中国科学技术大学
University of Science and Technology of China

支配树算法

直接支配节点

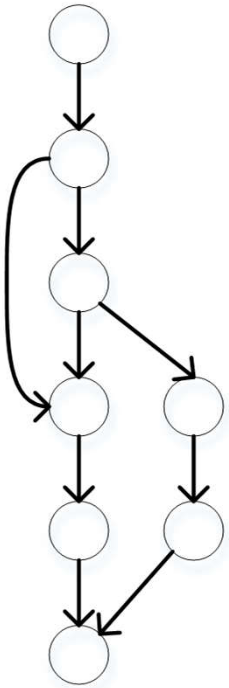
- 直接支配节点(immediate dominance, idom)
  - 对于a≠b，a直接支配b: a dom b 且不存在c≠a并且c≠b的节点c，使得a dom c且c dom b。
- 计算直接支配节点算法
  - Lengauer-Tarjan Algorithm (before 2017, LLVM)
  - Semi-NCA Algorithm (2017至今, LLVM)
    - 相关源码
      - include/llvm/IR/Dominators.h
      - include/llvm/Support/GenericDomTree.h
      - include/llvm/Support/GenericDomTreeConstruction.h

[**Lengauer-Tarjan Algorithm**] T. Lengauer and R. E. Tarjan. A fast algorithm for finding dominators in a flowgraph. ACM Transactions on Programming Languages and Systems, 1(1):121–41, 1979.

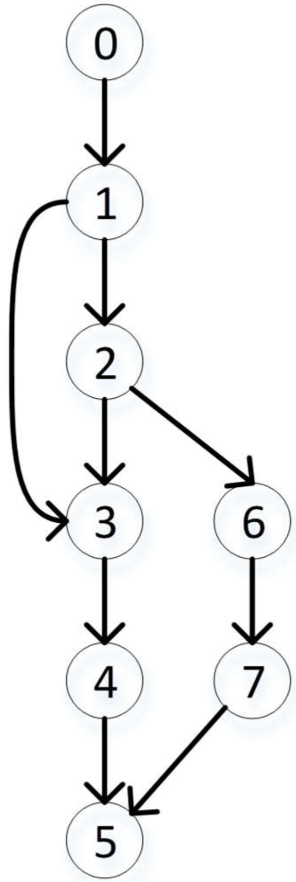
[**Semi-NCA Algorithm**] Loukas Georgiadis, Renato F. Werneck, Robert E. Tarjan, Spyridon Triantafyllis, David I. August. Finding dominators in practice. European Symposium on Algorithms 2004: 677–688, 2004.

半必经节点

- 深度优先先序遍历 ⇒ DFS Tree D
  - 按照遍历顺序，对节点进行编号
  - v < w: 遍历v在遍历w之前



⇒

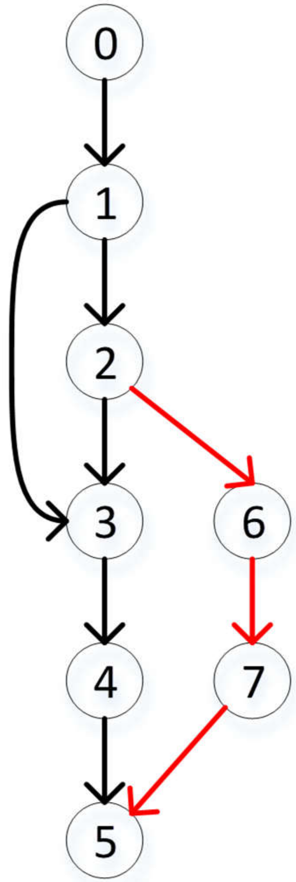


半必经节点

半必经路径 SDOM-PATH

$$P = (v_0 = v, v_1, v_2, \dots, v_k = w)$$

且对于$1 \le i \le k-1$，$v_i > w$，$v_0 < w$



如图所示对于节点5的半必经路径有：

2->6->7->5[图中红色标注]

4->5

注：$(v_0 = v, v_1, v_2, \dots, v_{k-1})$为DFS上的路径
v_{k-1}为v_k在图上的前驱

半必经节点

- 半必经节点semidominators

$$sdom(w) = min\{v|\exists SDOM - path\ from\ v\ to\ w\}$$

取w半必经路径中首点的最小值

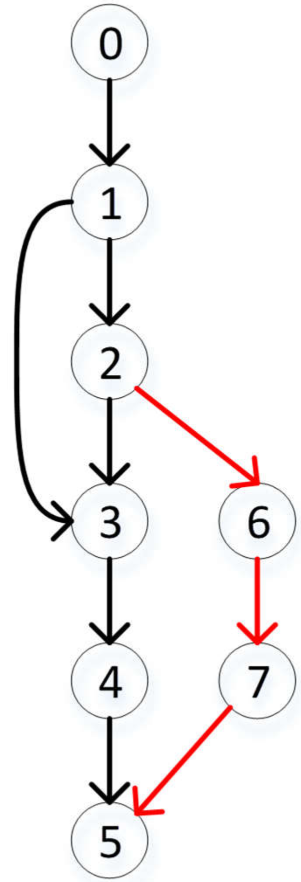
利用$sdom(w)$逼近$idom(w)$

如图所示对于节点5的半必经路径有：

2->6->7->5[图中红色标注]

4->5

$$sdom(5) = min\{2, 4\} = 2$$



半必经节点

单个半必经节点查找算法

- 对节点v进行以下操作

  1、找到所有前驱节点中序号最小的节点

  2、对序号大于v的前驱节点序号q

  若$\mathrm{sdom}(q) > v$，设$q=\mathrm{sdom}(q)$，递归第2步查找

  若$\mathrm{sdom}(q) < v$，返回q。

  3、求步骤1和2的最小值。

注意： 前驱节点对应于控制流图

SDOM-PATH 对应于 DFS Tree

Sdom/semi伪代码

中国科学技术大学
University of Science and Technology of China

Step 1: Initialisation

Number the vertices is depth first search order from 1 to n.

For each vertex v from 1 to n set:

parent[v]	:=	DFS tree parent of v
succs[v]	:=	list of successors
preds[v]	:=	list of predecessors
semi[v]	:=	v
idom[v]	:=	0
ancestor[v]	:=	0
best[v]	:=	v
bucket[v]	:=	0

Steps 2 and 3

```
FOR w = n TO 2 BY -1 DO
{       LET p = parent[w]

step2: FOR each v in preds[w] DO
       { LET u = EVAL(v)
         IF semi[w] >  semi[u] DO
            semi[w] := semi[u]
       }
       LINK(p, w)
 }
```

http://www.cse.unt.edu/~sweany/CSCE5934/HANDOUTS/LengaurTarjanOverheads.pdf

Sdom/semi伪代码


```
LET EVAL(v) = VALOF
{ LET a = ancestor[v]

  WHILE ancestor[a] DO
  { IF semi[v] > semi[a] DO v := a
    a := ancestor[a]
  }

  // v is now a vertex
  //   with smallest semidominator
  //   of any in the ancestor chain.

  RESULTIS v
}
LET LINK(v, w) BE ancestor[w] := v
```

Steps 2 and 3

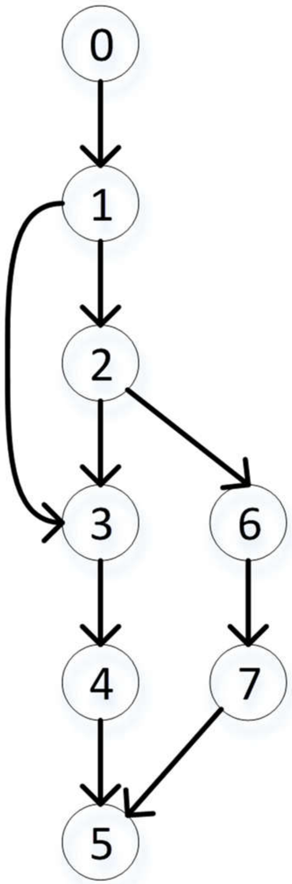
```
FOR w = n TO 2 BY -1 DO
{       LET p = parent[w]

step2: FOR each v in preds[w] DO
       { LET u = EVAL(v)
         IF semi[w] >  semi[u] DO
            semi[w] := semi[u]
       }
       LINK(p, w)
 }
```

http://www.cse.unt.edu/~sweany/CSCE5934/HANDOUTS/LengaurTarjanOverheads.pdf

示例

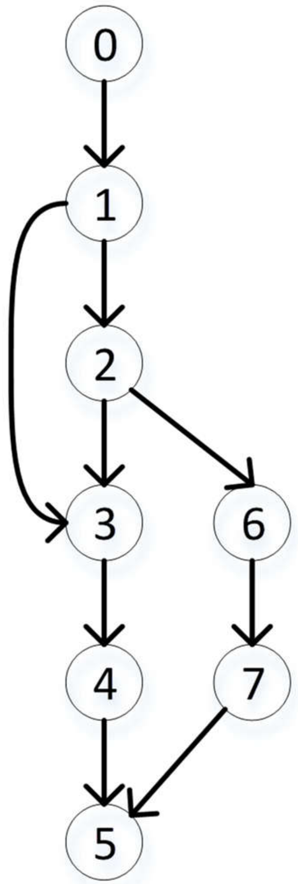
1、初始化、DFS树建立



vertex	parent	ancestor	Sdom
0	-	-	0
1	0	∅	1
2	1	∅	2
3	2	∅	3
4	3	∅	4
5	4	∅	5
6	2	∅	6
7	6	∅	7

示例

2、逆序处理节点，节点7



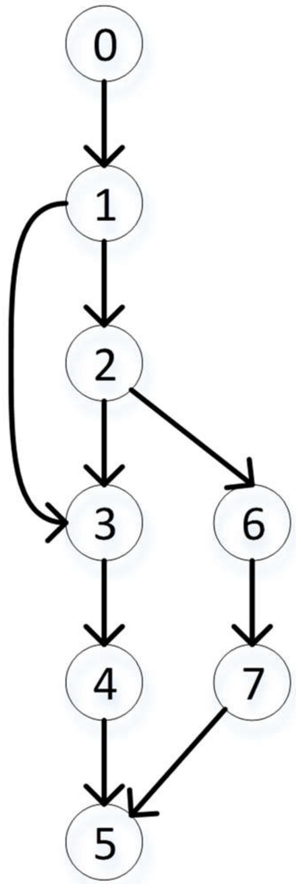

```
step2: FOR each v in preds[w] DO
       { LET u = EVAL(v)
         IF semi[w] >  semi[u] DO
            semi[w] := semi[u]
       }
```

w=7; v = pred[7]=6

u = eval(6)

vertex	parent	ancestor	Sdom
0	-	-	0
1	0	∅	1
2	1	∅	2
3	2	∅	3
4	3	∅	4
5	4	∅	5
6	2	∅	6
7	6	∅	7

2、逆序处理节点，节点7



```
LET EVAL(v) = VALOF
{ LET a = ancestor[v]

  WHILE ancestor[a] DO
  { IF semi[v] > semi[a] DO v := a
    a := ancestor[a]
  }
```

a= ancestor[6]=∅

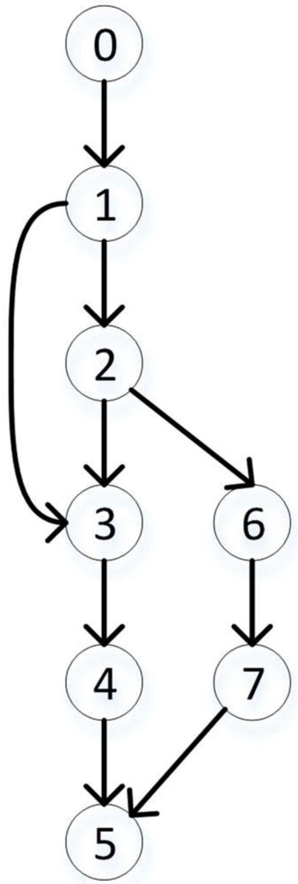
退出while

返回v = 6

vertex	parent	ancestor	Sdom
0	-	-	0
1	0	∅	1
2	1	∅	2
3	2	∅	3
4	3	∅	4
5	4	∅	5
6	2	∅	6
7	6	∅	7

示例

2、逆序处理节点，节点7




```
step2: FOR each v in preds[w] DO
       { LET u = EVAL(v)
         IF semi[w] >  semi[u] DO
            semi[w] := semi[u]
       }
```

w = 7; v = pred[7]=6

u = eval(6)=6

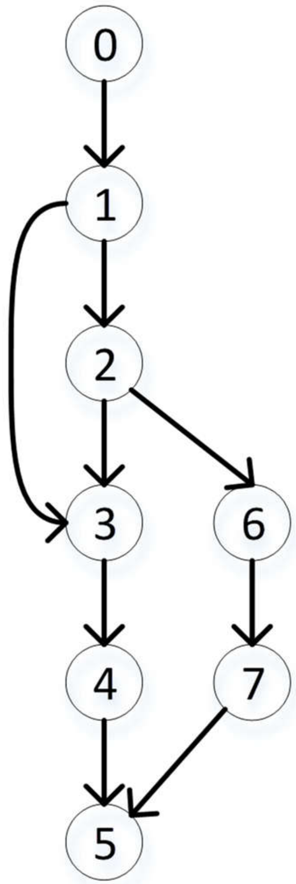
semi[7] > semi[6]

semi[7] = semi[6] = 6

vertex	parent	ancestor	Sdom
0	-	-	0
1	0	∅	1
2	1	∅	2
3	2	∅	3
4	3	∅	4
5	4	∅	5
6	2	∅	6
7	6	∅	6

示例

2、逆序处理节点，节点7



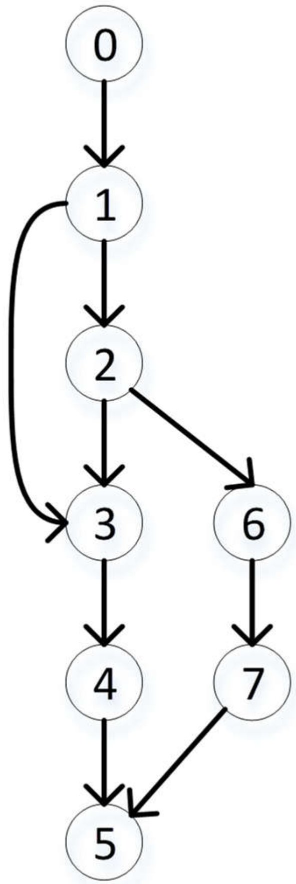
`LET LINK(v, w) BE ancestor[w] := v`

ancestor[7] = 6

vertex	parent	ancestor	Sdom
0	-	-	0
1	0	∅	1
2	1	∅	2
3	2	∅	3
4	3	∅	4
5	4	∅	5
6	2	∅	6
7	6	6	6

示例

3、逆序处理节点，节点6




```
step2: FOR each v in preds[w] DO
       { LET u = EVAL(v)
         IF semi[w] >  semi[u] DO
            semi[w] := semi[u]
       }
```

w = 6; v = pred[6]=2

u = eval(2)=2

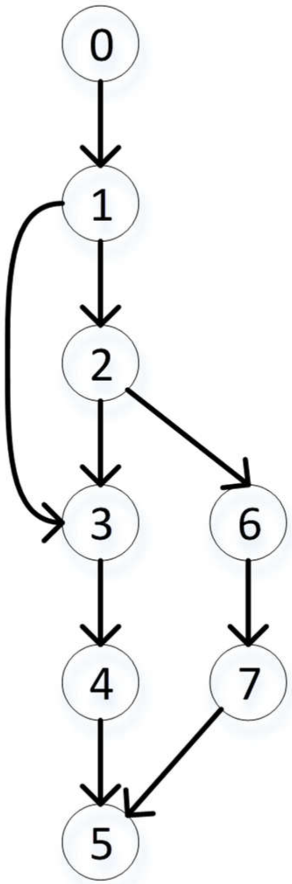
semi[6] > semi[2]

semi[6] = semi[2] = 2

vertex	parent	ancestor	Sdom
0	-	-	0
1	0	∅	1
2	1	∅	2
3	2	∅	3
4	3	∅	4
5	4	∅	5
6	2	2	2
7	6	6	6

示例

4、逆序处理节点，节点5




```
step2: FOR each v in preds[w] DO
       { LET u = EVAL(v)
           IF semi[w] >  semi[u] DO
              semi[w] := semi[u]
       }
```

w = 5; pred[5] = 4/7; v = 4

u = eval(4)=4

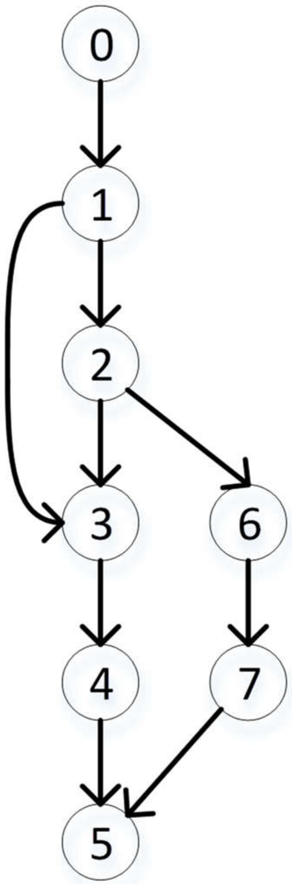
semi[5] > semi[4]

semi[5] = semi[4] = 4

vertex	parent	ancestor	Sdom
0	-	-	0
1	0	∅	1
2	1	∅	2
3	2	∅	3
4	3	∅	4
5	4	∅	4
6	2	2	2
7	6	6	6

示例

4、逆序处理节点，节点5




```
step2: FOR each v in preds[w] DO
       { LET u = EVAL(v)
         IF semi[w] >  semi[u] DO
            semi[w] := semi[u]
       }
```

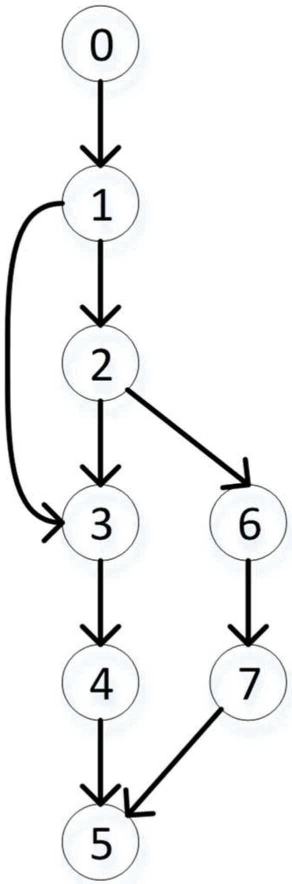
w = 5; pred[5] = 4/7; v = 7

u = eval(7)

vertex	parent	ancestor	Sdom
0	-	-	0
1	0	∅	1
2	1	∅	2
3	2	∅	3
4	3	∅	4
5	4	∅	4
6	2	2	2
7	6	6	6

示例

4、逆序处理节点，节点5




```
LET EVAL(v) = VALOF
{ LET a = ancestor[v]

  WHILE ancestor[a] DO
  { IF semi[v] > semi[a] DO v := a
    a := ancestor[a]
  }
```

v = 7; a= ancestor[7]=6

While ancestor[6] do

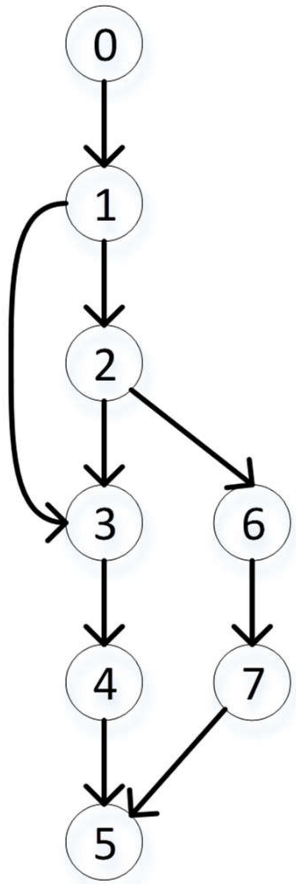
semi[7]>semi[6] v = 6

a = ancestor[6]=2

ancestor[2]= ∅，**退出返回v = 6**

vertex	parent	ancestor	Sdom
0	-	-	0
1	0	∅	1
2	1	∅	2
3	2	∅	3
4	3	∅	4
5	4	∅	4
6	2	2	2
7	6	6	6

4、逆序处理节点，节点5




```
step2: FOR each v in preds[w] DO
       { LET u = EVAL(v)
         IF semi[w] >  semi[u] DO
            semi[w] := semi[u]
       }
```

w = 5; pred[5] = 4/7; v = 7

u = eval(7) = 6

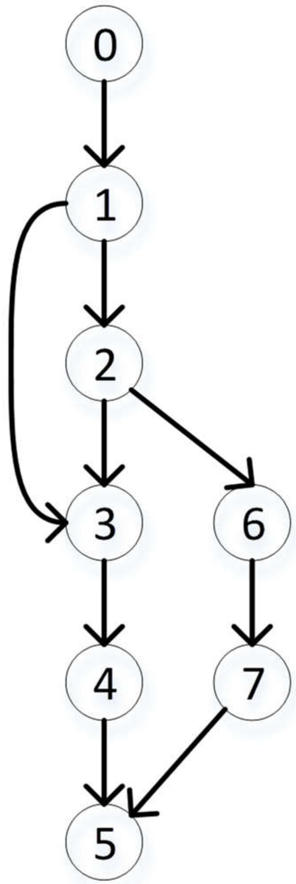
semi[5] > semi[6]

Semi[5] = semi[6] = 2

vertex	parent	ancestor	Sdom
0	-	-	0
1	0	∅	1
2	1	∅	2
3	2	∅	3
4	3	∅	4
5	4	∅	2
6	2	2	2
7	6	6	6

示例

4、逆序处理节点，节点5



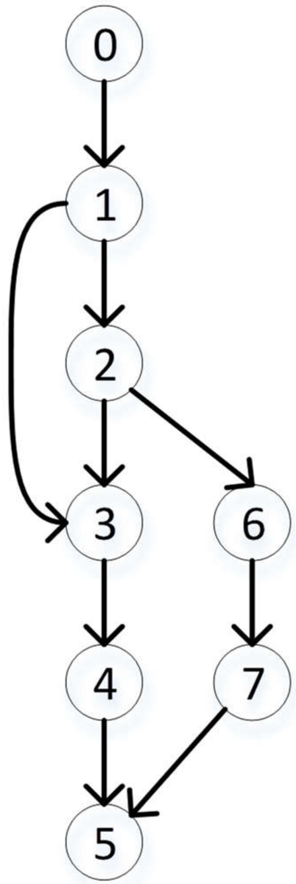

```
LET LINK(v, w) BE ancestor[w] := v
```

ancestor[5] = 4

vertex	parent	ancestor	Sdom
0	-	-	0
1	0	∅	1
2	1	∅	2
3	2	∅	3
4	3	∅	4
5	4	4	2
6	2	2	2
7	6	6	6

示例

5、类似方法处理4、3、2、1节点



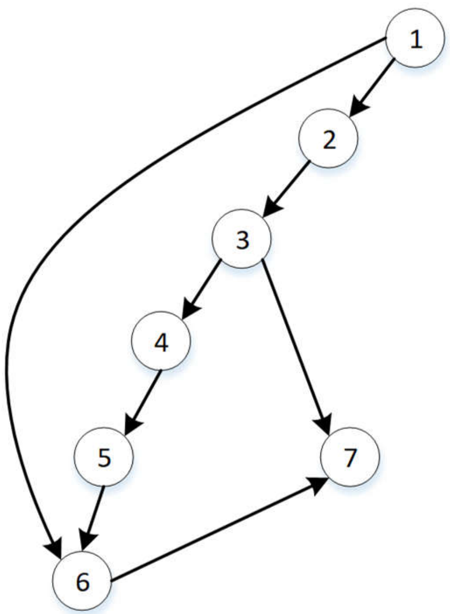
vertex	parent	ancestor	Sdom
0	-	-	0
1	0	0	0
2	1	1	1
3	2	2	1
4	3	3	3
5	4	4	2
6	2	2	2
7	6	6	6

Semi-NCA

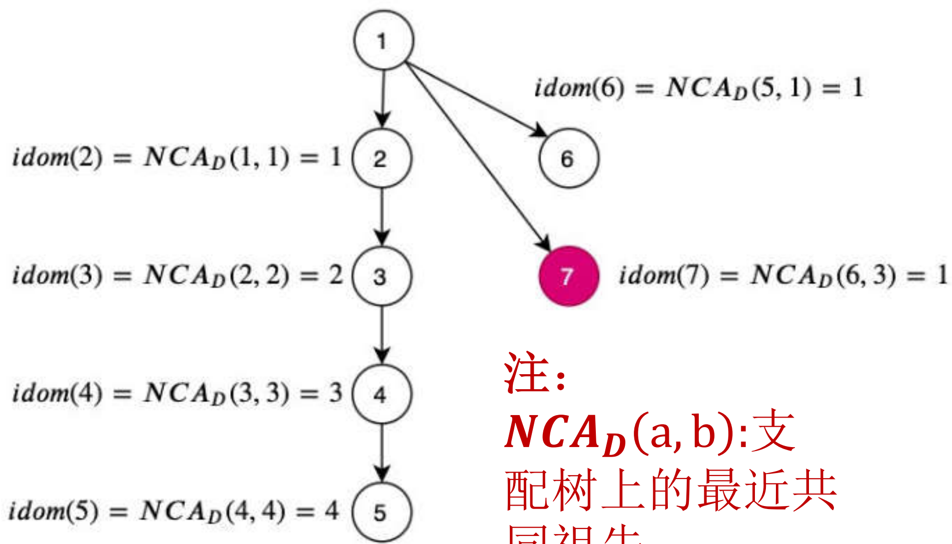
□ 算法

- 先序构建深度优先树T
- 按照深度优先树节点编号逆序遍历，计算节点的sdom
- 按照先序遍历，计算节点的idom，同时建立支配树D

$$idom(v) = NCA_D(parent_T(v), sdom(w))$$



$\Rightarrow$



注：
$\boldsymbol{NCA_D}$(a, b):支配树上的最近共同祖先

Algorithm 6 Semi-NCA

Create a DFS tree T.
Calculate semidominator for w
Create a tree D and initialize it with r as the root.
for $w \in V - \{r\}$ in preorder by the DFS **do**
    Ascend the path $r \xrightarrow{*}_D parent_T(v)$ and find the deepest vertex which number is smaller than or equal to $sdom(v)$. set this vertex as parent for v in D.
end for

Algorithms for Finding Dominators in Directed Graphs