并行编译与优化
Parallel Compiler and Optimization

计算机研究所编译系统室 沈洁

Lecture Four: Control Flow Analysis

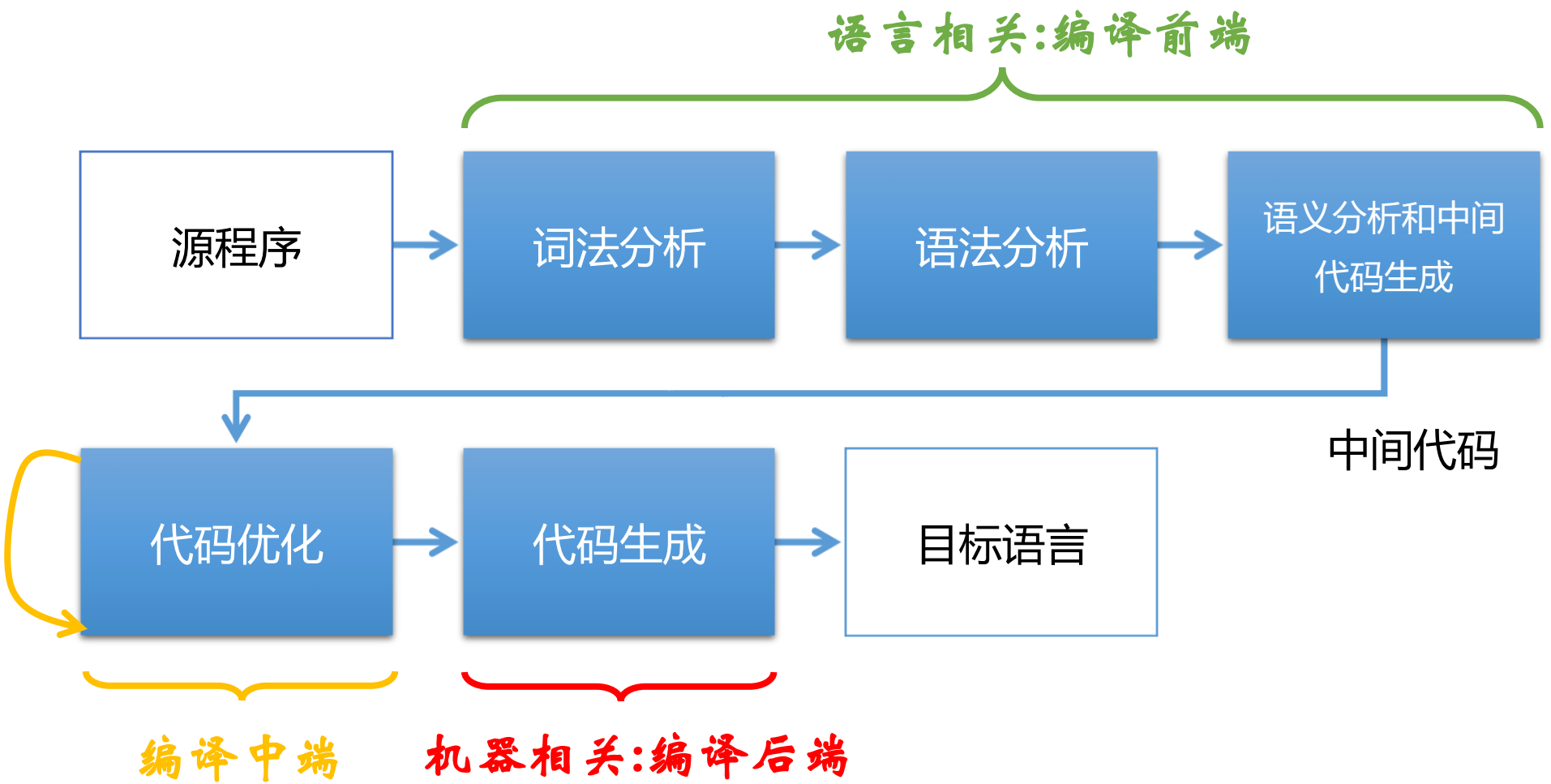
第四课：控制流分析

2023-03-19

复习：编译器结构

COMP CCRG Open MP

总体结构



复习：中间表示

COMP CCRG Open MP

中间表示代码

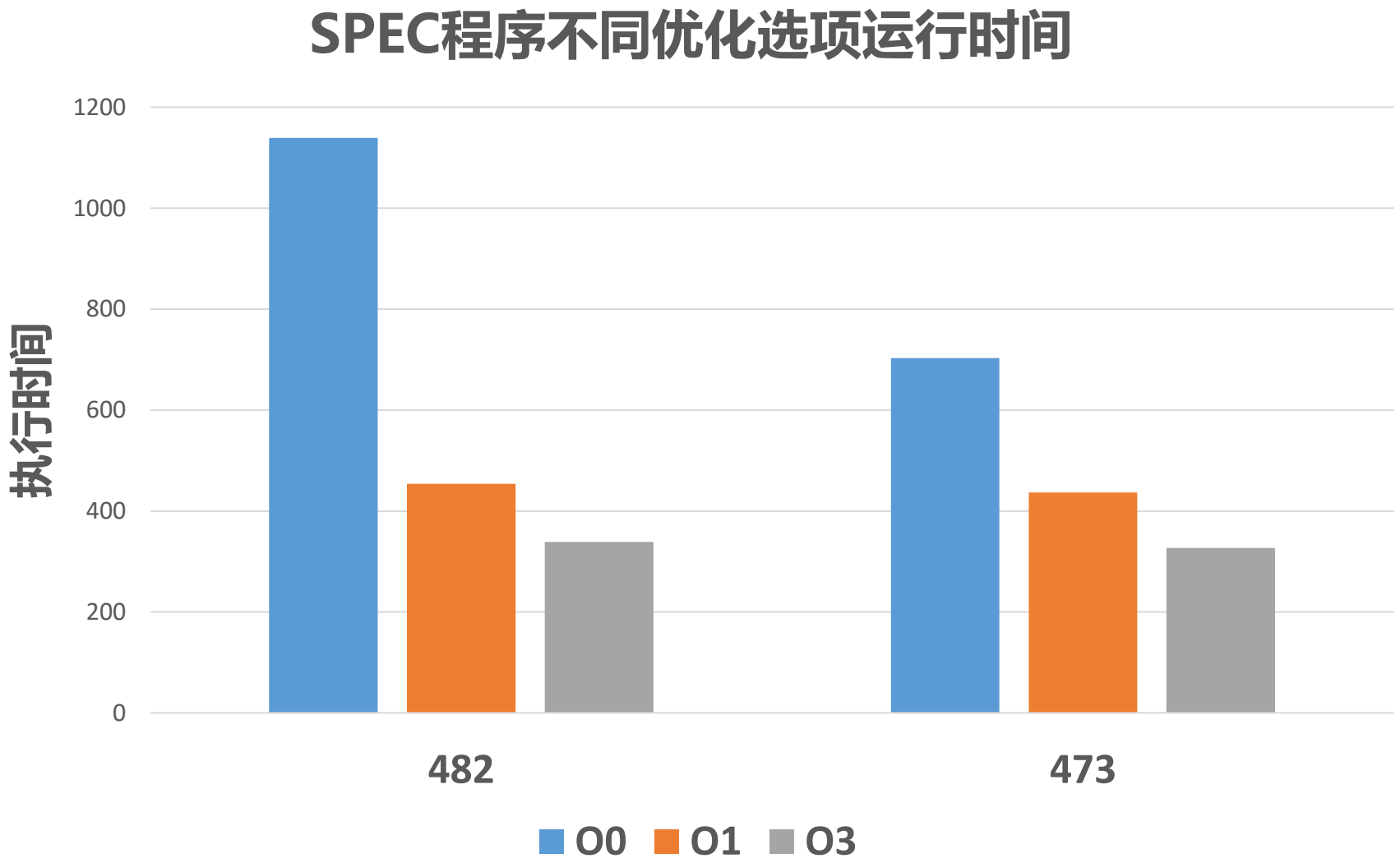
```
i=0;
for(i=0;i<n;i++)
     a = a+i;
if (n<b)
     a = b;
…
```

```
(1) i = 0;
(2) goto <L2>;
(3) <L1>:  a = a + i;
(4) i = i + 1;
(5) <L2>:  if (i < n) goto <L1>;
                      else goto <L3>;
(6) <L3>:  if (n < b) goto <L4>;
                      else goto <L5>;
(7) <L4>:  a = b;
(8) <L5>:  ……
```

三地址代码

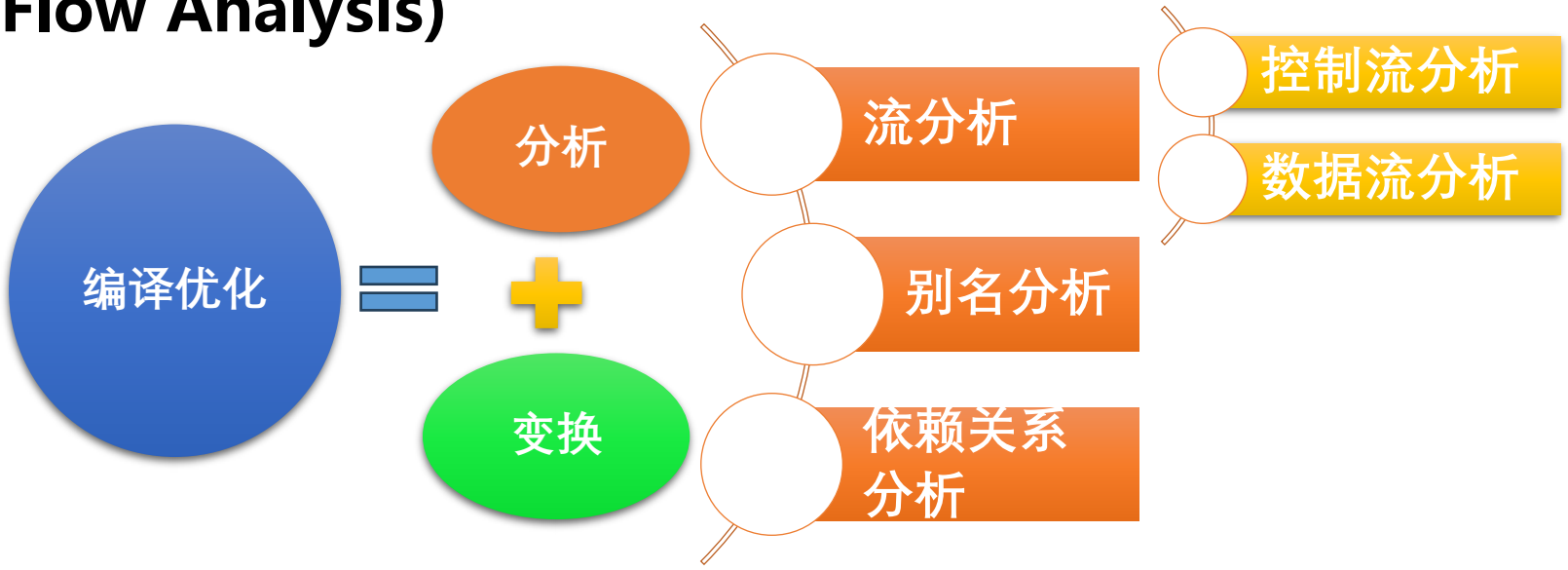
编译优化

- 优化是编译器的主要功能之一



编译优化

- 优化 = 分析 + 变换
- 分析是优化的基础
  - 流分析
    - 控制流分析(Control Flow Analysis)
    - 数据流分析(Data Flow Analysis)
  - 别名分析
  - 依赖关系分析



本次课：理解和掌握控制流分析方法

学习内容

COMP
CCRG Open MP

4.1 基本块

4.2 控制流图

4.3 必经关系

4.4 循环

4.5 加权控制流图、区域和轨迹

学习目标

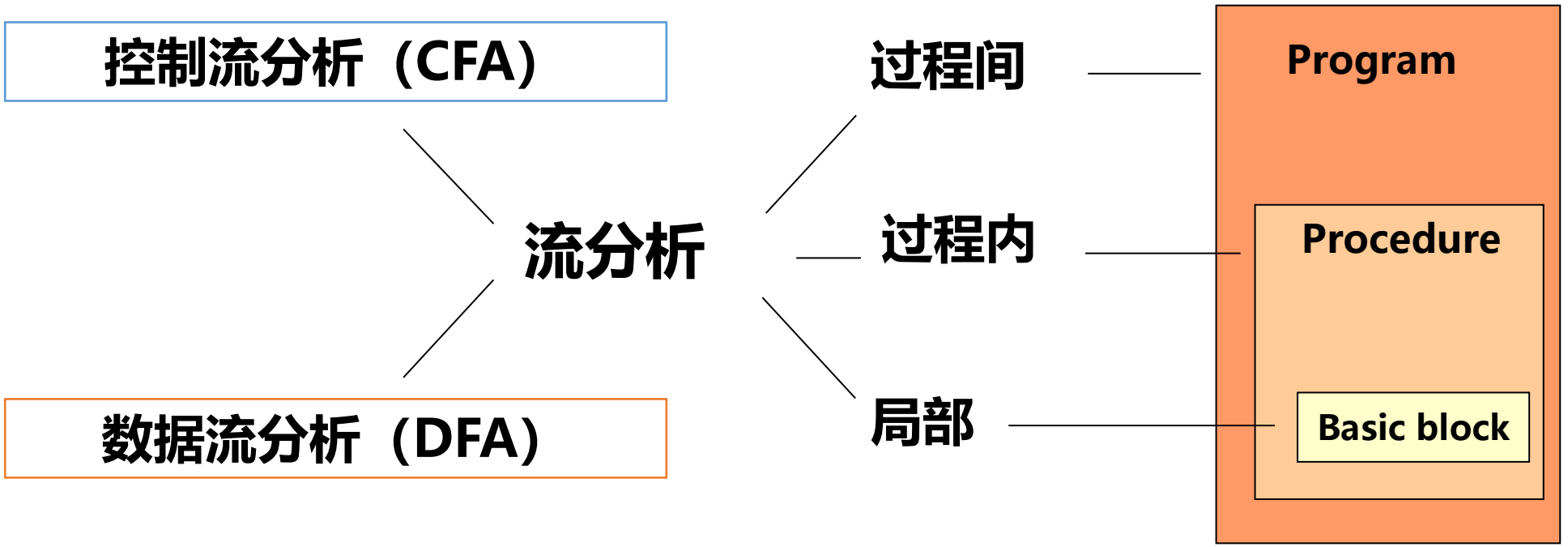
COMP CCRG Open MP

- 掌握识别基本块、构建控制流图、计算必经结点集合和找出循环的方法
- 熟悉加权控制流图
- 了解区域和轨迹

控制流分析

控制流分析

- 分析判断程序的控制结构，构建程序的控制流图(Control Flow Graph)



投票 最多可选1项

设置

控制流图中

A 结点是程序中的基本块，边表示程序必然走这条路径

B 结点是程序中的基本块，边表示程序可能走这条路径

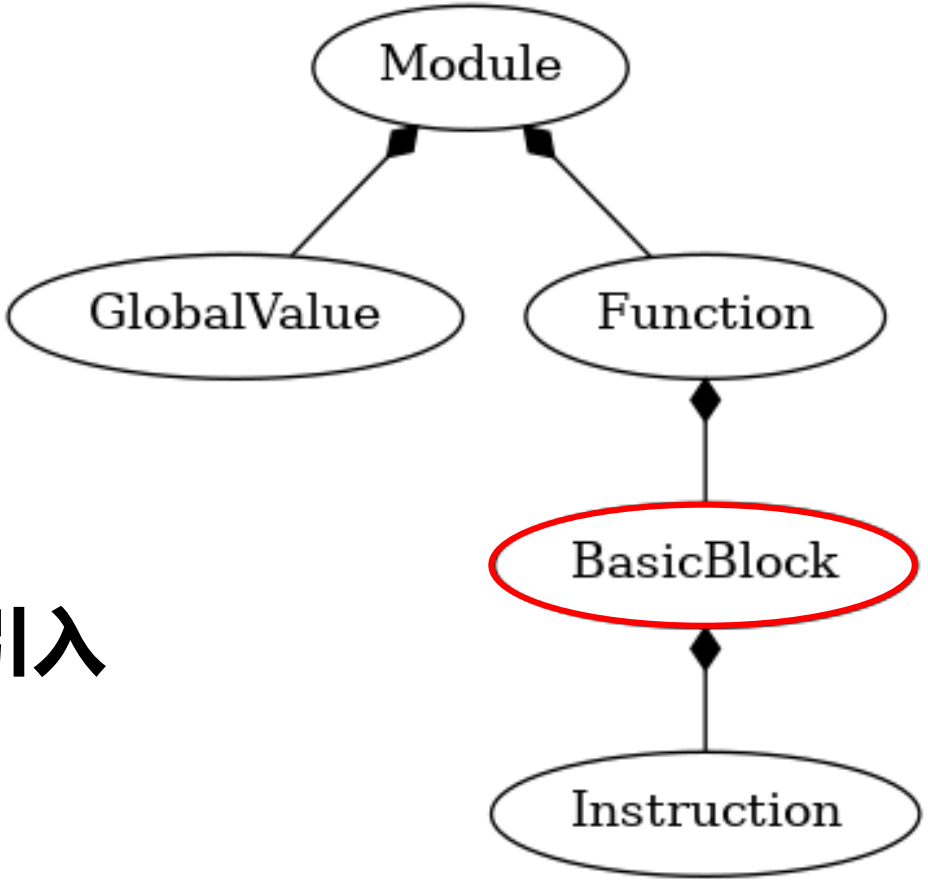
C 结点是程序中的函数，边表示程序必然走这条路径

D 结点是程序中的函数，边表示程序可能走这条路径

提交

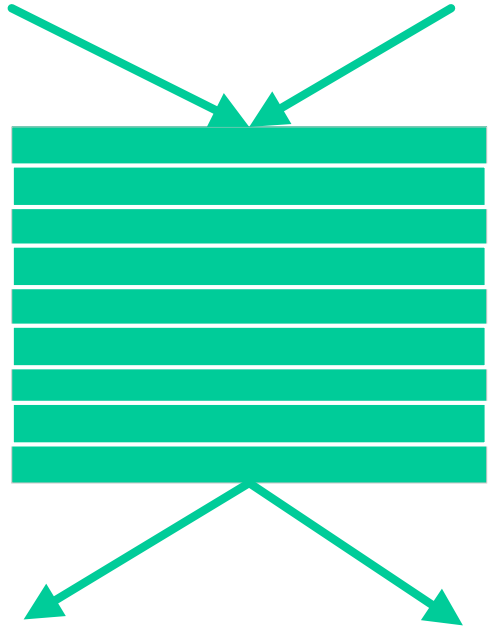
SysY IR的基本结构

- Module
  - 顶层的IR结构，对应CompUnit
- GlobalValue
  - 全局变量Decl
- Function
  - 函数定义FuncDef
- BasicBlock
  - 基本块，由If-else、while控制结构引入
- Instruction
  - 算术逻辑运算、比较、类型转换操作



4.1.1 基本块的基本概念

- 基本块(Basic Block)
  - 基本块是一段只能从它的开始处进入，结束处离开的顺序代码序列
  - 基本块只有最后一条语句可以是分支语句，并且只有第一条语句可以是分支的目标
  - 在基本块内部，除了第一条指令外，每条指令都只有一个前驱；除了最后一条指令外，每条指令只有一个后继



- 使用基本块代替指令，可以减少分析的时间和空间开销

4.1.2 识别基本块

- **第一步：扫描程序，使用下列规则识别基本块的首语句 (leader statement)**

(规则1) 程序/函数的第一条语句

(规则2) 分支语句的目标 (对大多数中间表示代码，分支的目标是带有标号的语句)

(规则3) 任何紧跟在分支或者return语句后面的语句

4.1.2 识别基本块

```
i=0;
for(i=0;i<n;i++)
     a = a+i;
if (n<b)
     a = b;
…
```

规则1

```
(1) i = 0;
(2) goto <L2>;
(3) <L1>:  a = a + i;
(4) i = i + 1;
(5) <L2>:  if (i < n) goto <L1>;
                      else goto <L3>;
(6) <L3>:  if (n < b) goto <L4>;
                      else goto <L5>;
(7) <L4>:  a = b;
(8) <L5>:  ……
```

三地址代码

4.1.2 识别基本块

```
i=0;
for(i=0;i<n;i++)
     a = a+i;
if (n<b)
     a = b;
…
```

规则1 (1) i = 0;

(2) goto <L2>;

规则2 (3) <L1>: a = a + i;

(4) i = i + 1;

规则2 (5) <L2>: if (i < n) goto <L1>;
else goto <L3>;

规则2 (6) <L3>: if (n < b) goto <L4>;
else goto <L5>;

规则2 (7) <L4>: a = b;

规则2 (8) <L5>: ……

三地址代码

4.1.2 识别基本块

- **第一步：识别首语句**
- **第二步：每个首语句对应的基本块包括首语句**
  - **（1）到下一个首语句之间（不包括下一个首语句）的所有语句，**
    或者
  - **（2）到程序的结尾**

4.1.2 识别基本块

```
i=0;
for(i=0;i<n;i++)
     a = a+i;
if (n<b)
     a = b;
…
```

B1
```
(1) i = 0;
(2) goto <L2>;
```

B2
```
(3) <L1>:  a = a + i;
(4) i = i + 1;
```

B3
```
(5) <L2>:  if (i < n) goto <L1>;
                      else goto <L3>;
```

B4
```
(6) <L3>:  if (n < b) goto <L4>;
                      else goto <L5>;
```

B5
```
(7) <L4>:  a = b;
```

B6
```
(8) <L5>:  ……
```

三地址代码

学习内容

4.1 基本块

4.2 控制流图

4.3 必经关系

4.4 循环

4.5 加权控制流图、区域和轨迹

4.2.1 控制流图的基本概念

- 控制流图(Control Flow Graph ,CFG)是一个有向流图 $G=(N,E)$，其中：
  - 结点N表示基本块
  - 边E表示程序的控制流向
- 首语句是第一条语句的基本块称为开始结点(start node)
- CFG没有给出数据的任何信息，CFG中的边只表示程序可能走这条路径

4.2.2 构建控制流图

- 如果基本块B1和B2满足下面的条件之一，则存在一条从B1到B2的有向边

  (条件1) 从B1的最后一条语句可以跳转到B2的第一条语句

  (条件2) B2紧跟在B1之后，并且B1的最后一条指令不是无条件转移

4.2.2 构建控制流图

B1
（1）i = 0;
（2）goto B3

B2
（3）<L1>: a = a + i;
（4）i = i + 1;

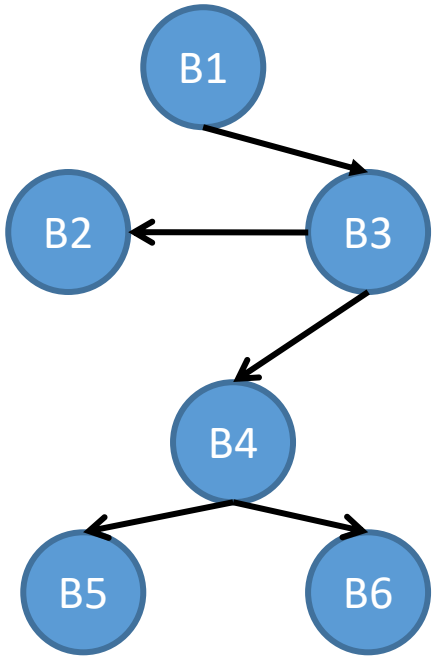
B3
（5）<L2>: if (i < n) goto B2;
else goto B4

B4
（6）<L3>: if (n < b) goto B5
else goto B6

B5
（7）<L4>: a = b;

B6
（8）<L5>: ……

条件1



4.2.2 构建控制流图

B1
（1）i = 0;
（2）goto B3

B2
（3）<L1>: a = a + i;
（4）i = i + 1;

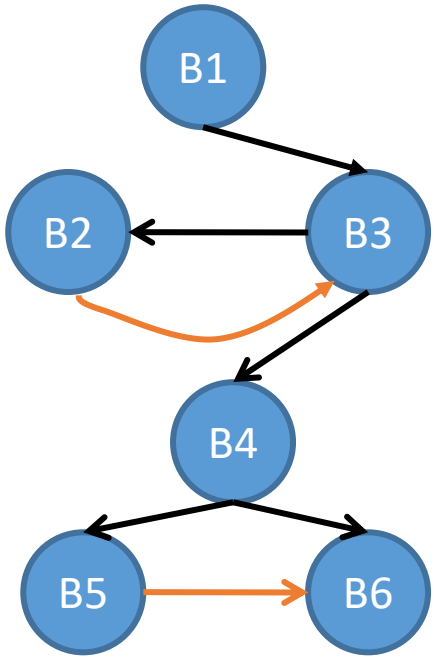
B3
（5）<L2>: if (i < n) goto B2;
else goto B4

B4
（6）<L3>: if (n < b) goto B5
else goto B6

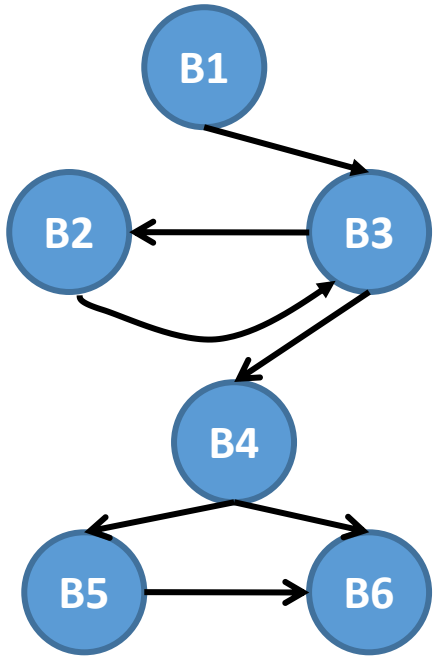
B5
（7）<L4>: a = b;

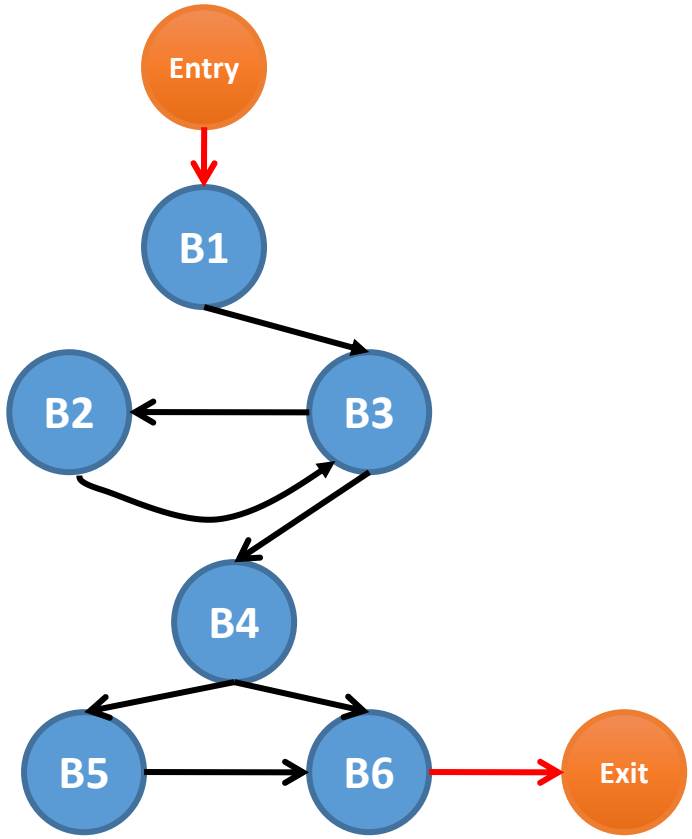
B6
（8）<L5>: ……

条件2



4.2.3 控制流图中的特殊结点





入口结点(Entry node): Entry结点到开始结点有一条边

出口结点(Exit node): CFG中所有没有后继的结点到Exit结点有一条边

4.2.4 前驱和后继

- 定义: 控制流图 $G = (N, E, s)$，其中：

  N: 结点集合

  E: 边的集合

  s: 入口结点

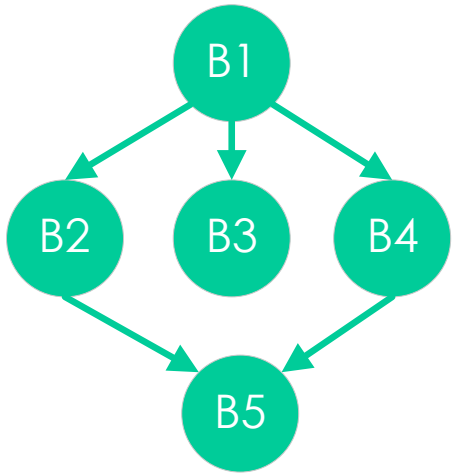
  有 $a \in N, b \in N$.

$$pred[b] = \{a \in N \mid \exists e \in E, e = a \rightarrow b\}$$

$$succ[b] = \{a \in N \mid \exists e \in E, e = b \rightarrow a\}$$

a 是**分支结点** $if\ succ[a] > 1$

a 是**汇合结点** $if\ pred[a] > 1$



4.2.4 前驱和后继

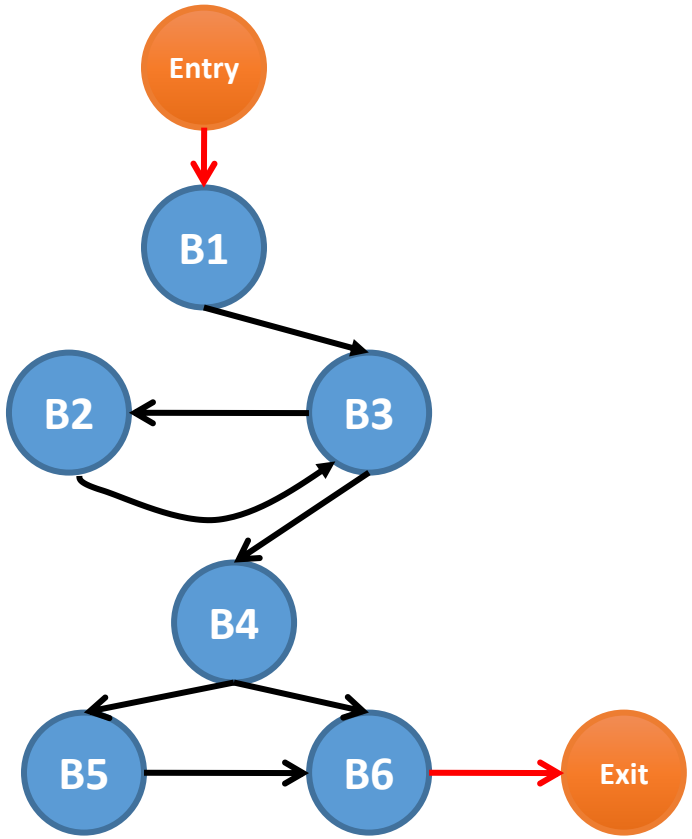
```
i=0;
for(i=0;i<n;i++)
     a = a+i;
if (n<b)
     a = b;
…
```

Pred(Entry) = ∅
Succ(Entry) = {B1}

Pred(B1) = {Entry}
Succ(B1) = {B3}

Pred(B3) = {B1,B2} 既是汇合结点
Succ(B3) = {B2,B4} 也是分支结点

Pred(Exit) = {B6}
Succ(Exit) = ∅



学习内容

COMP
CCRG Open MP

4.1 基本块

4.2 控制流图

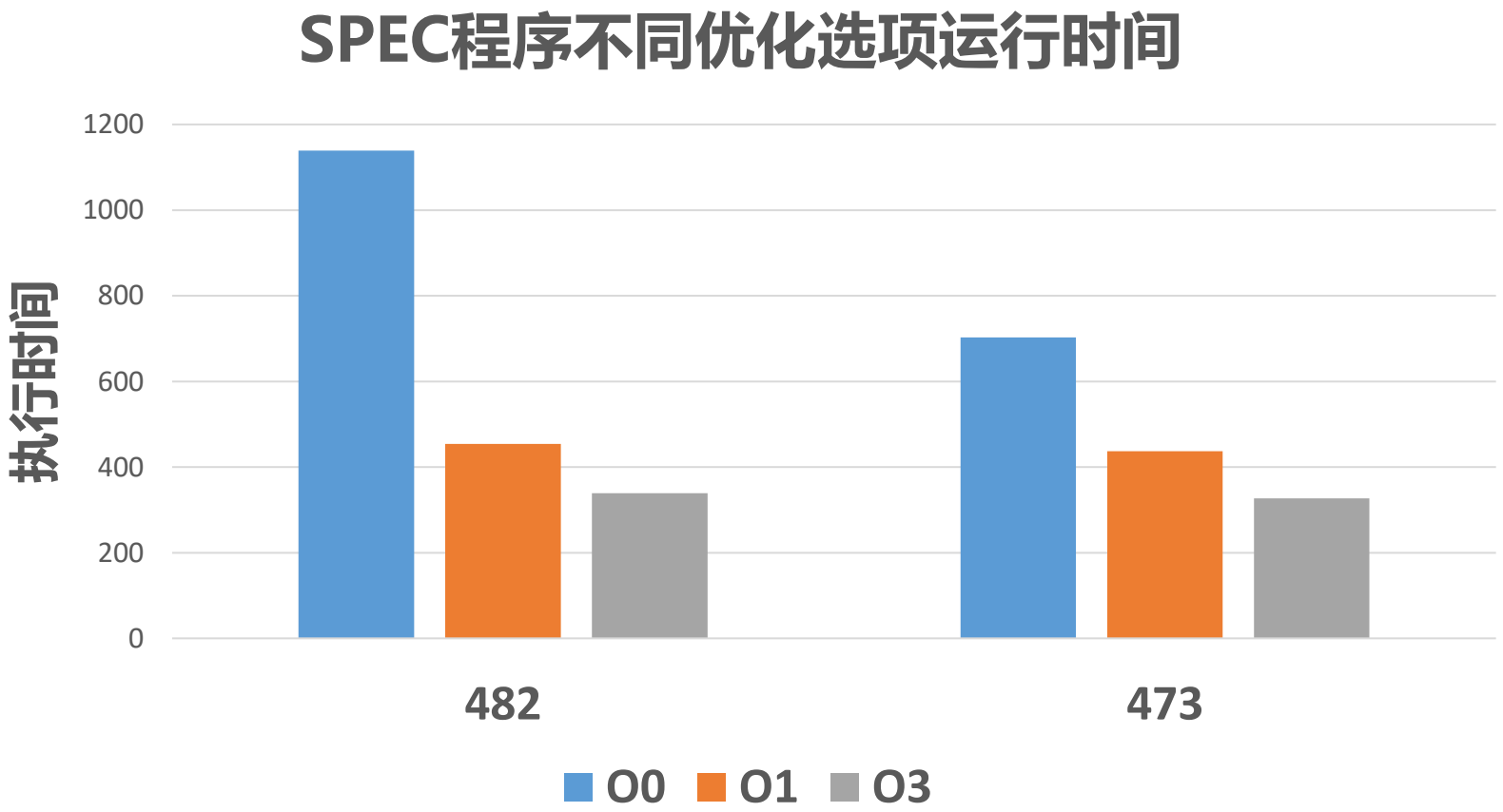
4.3 必经关系

4.4 循环

4.5 加权控制流图、区域和轨迹

为什么引入必经结点关系

- 目的之一：找出循环
  - 循环占据了SPEC CPU程序90%以上的执行时间，是性能瓶颈

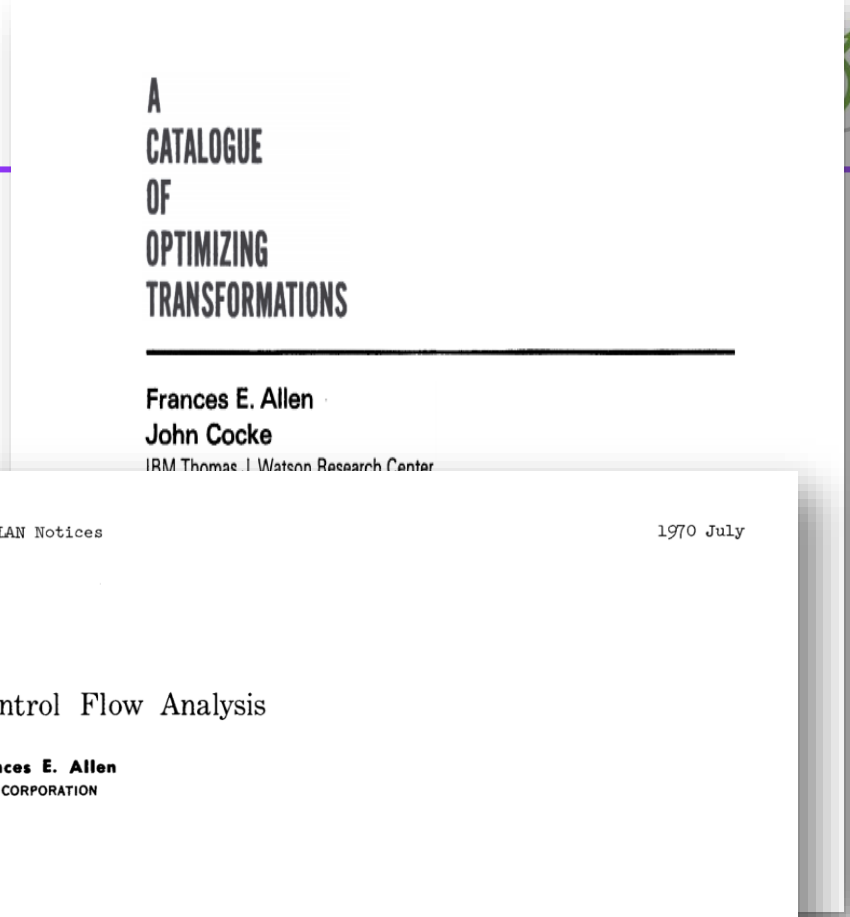


Frances E. Allen

- 全球首位女性图灵奖获得者 (2006获奖)

"pioneering contributions to the theory and practice of optimizing compiler techniques that laid the foundation for modern optimizing compilers and automatic parallel execution."

- 毕业后加入IBM实验室
- 研究编译器这件"小事"，45年而已
- 编程与登山一样，充满挑战



A
CATALOGUE
OF
OPTIMIZING
TRANSFORMATIONS

Frances E. Allen
John Cocke
IBM Thomas J. Watson Research Center

SIGPLAN Notices 1970 July

Control Flow Analysis

Frances E. Allen
IBM CORPORATION

INTRODUCTION

Any static, global analysis of the expression and data relationships in a program requires a knowledge of the control flow of the program. Since one of the primary reasons for doing such a global analysis in a compiler is to produce optimized programs, control flow analysis has been embedded in many compilers and has been described in several papers. An early paper by Prosser [5] described the use of Boolean matrices (or, more particularly, connectivity matrices) in flow analysis. The use of "dominance" relationships in flow analysis was first introduced by Prosser and much expanded by Lowry and Medlock [6]. References [6,8,9] describe compilers which use various forms of control flow analysis for optimization. Some recent developments in the area are reported in [4] and in [7].

★ 银河、天河编译系统研发也已46年
陈火旺院士主持为银河-I研制支持向量扩展的Fortran编译器

4.3.1 必经结点

COMP CCRG Open MP

也称为支配结点

> CFG中，如果从入口结点到结点b的每一条路径都经过结点a，则称a是b的必经结点。

b的所有必经结点构成b的必经结点集合dom(b)

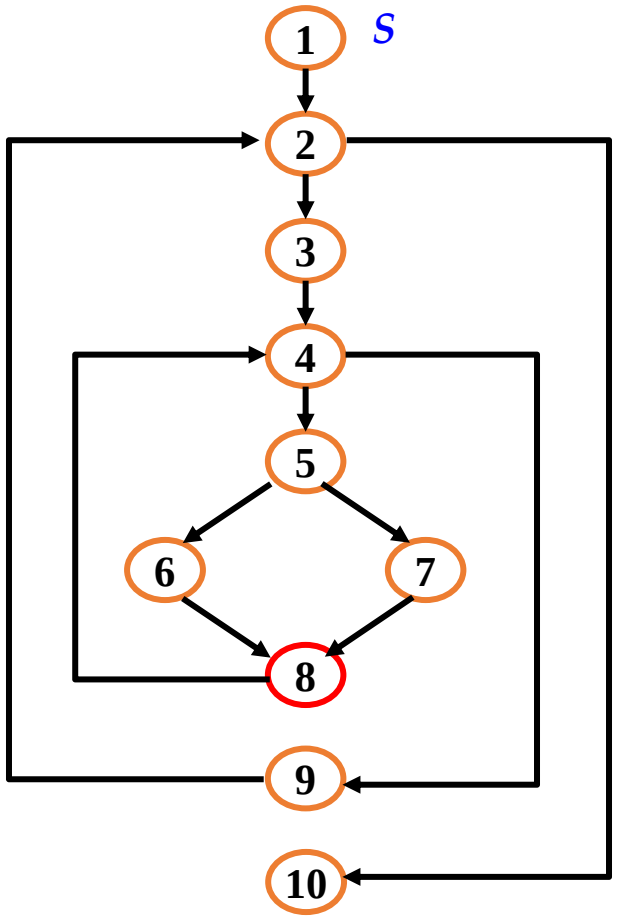
必经结点关系是一种**偏序**关系：

1. **自反的**, $a \in dom(a)$
2. **传递的**, $a\ dom\ b$ & $b\ dom\ c$，则$a\ dom\ c$
3. **反对称的**, if $a\ dom\ b$ & $b\ dom\ a$，则$a = b$

4.3.2 必经关系

1. 如果从入口结点到b的每一条路径，都包括a，
   a是b的必经结点，记作 $a \le b$ 或 (a, b)
2. 如果$a \le b$ 且 $a \ne b$
   a是b的严格必经结点， $a < b$ 或 a sdom b
3. 如果$a < b$ 且不存在一个结点 c ∈N，满足 a<c< b
   a是b的直接必经结点，$a <_i b$ ，也记作a = idom (b) 或 a idom b
4. 直接必经结点是唯一的

4.3.2 必经关系



必经关系:

{ (1, 1), (1, 2), (1, 3), (1,4) … ,(1,10)
(2, 2), (2, 3), …, (2, 10)
(3, 3),(3, 4),…, (3,9)
(4, 4),(4, 5),…, (4,9)
(5, 5),(5, 6) ,(5, 7),(5,8)
(6, 6), (7, 7) ,(8, 8) ,(9, 9) ,(10, 10)
}

直接必经关系:

$1 <_i 2$, $2 <_i 3$, $3 <_i 4$, $4 <_i 5$, $4 <_i 9$,
$5 <_i 6$, $5 <_i 7$, $5 <_i 8$, $2 <_i 10$

必经结点集合:

dom (1) = {1}
dom (2) = {1, 2}
dom (3) = {1, 2, 3}
dom (10) = {1, 2, 10)

4.3.3 计算必经结点集合

基本思想

- 结点b是b的必经结点
- 如果a是b的唯一前驱，则a是b的必经结点
- 如果a是b所有前驱的必经结点，则a是b的必经结点

$$\forall p \in pred[b], \text{if } a \le p, \text{则 } a \le b$$

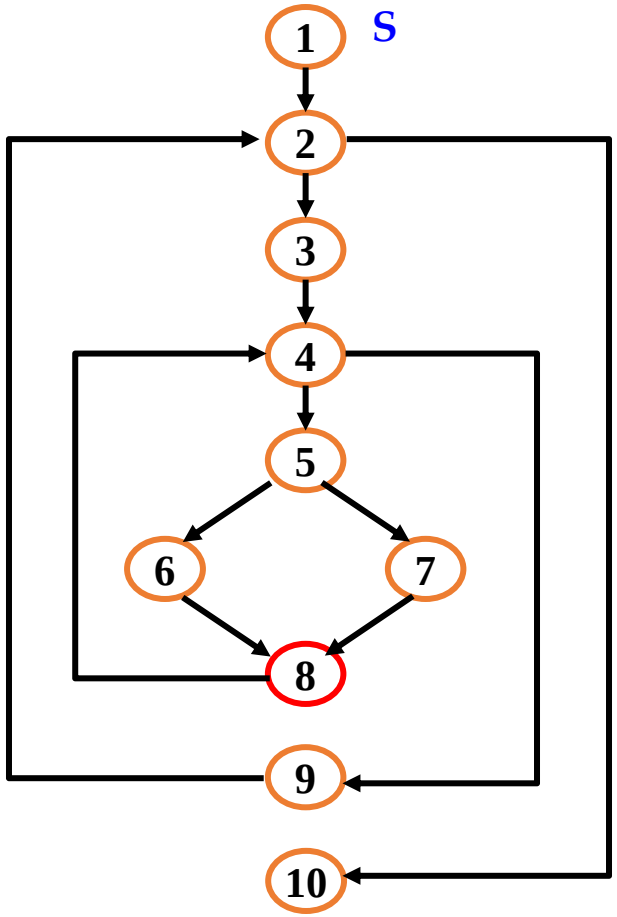
4.3.3 计算必经结点集合

- 求结点a的必经结点集合dom[a]
  - 对于入口结点s，dom[s] = {s}
  - 对于结点$a \neq s$，dom[a] = {a} ∪ ($\cap_{p \in pred[a]}$ dom[p])

```
dom(s) = {s}
for n ∈ N – {s} do dom(n) = N
repeat
   changed = false
   for n ∈ N – {s} {
     olddom = dom(n)
     dom(n) = {n} ∪ ∩p ∈ pred(n) dom(p)
      if dom(n) ≠ olddom then changed = true
   }
until changed = false
```

计算复杂性：$O(N^2)$

4.3.3 计算必经结点集合



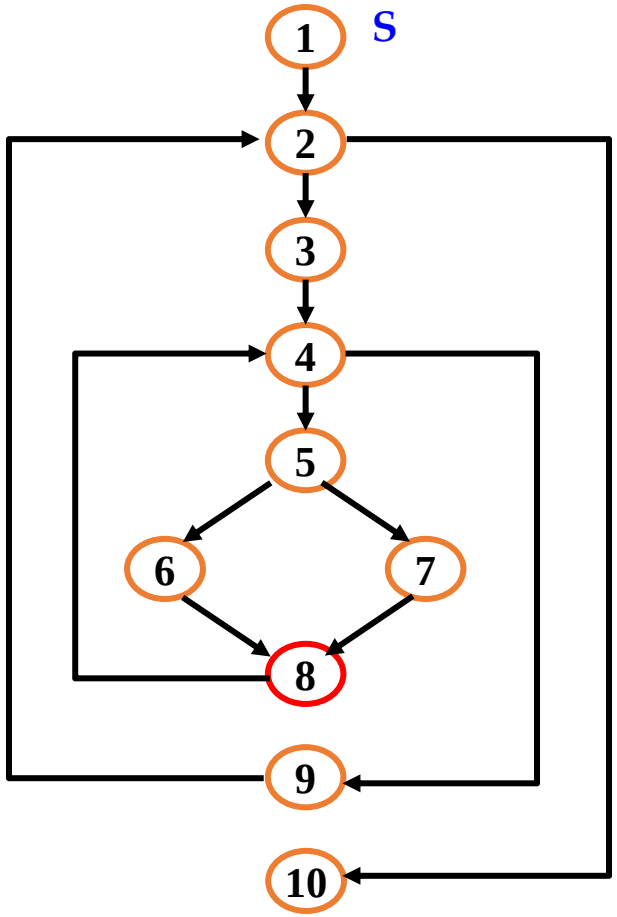
必经结点集合:

结点a	dom(a)	备注
1	{1}	1是入口结点
2	{1,2}	2的唯一前驱是1
3	{1,2,3}	3的唯一前驱是2
4	{1,2,3,4}	4的唯一前驱是3
5	{1,2,3,4,5}	5的唯一前驱是4
6	{1,2,3,4,5,6}	6的唯一前驱是5
7	{1,2,3,4,5,7}	7的唯一前驱是5
8		
9		
10		

填空题 1分

设置

dom(8)=[填空1]

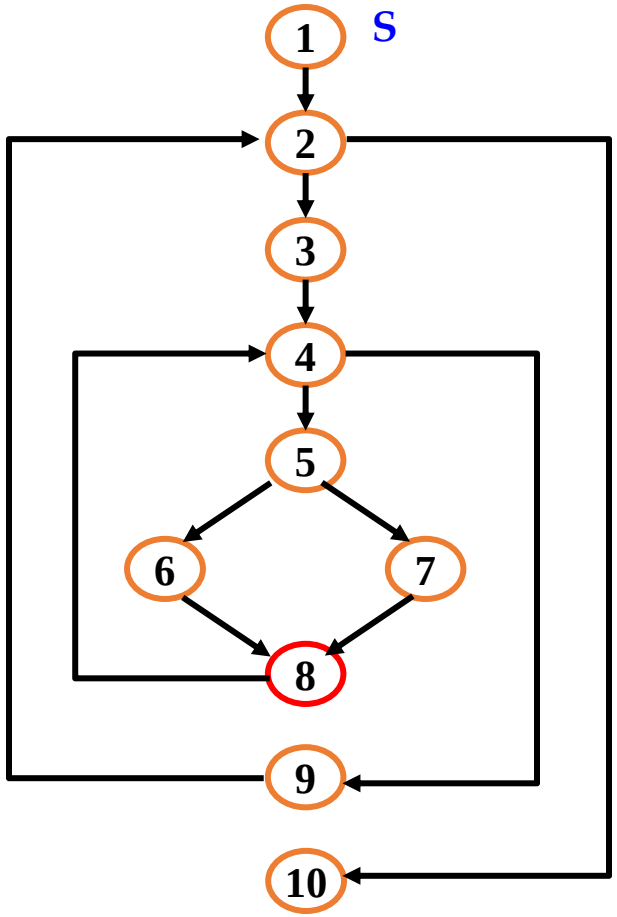


必经结点集合:

结点a	dom(a)	备注
1	{1}	1是入口结点
2	{1,2}	2的唯一前驱是1
3	{1,2,3}	3的唯一前驱是2
4	{1,2,3,4}	4的唯一前驱是3
5	{1,2,3,4,5}	5的唯一前驱是4
6	{1,2,3,4,5,6}	6的唯一前驱是5
7	{1,2,3,4,5,7}	7的唯一前驱是5
8		
9		
10		

作答

4.3.3 计算必经结点集合

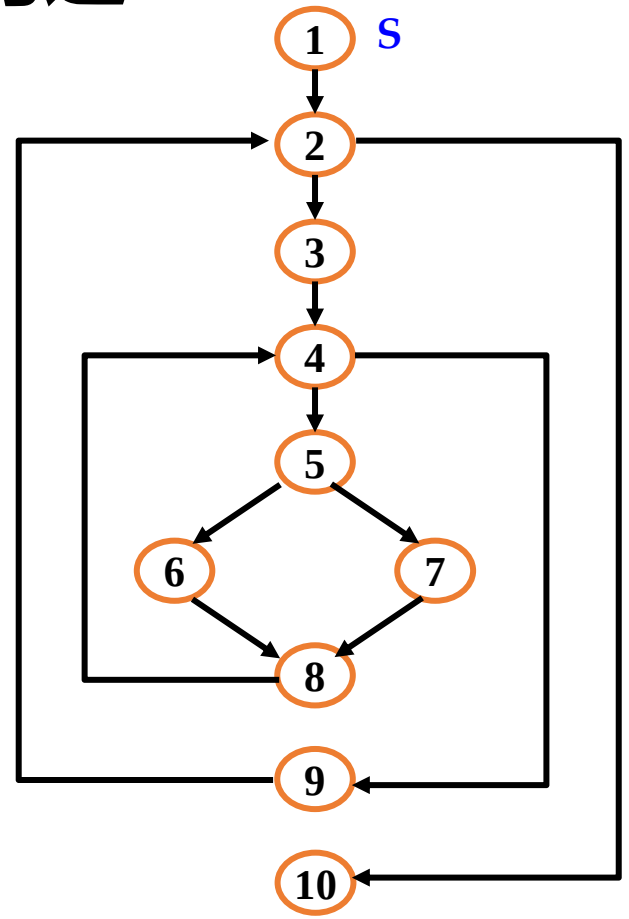


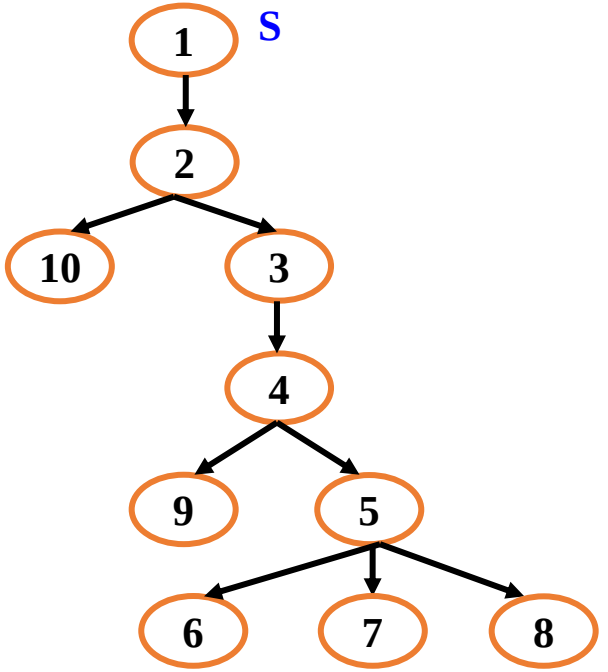
必经结点集合:

结点a	dom(a)	备注
1	{1}	1是入口结点
2	{1,2}	2的唯一前驱是1
3	{1,2,3}	3的唯一前驱是2
4	{1,2,3,4}	4的唯一前驱是3
5	{1,2,3,4,5}	5的唯一前驱是4
6	{1,2,3,4,5,6}	6的唯一前驱是5
7	{1,2,3,4,5,7}	7的唯一前驱是5
8	{1,2,3,4,5,8}	8的前驱是6和7
9	{1,2,3,4,9}	9的前驱是4
10	{1,2,10}	10的前驱是2

4.3.4 必经结点树

- 必经结点树包含CFG所有结点，每个结点a的直接必经结点到a有一条有向边

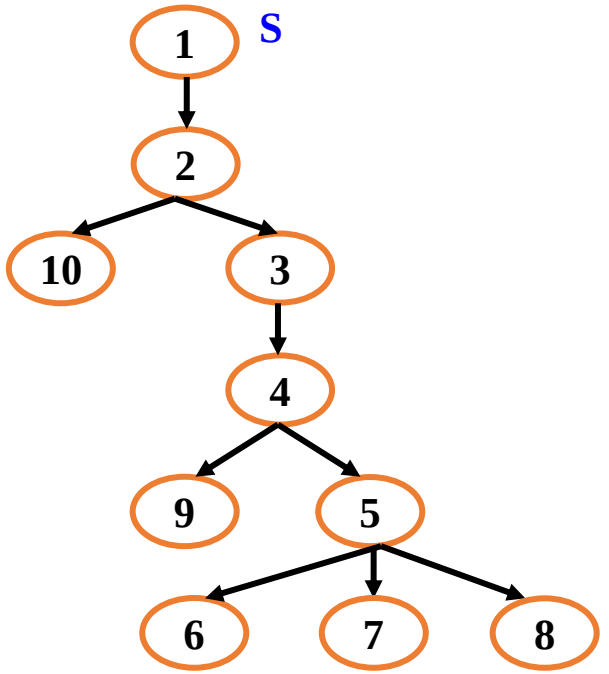




必经结点树

4.3.4 必经结点树

- **必经结点树包含CFG所有结点，每个结点*a*的直接必经结点到*a*有一条有向边**
  - 入口结点s是根
  - 每个结点是其后代的必经结点



必经结点树

学习内容

COMP
CCRG Open MP

4.1 基本块

4.2 控制流图

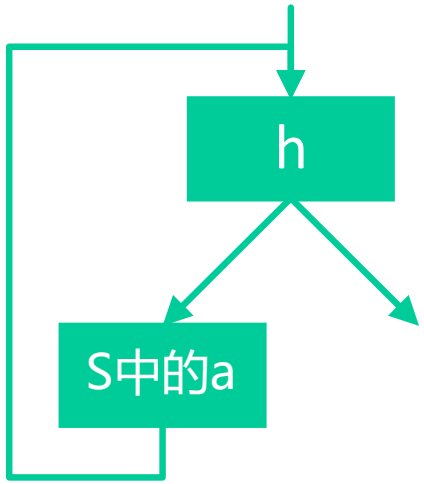
4.3 必经关系

4.4 循环

4.5 加权控制流图、区域和轨迹

4.4.1 循环的基本概念

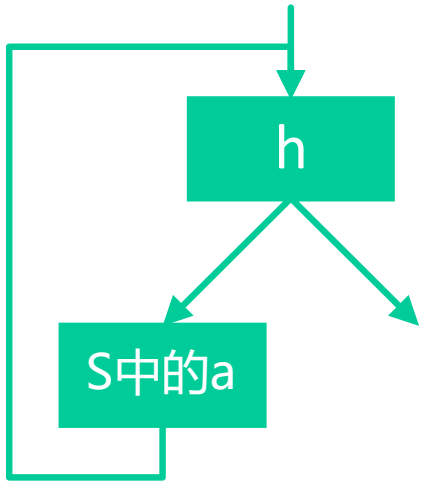
- 在控制流图中，循环是满足下列条件的结点集合S：
  - S包括一个头结点h
  - 对S中任意结点a，都有一条有向边组成的路径从a到h
  - 从h到S中任意结点a，都存在一条路径
  - 从S之外的任意结点到达S中的结点，都必须经过h



4.4.1 循环的基本概念

循环是CFG中的强连通部分

- 循环是控制流图的一个强连通子图，在这个子图中，每个结点都可以通过一条路径到达另一个任意结点
- 头节点是强连通子图中所有结点的必经结点



4.4.2 找出循环

- 向后边
  - 控制流图中，如果结点$a < b$，边$b \rightarrow a$称为回边
- 循环
  - 给定一条回边$b \rightarrow a$，和它关联的循环是指以a为必经结点，并且能够不经过a到达b的所有结点构成的循环，a为头结点
- 找出循环的方法

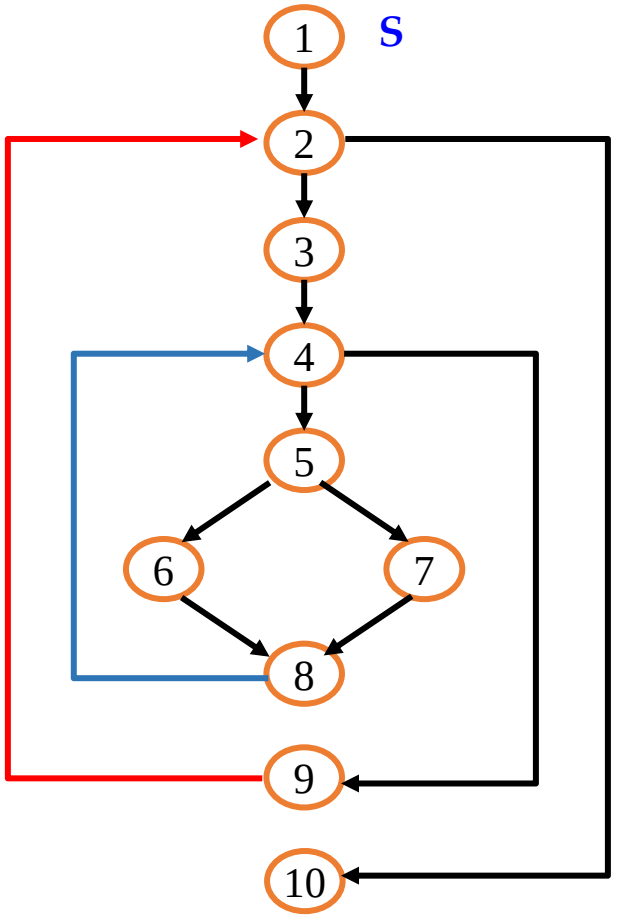
  (1) 找出回边$b \rightarrow a$

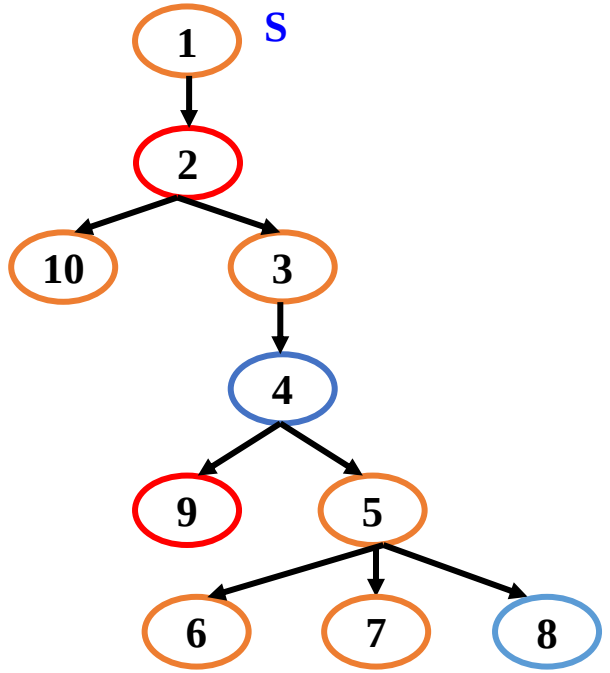
  (2) 找出以a为必经结点的所有结点

  (3) 从(2)中选出可以从这些结点到达b，但不经过a的所有结点

4.4.2 找出循环

(1) 找出回边$b \rightarrow a$



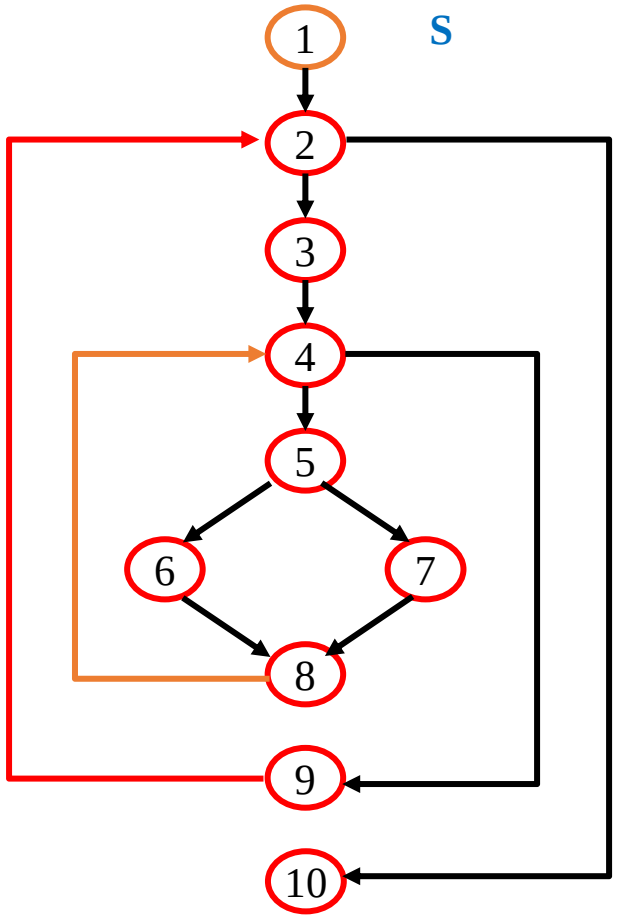


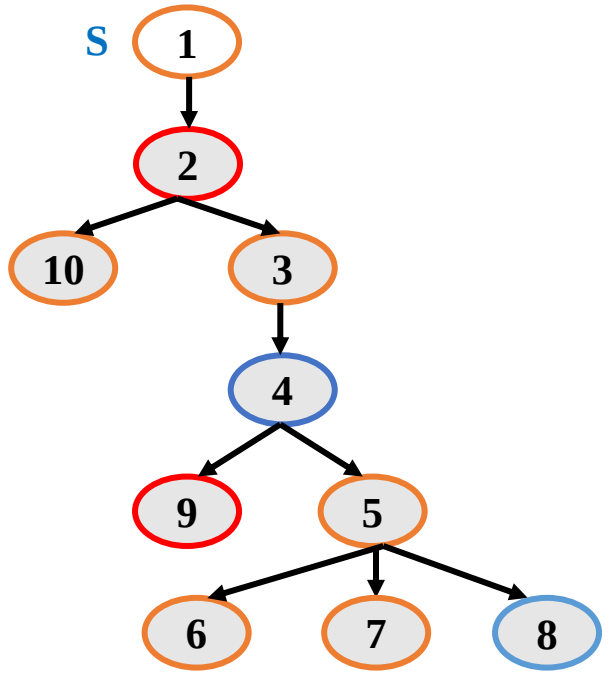
回边：

9→2， 8→4

4.4.2 找出循环

(2) 找出以*a*为必经结点的所有结点

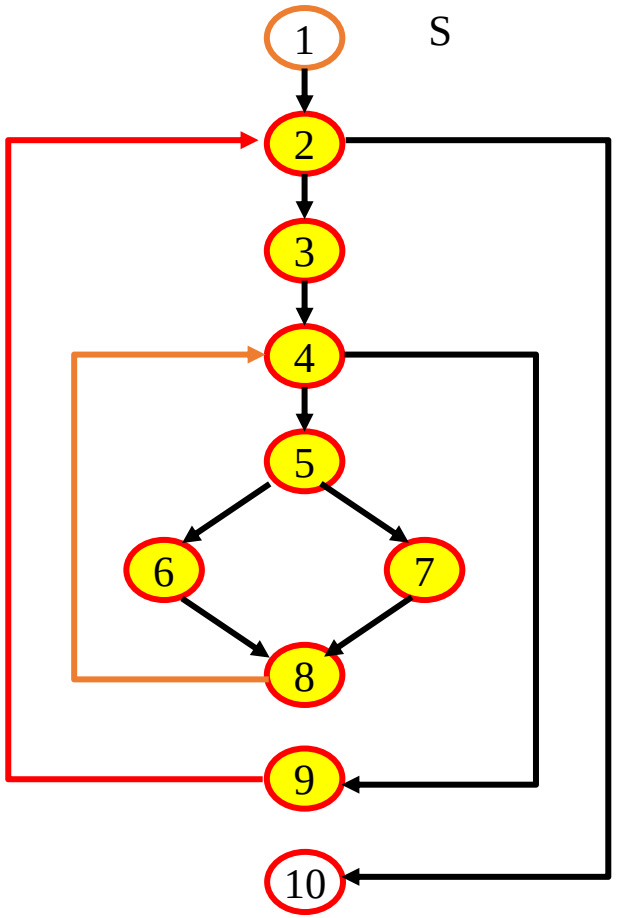




回边9→2：(2, 2), (2, 3), …, (2, 10)
回边8→4：(4, 4), (4, 5), …, (4, 9)

4.4.2 找出循环

(3) 从(2)中选出可以从这些结点到达*b*，但不经过*a*的所有结点

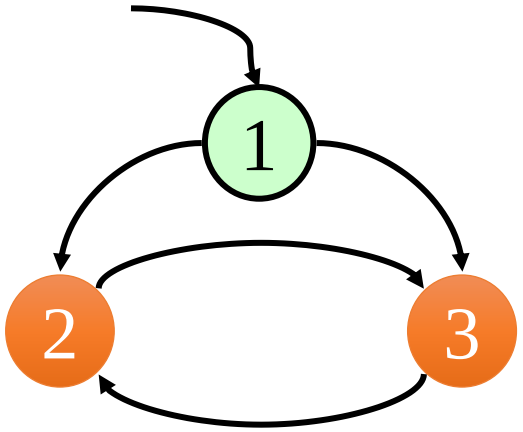


Loop1：{2, 3, 4, 5, 6, 7, 8, 9}

Loop2：{4, 5, 6, 7, 8}

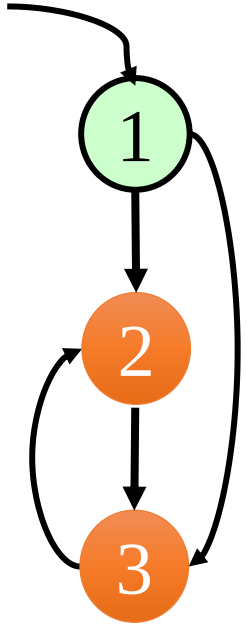
4.4.3 自然循环

- 自然循环只有一个头结点，即唯一的入口



强连通子图

是自然循环吗?



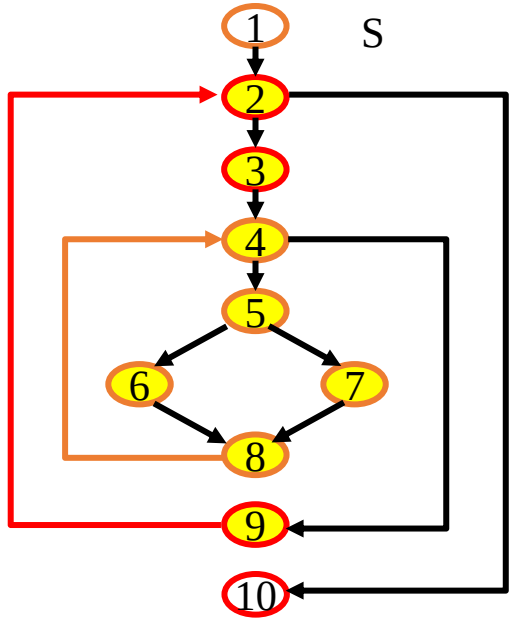
以深度为主的表示方式

有回边3→2, 但2不是3的必经结点

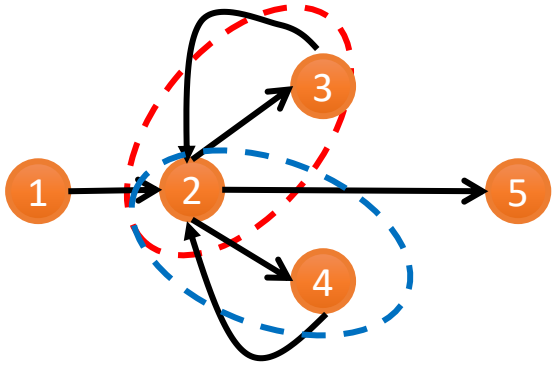
回边3→2定义的循环不是自然循环，因为这个循环有两个入口（结点2和结点3）

4.4.4 内层(嵌套)循环

- 假设A 和 B是循环，头结点分别为 a、b
  - a ≠ b
  - b ∈ A
- 则
  - B是A的真子集
  - B是A的嵌套循环，即B是A的内层循环



- 思考?



哪一个是内层循环，如何处理?

优化方法之一: 循环合并

学习内容

COMP CCRG Open MP

4.1 基本块

4.2 控制流图

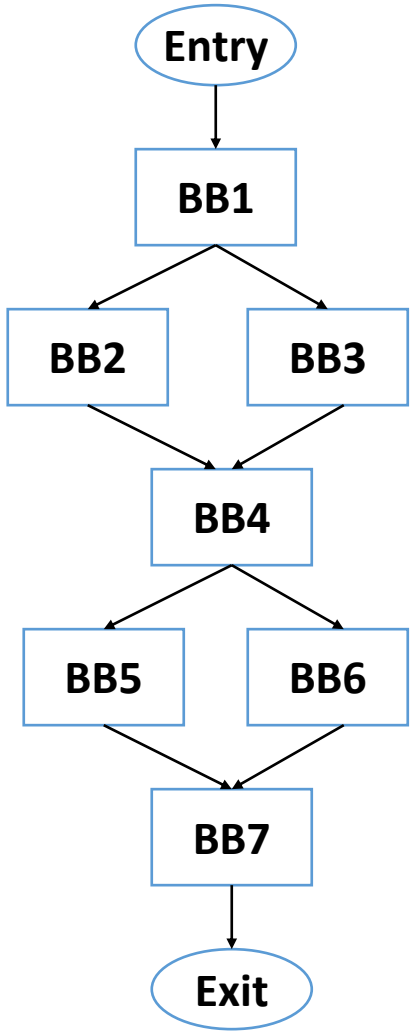
4.3 必经关系

4.4 循环

4.5 加权控制流图、区域和轨迹

4.5.1 加权控制流图

- CFG中的边仅仅意味着程序**可能**走这条路径



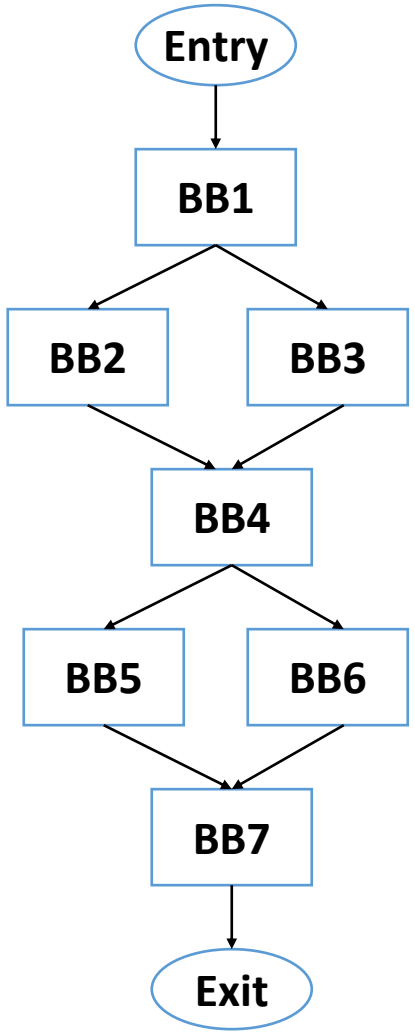
代码编写顺序

基本块排序

思考：这样的排序有什么问题?

4.5.1 加权控制流图

- CFG中的边仅仅意味着程序**可能**走这条路径



假设：BB3、BB6执行频率低

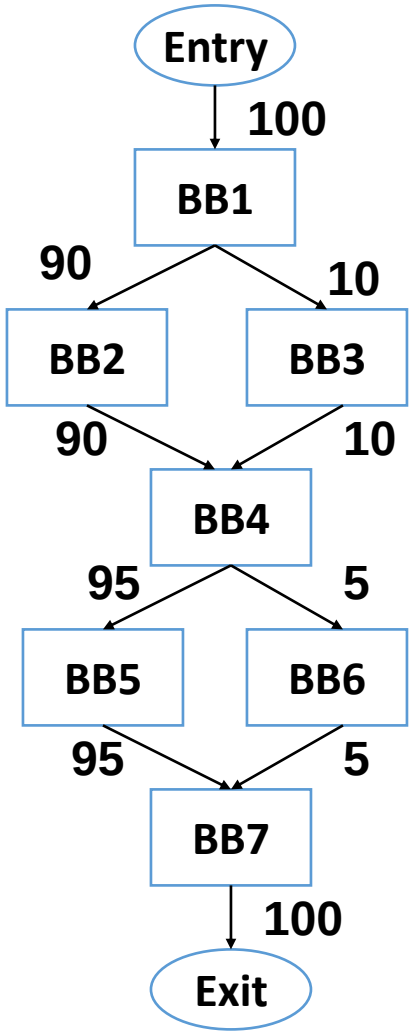
基本块排序

方便指令调度

开发基本块间的指令级并行

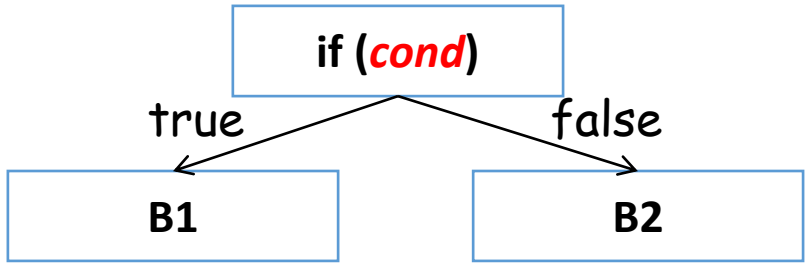
4.5.1 加权控制流图

- CFG的每一条边标记执行频率
  - 边的执行频率精度高于结点执行频率
- 如何获取执行频率
  - 控制流剖视信息(Profile)
    - 结点剖视信息
    - 边剖视信息
    - 路径剖视信息
  - 静态分支预测



4.5.1 加权控制流图

静态分支预测



启发信息

	condition	frequency
Pointer	!= NULL	85
operator	> 0	79
	!=	71
	fp_opcode	90
Successor	call	71
	loop	always

gcc/gcc/predict.def

4.5.2 区域 (Region)

- 被编译器当作整体对待的指令集合
  - 基本块
  - 循环
  - 函数/过程……

- 为什么引入区域?
  - 基本块包含的指令过少，不足以开发足够的并行
  - 函数/过程控制流过于复杂
  - 同时，函数/过程内的只有部分代码是“热代码”

4.5.2 区域 (Region)

- 希望区域具有的属性
  - 简单的控制流（最理想的就是基本块尽可能的大！）
  - “热”代码和“冷”代码分离
  - 区域包含的代码适中

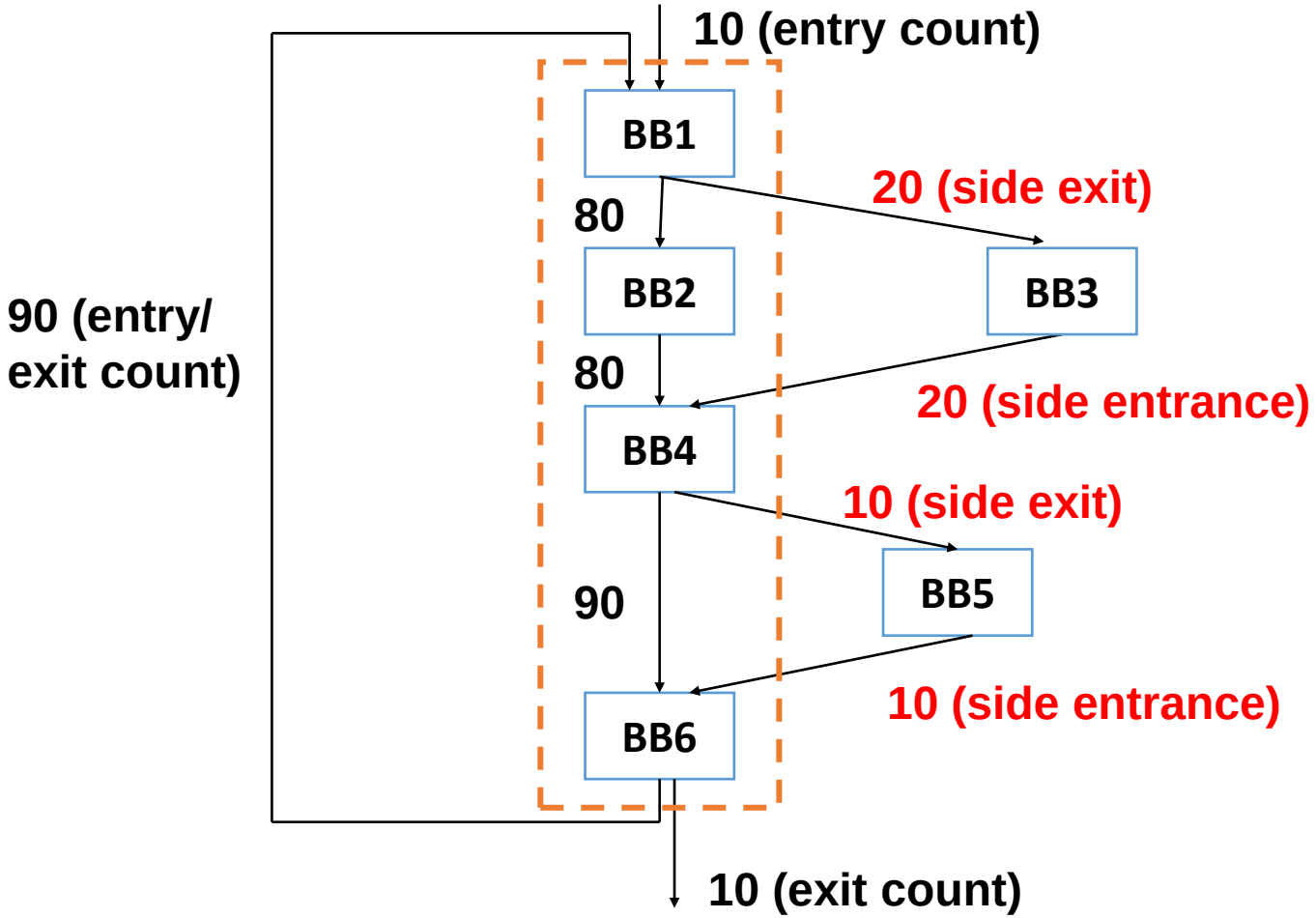
- 典型的区域
  - 轨迹（Trace）
  - 超块（Superblock）
  - Hyperblock

4.5.3 轨迹 (Trace)

- 能够顺序执行的基本块集合
  - 控制流图中的无环子图
- 构建轨迹的原则
  - 无环，不包含向后边
  - 频繁执行的路径
  - 一条轨迹的长度不能过长
  - 一个基本块只允许出现在一个轨迹中

4.5.3 轨迹 (Trace)

- 编译器将轨迹看作是基本块，做指令调度
  - 需要处理入口和出口副作用！！



本讲小结

4.1 基本块

- **识别基本块的方法**：找到首语句，划分基本块

4.2 控制流图

- **构建控制流图的方法**：构建基本块之间的有向边
- 入口、出口、前驱、后继、分支、汇合结点

4.3 必经关系

- 必经结点、必经关系
- **计算必经结点集合的方法**：$\mathrm{dom}[a] = \{a\} \cup (\cap_{p \in pred[a]} \mathrm{dom}[p])$

本讲小结

- 4. 循环
  - **找出循环的方法**：找回边$b \rightarrow a$，找出以a为必经结点，并且能够不经过a到达b的所有结点
- 5. 加权控制流图、区域和轨迹
  - 控制流图中的边仅意味着程序**可能**执行的路径，在边上标记执行频率
  - 区域和轨迹：合并基本块蜕化成新的结点，以开发指令级并行

作业

- 对如下中间表示描述的代码段，识别出基本块，画出控制流图，并识别控制流图中的循环

```
S1      a ← i
S2      c ← d
S3      i ← i + 5
S4      if(i > 5) goto S8
S5      f ← 10
S6      b ← f
S7      goto S14
S8      a ← a * c
S9      if(j == M) goto S11
S10     d ← a – c
S11     j ← j + 1
S12     b← a + j
S13     if(j > 10) goto S8
S14     print a, b
```

参考资料

- 《高级编译器设计与实现》（鲸书）第7章
- 参考论文
  - Thomas Lengauer and Robert Endre Tarjan. A fast algorithm for finding dominators in a flowgraph. ACM Trans. Program. Lang. Syst. 1, 1 (1979), 121–141.
  - D Harel. A linear algorithm for finding dominators in flow graphs and related problems. In Proceedings of the seventeenth annual ACM symposium on Theory of computing (STOC '85), 185–194.
  - Thomas Ball and James R. Larus. Branch prediction for free. In Proceedings of the ACM SIGPLAN 1993 conference on Programming language design and implementation (PLDI '93), 300–313.