第10章(Part 1)　代码优化

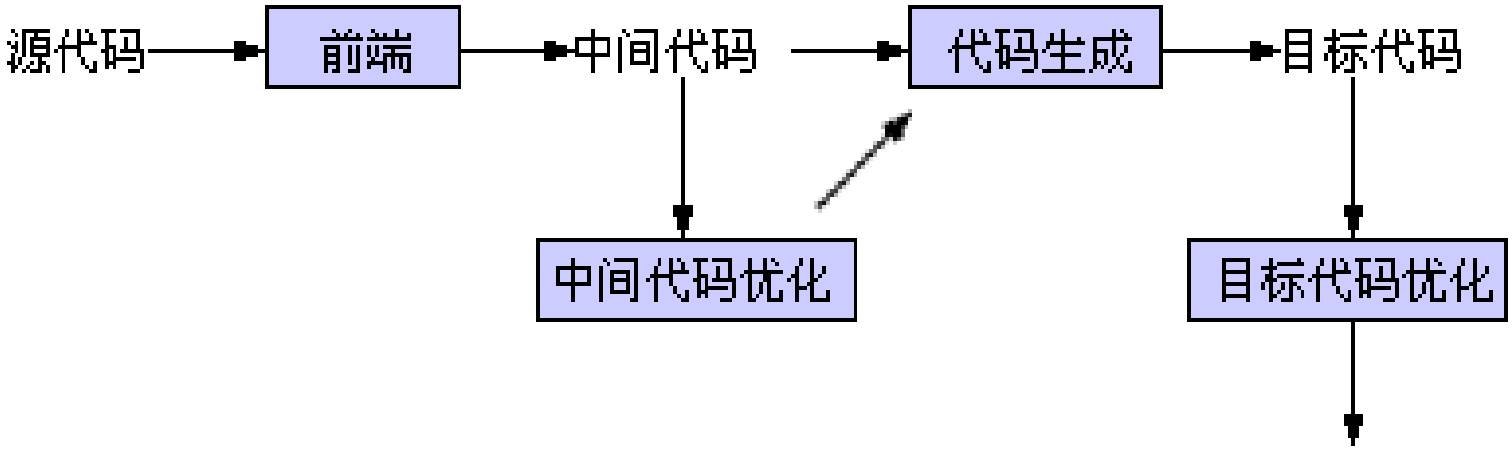
1 代码优化技术简介

某些编译程序在中间代码或目标代码生成之后要对生成的代码进行优化。

所谓**优化**，实质上是对代码进行等价变换，使得变换后的代码运行结果与变换前代码运行结果相同，而运行速度加大或占用存储空间少，或两者都有。

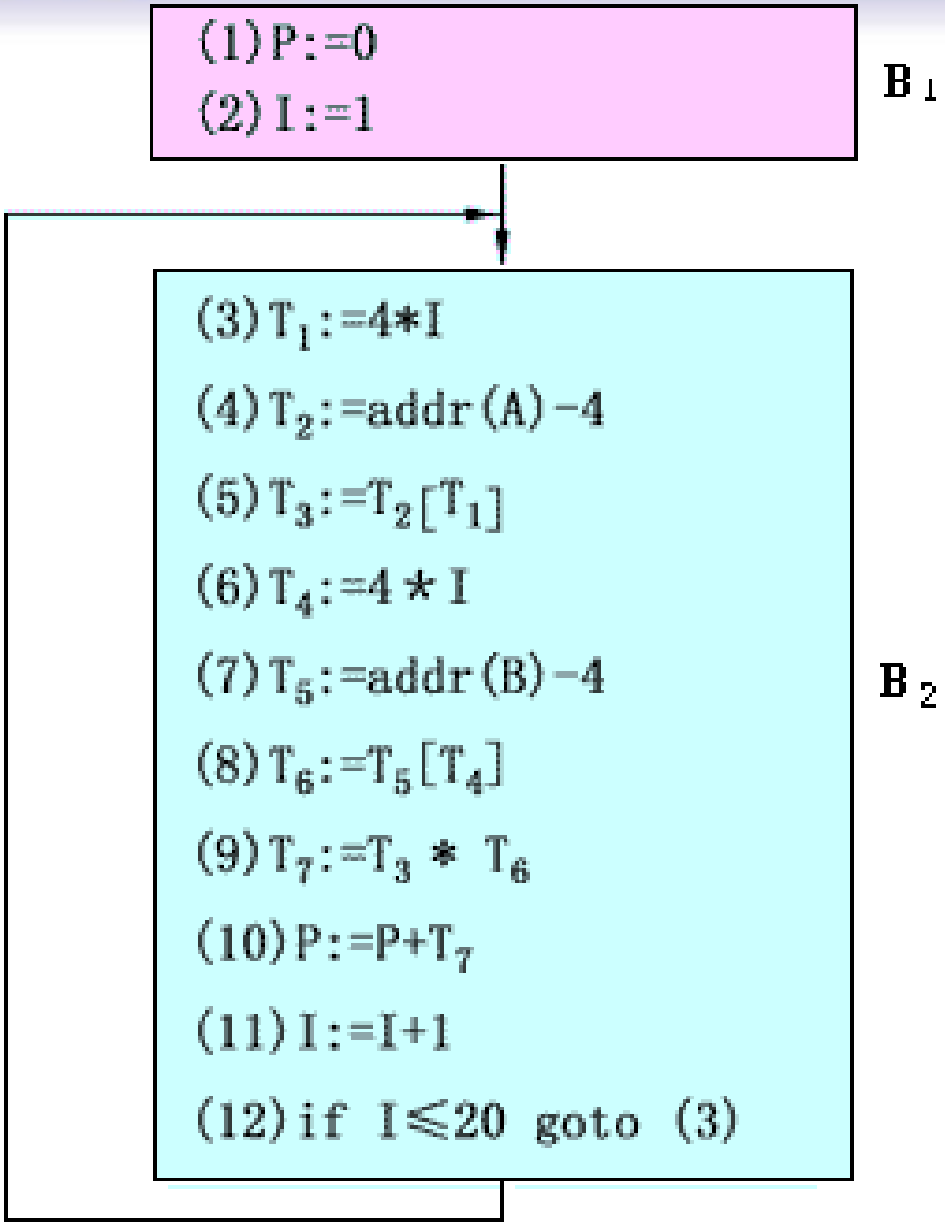
优化可在编译的不同阶段进行，对同一阶段，涉及的程序范围也有所不同，在同一范围内，可进行多种优化。

编译的优化工作阶段



给定下面的源程序：

```
P:=0
for I:=1 to 20 do
    P:=P+A[I]*B[I];
```



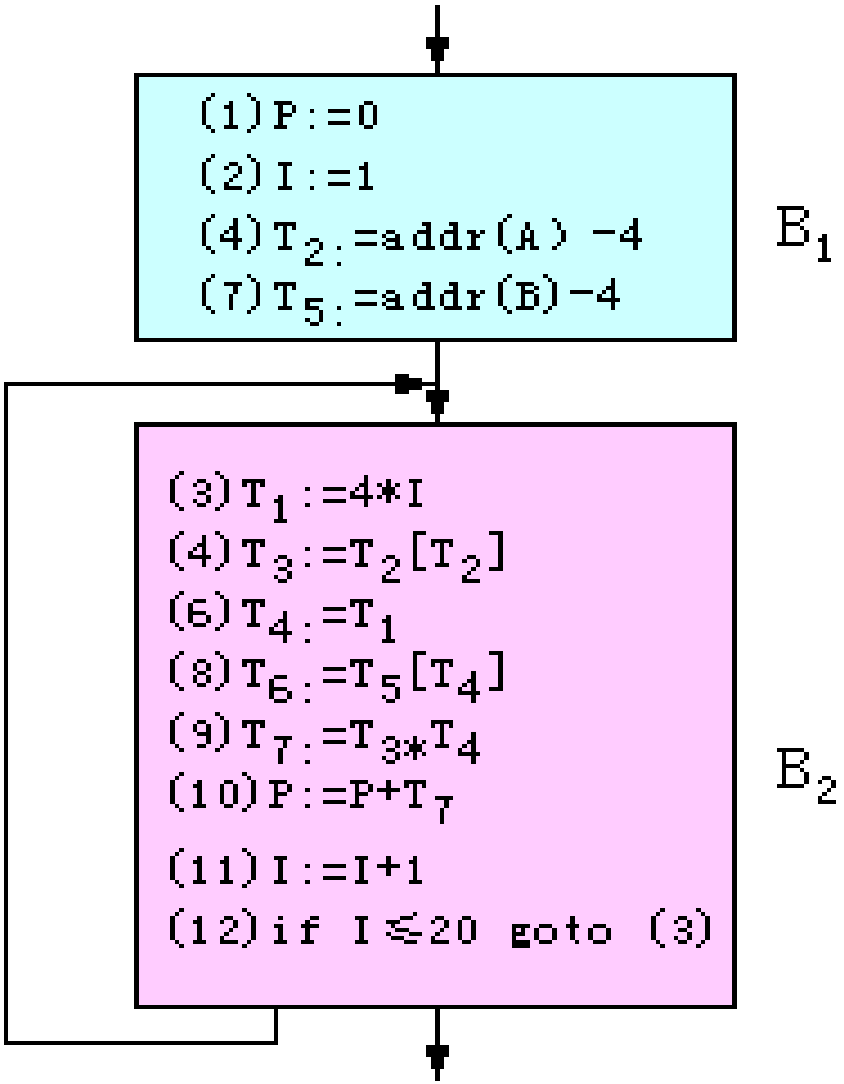
删除多余运算（删除公共子表达式）

优化的目的在于使生成的目标代码较少而执行速度较快。

代码外提

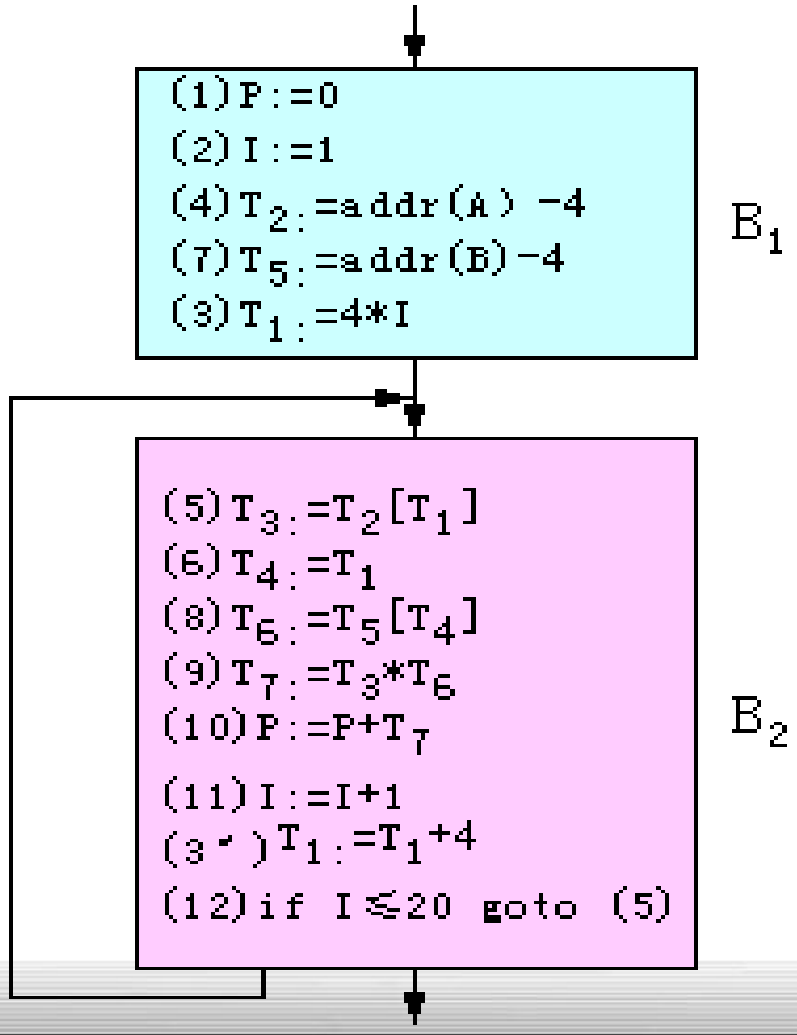
减少循环中代码总数的一个重要办法是循环不变代码外提。这种变换把循环不变运算，即其结果独立于循环执行次数的表达式，提到循环的前面,使之只在循环外计算一次。

删除公共子表达式和代码外提



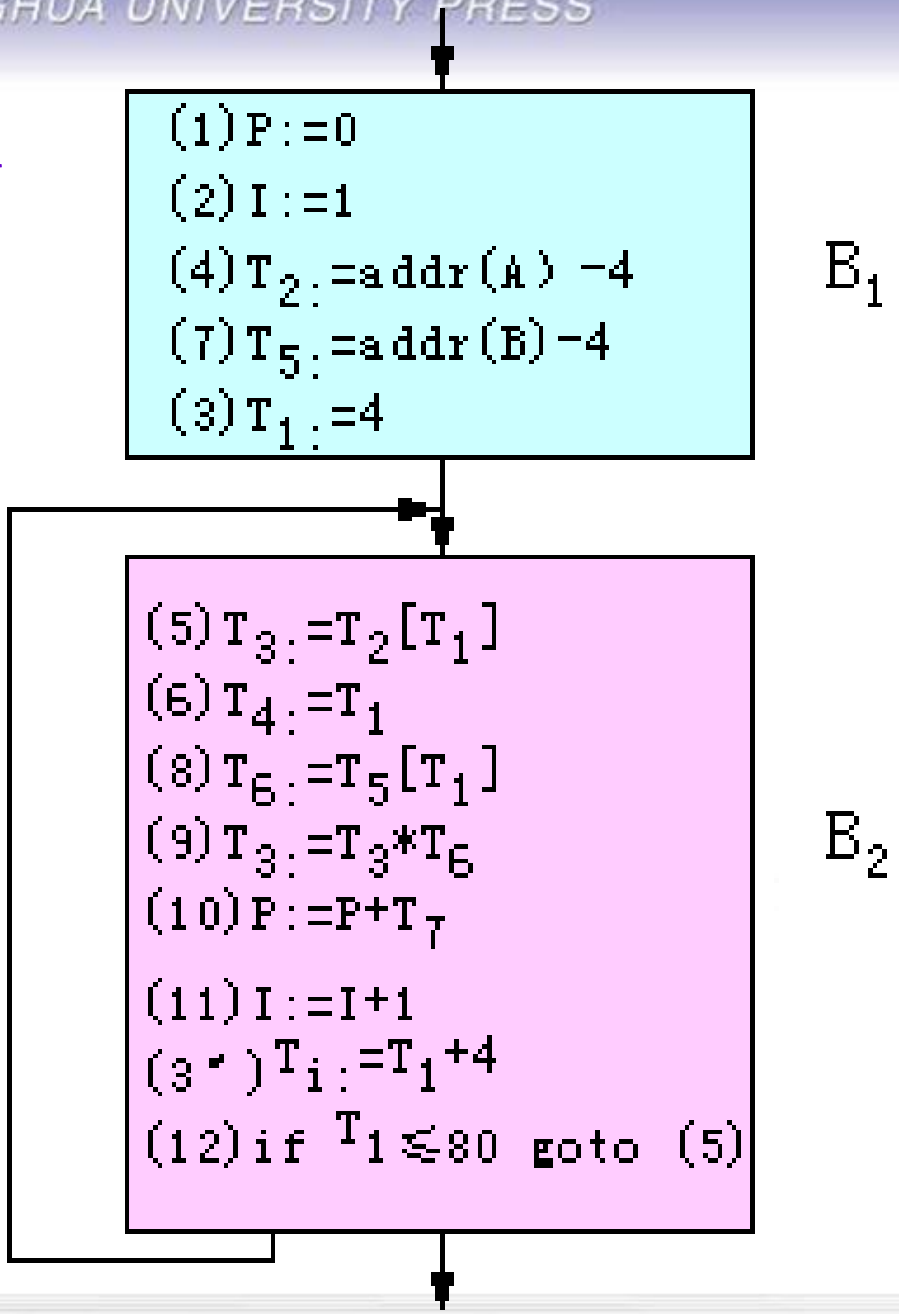
强度削弱

强度削弱的思想是把强度大的运算换算成强度小的运算。如把乘法运算换成加法运算等。

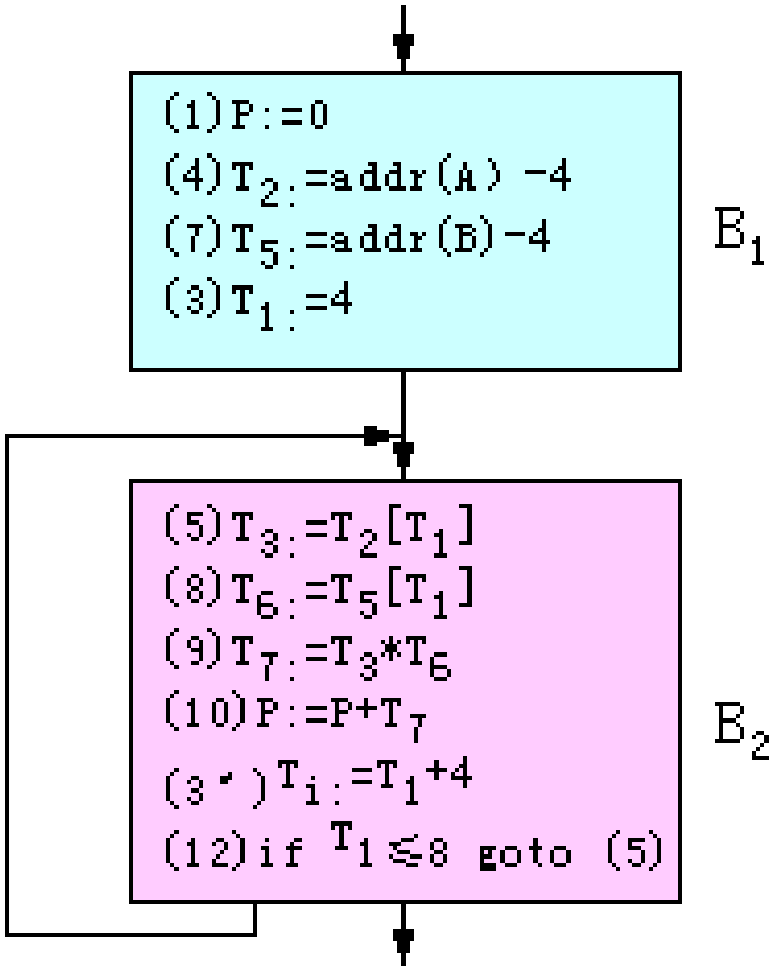


变换循环控制条件

合并已知量与复写传播



删除无用赋值


(1) P:=0
(4) T2:=addr(A)-4
(7) T5:=addr(B)-4
(3) T1:=4
B1
(5) T3:=T2[T1]
(8) T6:=T5[T1]
(9) T7:=T3*T6
(10) P:=P+T7
(3') Ti:=T1+4
(12) if T1≤8 goto (5)
B2

2 局部优化

局部优化是指基本块内的优化。

所谓**基本块**，是指程序中一个顺序执行的语句序列，其中只有一个入口语句和一个出口语句。

控制流只能从其入口语句进入，从其出口语句退出，没有中途停止或分支。**对于一个给定的程序，把它划分为一系列的基本块，在各个基本块范围内分别进行优化。**

基本块的划分

基本块的**入口语句**：

① 程序的第一个语句；或者，

② 条件转移语句或无条件转移语句的转移目标语句；或者，

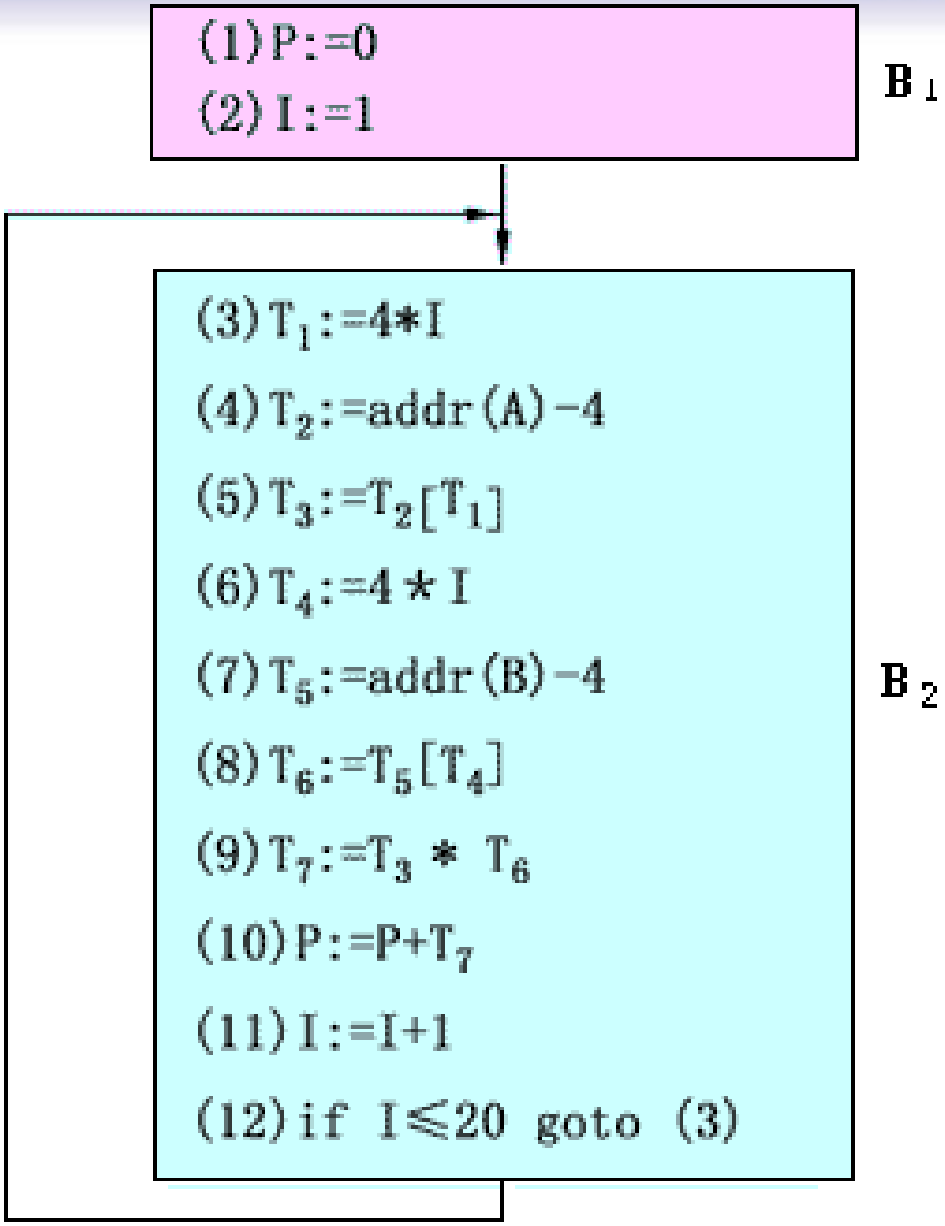
③ 紧跟在条件转移语句后面的语句。

划分中间代码为基本块的算法步骤如下：

① 求出四元式程序中各个基本块的入口语句。

② 对每一入口语句，构造其所属的基本块。它是由该入口语句到下一入口语句（不包括下一入口语句），或到一转移语句（包括该转移语句），或到一停语句（包括该停语句）之间的语句序列组成的。

③ 凡未被纳入某一基本块的语句，都是程序中控制流程无法到达的语句，因而也是不会被执行到的语句，可以把它们删除。


(1) P:=0
(2) I:=1
B1
(3) T1:=4*I
(4) T2:=addr(A)-4
(5) T3:=T2[T1]
(6) T4:=4*I
(7) T5:=addr(B)-4
(8) T6:=T5[T4]
(9) T7:=T3 * T6
(10) P:=P+T7
(11) I:=I+1
(12) if I≤20 goto (3)
B2

基本块的变换

很多变换可作用于基本块而不改变它计算的表达式集合，这样的变换对改进代码的质量是很有用的。有两类重要的局部等价变换可用于基本块；它们是**保结构的变换**和**代数变换**。

基本块的主要保结构变换：

① 删除公共子表达式
② 删除无用代码
③ 重新命名临时变量
④ 交换语句次序

若有语句t:=b+c，其中t是临时变量。如果把这个语句改为u:=b+c，其中u是新的临时变量，并且把这个t的所有引用改成u，那么**基本块的运算结果不变**。

如果基本块有两个相邻的语句：

t1:=b+c

t2:=x+y

当且仅当x和y都不是t1，b和c都不是t2时，我们可以交换这两个语句的次序。

代数变换可以把基本块计算的表达式集合变换成代数等价的集合。其中常用的变换是那些可以**简化表达式**或**用较快运算代替较慢运算**的变换。

如x:=x+0或x:=x*1可删除。

又如x:=y2**

指数算符通常要用函数调用来实现。若使用代数变换，这个语句可由快速、等价的语句

x:=y*y来代替。

基本块的DAG表示

在一个有向图中，称任一有向边ni→nj（或表示为有序对（ni，nj））中的结点ni为结点nj的前驱（父结），结点nj为结点ni的后继（子结）。又称任一有向边序列n1→n2，n2→n3，…，nk-1→nk为**从结点n1到结点nk的一条通路**。

如果其中n1 = nk，则称该通路为**环路**。该结点序列也记为（n1，n2，…，nk）。如果有向图中任一通路都不是环路，则称该有向图为无环路有向图，简称**DAG**。

考虑一种其结点带有下述标记或附加信息的DAG：

① **图的叶结点**，即无后继的结点，以一标识符（变量名）或常数作为标记，表示这个结点代表该变量或常数的值。如果叶结点用来代表某变量A的地址，则用addr（A）作为这个结点的标记。通常把叶结点上作为标记的标识符加上下标0，以表示它是该变量的初值。

② **图的内部结点**，即有后继的结点，以一运算符作为标记，结点代表应用该运算符对其后继结点所代表的值进行运算的结果。

③ 图中各个结点上可能附加一个或多个标识符，表示这些变量具有该结点所代表的值。

上述DAG可用来描述计算过程，又称为描述计算过程的DAG。

四元式与DAG结点

四元式

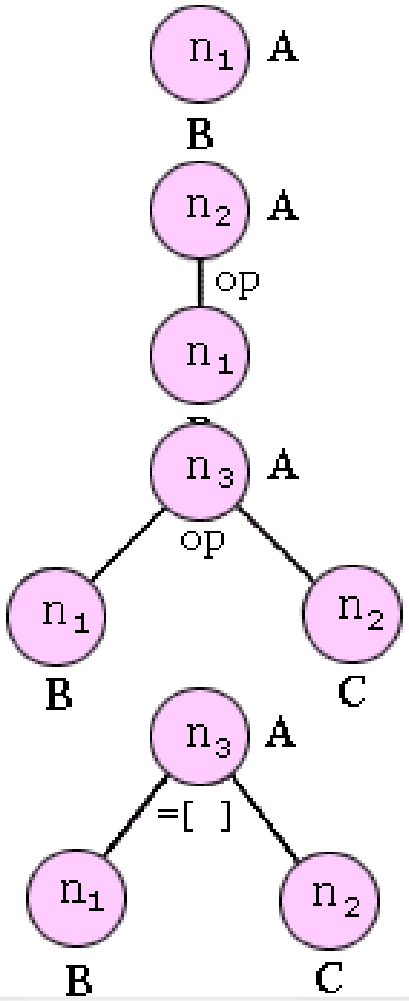
0型 (0) A: =B (: =, B, -, A)

1型 (1) A: =op B (op, B, -, A)

2型 (2) A: =B op C (op, B, C, A)

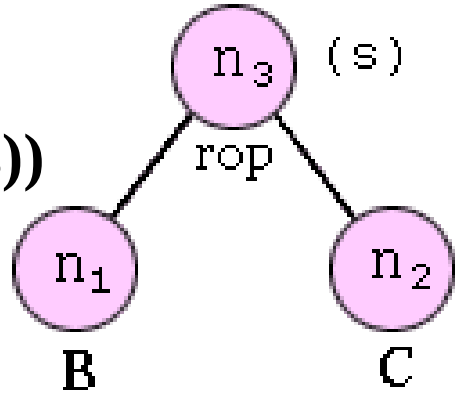
2型 (3) A: =B[C] (=[], B[C], -, A)

DAG结点

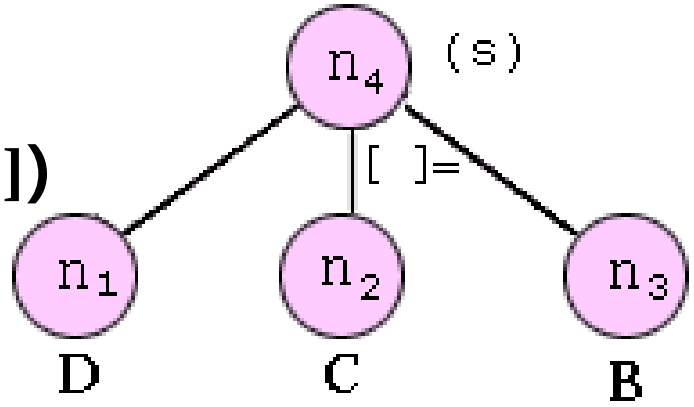


四元式与DAG结点

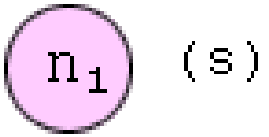
(4) if B rop C goto (s) (jrop, B, C, (s))



3型 (5) D[C]:=B ([]=, B, -, D[C])



(6) goto (s) (j, -, -, (s))



仅含0，1，2型四元式的基本块的DAG构造算法：

首先，DAG为空。

对基本块的每一四元式，依次执行：

1. 如果NODE（B）无定义，则构造一标记为B的叶结点并定义NODE（B）为这个结点；
   如果当前四元式是0型，则记NODE（B）的值为n，转4。
   如果当前四元式是1型，则转2.（1）。
   如果当前四元式是2型，则：（I）如果NODE（C）无定义，则构造一标记为C的叶结点并定义NODE（C）为这个结点，（II）转2.（2）。

2.

(1) 如果NODE（B）是标记为常数的叶结点，则转2.（3），否则转3.（1）。

(2) 如果NODE（B）和NODE（C）都是标记为常数的叶结点，则转2.（4），否则转3.（2）。

(3) 执行op B（即合并已知量），令得到的新常数为P。如果NODE（B）是处理当前四元式时新构造出来的结点，则删除它。如果NODE（P）无定义，则构造一用P做标记的叶结点n。置NODE（P）＝n，转4.。

(4) 执行B op C(即合并已知量)，令得到的新常数为P。如果NODE（B）或NODE（C）是处理当前四元式时新构造出来的结点，则删除它。如果NODE（P）无定义，则构造一用P做标记的叶结点n。置NODE（P）＝n，转4.。

3.

（1） 检查DAG中是否已有一结点，其唯一后继为NODE（B），且标记为op（即找公共子表达式）。如果没有，则构造该结点n，否则就把已有的结点作为它的结点并设该结点为n，转4.。

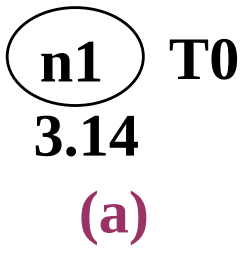
（2） 检查DAG中是否已有一结点，其左后继为NODE（B），右后继为NODE（C），且标记为op(即找公共子表达式)。如果没有，则构造该结点n，否则就把已有的结点作为它的结点并设该结点为n。转4.。

4．如果NODE（A）无定义，则把A附加在结点n上并令NODE（A）＝n；否则先把A从NODE（A）结点上的附加标识符集中删除（注意，如果NODE（A）是叶结点，则其标记A不删除），把A附加到新结点n上并令NODE（A）＝n。转处理下一四元式。

例 试构造以下基本块G的DAG。

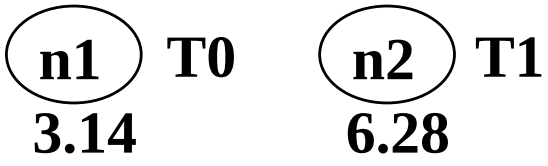
(1) T0：=3.14
(2) T1：=2 * T0
(3) T2：=R + r
(4) A：=T1 * T2
(5) B：=A
(6) T3：=2 * T0
(7) T4：=R + r
(8) T5：=T3 * T4
(9) T6：=R - r
(10) B：=T5 * T6

(1) T0：=3.14
(2) T1：=2 * T0
(3) T2：=R + r
(4) A：=T1 * T2
(5) B：=A
(6) T3：=2 * T0
(7) T4：=R + r
(8) T5：=T3 * T4
(9) T6：=R – r
(10) B：=T5 * T6



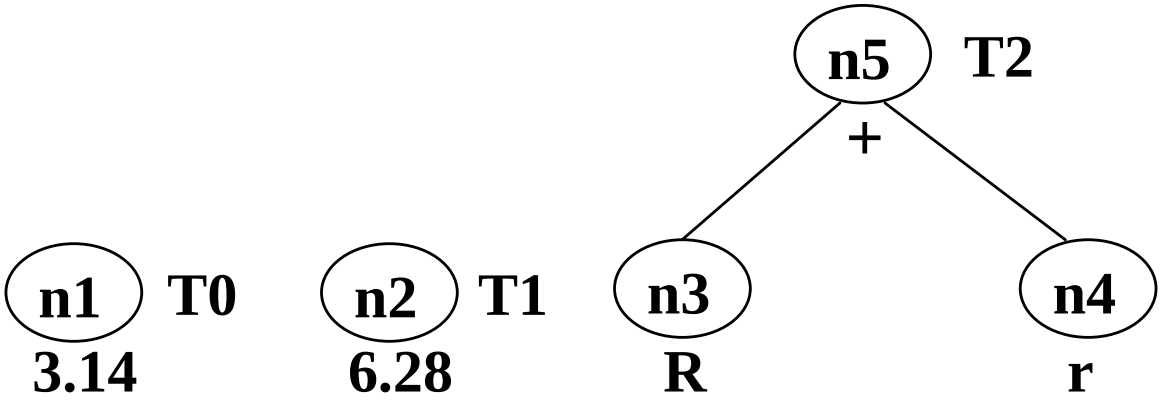
(a)

(1) T0：=3.14
(2) T1：=2 * T0
(3) T2：=R + r
(4) A：=T1 * T2
(5) B：=A
(6) T3：=2 * T0
(7) T4：=R + r
(8) T5：=T3 * T4
(9) T6：=R – r
(10) B：=T5 * T6



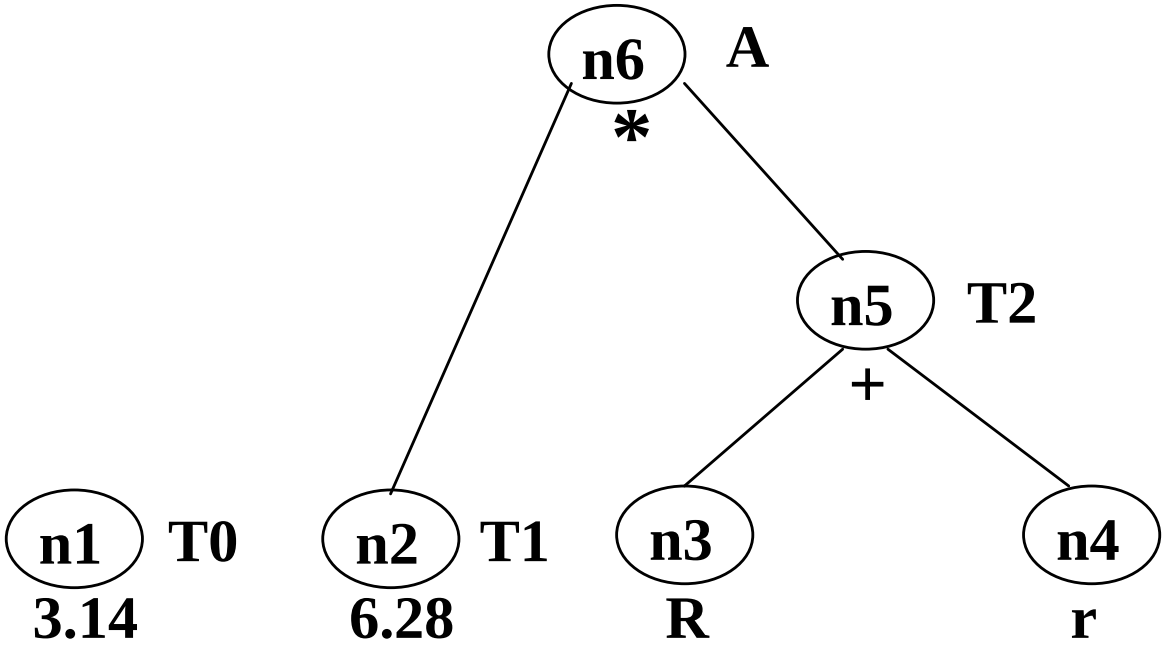
(b)

(1) T0：=3.14
(2) T1：=2 * T0
(3) T2：=R + r
(4) A：=T1 * T2
(5) B：=A
(6) T3：=2 * T0
(7) T4：=R + r
(8) T5：=T3 * T4
(9) T6：=R – r
(10) B：=T5 * T6



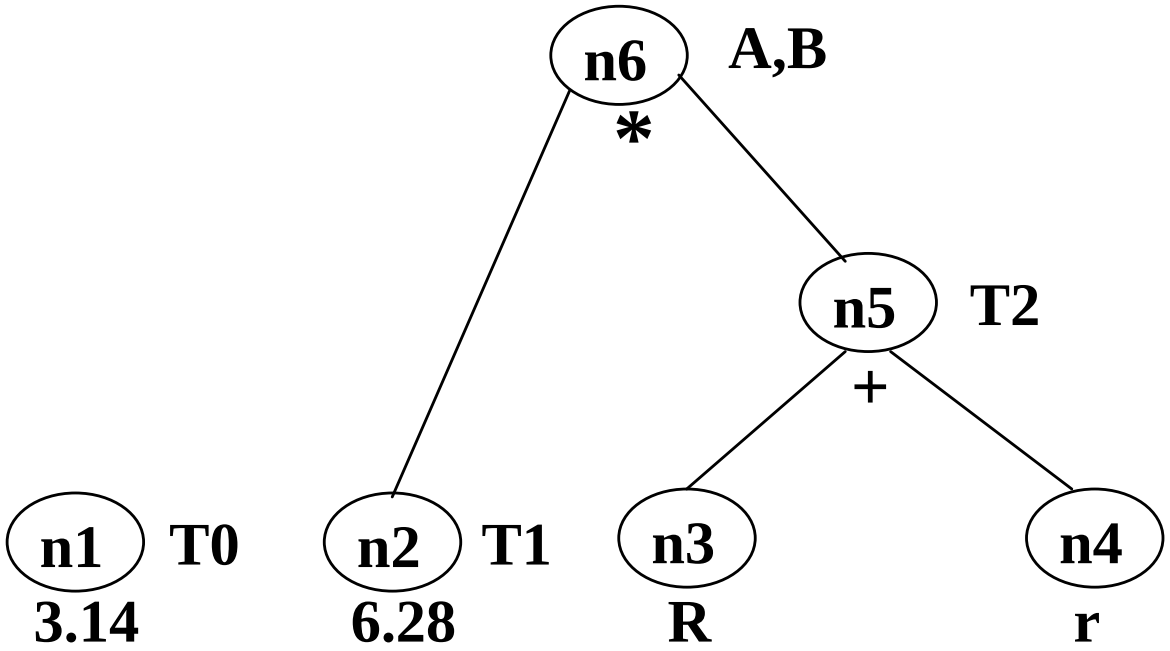
(c)

(1) T0：=3.14
(2) T1：=2 * T0
(3) T2：=R + r
(4) A：=T1 * T2
(5) B：=A
(6) T3：=2 * T0
(7) T4：=R + r
(8) T5：=T3 * T4
(9) T6：=R – r
(10) B：=T5 * T6



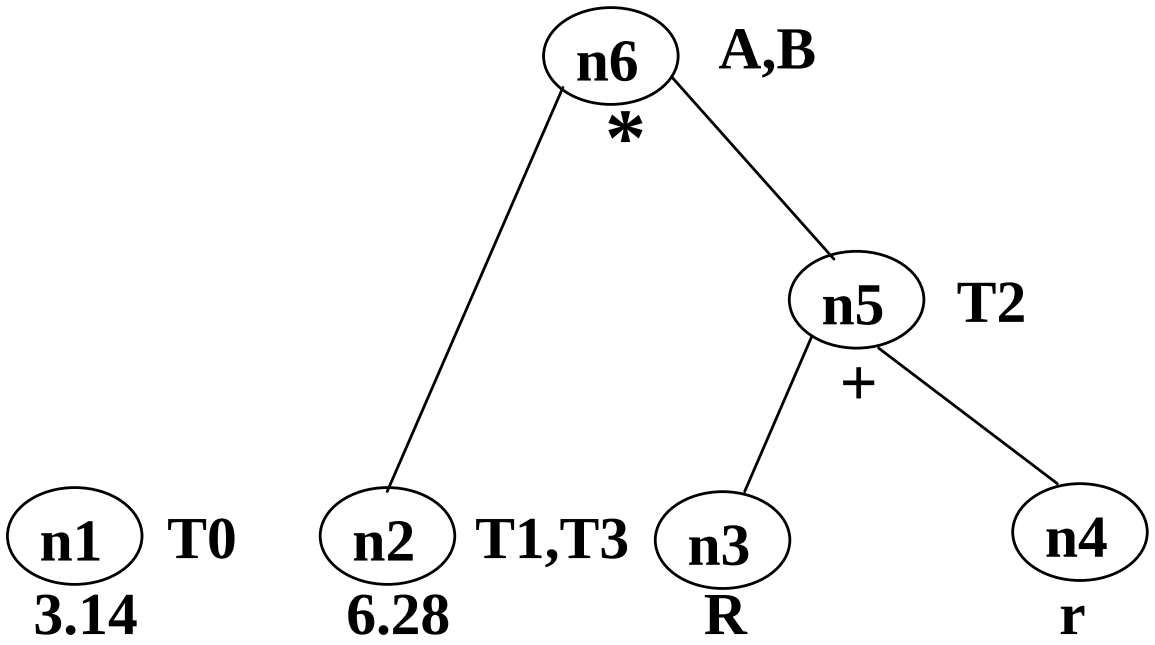
(d)

(1) T0：=3.14
(2) T1：=2 * T0
(3) T2：=R + r
(4) A：=T1 * T2
(5) B：=A
(6) T3：=2 * T0
(7) T4：=R + r
(8) T5：=T3 * T4
(9) T6：=R – r
(10) B：=T5 * T6



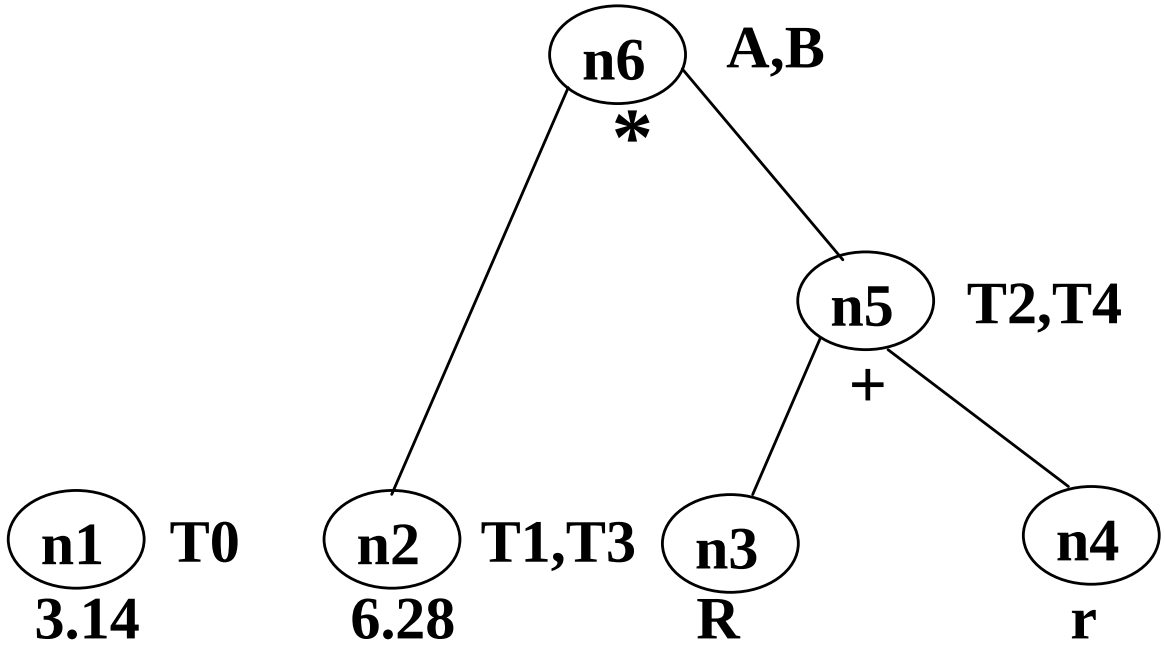
(e)

(1) T0：=3.14
(2) T1：=2 * T0
(3) T2：=R + r
(4) A：=T1 * T2
(5) B：=A
(6) T3：=2 * T0
(7) T4：=R + r
(8) T5：=T3 * T4
(9) T6：=R – r
(10) B：=T5 * T6



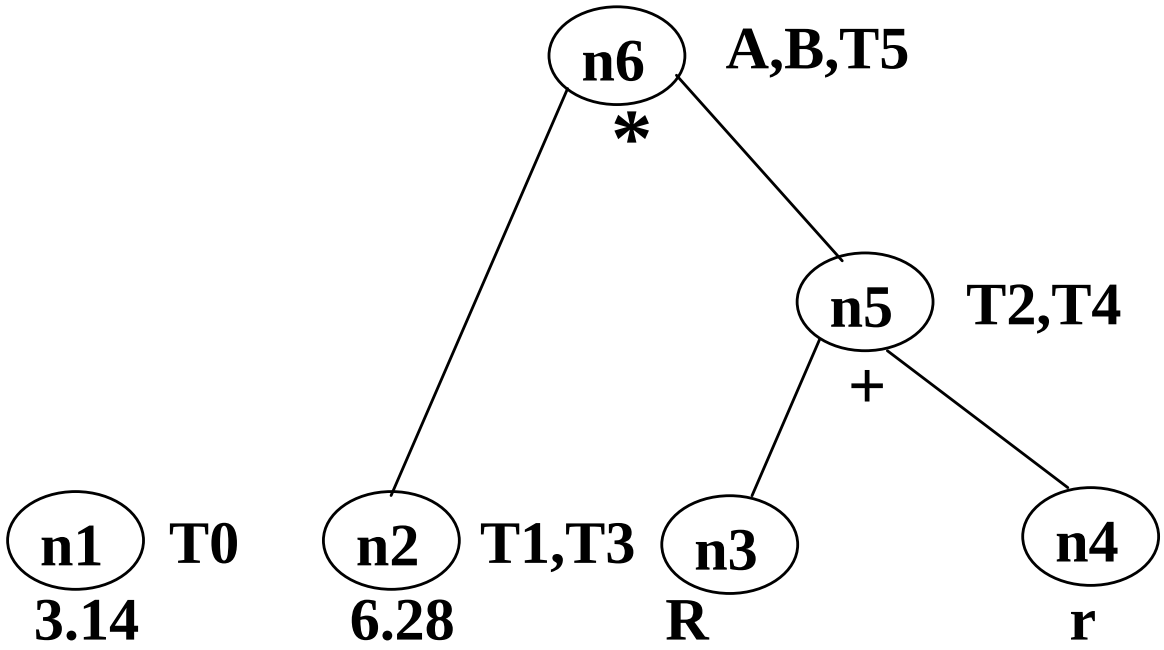
(f)

(1) T0：=3.14
(2) T1：=2 * T0
(3) T2：=R + r
(4) A：=T1 * T2
(5) B：=A
(6) T3：=2 * T0
(7) T4：=R + r
(8) T5：=T3 * T4
(9) T6：=R – r
(10) B：=T5 * T6



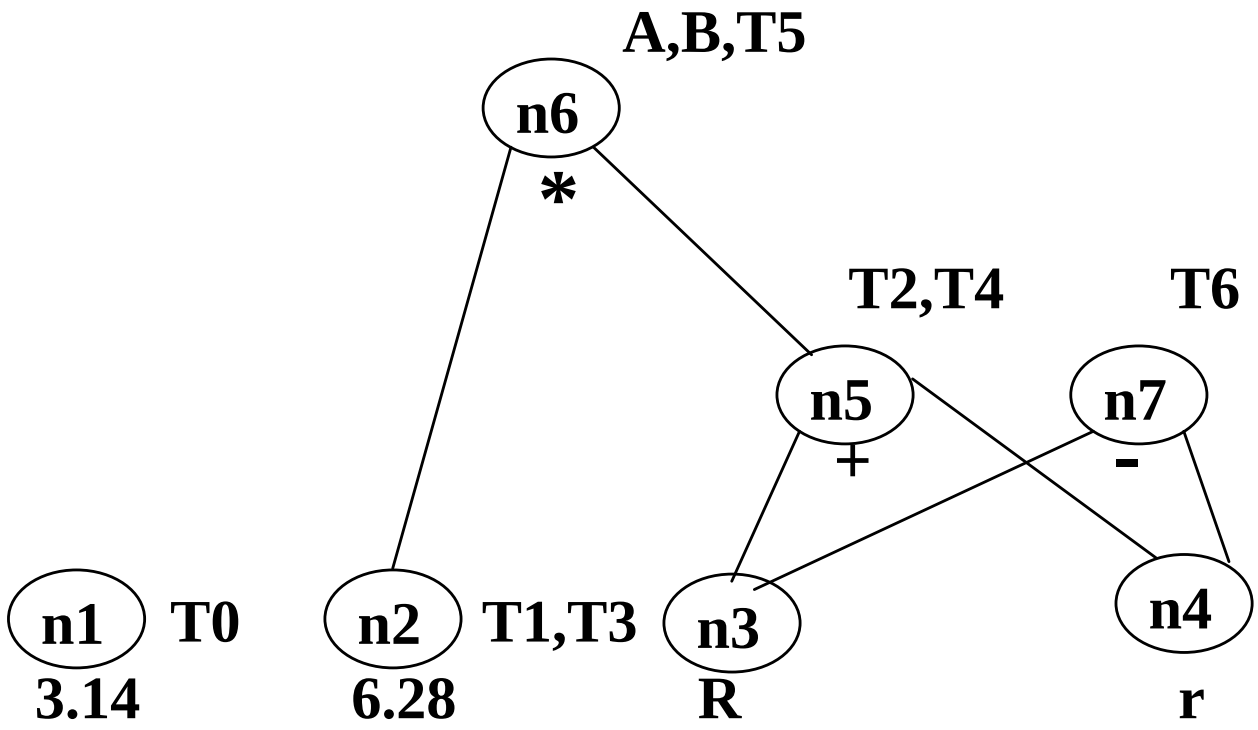
(g)

(1) T0：=3.14
(2) T1：=2 * T0
(3) T2：=R + r
(4) A：=T1 * T2
(5) B：=A
(6) T3：=2 * T0
(7) T4：=R + r
(8) T5：=T3 * T4
(9) T6：=R – r
(10) B：=T5 * T6



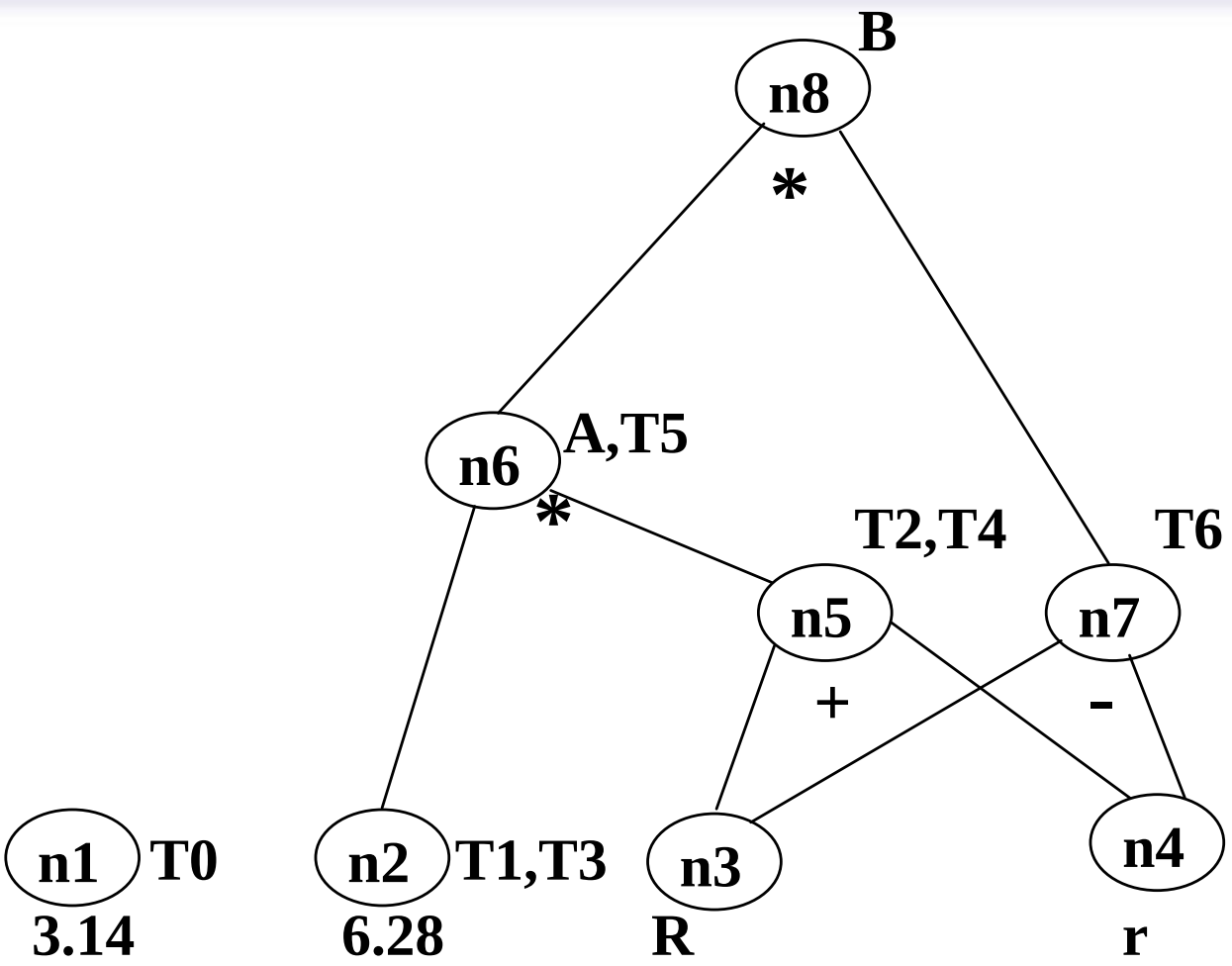
(h)

(1) T0：=3.14
(2) T1：=2 * T0
(3) T2：=R + r
(4) A：=T1 * T2
(5) B：=A
(6) T3：=2 * T0
(7) T4：=R + r
(8) T5：=T3 * T4
(9) T6：=R – r
(10) B：=T5 * T6



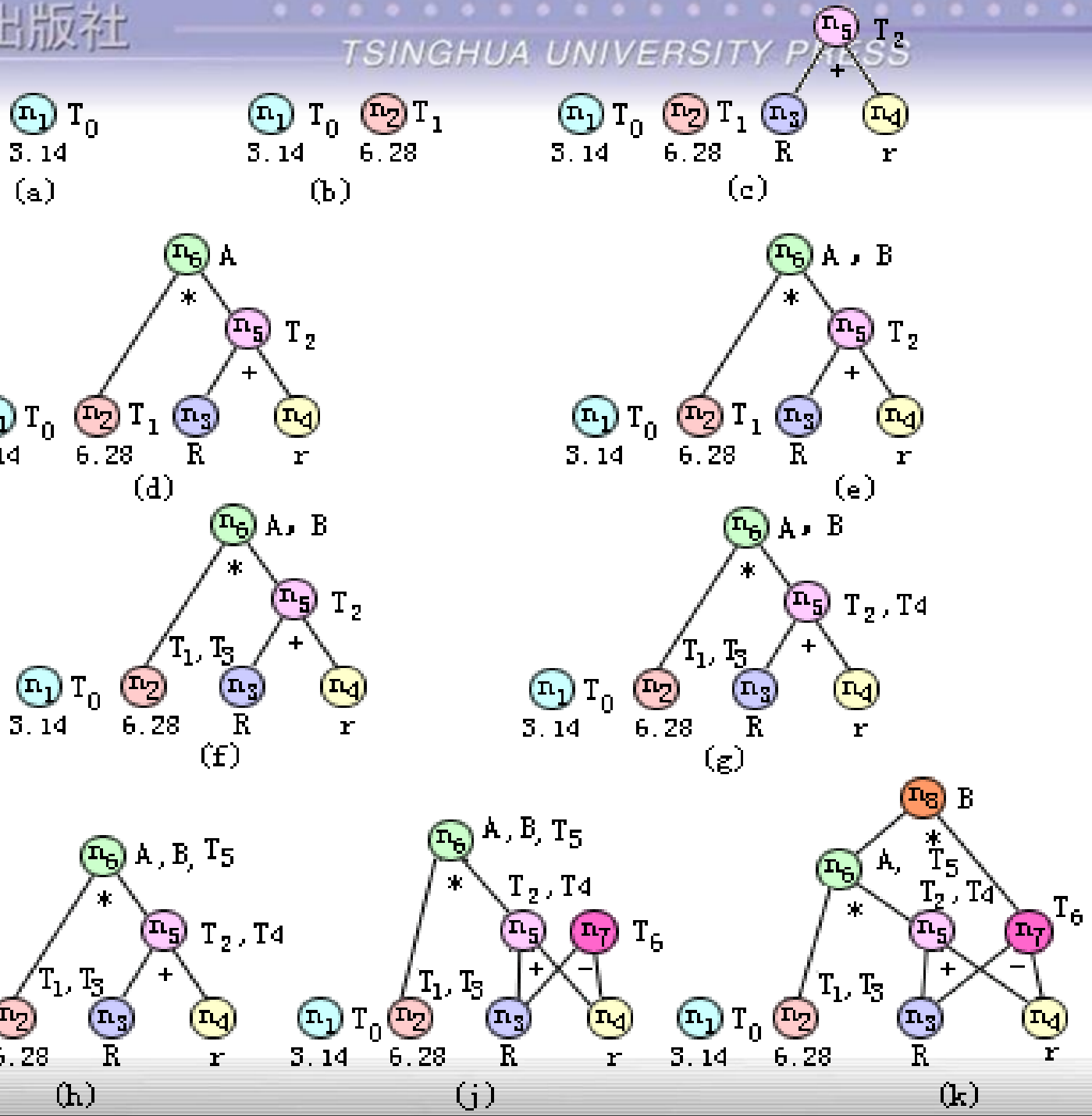
(i)

(1) T0：=3.14
(2) T1：=2 * T0
(3) T2：=R + r
(4) A：=T1 * T2
(5) B：=A
(6) T3：=2 * T0
(7) T4：=R + r
(8) T5：=T3 * T4
(9) T6：=R – r
(10) B：=T5 * T6



(j)

清华大学出版社 TSINGHUA UNIVERSITY PRESS



DAG的应用

将四元式表示成相应的DAG后，可以利用DAG来进行优化。

对于**步骤2**，如果参与运算的对象都是编译时的已知量，则它**并不生成计算该结点值的内部结点，而是执行该运算，将计算结果生成一个叶结点**。显然，步骤2起到了合并已知量的作用。

步骤3的作用是检查公共子表达式，**对具有公共子表达式的所有四元式，它只产生一个计算该表达式值的内部结点**，而把那些被赋值的变量标识符附加到该结点上。这样可删除多余运算。

步骤4具有**删除无用赋值的作用**。如果某变量被赋值后，在它被引用前又被重新赋值，则步骤4把该变量从具有前一个值的结点上删除。

对基本块G的DAG按原来构造其结点的顺序，重新写成四元式，得到以下的四元式序列G′：

(1) T0：=3.14
(2) T1：=2 * T0
(3) T2：=R + r
(4) A：=T1 * T2
(5) B：=A
(6) T3：=2 * T0
(7) T4：=R + r
(8) T5：=T3 * T4
(9) T6：=R – r
(10) B：=T5 * T6

(1) T0：=3.14
(2) T1：=6.28
(3) T3：=6.28
(4) T2：=R+r
(5) T4：=T2
(6) A：=6.28*T2
(7) T5：=A
(8) T6：=R-r
(9) B：=A*T6

1．G中的代码（2）和（6）的已知量都已合并。
2．G中（5）的无用赋值已被删除。
3．G中（3）和（7）的公共子表达式R+r只被计算一次，删除了多余运算。

3 控制流分析和循环优化

循环，粗略地说，就是程序中那些可能反复执行的代码序列。

循环中的代码要反复执行，因而为了提高目标代码的效率必须着重考虑循环的代码优化。

程序流图

一个**控制流程图**，简称**流图**，就是具有唯一首结点的有向图。所谓**首结点**，就是从它开始到控制流程图中任何结点都有一条通路的结点。

可以把一个控制流程图表示成一个三元组**G = （N，E，n0）**，其中，N代表图中所有结点集，E代表图中所有有向边集，n0代表首结点。

一个程序可用一个流图来表示。

流图中的**有限结点集N**就是程序的基本块集，流图中的结点就是程序中的基本块。流图的首结点就是包含程序第一个语句的基本块。

程序流图中的**有向边集E**是这样构成的：

假设流图中结点i和结点j分别对应于程序的基本块i和基本块j，则当下述条件1)或2)有一个成立时，从结点i有一有向边引向结点j：

1) 基本块j在程序中的位置紧跟在基本块i之后，并且基本块的出口语句不是无条件转移语句goto（s）或停语句。

2) 基本块i的出口语句是goto（s）或if…goto（s），并且（s）是基本块j的入口语句。

考虑以下程序段:

(1) read x

(2) read y

(3) r：=x mod y

(4) if r=0 goto (8)

(5) x：=y

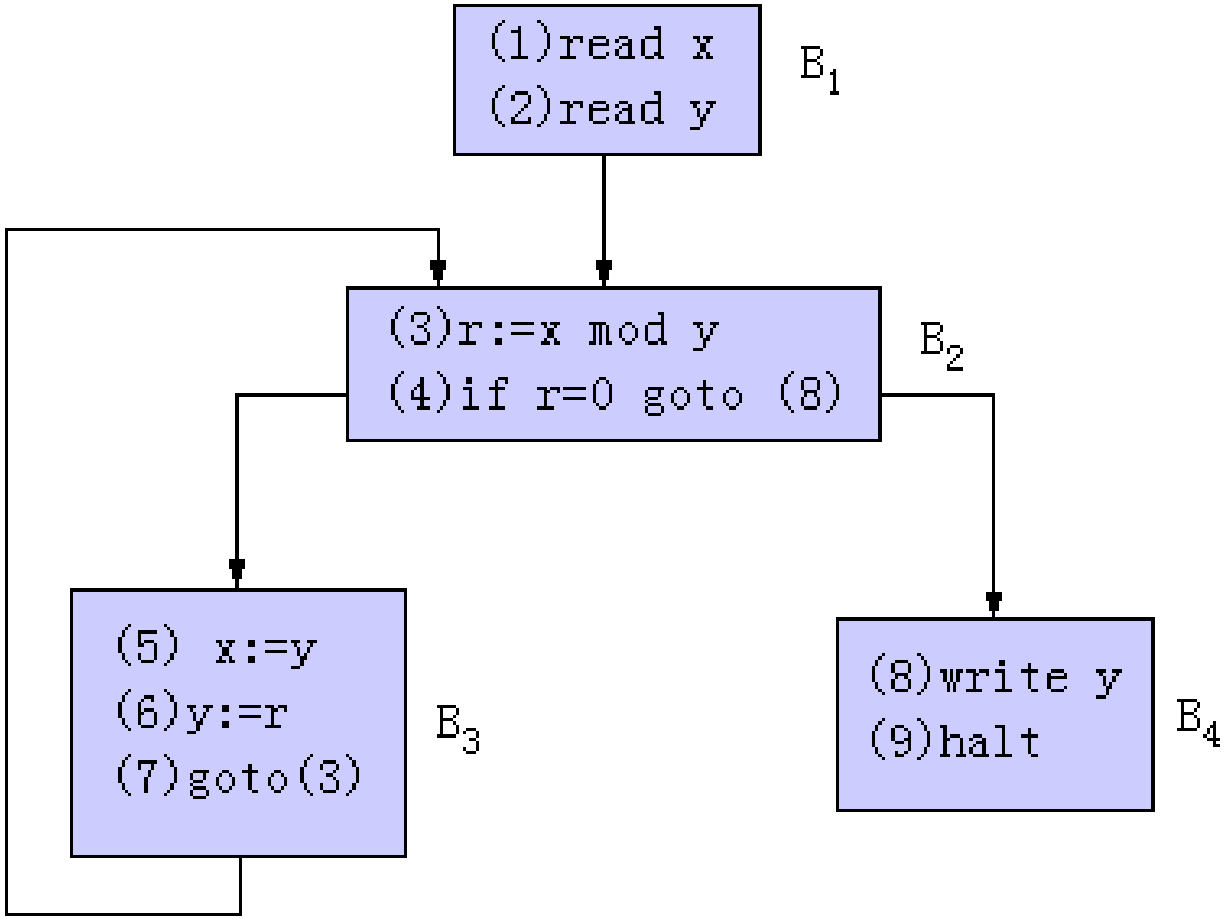
(6) y：=r

(7) goto (3)

(8) write y

(9) halt

程序流图


(1)read x
(2)read y
B1
(3)r:=x mod y
(4)if r=0 goto (8)
B2
(5) x:=y
(6)y:=r
(7)goto(3)
B3
(8)write y
(9)halt
B4

循环的查找

循环结构的定义，具有下列性质的结点序列为一个循环：

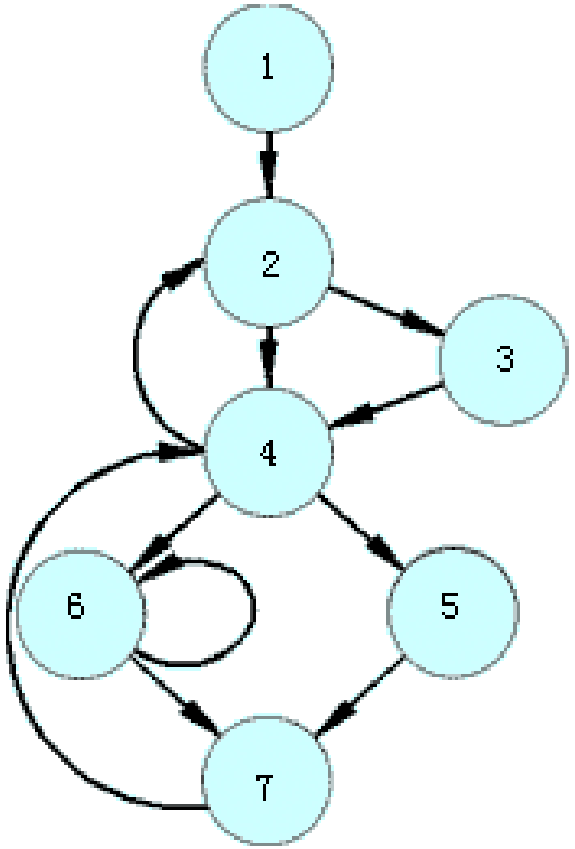
① 它们是强连通的。也即，其中任意两个结点之间，必有一条通路，而且该通路上各结点都属于该结点序列。如果序列只包含一个结点，则必有一有向边从该结点引到其自身。

② 它们中间有且只有一个是入口结点。所谓**入口结点**，是指序列中具有下述性质的结点：

从序列外某结点，有一有向边引到它，或者它就是程序流图的首结点。

循环就是程序流图中具有唯一入口结点的强连通子图，从循环外要进入循环，必须首先经过循环的入口结点。

程序流图



循环的查找

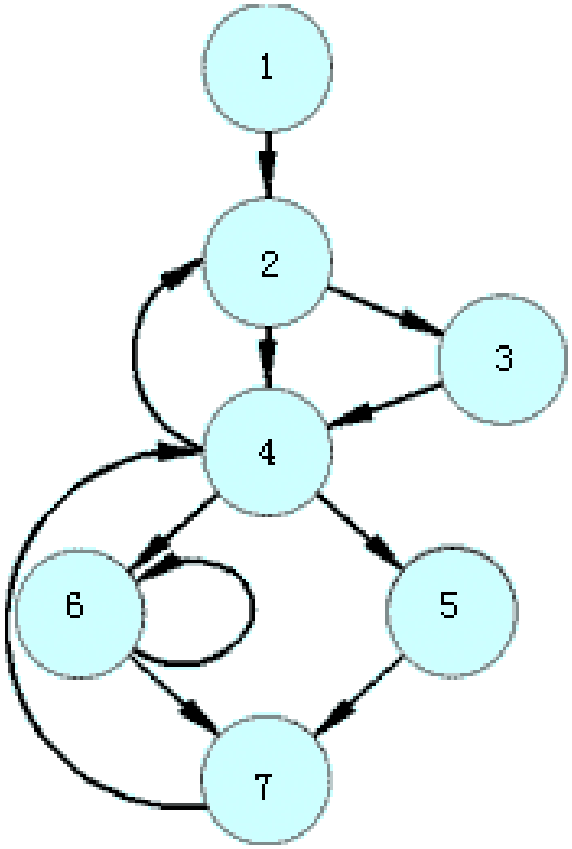
为了找出程序流图中的循环，需要分析流图中结点的控制关系。

在程序流图中，对任意两个结点m和n，如果从流图的首结点出发，到达n的任一通路都要经过m，则称m是n的必经结点，记为**m DOM n**。流图中结点n的所有必经结点的集合，称为结点n的必经结点集，记为**D（n）**。

关系DOM是一个偏序关系。任何结点n的必经结点集D（n）是一个有序集。

考虑上述程序流图：

D(1)={1}
D(2)={1,2}
D(3)={1,2,3}
D(4)={1,2,4}
D(5)={1,2,4,5}
D(6)={1,2,4,6}
D(7)={1,2,4,7}



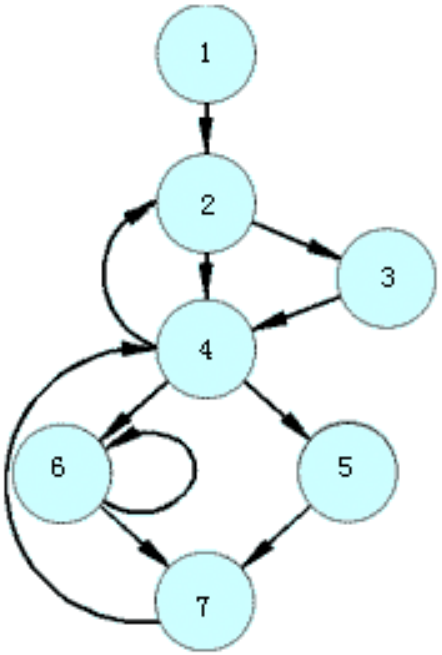
循环的查找算法

假设a→b是流图中的一条有向边，如果b DOM a，则称a→b是流图中的一条**回边**。

如果已知有向边n→d是回边，那么就可以求出由它组成的循环。该循环就是由结点d、结点n以及有通路到达n而该通路不经过d的所有结点组成，并且d是该循环的唯一入口结点。

由流图可以看出，有向边6→6、7→4、4→2是回边。

由回边6→6组成的循环就是｛6｝,由回边7→4组成的循环是｛4，5，6，7｝；由回边4→2组成的循环是｛2，3，4，5，6，7｝。



循环优化

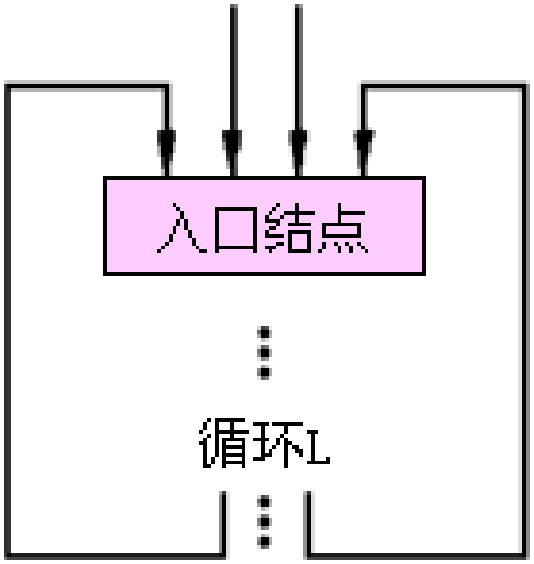
循环优化的三种重要技术是**代码外提、强度削弱与删除归纳变量**。

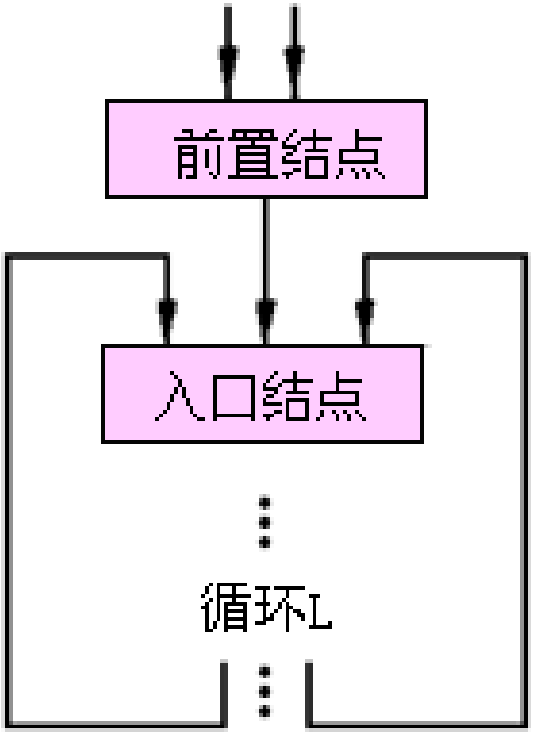
代码外提是减少循环中代码数目的一个重要办法。

把**循环不变运算**，即产生的结果独立于循环执行次数的表达式，放到循环的前面。这里讨论的循环只存在一个入口。

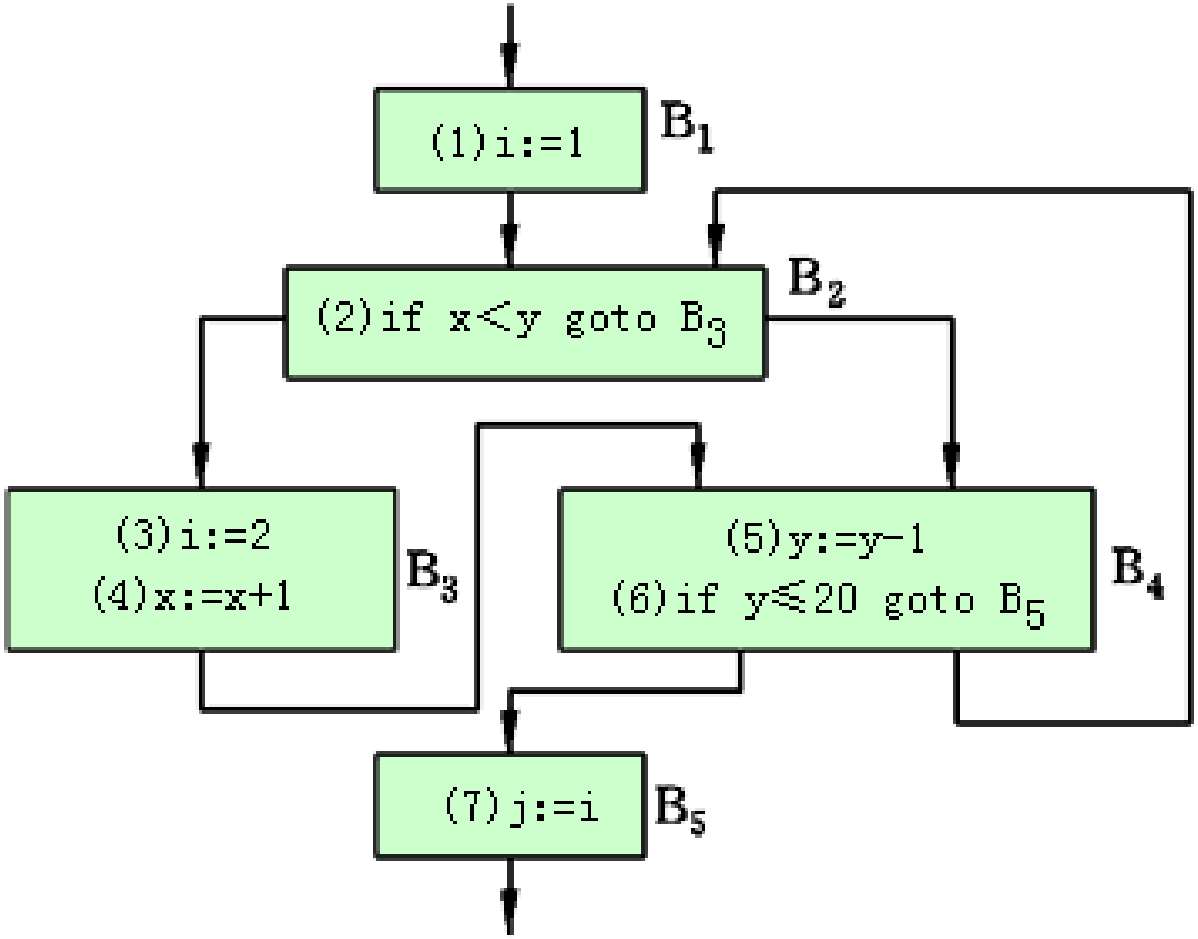
实行代码外提时，在循环的入口结点前面建立一个新结点（基本块），称之为循环的**前置结点**。循环的前置结点以循环的入口结点为其唯一后继，原来流图中从循环外引到循环入口结点的有向边，改成引到循环前置结点。

代码外提的流图

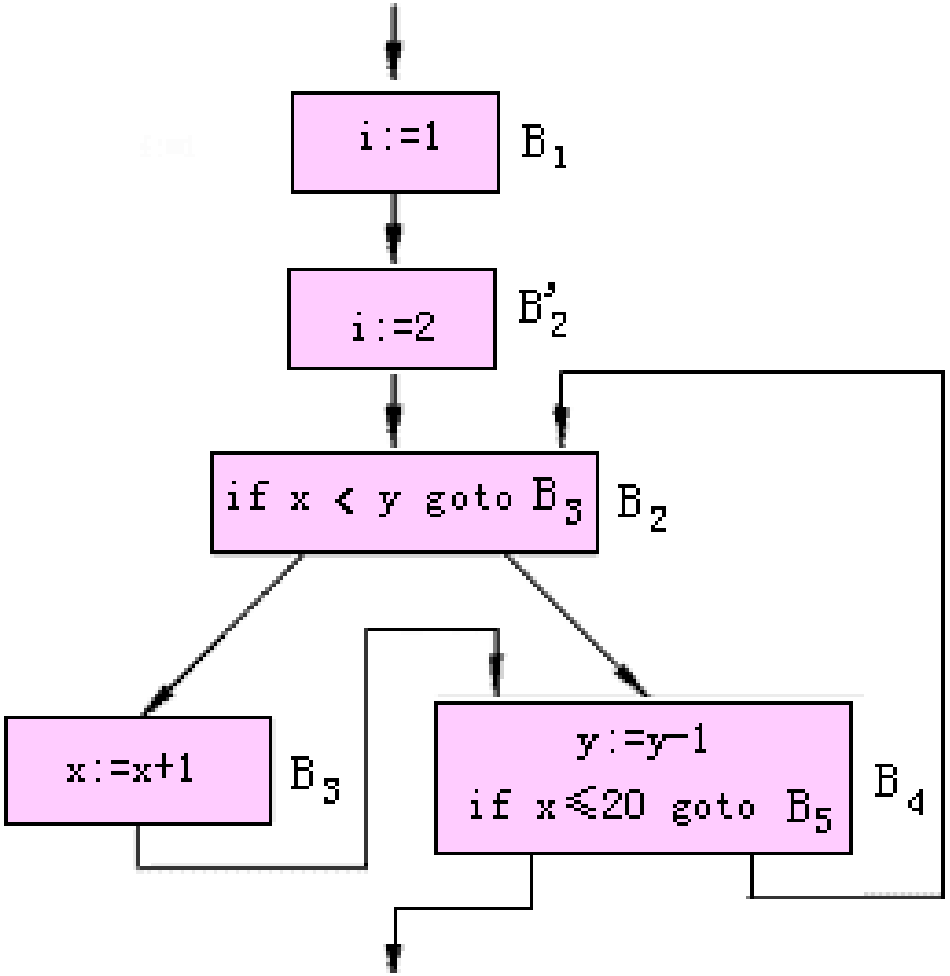




程序流图


(1)i:=1
B1
(2)if x<y goto B3
B2
(3)i:=2
(4)x:=x+1
B3
(5)y:=y-1
(6)if y≤20 goto B5
B4
(7)j:=i
B5

代码外提后的程序流图

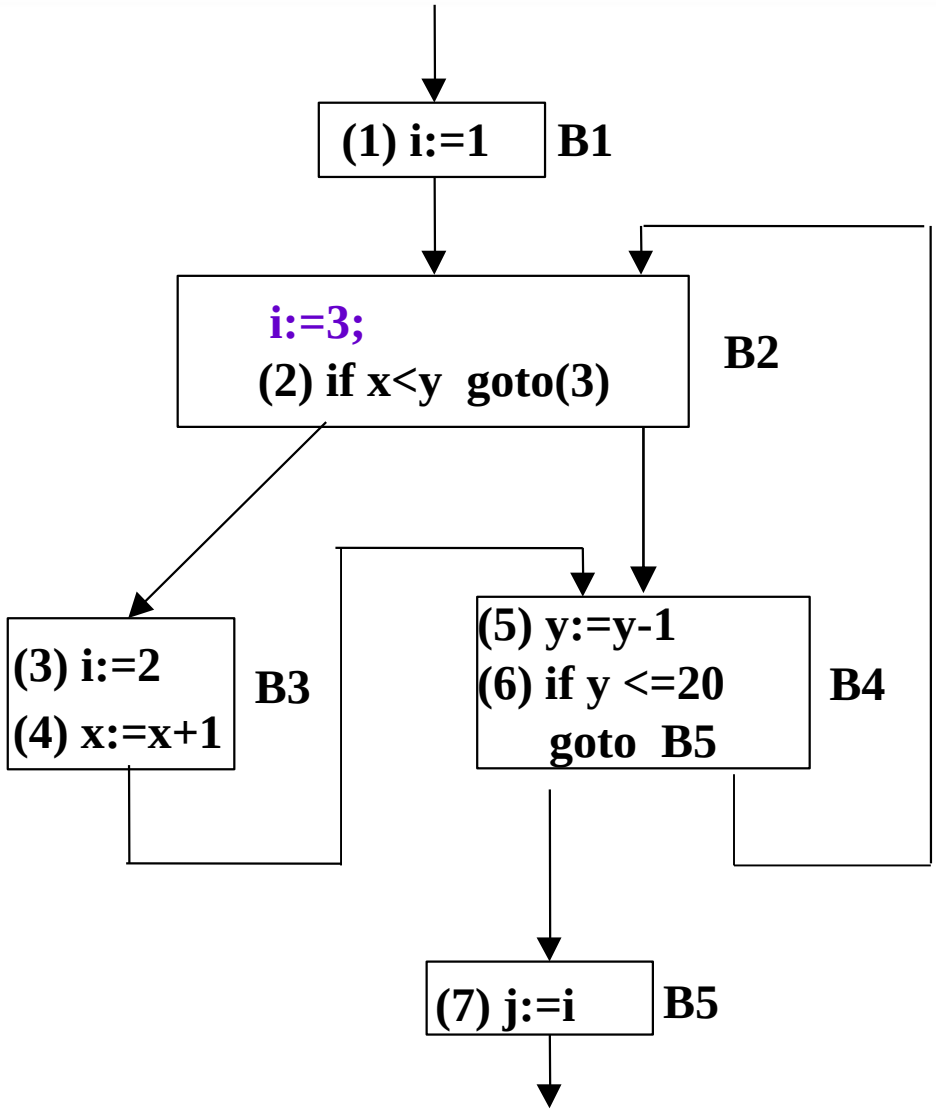


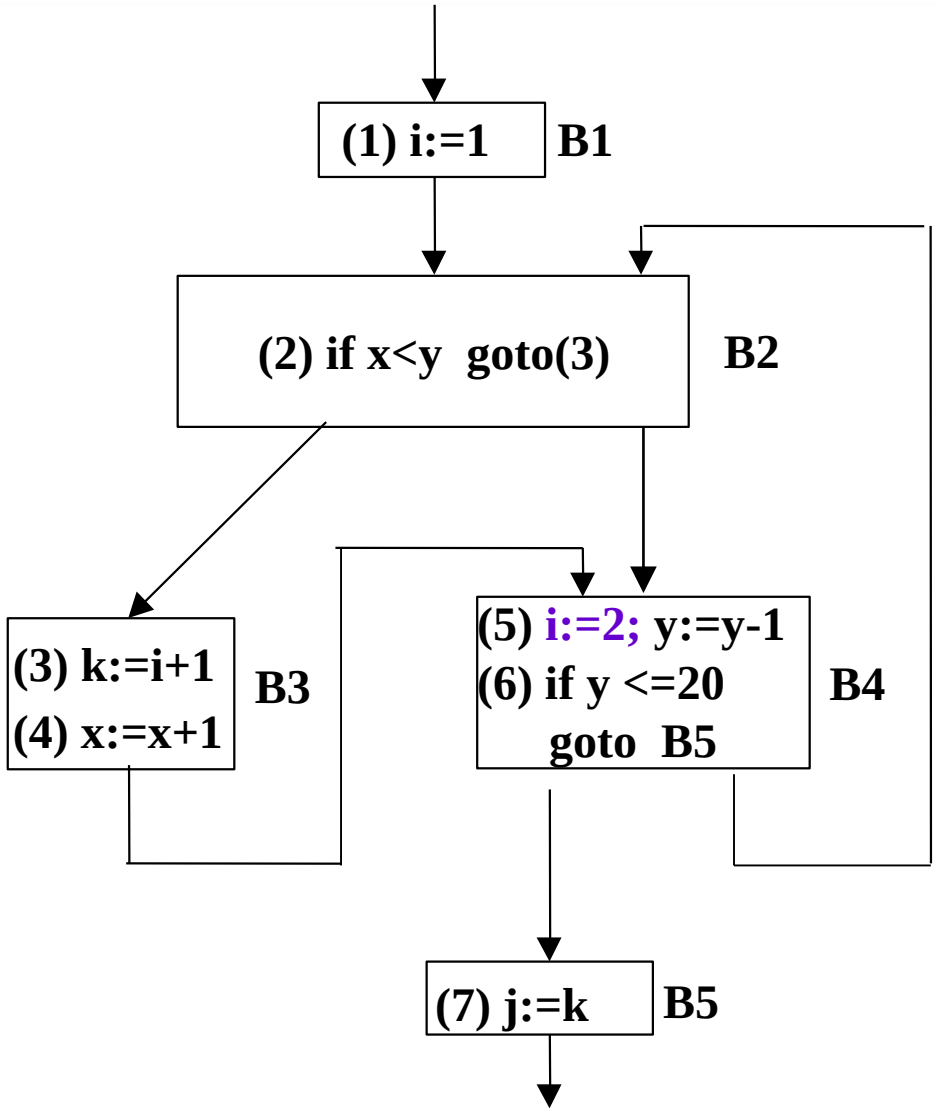
外提的条件

对每一循环不变运算s: A:=B op C或A:= op B 或A:=B，检查其是否满足以下条件(a)或(b):

(a) (1) S所在的结点是循环L的所有出口的必经结点；(2) A在L中其它地方未再定值；(3) L中所有的引用点只有s中A的定值才能到达。

(b) A在离开L后是活跃的，且(a)中(1)(2)满足。


(1) i:=1
B1
i:=3;
(2) if x<y goto(3)
B2
(3) i:=2
(4) x:=x+1
B3
(5) y:=y-1
(6) if y <=20
goto B5
B4
(7) j:=i
B5


(1) i:=1
B1
(2) if x<y goto(3)
B2
(3) k:=i+1
(4) x:=x+1
B3
(5) i:=2; y:=y-1
(6) if y <=20
goto B5
B4
(7) j:=k
B5

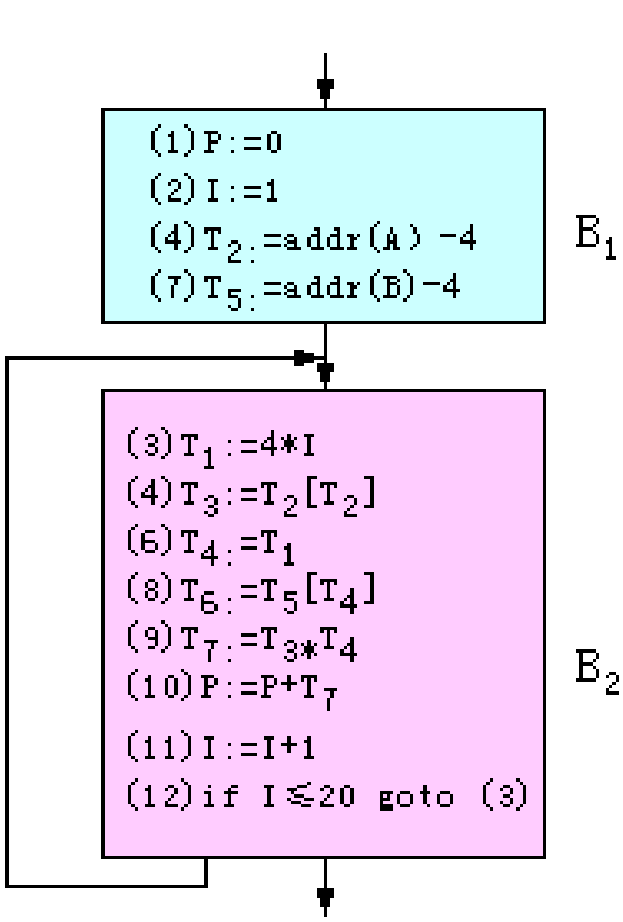
强度削弱与删除归纳变量

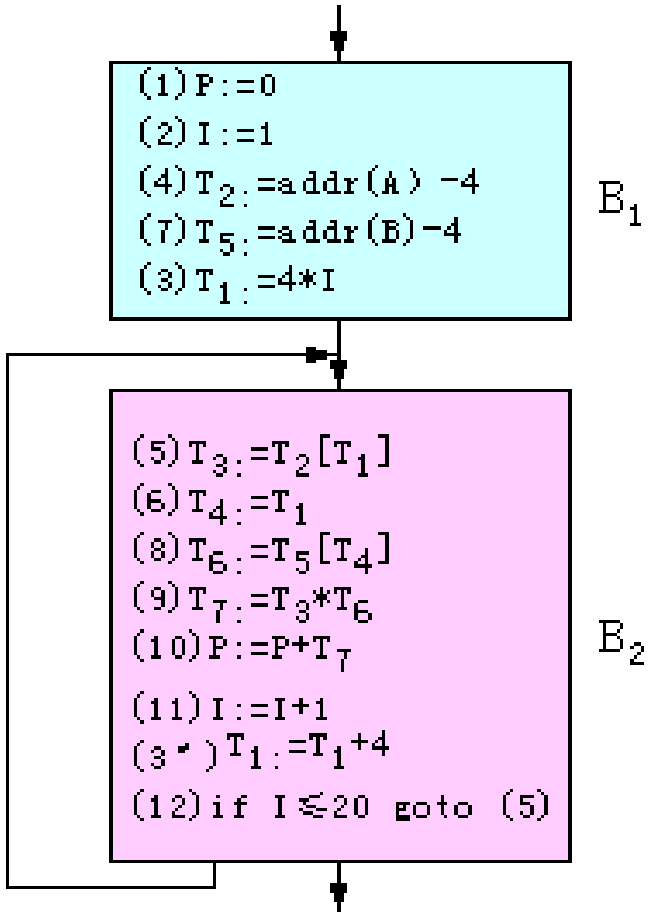
如果循环中对变量I只有唯一的形如I:=I+/-C的赋值，且其中C为循环不变量，则称I为循环中的**基本归纳变量。**

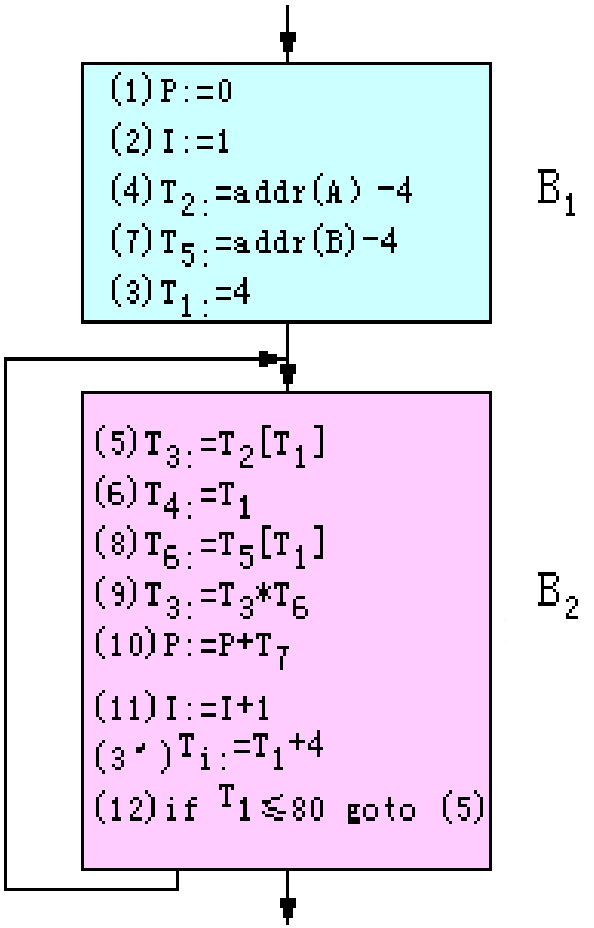
如果I是循环中的一基本归纳变量，J在循环中的定值总可以划归为I的同一线形函数，也即J=C1*I+/-C2，其中C1和C2都是循环不变量，则称J为**归纳变量**，并称J与I同族。

一个基本归纳变量除用于自身的递归定值外，往往只在循环中用来计算其它归纳变量以及控制循环的进行。

于是**可用与循环控制条件中的基本归纳变量同族的某一归纳变量来替换，然后删除基本归纳变量的递归定值**。


(1) P:=0
(2) I:=1
(4) T2:=addr(A)-4
(7) T5:=addr(B)-4
B1
(3) T1:=4*I
(4) T3:=T2[T2]
(6) T4:=T1
(8) T6:=T5[T4]
(9) T7:=T3*T4
(10) P:=P+T7
(11) I:=I+1
(12) if I≤20 goto (3)
B2


(1) P:=0
(2) I:=1
(4) T2:=addr(A)-4
(7) T5:=addr(B)-4
(3) T1:=4*I
B1
(5) T3:=T2[T1]
(6) T4:=T1
(8) T6:=T5[T4]
(9) T7:=T3*T6
(10) P:=P+T7
(11) I:=I+1
(3') T1:=T1+4
(12) if I≤20 goto (5)
B2


(1) P:=0
(2) I:=1
(4) T2:=addr(A)-4
(7) T5:=addr(B)-4
(3) T1:=4
B1
(5) T3:=T2[T1]
(6) T4:=T1
(8) T6:=T5[T1]
(9) T3:=T3*T6
(10) P:=P+T7
(11) I:=I+1
(3') Ti:=T1+4
(12) if T1≤80 goto (5)
B2

4 数据流分析与全局优化

基本概念：

1. 到达-定值
2. 引用-定值链（ud链）
3. 活跃变量
4. 定值-引用链（du链）

到达-定值：所谓变量A在某点d**定值到达**另一点是指流图中从d有一通路到达u且该通路上没有A的其它定值。

引用-定值链（ud链）：若流图中某点u引用了变量A的值，则把能到达u的A的所有定值点的全体，称作A在引用点u的**引用-定值链**。

活跃变量：对流图中某变量A和某点p而言，如果存在一条从p开始的通路，其中引用了A在p点的值，则称A在点p是**活跃**的。否则，称A在点p是死亡的。

定值-引用链（du链）：与ud链对应，对变量A在点p的定值，计算该定值能到达的A的所有引用点。这些点的集合称为A在点p的**定值-引用链**。

数据流方程的一般形式

数据流信息可由建立和求解数据流方程来收集。

数据流方程的一般形式为：

$$\mathbf{out[s] = (in[s] - kill[s]) \cup gen[s]} \quad （1）$$

方程的含义是：当控制流通过一个语句时，在语句末尾的信息是进入这个语句的信息扣除本语句注销的信息并加上此语句产生的信息。

数据流方程的建立和求解依赖于:

1. 产生与注销依赖于所处理的信息。有些问题可能逆向控制流来求解。方程形式为:

$$in[s] = (out[s] - kill[s]) \cup gen[s] \quad （2）$$

2. 方程通常建立在基本块一级而非语句一级，因为基本块有唯一出口。
3. 对过程调用、指针赋值和数组变量赋值语句需要特定处理。

数据流方程的选用（1）or （2）取决于：

1. 数据流的方向：正向还是反向分析。
2. 合流算符。合流算符是指当多条边到达基本块B时，由B前驱结点的out集计算in[B]时所采用的运算符，交还是并。

到达-定值数据流方程

到达-定值数据流方程为：

$$out[B] = (in[B] - kill[B]) \cup gen[B]$$

$$in[B] = \cup out[p],\ p \in P[B]$$

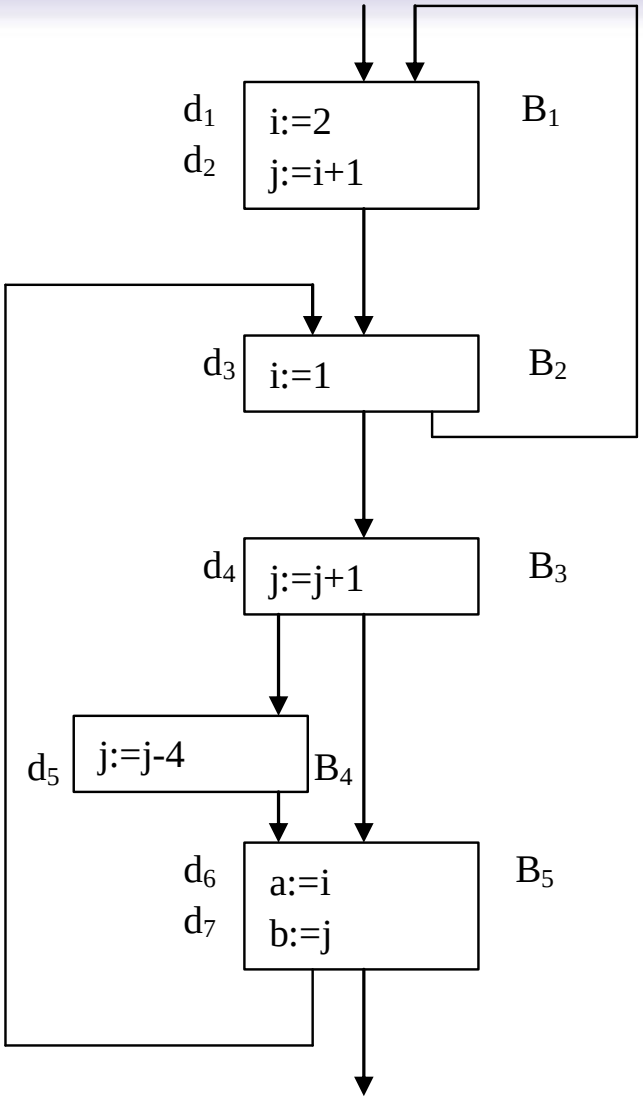
其中，P[B]：B的所有前驱基本块；

gen[B]：B中定值并到达B出口之后的所有定值点集；

kill[B]：B之外的那些定值点集，其定值的变量在B中又重新定值；

in[B]:到B入口之前的各变量的所有定值点集；

out[B]:到达B出口之后各变量的所有定值点集；



流　　图

	gen	位向量	kill	位向量
B1	$\{d_1, d_2\}$	1100000	$\{d_3, d_4, d_5\}$	0011100
B2	$\{d_3\}$	0010000	$\{d_1\}$	1000000
B3	$\{d_4\}$	0001000	$\{d_2, d_5\}$	0100100
B4	$\{d_4\}$	0000100	$\{d_2, d_4\}$	0101000
B5	{ }	0000000	{ }	0000000

计算gen和kill

	in[B]	out[B]
B1	0111100	1100000
B2	1111100	0111100
B3	0111100	0011000
B4	0011000	0010100
B5	0011100	0011100

计算结果

```
begin
    for  i:= 1 to n do begin
            in[ Bi] :=∅;
            out[Bi ] := gen[Bi ];
    end;
    change := true;
    while change do begin
        change := false;
        for i := 1 to n do begin
            newin := ∪ out[p];     //p∈P[Bi]
            if newin ≠ in[Bi] then begin
                change := ture;
                in[Bi] := newin;
                out[Bi] := (in[Bi] – kill[Bi]) ∪gen[Bi]
             end
         end
    end
end
```

迭代算法

开始，in[B1]=in[B2]= in[B3]=in[B4]=in[B5]=0000000

out[B1]=1100000 out[B2]=0010000 out[B3]=0001000

out[B4]=0000100 out[B5]=0000000

第一次迭代：

B1： newin=out[B2]=0010000 ≠ in[B1]，所以

change=true，in[B1]=0010000

out[B1]=in[B1]-kill[B1] ∪ gen[B1]=0010000-0011100

∪1100000=1100000

B2： newin=out[B1] ∪ out[B5]=1100000 ∪ 0000000 ≠ in[B2]

所以， change=true，in[B2]=1100000

out[B2]=1100000-1000000 ∪ 0010000=0110000

	in[B]	out[B]	in[B]	out[B]	in[B]	out[B]	in[B]	out[B]
B1	0010000	1100000	0110000	1100000	0111100	1100000	0111100	1100000
B2	1100000	0110000	1111100	0111100	1111100	0111100	1111100	0111100
B3	0110000	0011000	0111100	0011000	0111100	0011000	0111100	0011000
B4	0011000	0010100	0011000	0010100	0011000	0010100	0011000	0010100
B5	0011100	0011100	0011100	0011100	0011100	0011100	0011100	0011100

第1次　　第2次　　第3次　　第4次

in[B]和out[B]可用来计算各变量在任何点的ud链。

ud链还可用于常数传播和合并已知量。

可用表达式及其数据流方程

可用表达式：如果从首结点到p的每条路径上（不必是无环）都计算X op Y，并且每条通路上最后一个这样的计算和p之间没有对X和Y的定值，则称表达式X op Y在点p可用。

对可用表达式，如果一个基本块对X或Y赋值（或可能赋值），而且随后没有重新计算X op Y，则称**基本块注销表达式**。如果它计算X op Y，而且随后又没有重新定义X或Y，则称**基本块产生表达式**。

可用表达式可用于**寻找公共子表达式**。

计算基本块产生的可用表达式集合

在块中从头到尾扫描块中的语句。如果在点p可用表达式集合是A，q是p的下一点，p和q之间的语句是：X:=Y op Z，则点q的可用表达式可计算如下：

1. 将表达式Y op Z加入A；
2. 删除A中任何含X的表达式。

语句	可用表达式
	none
1. a:=b+c	
	only b+c
2. b:=a-d	
	only a-d
3. c:=b+c	
	only a-d
4. d:=a-d	
	none

表达式数据流方程

$out[B] = (in[B] - kill[B]) \cup gen[B]$

$in[B] = \cap out[p]$, B不是首结点, $p \in P[B]$

$in[B_1] = \Phi$, B_1是首结点

活跃变量数据流方程

in(B)：在基本块B入口处活跃的变量的集合。

out(B)：基本块B的出口处的活跃变量的集合。in和out并不是相互独立的，令S(B)为流图中基本块B的后继的集合，则有out(B) $=\cup$ in(s)，s$\in$S(B)。如果基本块没有后继，则其out为空。

def(B)：在B中定值的且定值前未曾在B中使用过的变量集。如果一个变量在基本块B的出口处为活跃的且v$\notin$def(B)，则它在B的入口处也是活跃的；即，in(B) $\supseteq$ out(B) - def(B)。

use(B)：B中引用的且引用前未曾在B中定值的变量的集合。这个集合由基本块B中的语句唯一确定。容易看出，如果v$\in$use(B)，则v$\in$in(B)；即in(B) $\supseteq$ use(B)。

活跃变量数据流方程

$$in(B) = (out(B) - def(B)) \cup use(B)$$

$$out(B) = \cup in(s),\ s \in S(B)$$

一个变量在某基本块入口处为活跃的，则一定有：或者它在该块中定值前有引用，或者它在该块的出口处活跃且没有被该块定值。

复写传播

复写：形如x:=y的赋值；

如能找出复写语句s: x:=y中x定值的所有引用点，然后用y替代x，则可以删除此复写语句，这称为**复写传播**。

但其中x的每个引用点u必须满足：

1. 语句s是到达u的唯一x定值，即x在引用点u的ud链只含s。
2. 从s到u的每条通路，包括穿过u若干次的通路（但没多次穿过）上，没有对y的重新定值。（否则，将x替换成y时，语义出错。）

in[B]：满足下述条件的所有复写s: x:=y的集合，从首结点到基本块B入口之前的每一通路上都包含复写s: x:=y，并且每一通路上最后出现的复写s: x:=y到B入口之前未曾对x或y重新定值。

out[B]：满足下述条件的所有复写s: x:=y的集合，从首结点到基本块B出口之后的每一通路上都包含复写s: x:=y，并且每一通路上最后出现的复写s: x:=y到B出口口之前未曾对x或y重新定值。

E-gen[B]：基本块中满足下述条件的所有复写s: x:=y的集合，在B中s的后面未曾对x或y重新定值。

E-kill[B]：流图中满足下述条件的所有复写s: x:=y的集合，其中s在B外面， x或y在B中被重新定值。

数据流方程如下：

$$out[B] = (in[B] - E\text{-}kill[B]) \cup E\text{-}gen[B]$$

$$in[B] = \cap\, out[p],\ B\text{不是首结点}, p \in P[B]$$

$$in[B_1] = \Phi,\ B_1\text{是首结点}$$

作业

1

2

3

6 (1)