流图中的循环
Part1：概念、识别方法和应用

李 诚

国家高性能计算中心(合肥)、信息与计算机国家级实验教学示范中心

计算机科学与技术学院

2025年11月12日

主要内容

- 理解循环的重要意义
- 循环的一些基本概念
  - 支配结点
  - 深度优先排序
  - 回边
- 自然循环及其识别

流图中的循环

- 标识循环并对循环专门处理的重要性
  - 程序执行的大部分时间消耗在循环上
  - 循环会影响程序分析的运行时间
  - 改进循环性能的优化会对程序执行产生显著影响

举例——循环展开

loop unrolling(循环展开)

- 通过将循环体内的代码复制多次，增加loop每一次迭代的步长
- 减少循环分支指令执行的次数
- 增大处理器指令调度的空间（指令并行、流水线）
- 增加了寄存器的重用

```
for (i = 0; i < N; i++) {
    for (j = 0; j < N; i++) {
        A[i][j] = A[i][j] + B[i][j] * C[i][j];
    }
}
```

对j层进行循环展开

```
for (i = 0; i < N; i++) {
    for (j = 0; j < N; i+=4) {
        A[i][j] = A[i][j] + B[i][j] * C[i][j];
        A[i][j+1] = A[i][j+1] + B[i][j+1] * C[i][j+1];
        A[i][j+2] = A[i][j+2] + B[i][j+2] * C[i][j+2];
        A[i][j+3] = A[i][j+3] + B[i][j+3] * C[i][j+3];
    }
}
```

举例——循环展开

loop unrolling(循环展开)

- 通过将循环体内的代码复制多次，增加loop每一次迭代的步长
- 减少循环分支指令执行的次数
- 增大处理器指令调度的空间（指令并行、流水线）
- 增加了寄存器的重用

```
for (i = 0; i < N; i++) {
    for (j = 0; j < N; i++) {
        A[i][j] = A[i][j] + B[i][j] * C[i][j];
    }
}
```

对j层进行循环展开

```
for (i = 0; i < N; i++) {
    for (j = 0; j < N; i+=4) {
        A[i][j] = A[i][j] + B[i][j] * C[i][j];
        A[i][j+1] = A[i][j+1] + B[i][j+1] * C[i][j+1];
        A[i][j+2] = A[i][j+2] + B[i][j+2] * C[i][j+2];
        A[i][j+3] = A[i][j+3] + B[i][j+3] * C[i][j+3];
    }
}
```

在代码尺寸和性能之间做了权衡

举例——循环展开

- clang 含有循环计算的源代码.c –O1 –funroll-loops –emit-llvm –S –Rpass=loop-unroll

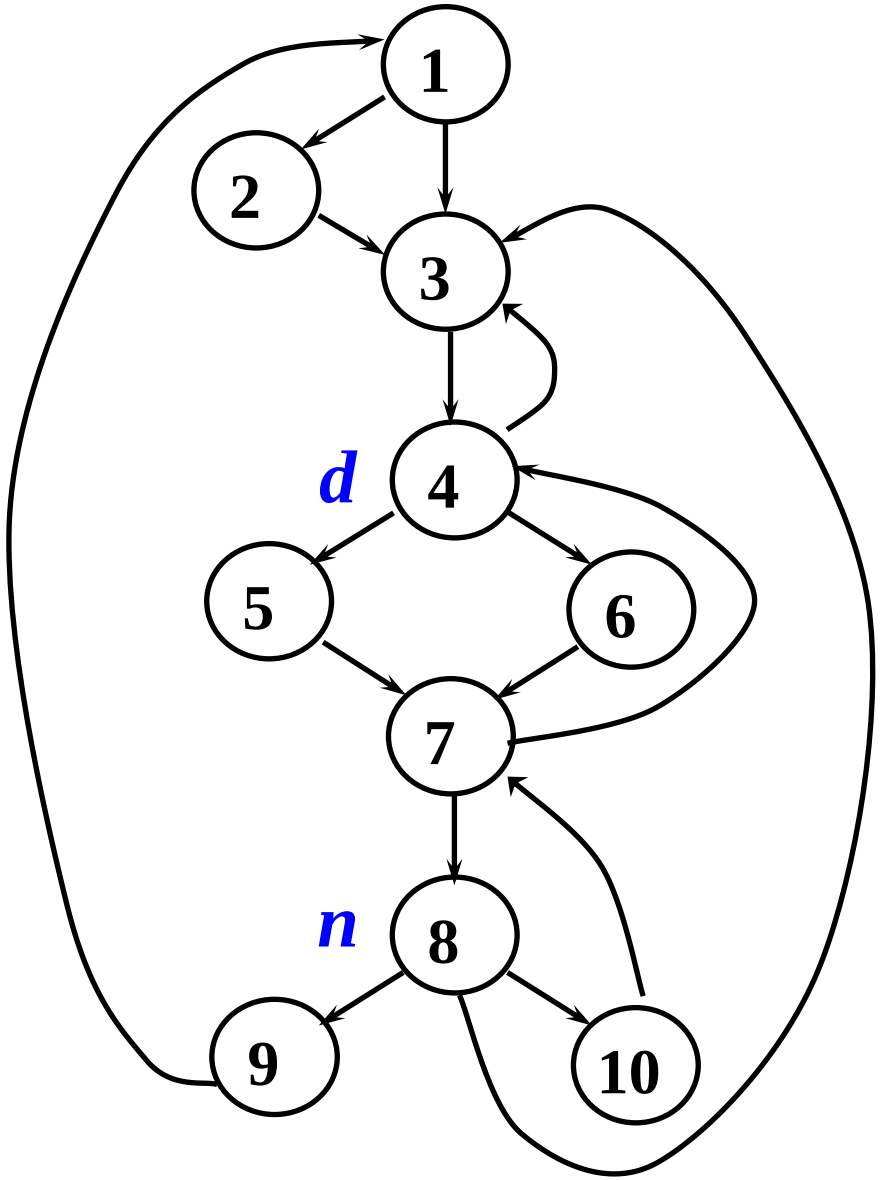
选项	功能
-funroll-loops	打开循环展开
-fno-unroll-loops	关闭循环展开
-mllvm –unroll-max-count	为部分和运行时展开设置最大展开次数
-mllvm –unroll-count	确定展开次数
-mllvm –unroll-threshold	设置循环展开的成本限制
-mllvm –unroll-remainder	允许循环展开后有尾循环
-Rpass=loop-unroll	显示循环展开的优化信息
-Rpass-missed=loop-unroll	显示循环展开失败的信息

主要内容

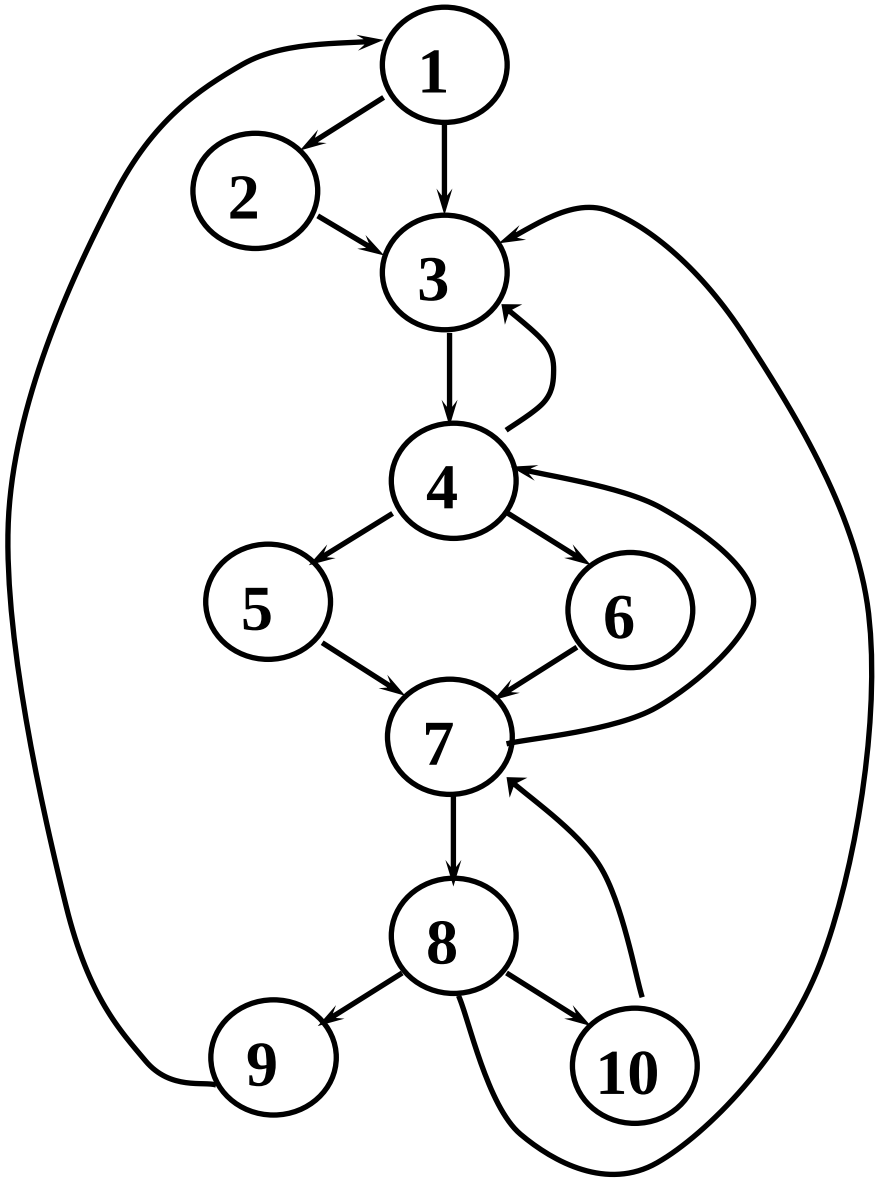
- 理解循环的重要意义
- 循环的一些基本概念
  - 支配结点
  - 深度优先排序
  - 回边
- 自然循环及其识别

支配结点(Dominators)

- d是n的支配结点:
  - 若从初始结点起，每条到达*n*的路径都要经过*d*，写成*d dom n*
- 结点是它本身的支配结点
- 循环的入口是循环中所有结点的支配结点

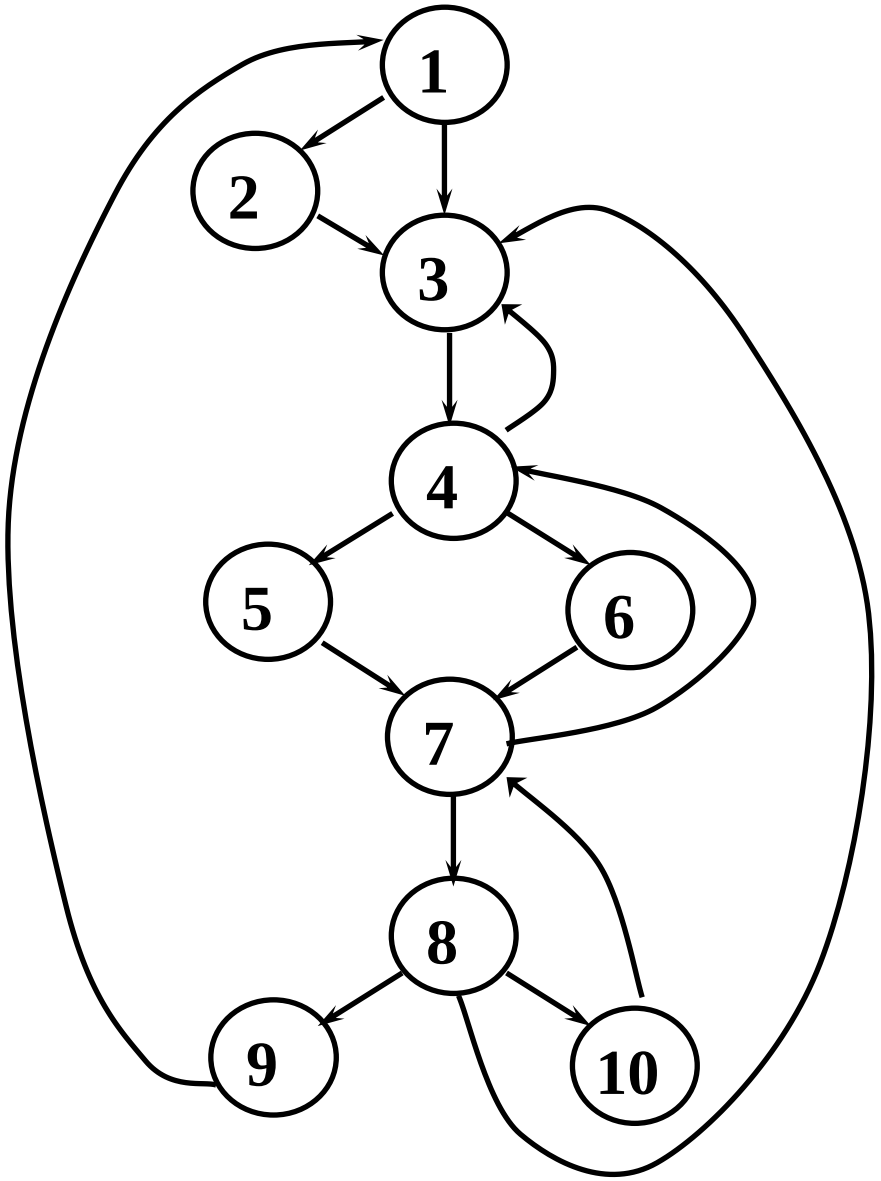


支配结点(Dominators)-举例

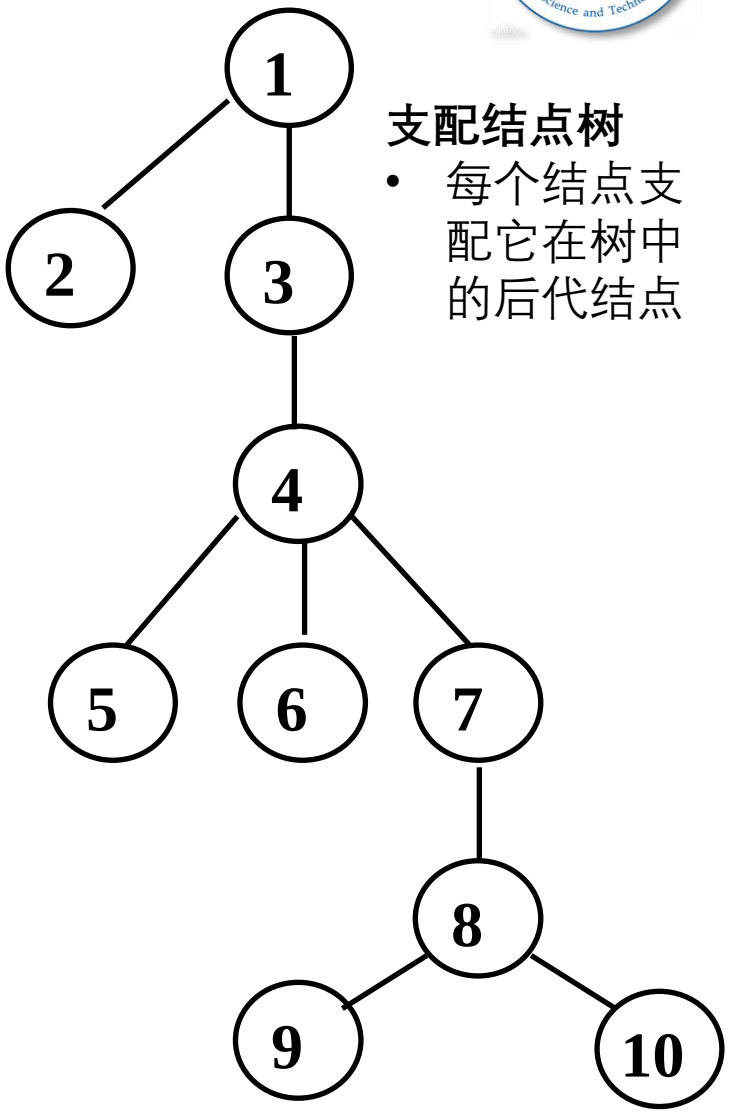


支配结点	支配对象
1	1-10
2	2
3	3-10
4	4-10
5	5
6	6
7	7-10
8	8-10
9	9
10	10

支配结点(Dominators)-举例



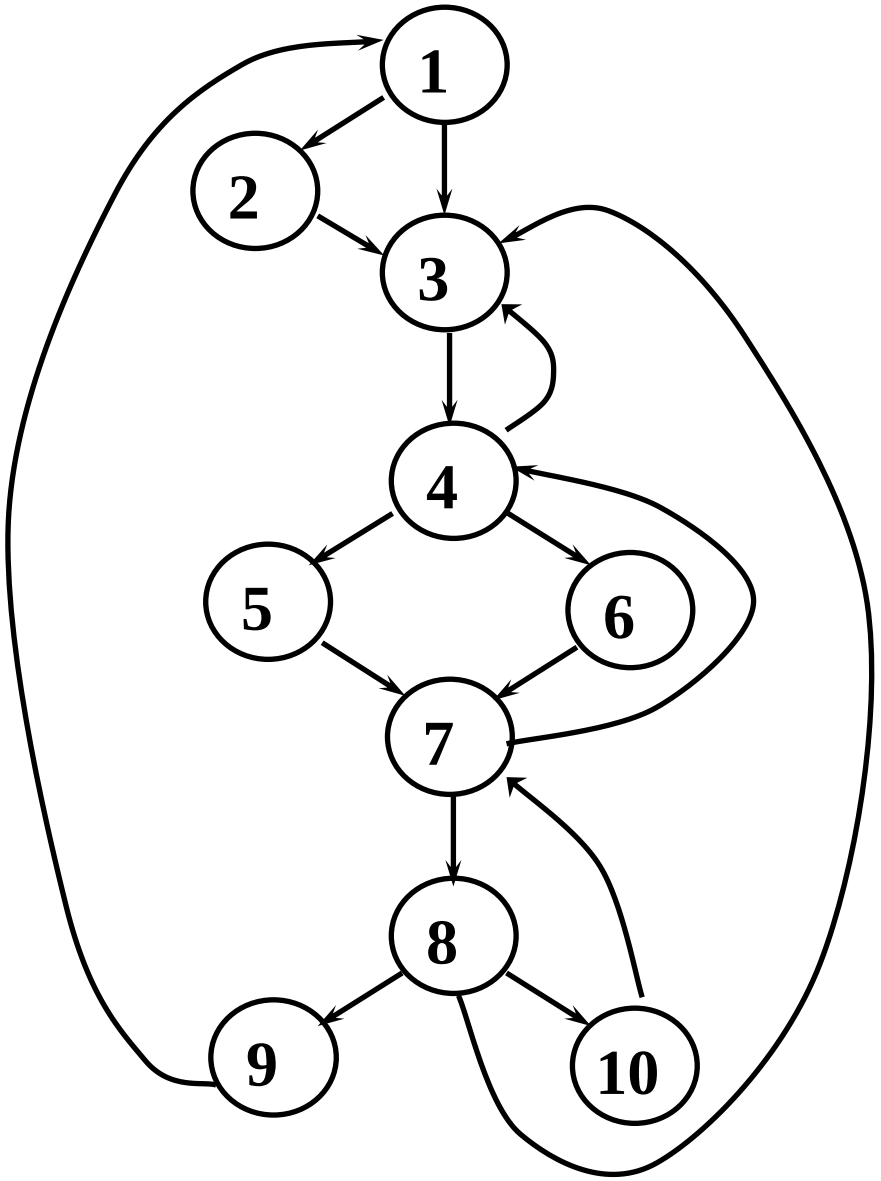
支配结点	支配对象
1	1-10
2	2
3	3-10
4	4-10
5	5
6	6
7	7-10
8	8-10
9	9
10	10



支配结点树

- 每个结点支配它在树中的后代结点

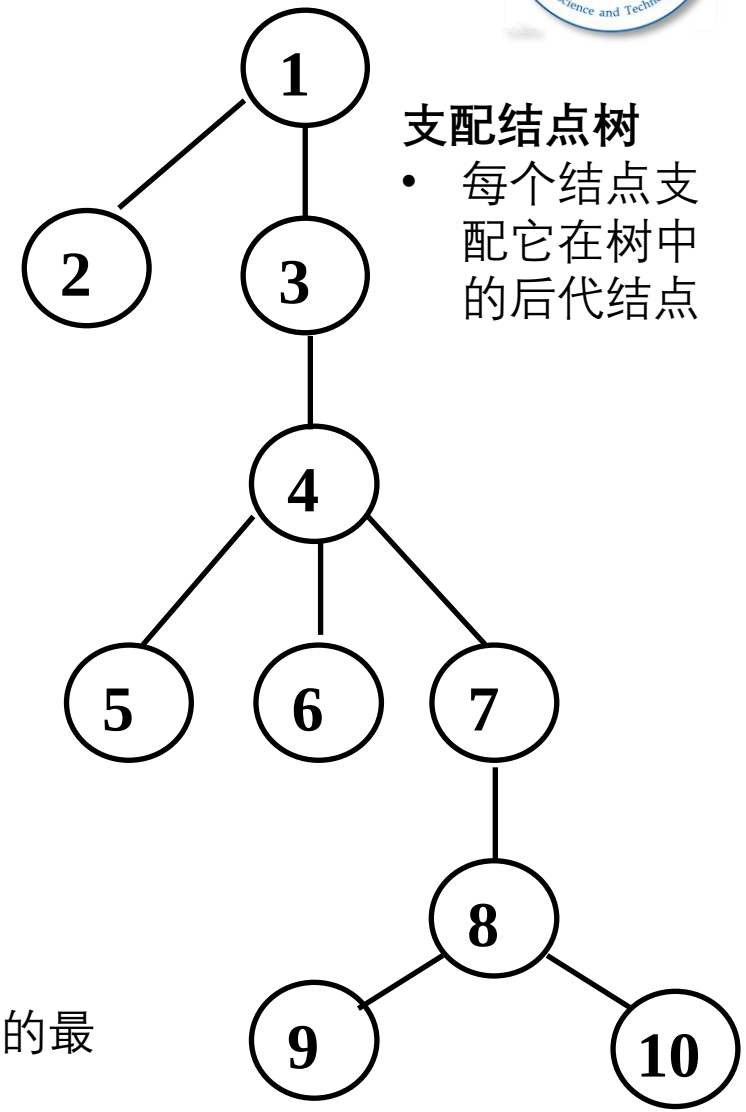
支配结点(Dominators)-举例



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

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



支配结点树

- 每个结点支配它在树中的后代结点

支配结点计算的数据流分析

- 正向数据流分析
- 支配结点计算的数据流方程
  - 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
- 边界条件：OUT[Entry] = {Entry}
- 初始化条件：OUT[B] = N
  - 除了Entry以外的B，N为支配结点的全集
- 约束方程
  - $OUT[B] = IN[B] \cup \{B\}$ (B not Entry)
  - $IN[B] = \cap_{P\text{是}B\text{的前驱}} OUT[P]$ (B not Entry)

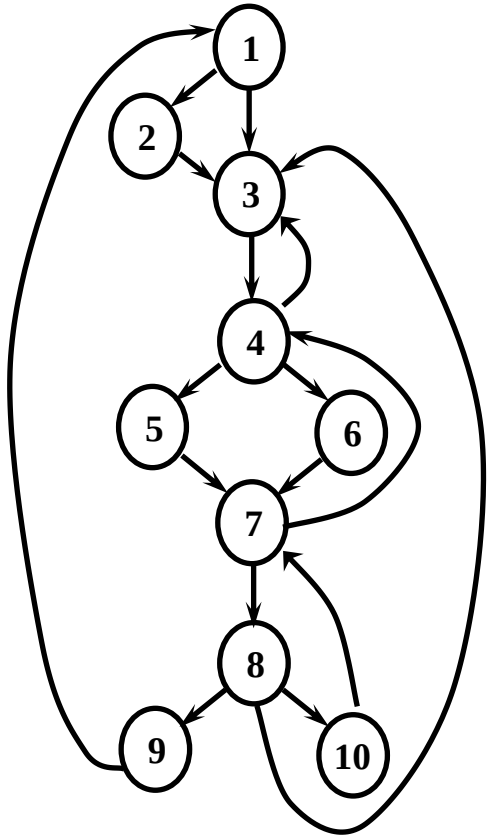
支配结点计算的数据流分析

- 目标：寻找支配结点
- 输入：流图G，G的结点集N
- 输出：每个N中的结点n，支配n的所有结点的集合D(n)
- 边界条件：OUT[Entry] = {Entry}, 初始化：OUT[B] = N
- 正向分析

```
while(某个OUT值变化){
        for(除Entry外的块B){
                IN[B] = ∩ P是B的前驱 (OUT[P]) //交集
                OUT[B]  =  {B} ∪ IN[B]
        }
}
```

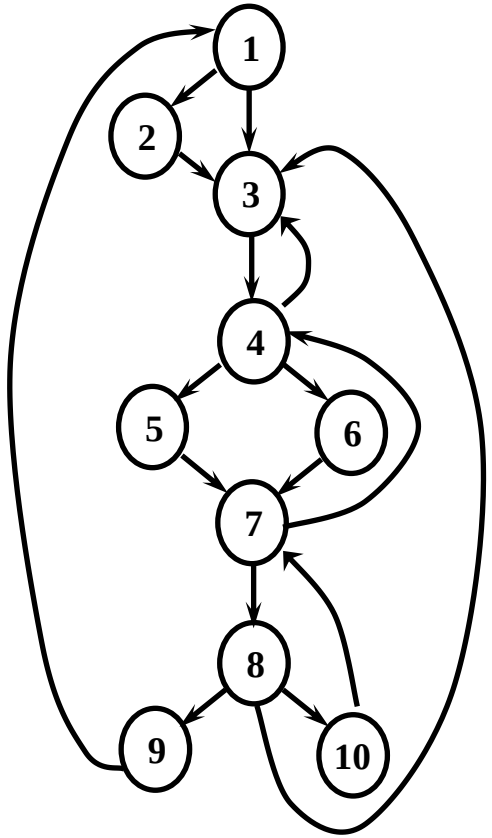
支配结点计算的数据流分析

```
while(某个OUT值变化){
    for(除Entry外的块B){
        IN[B] = ∩ P是B的前驱 (OUT[P]) //交集
        OUT[B]  =  {B} ∪ IN[B]
    }
}
```



支配结点计算的数据流分析

```
while(某个OUT值变化){
    for(除Entry外的块B){
        IN[B] = ∩ P是B的前驱 (OUT[P]) //交集
        OUT[B]  =  {B} ∪ IN[B]
    }
}
```



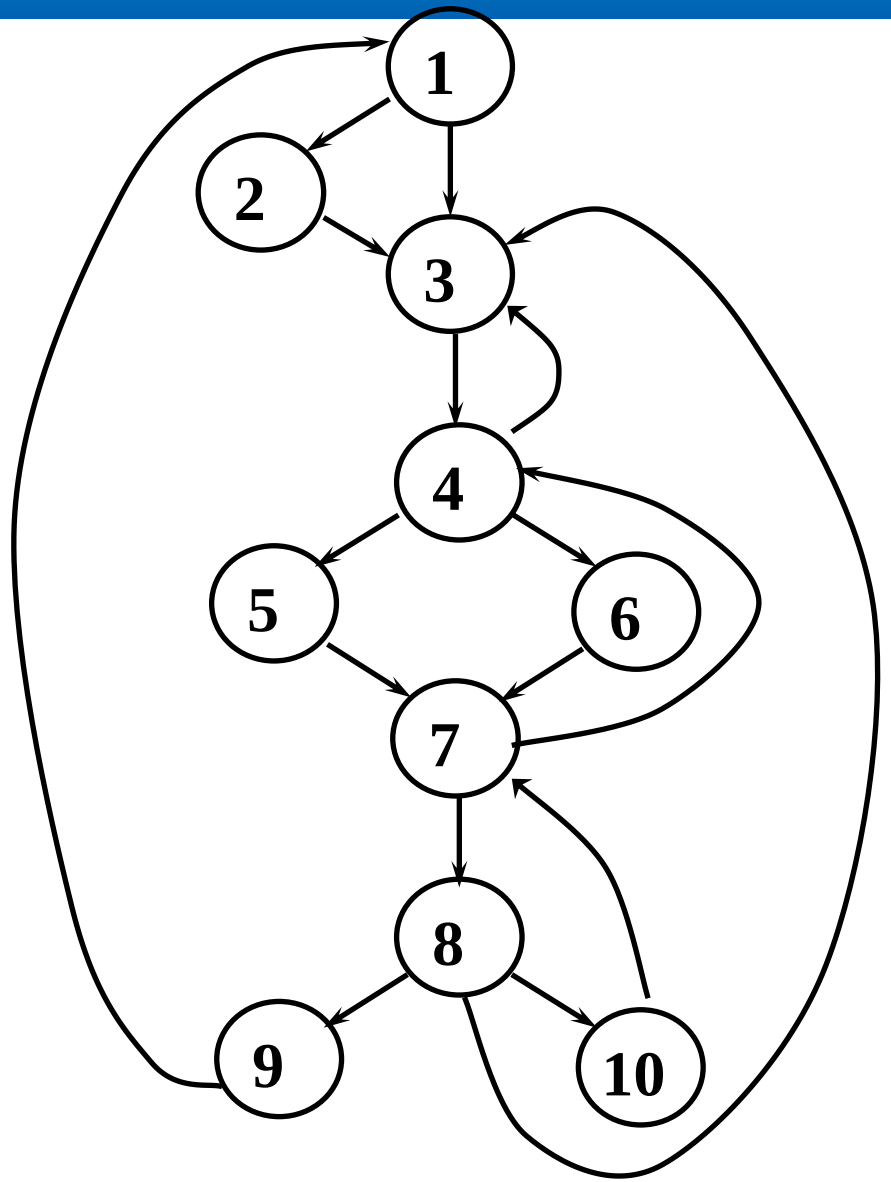
	$OUT^0[B]$	$IN^1[B]$	$OUT^1[B]$
Entry	{Entry}		
1	N	{Entry}	{Entry, 1}
2	N	{Entry, 1}	{Entry, 1, 2}
3	N	{Entry, 1}	{Entry, 1, 3}
4	N	{Entry, 1, 3}	{Entry, 1, 3, 4}
5	N	{Entry, 1, 3, 4}	{Entry, 1, 3, 4, 5}
6	N	{Entry, 1, 3, 4}	{Entry, 1, 3, 4, 6}
7	N	{Entry, 1, 3, 4}	{Entry, 1, 3, 4, 7}
8	N	{Entry, 1,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

主要内容

- 理解循环的重要意义
- 循环的一些基本概念
  - 支配结点
  - 深度优先排序
  - 回边
- 自然循环及其识别

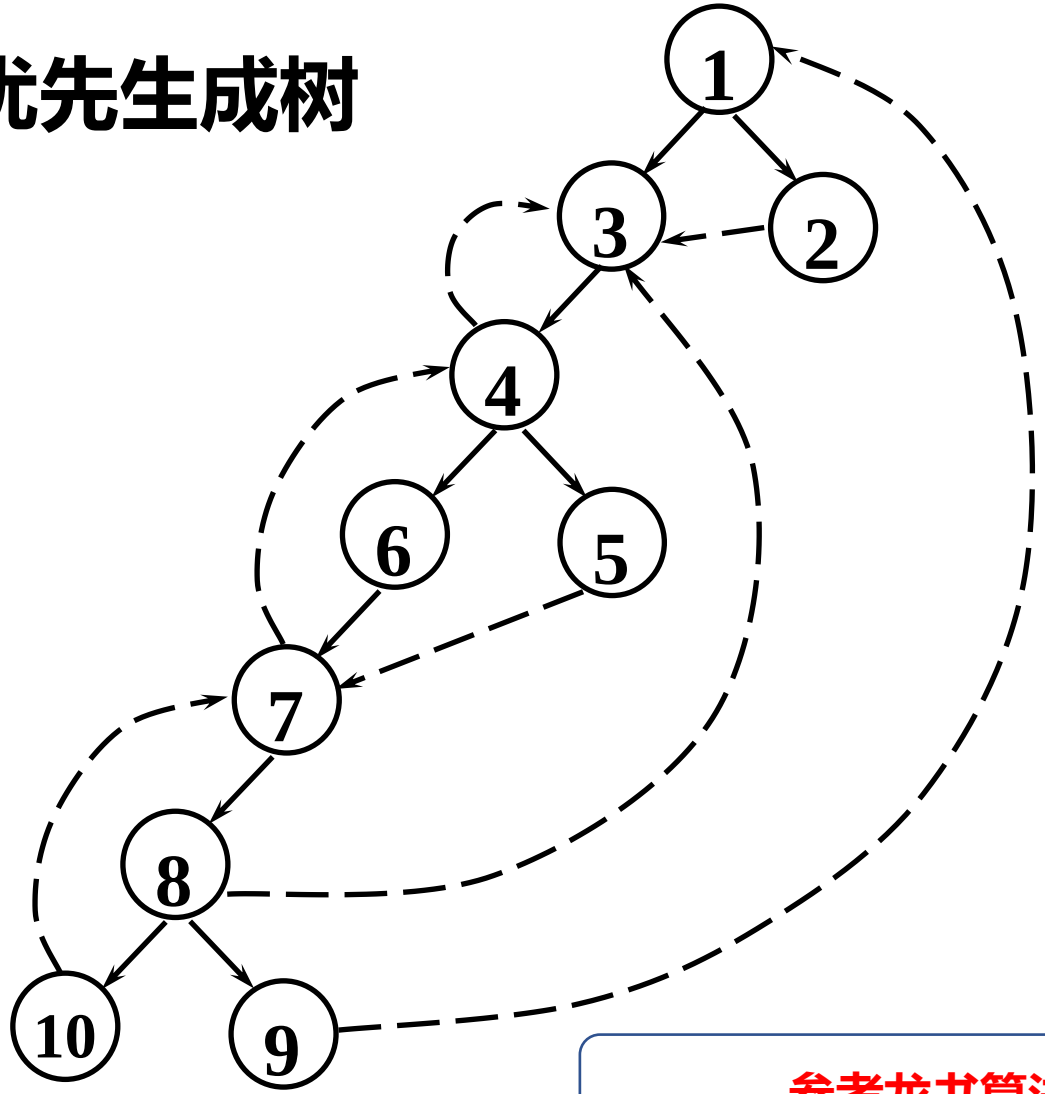
深度优先排序

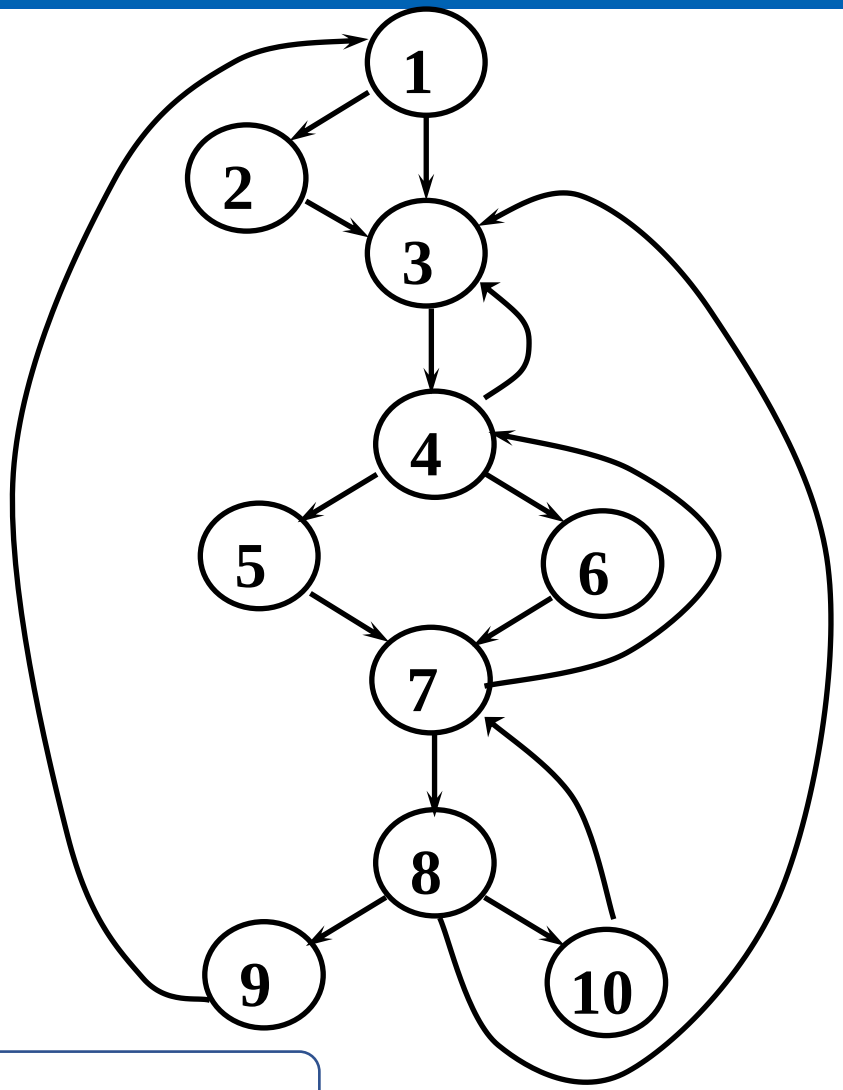
- **深度优先搜索**
  - 先序遍历（先左后右）
    - 1,3,4,6,7,8,10,9,5,2
  - 后序遍历
    - 10,9,8,7,6,5,4,3,2,1
- **深度优先排序正好与后序遍历相反**
  - 1,2,3,4,5,6,7,8,9,10
  - 先右后左



深度优先表示

- 深度优先生成树





参考龙书算法9.41

深度优先生成树中的边

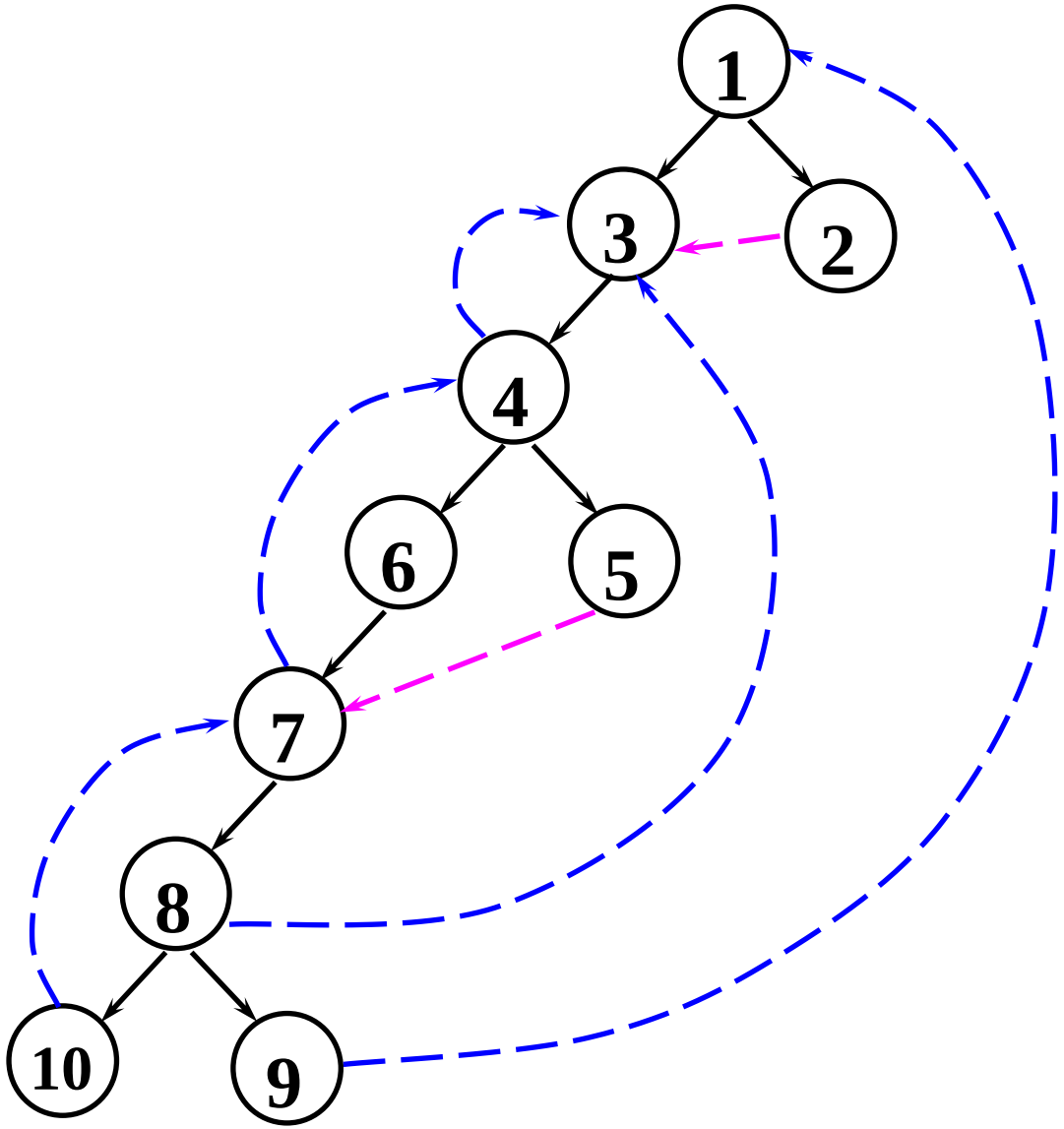
- **前进边**
  - 深度优先生成树的边
- **后撤边**
  - 指向祖先节点

    4 → 3、7 → 4、10 → 7、

    8 → 3和9 → 1

- **交叉边**
  - 在树中互不为祖先

    2 → 3和5 → 7

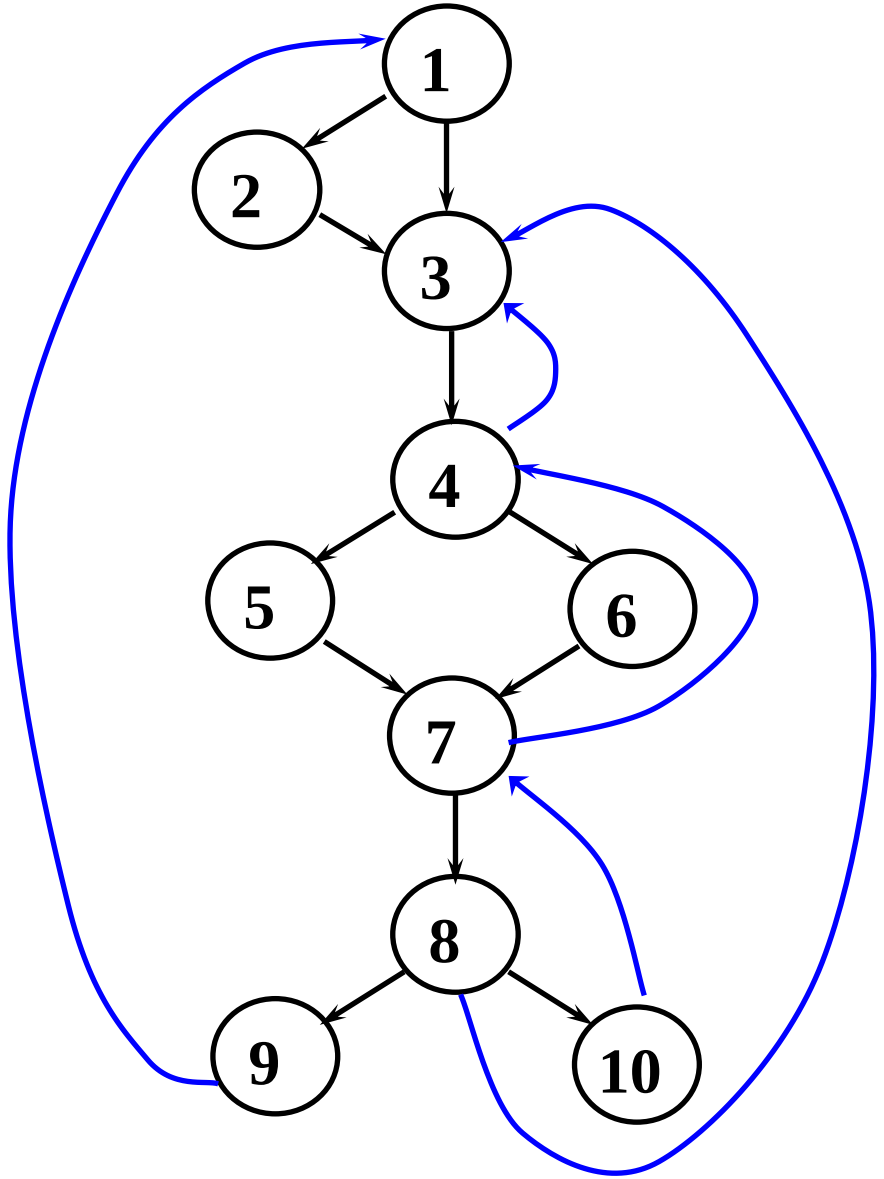


主要内容

- 理解循环的重要意义
- 循环的一些基本概念
  - 支配结点
  - 深度优先排序
  - 回边
- 自然循环及其识别

回边和可归约性

- **回边**
  - 如果有$a\ dom\ b$，那么边$b \rightarrow a$叫做回边



回边和可归约性

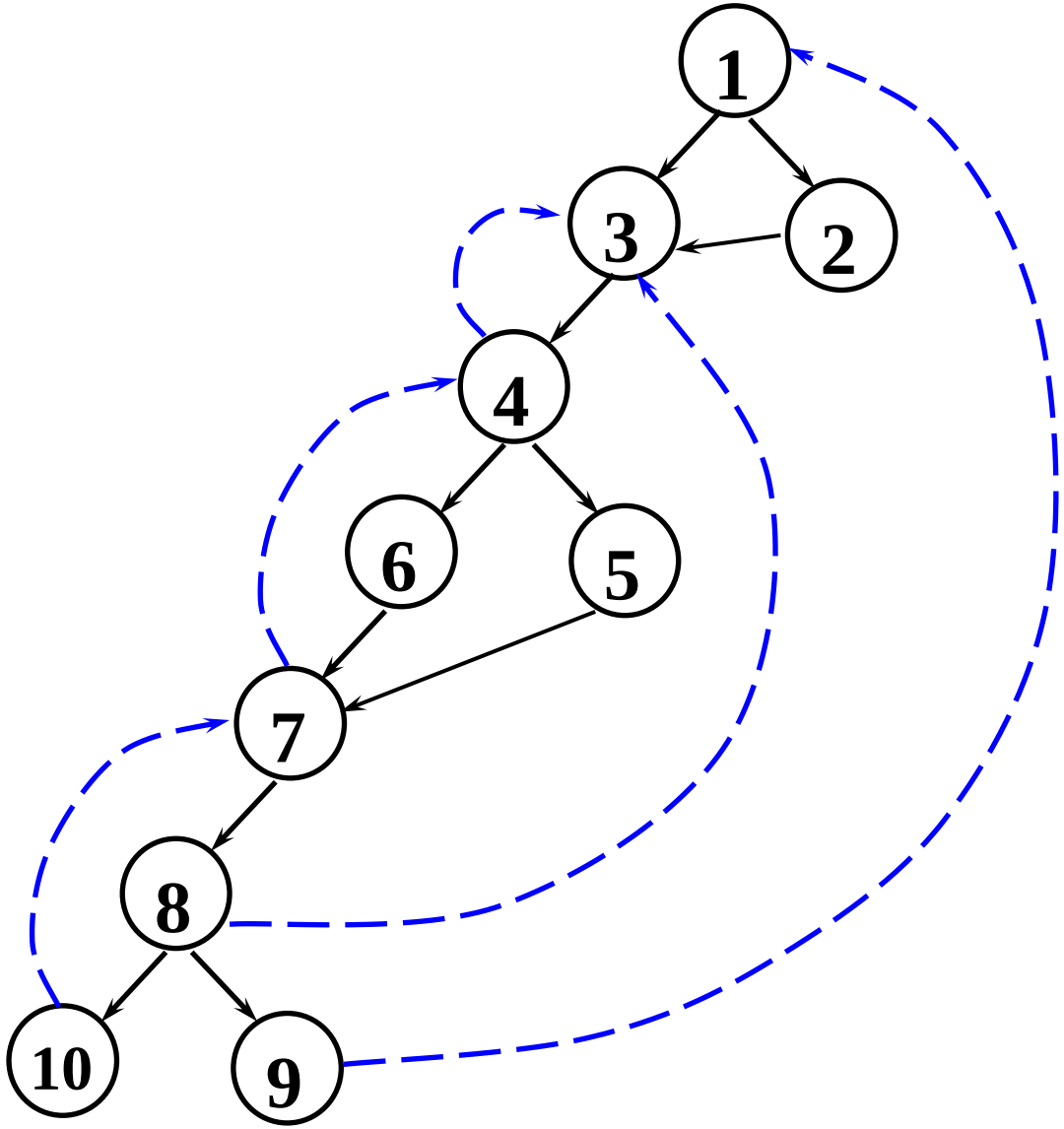
- **回边**
  - 如果有$a\ dom\ b$，那么边$b \rightarrow a$叫做回边
  - 如果流图可归约，则后撤边正好就是回边

后撤边集合

4 → 3、7 → 4、10 → 7、8 → 3和9 → 1

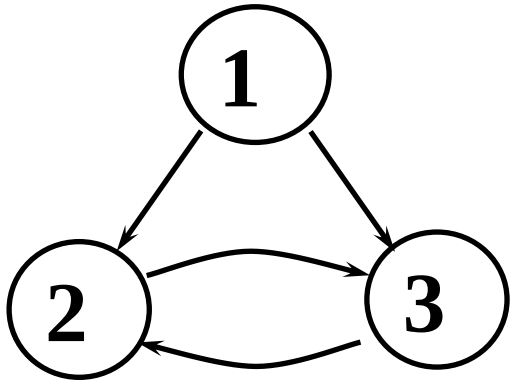
回边集合

4 → 3、7 → 4、10 → 7、8 → 3和9 → 1



回边和可归约性

- **可归约流图**
  - 如果把一个流图中所有回边删掉后，剩余的图无环
- **例：不可归约流图**
  - 开始结点是1
  - 2 →3和3 →2都不是回边
  - 该图不是无环的
  - 从结点2和3两处都能进入由它们构成的环



主要内容

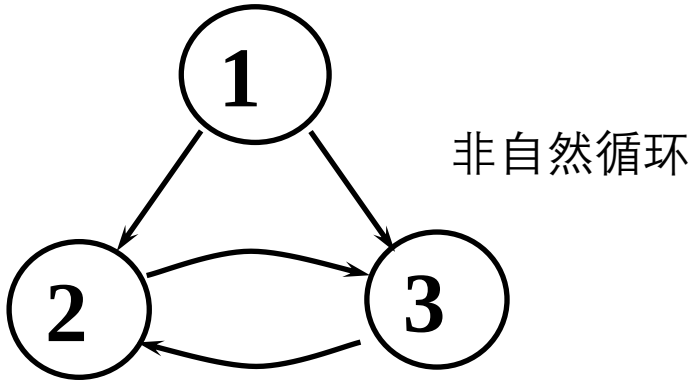
- 理解循环的重要意义
- 循环的一些基本概念
  - 支配结点
  - 深度优先排序
  - 回边
- **自然循环及其识别**

自然循环

- **在源程序中，循环可以用多种形式描述**
  - for, while, goto等
- **但从程序分析的角度来看，循环的代码形式并不重要，重要的是它是否具有一些易于优化的性质。**
  - 如果循环的入口结点唯一，那么我们就可以假设某些初始条件在循环的每一次迭代开头成立。

自然循环

- **自然循环的性质**
  - 有唯一的入口结点，叫做首结点，首结点支配该循环中所有结点
  - 至少存在一条回边进入该循环首结点
- **自然循环是一种适合于优化的循环。**

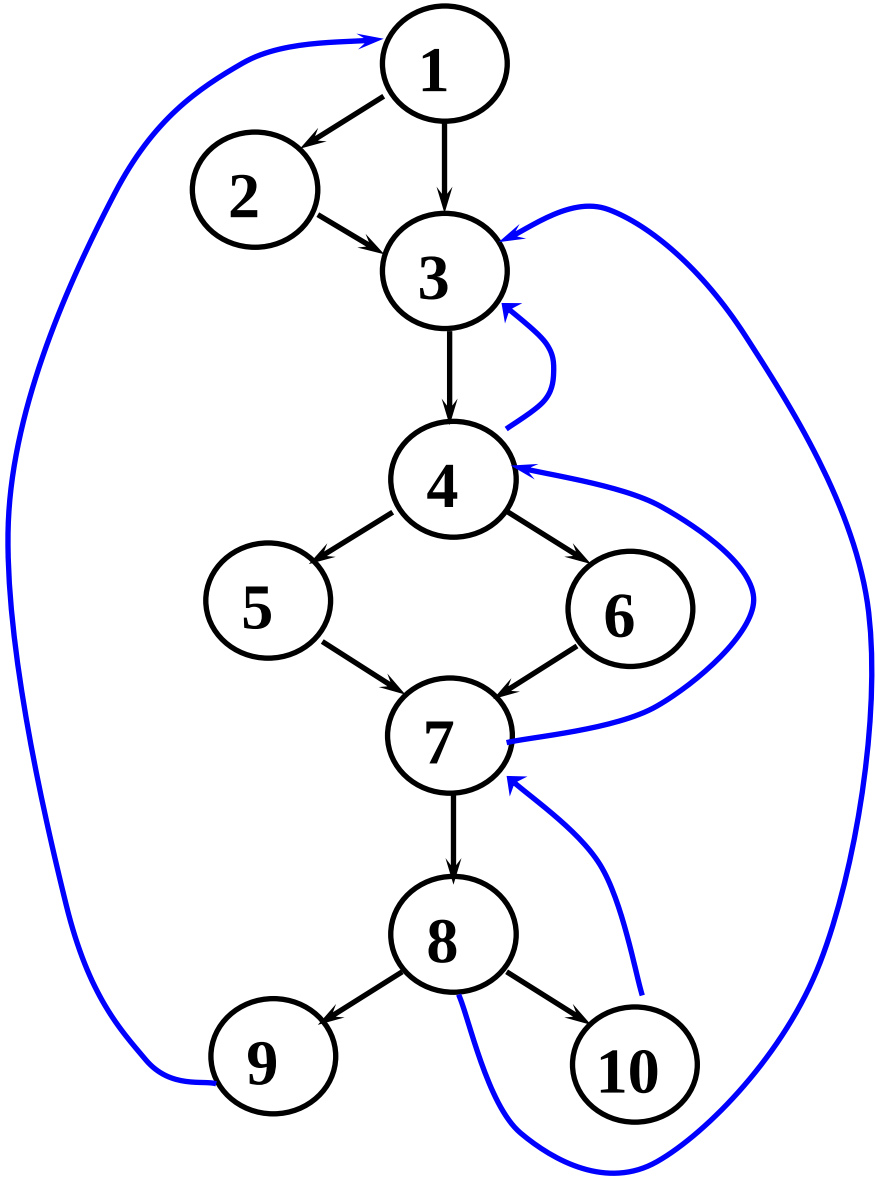


非自然循环

回边与自然循环识别

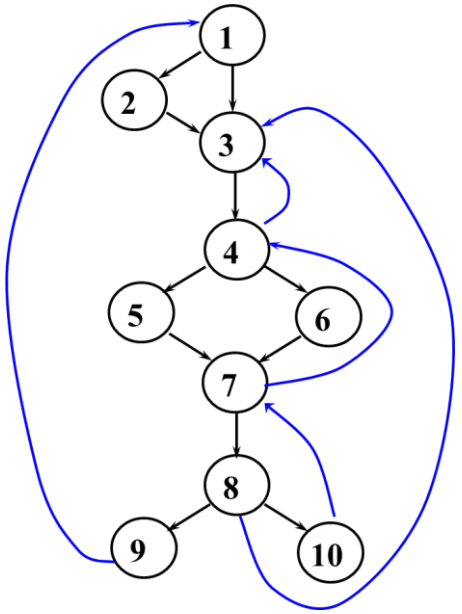
- **回边**
  - 如果有$a\ dom\ b$，那么

    边$b \rightarrow a$叫做回边
- **回边$n \rightarrow d$确定的自然循环**
  - d加上不经过d能到达n的所有结点
  - 结点d是该循环的首结点



自然循环的识别

- 给定一条回边$n \to d$，它所对应自然循环包含d加上不经过d能到达n的所有结点，且结点d是该循环的首结点



回边	自然循环
$4 \to 3$	{3,4,5,6,7,8,10}
$7 \to 4$	{4,5,6,7,8,10}
$8 \to 3$	{3,4,5,6,7,8,10}
$9 \to 1$	{1-10}
$10 \to 7$	{7,8,10}

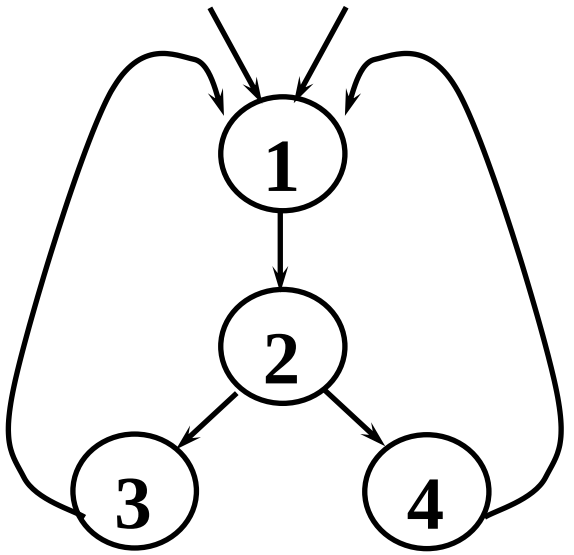
自然循环识别的算法

- **输入：流图G和回边$n \rightarrow d$**
- **输出：回边对应的自然循环的所有结点集合**
- **算法：**

```
stack = {}; loop = {n, d}; push{stack, n};
while(stack != {}){
    m = top{stack}; pop{stack};
    for(p: p是m的前驱){
        if(p not in loop){
            loop = loop U {p};
            push(stack, p);
        }
    }
}
```

流图中的循环

- **内循环**
  - 若一个循环的结点集合是另一个循环的结点集合的子集
  - 两个循环有相同的首结点，但并非一个结点集是另一个的子集，则看成一个循环



一起努力
打造国产基础系统软件体系！

李 诚

国家高性能计算中心(合肥)、信息与计算机国家级实验教学示范中心

计算机科学与技术学院

2025年11月12日