  $y$  的引用代替对  $x$  的引用(复制传播), 那么可以删除复制语句  $x=y$

➤ 例



- 在  $x$  的引用点  $u$  用  $y$  代替  $x$  (复制传播) 的条件
  - 复制语句  $s: x=y$  在  $u$  点 “可用”

## 删除复制语句的算法

- 输入：流图 $G$ 、 $du$ 链、各基本块 $B$ 入口处的可用复制语句集合
- 输出：修改后的流图
- 方法：
  - 对于每个复制语句 $x=y$ ，执行下列步骤
    - ① 根据 $du$ 链找出该定值所能够到达的那些对 $x$ 的引用
    - ② 确定是否对于每个这样的引用， $x=y$ 都在 $IN[B]$ 中( $B$ 是包含这个引用的基本块)，并且 $B$ 中该引用的前面没有 $x$ 或者 $y$ 的定值
    - ③ 如果 $x=y$ 满足第②步的条件，删除 $x=y$ ，且把步骤①中找到的对 $x$ 的引用用 $y$ 代替



## 代码移动

- 循环不变计算的检测
- 代码外提

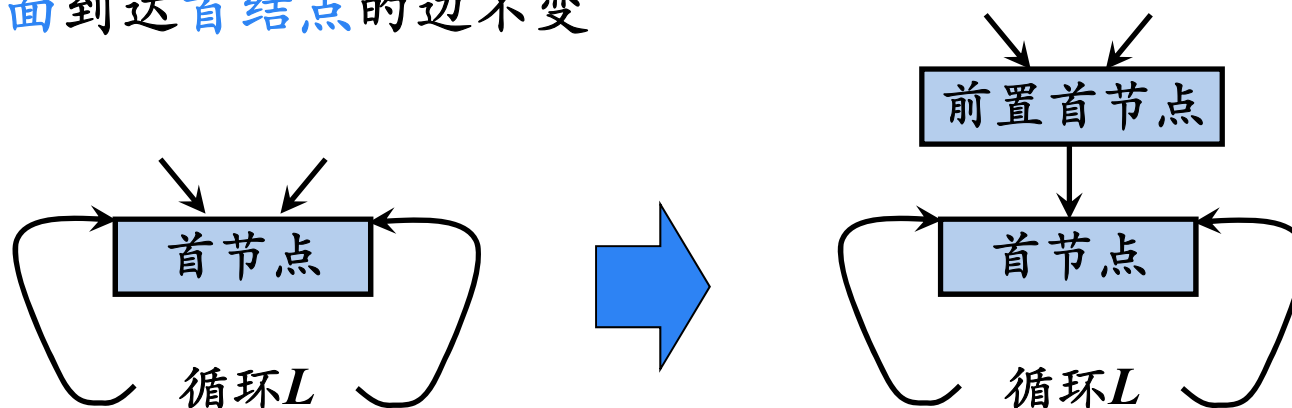
## 循环不变计算检测算法

- 输入：循环 $L$ ，每个三地址指令的 $ud$ 链
- 输出： $L$ 的循环不变计算语句
- 方法
  1. 将下面这样的语句标记为“不变”：语句的运算分量或者是常数，或者其所有定值点都在循环 $L$ 外部
  2. 重复执行步骤(3)，直到某次没有新的语句可标记为“不变”为止
  3. 将下面这样的语句标记为“不变”：先前没有被标记过，且所有运算分量或者是常数，或者其所有定值点都在循环 $L$ 外部，或者只有一个到达定值，该定值是循环中已经被标记为“不变”的语句

## 代码外提

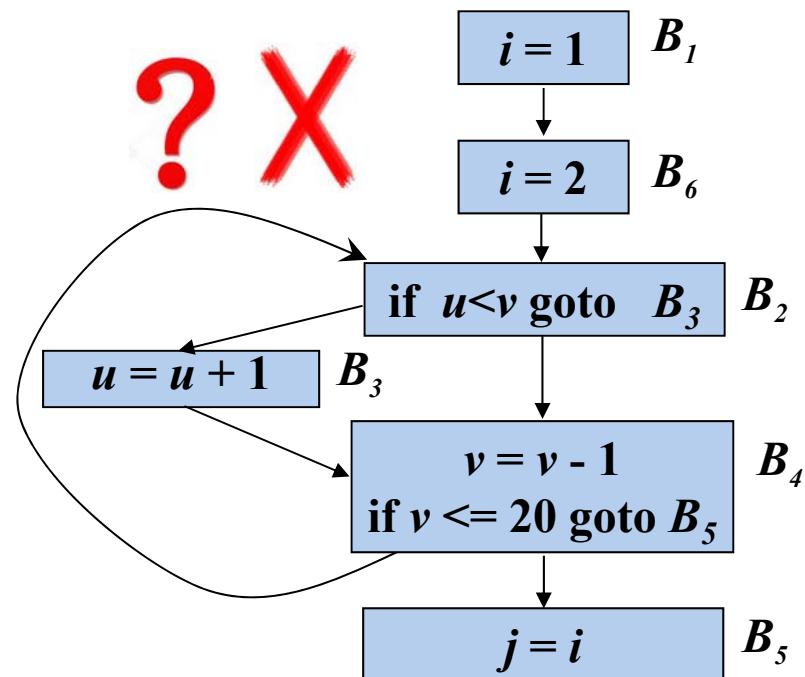
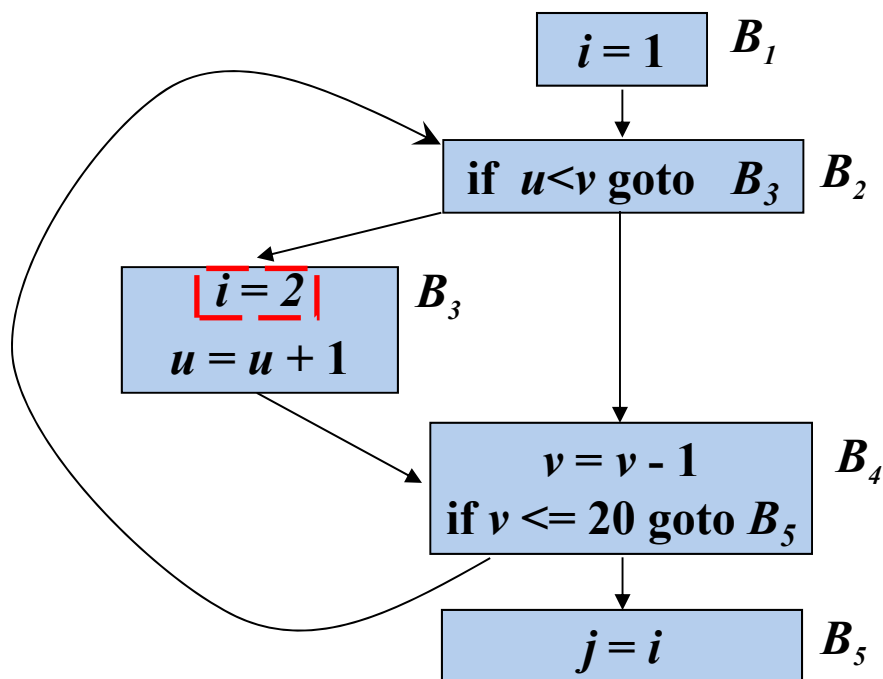
### ➤ 前置首结点 (*preheader*)

- 循环不变计算将被移至首结点之前，为此创建一个称为**前置首结点**的**新块**。前置首结点的**唯一后继**是**首结点**，并且原来从**循环L**外到达**L首结点**的边都改成进入**前置首结点**。从**循环L**里面到达**首结点**的边不变



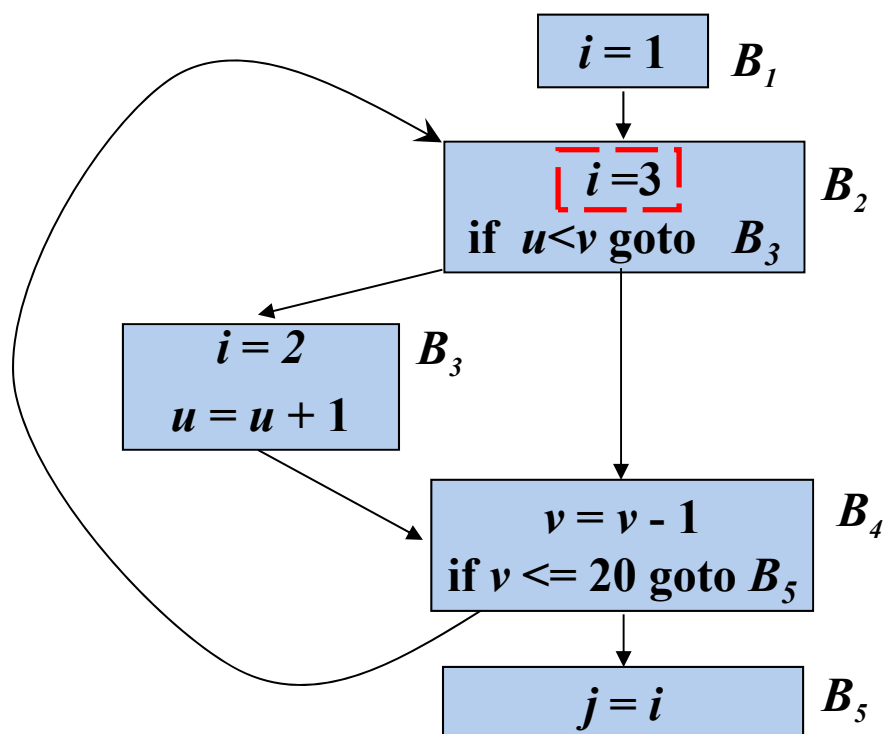
## 循环不变计算语句 $s : x = y + z$ 移动的条件

(1)  $s$ 所在的基本块是循环所有出口结点(有后继结点在循环外的结点)的支配结点



## 循环不变计算语句 $s : x = y + z$ 移动的条件

(2) 循环中 没有其它语句对  $x$  赋值



➤ 外提前

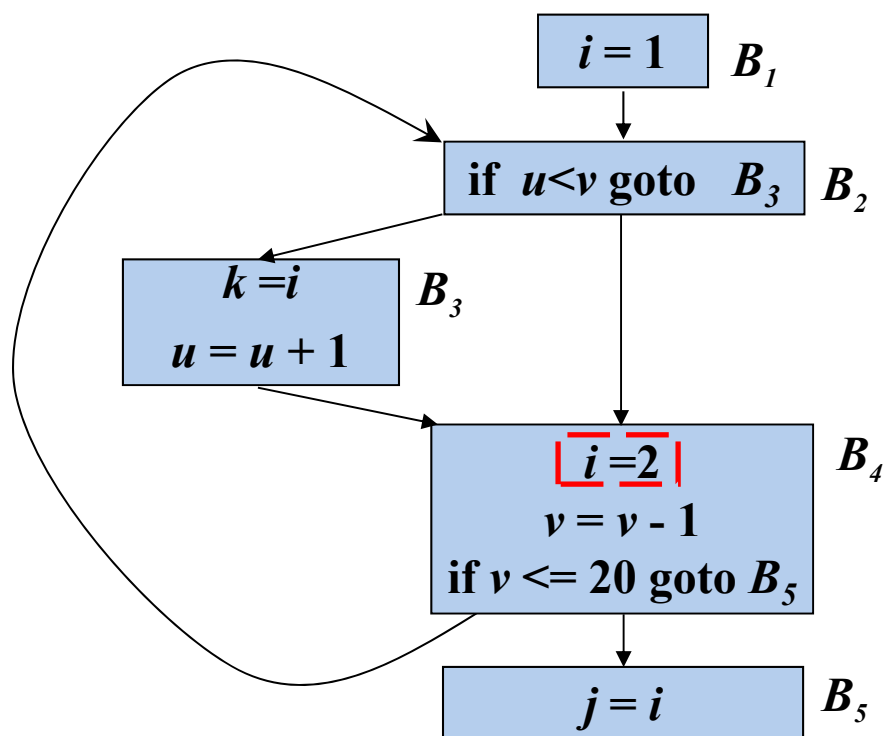
➤  $j$  的值是否等于 2 取决于循环最后一次迭代时，是否执行了  $B_3$

➤ 外提后

➤ 只要  $B_3$  执行过一次， $j$  的值就等于 2

## 循环不变计算语句 $s : x = y + z$ 移动的条件

(3) 循环中对 $x$ 的引用仅由 $s$ 到达



➤ 外提前

➤  $k$  的值有可能等于1,  
也可能等于2

➤ 外提后

➤  $k$  的值只能等于2

## 代码移动算法

➤ 输入：循环 $L$ 、 $ud$ 链、支配结点信息

➤ 输出：修改后的循环

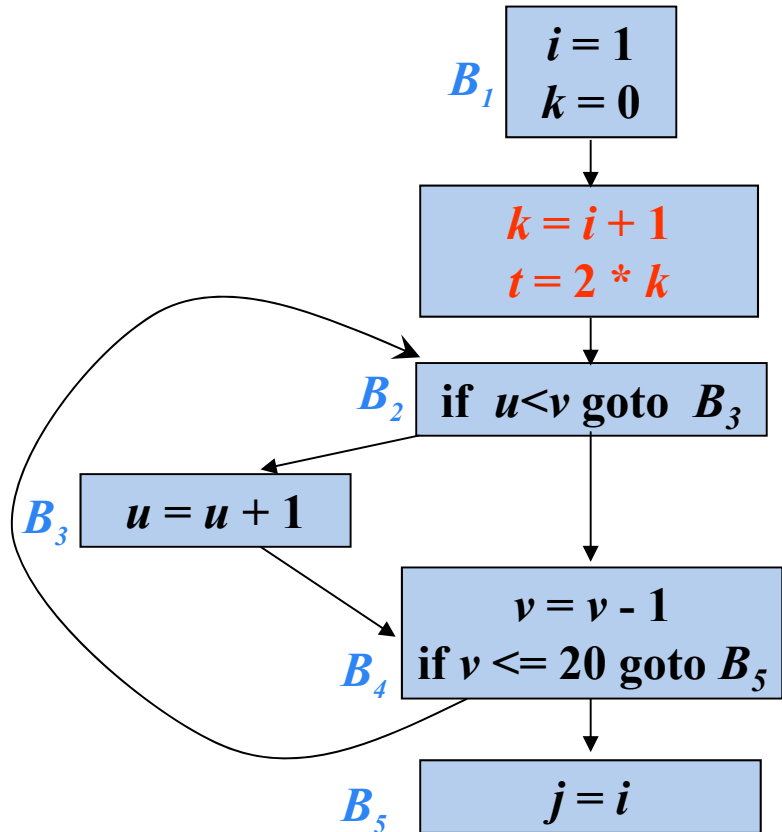
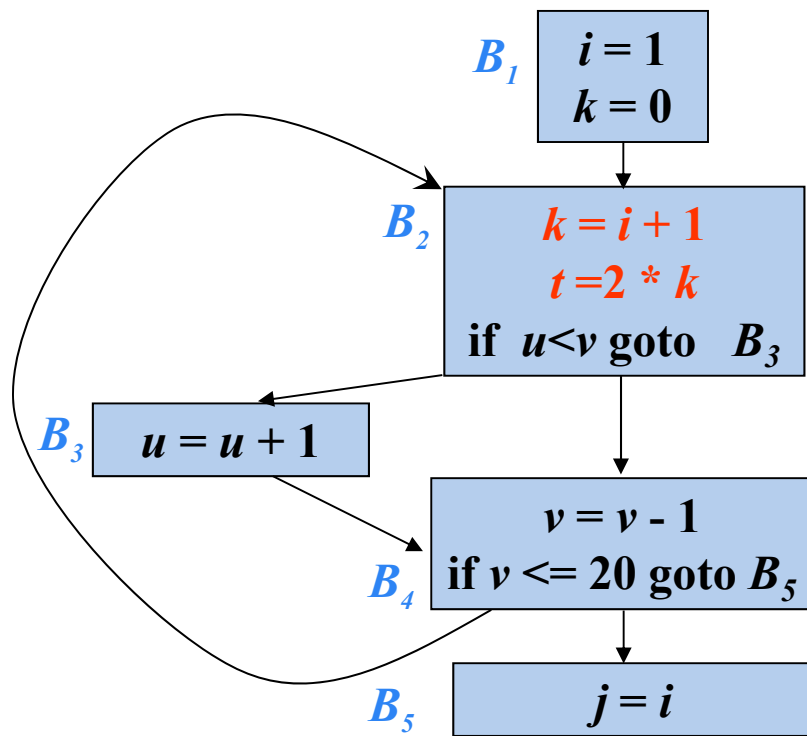
➤ 方法：

1. 寻找循环不变计算

2. 对于步骤(1)中找到的每个循环不变计算，检查是否满足上面的三个条件

3. 按照循环不变计算找出的次序，把所找到的满足上述条件的循环不变计算外提到前置首结点中。如果循环不变计算有分量在循环中定值，只有将定值点外提后，该循环不变计算才可以外提

# 例





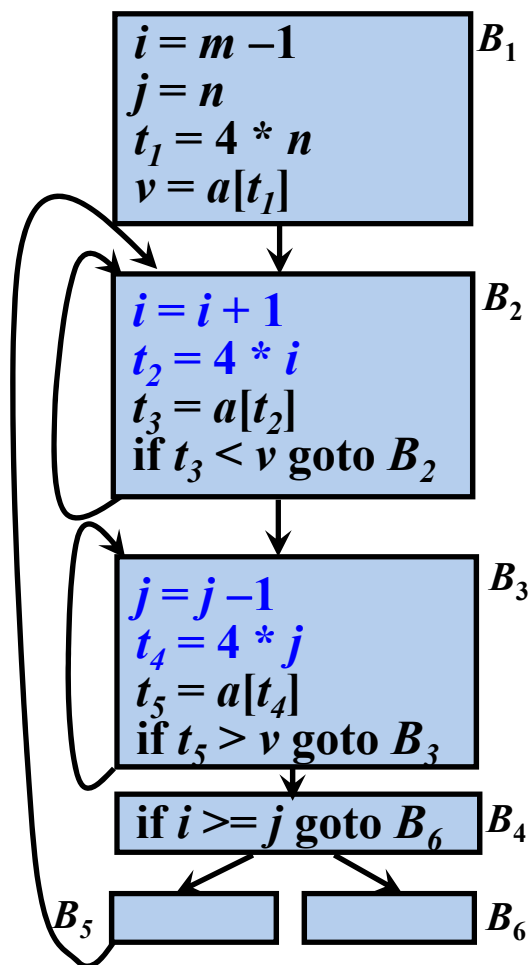
## 作用于归纳变量的强度削弱

- 对于一个变量 $x$ ，如果存在一个正的或负的常量 $c$ ，使得每次 $x$ 被赋值时，它的值总是增加 $c$ ，则称 $x$ 为归纳变量
- 如果循环 $L$ 中的变量 $i$ 只有形如 $i = i + c$ 的定值( $c$ 是常量)，则称 $i$ 为循环 $L$ 的基本归纳变量
- 如果 $j = c \times i + d$ ，其中 $i$ 是基本归纳变量， $c$ 和 $d$ 是常量，则 $j$ 也是一个归纳变量，称 $j$ 属于 $i$ 族
- 基本归纳变量 $i$ 属于它自己的族

## 作用于归纳变量的强度削弱

- 对于一个变量 $x$ ，如果存在一个正的或负的常量 $c$ ，使得每次 $x$ 被赋值时，它的值总是增加 $c$ ，则称 $x$ 为归纳变量
- 如果循环 $L$ 中的变量 $i$ 只有形如 $i = i + c$ 的定值( $c$ 是常量)，则称 $i$ 为循环 $L$ 的基本归纳变量
- 如果 $j = c \times i + d$ ，其中 $i$ 是基本归纳变量， $c$ 和 $d$ 是常量，则 $j$ 也是一个归纳变量，称 $j$ 属于 $i$ 族
- 每个归纳变量都关联一个三元组。如果 $j = c \times i + d$ ，其中 $i$ 是基本归纳变量， $c$ 和 $d$ 是常量，则与 $j$ 相关联的三元组是 $(i, c, d)$

# 例



$i$ 是基本归纳变量

$t_2$ 是 $i$ 族归纳变量, 可以表示为 $(i, 4, 0)$

$j$ 是基本归纳变量

$t_4$ 是 $j$ 族归纳变量, 可以表示为 $(j, 4, 0)$

## 归纳变量检测算法

- 输入：带有循环不变计算信息和到达定值信息的循环 $L$
- 输出：一组归纳变量
- 方法：
  1. 扫描 $L$ 的语句，找出所有基本归纳变量。在此要用到循环不变计算信息。与每个基本归纳变量 $i$ 相关联的三元组是 $(i, 1, 0)$

## 归纳变量检测算法 (续)

2: 寻找 $L$ 中只有一次定值的变量 $k$ ，它具有下面的形式:

$k=c'\times j+d'$ 。其中 $c'$ 和 $d'$ 是常量， $j$ 是基本的或非基本的归纳变量

- 如果 $j$ 是基本归纳变量，那么 $k$ 属于 $j$ 族。 $k$ 对应的三元组可以通过其定值语句确定
- 如果 $j$ 不是基本归纳变量，假设其属于 $i$ 族， $k$ 的三元组可以通过 $j$ 的三元组和 $k$ 的定值语句来计算，此时我们还要求:
  - 循环 $L$ 中对 $j$ 的唯一一定值和对 $k$ 的定值之间没有对 $i$ 的定值
  - 循环 $L$ 外没有 $j$ 的定值可以到达 $k$

这两个条件是为了保证对 $k$ 进行赋值的时候， $j$ 当时的值一定等于 $c*(i当时的值)+d$

$$j=c\times i+d$$

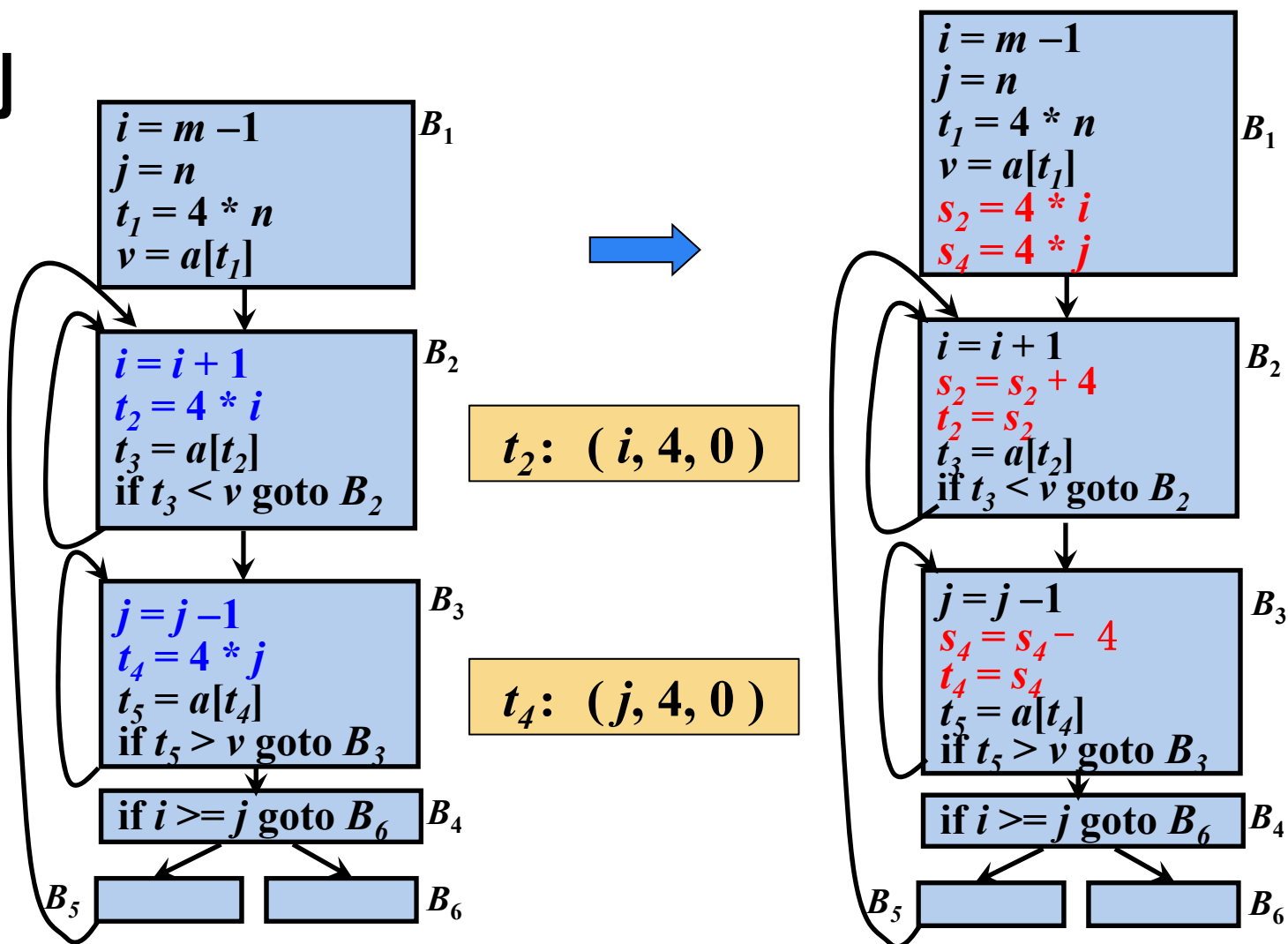
$$k=c'\times j+d'$$

## 作用于归纳变量的强度削弱算法

- 输入：带有到达定值信息和已计算出的归纳变量族的循环 $L$
- 输出：修改后的循环
- 方法：对于每个基本归纳变量 $i$ ，对其族中的每个归纳变量 $j$ ： $(i, c, d)$   
执行下列步骤
  1. 建立新的临时变量 $t$ 。如果变量 $j_1$ 和 $j_2$ 具有相同的三元组，则只为它们建立一个新变量
  2. 用 $j=t$ 代替对 $j$ 的赋值
  3. 在 $L$ 中紧跟定值 $i=i+n$ 之后，添加 $t=t+c*n$ 。将 $t$ 放入 $i$ 族，其三元组为 $(i, c, d)$
  4. 在前置节点的末尾，添加语句 $t=c*i$ 和 $t=t+d$ ，使得在循环开始的时候 $t=c*i+d=j$



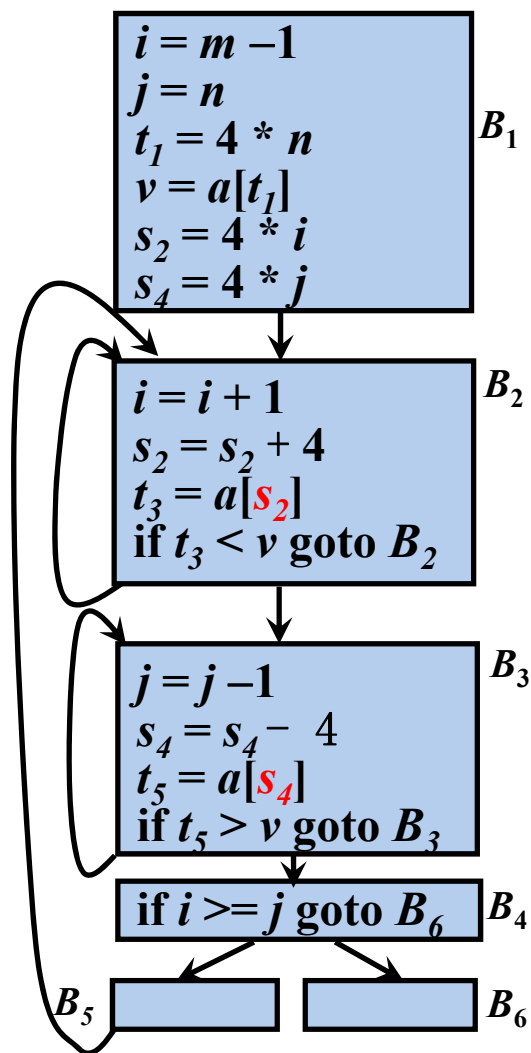
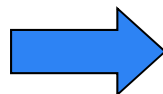
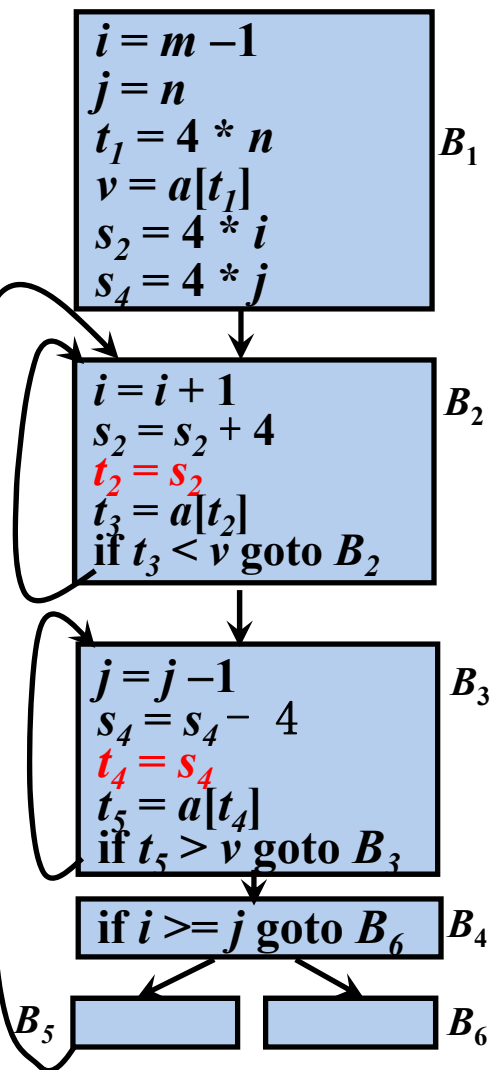
例



## 归纳变量的删除

- 对于在强度削弱算法中引入的复制语句 $j=t$ ，如果在归纳变量 $j$ 的所有引用点都可以用对 $t$ 的引用代替对 $j$ 的引用，并且 $j$ 在循环的出口处不活跃，则可以删除复制语句 $j=t$





## 归纳变量的删除

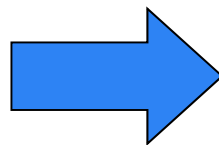
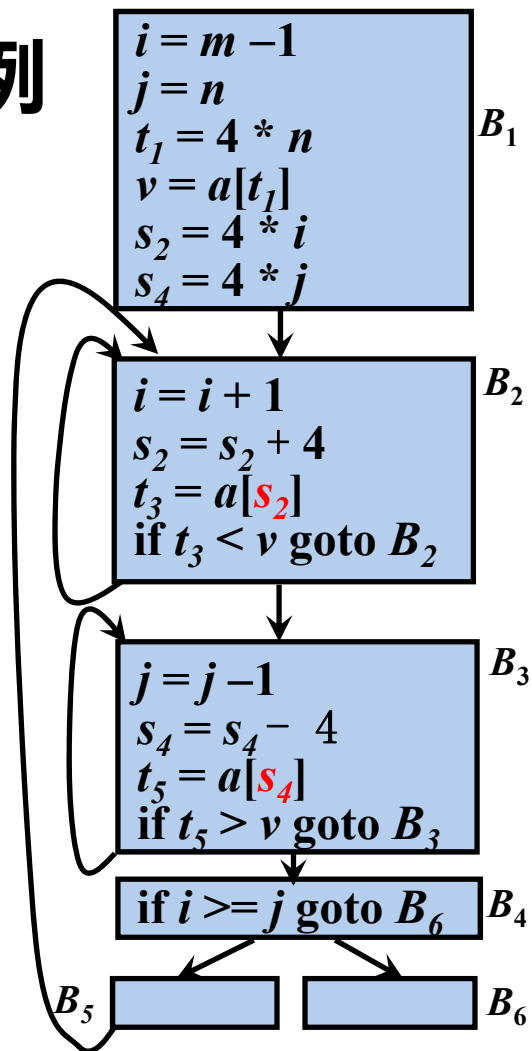
- 对于在强度削弱算法中引入的复制语句 $j=t$ ，如果在归纳变量 $j$ 的所有引用点都可以用对 $t$ 的引用代替对 $j$ 的引用，并且 $j$ 在循环的出口处不活跃，则可以删除复制语句 $j=t$
- 强度削弱后，有些归纳变量的作用只是用于测试。如果可以用对其它归纳变量的测试代替对这种归纳变量的测试，那么可以删除这种归纳变量

## 删除仅用于测试的归纳变量

- 对于仅用于测试的基本归纳变量 $i$ ，取 $i$ 族的某个归纳变量 $j$  (尽量使得 $c$ 、 $d$ 简单，即 $c=1$ 或 $d=0$ 的情况)。把每个对 $i$ 的测试替换成为对 $j$ 的测试
- $(relop\ i\ x\ B)$  替换为  $(relop\ j\ c*x+d\ B)$ ，其中 $x$ 不是归纳变量，并假设 $c>0$
- $(relop\ i_1\ i_2\ B)$ ，如果能够找到三元组 $j_1(i_1, c, d)$ 和 $j_2(i_2, c, d)$ ，那么可以将其替换为 $(relop\ j_1\ j_2\ B)$  (假设 $c>0$ )。否则，测试的替换可能是没有价值的
- 如果归纳变量 $i$ 不再被引用，那么可以删除和它相关的指令



例



$s_2: (i, 4, 0)$   
 $s_4: (j, 4, 0)$

