L11：线索二叉树

吉林大学计算机学院
谷方明
fmgu2002@sina.com

学习目标

- 掌握中序线索二叉树
- 理解先序和后序线索二叉树
- 了解单边线索二叉树

二叉树的空间效率

- 二叉树的二叉链表存储结构中有很多空链接。
  - ✓ 设二叉树有n个结点，则其对应的二叉链表存储结构共有n+1个都是空链接（ $2n – (n-1)$ ）。
  - ✓ 二叉链表存储结构的固有问题。
- 如何更有效地利用空链接所占的内存空间？

- **Perlis AJ和Thornton C**设计了一种巧妙地使用空链接的方法：用线索（**Thread**）代替空链接连向树的其它部分，辅助二叉树的遍历，即线索二叉树。

线索的规定

原链接	线索表示
Left(P) =Λ	Left(P) = P之前驱
Left(P) = Q≠Λ	Left(P) =Q
Right(P) =Λ	Right(P) = P之后继
Right(P) = Q≠Λ	Right(P) = Q

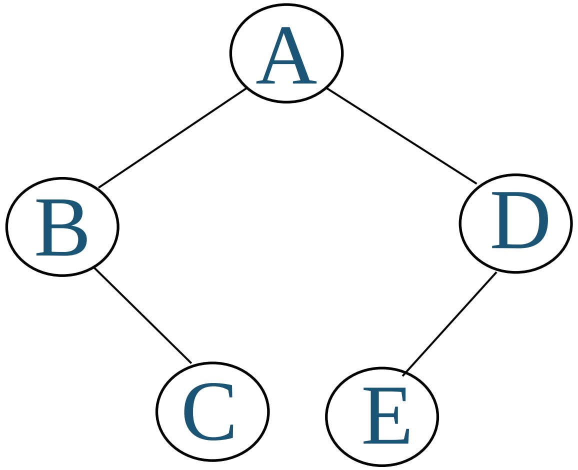
结点结构

- 为区分线索和指针，引进标识域*LThread*和*RThread*.

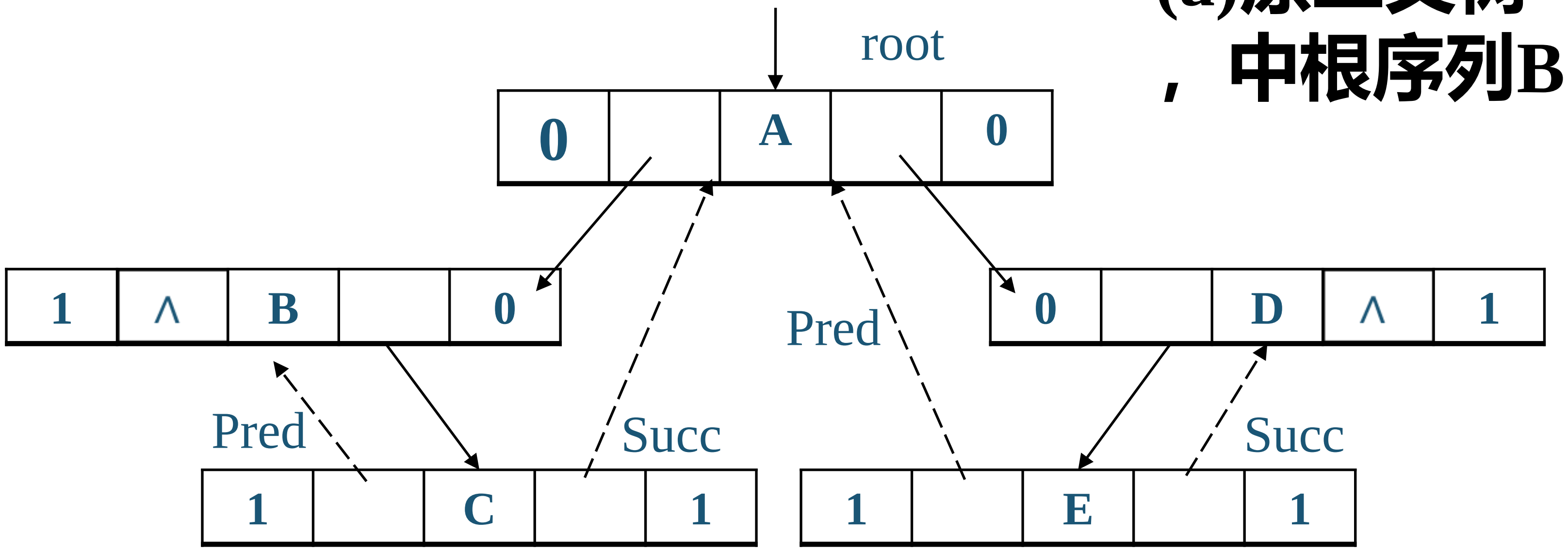
LThread	Left	Data	Right	RThread

- 若结点*t*有左儿子，则*Left*指向 *t* 的左儿子，且*LThread*的值为**0**；若 ***t*** 没有左儿子，则*Left*指向 ***t*** 的前驱结点，且*LThread*的值为**1**（此时称*Left*为线索）；**RThread**的规定类似

中序线索二叉树



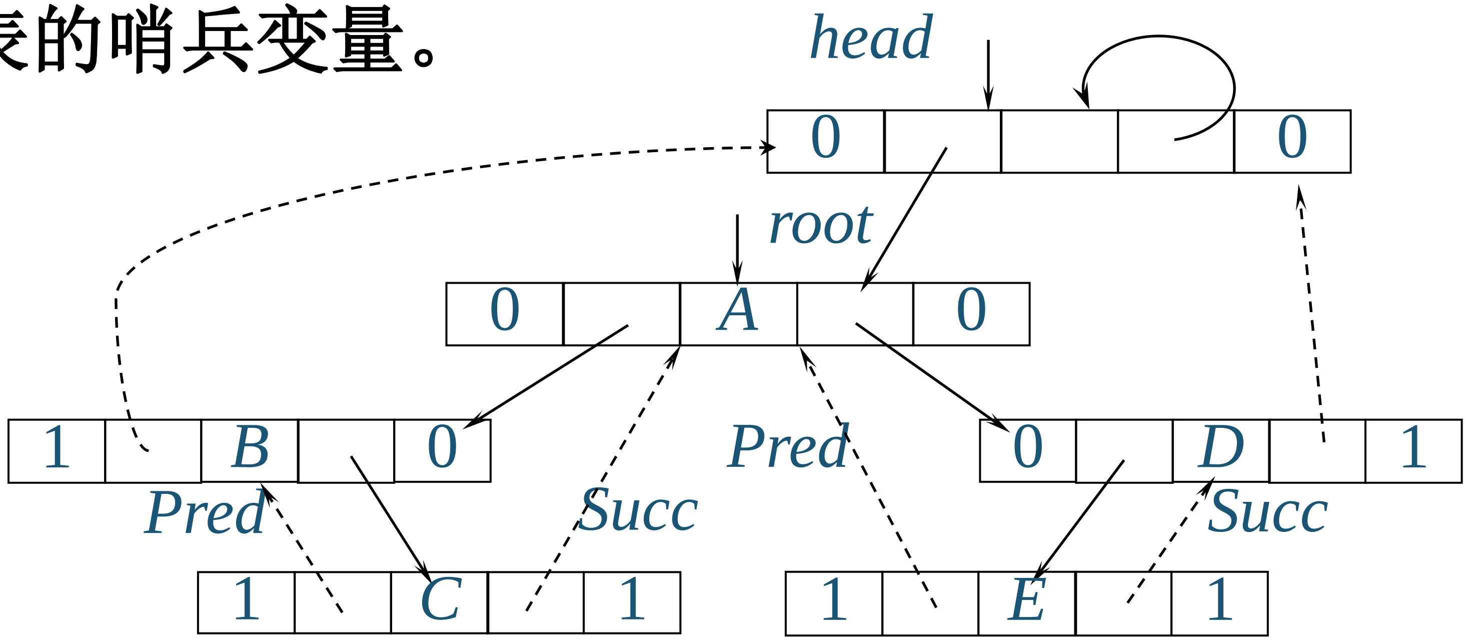
(a)原二叉树
，中根序列BCAED



(b)中序线索二叉树

带表头结点的线索二叉树

- 中序线索二叉树中，有两个空指针，分别是中根遍历序列的第一个结点的**Left**指针和最后一个结点的**Right**指针。可设置一个表头结点**head**，让这两个空指针都指向表头结点。其作用类似于链表的哨兵变量。



线索辅助中根遍历

1. 查找中根序列的第一个结点；
2. 访问该结点；查找中序后继结点。重复执行该操作，直到遍历结束。

查找中根序列的第一个结点

算法FIO(*t* . *q*)

/*查找t指向的的二叉树*T**的中根序列的首结点，并用*q*指向*/

FIO1.[初始化]

  $q \leftarrow t$.

  IF t = Λ THEN RETURN. *//有表头可不判*

FIO2. [找二叉树中根序列的第一个被访问结点]

  WHILE $LThread(q) = 0$ **DO** $q \leftarrow Left(q)$. ▌

- 算法FIO的时间复杂度O(n)

查找中序后继结点

算法NIO*(t, p. q)

/*在t指向的二叉树T*中搜索结点p的中序后继并令q指向，t、p不空*/

NIO*1. [*Right* (p)为线索]

$q \leftarrow$*Right* (p) .

IF *RThread*(p) = 1 THEN RETURN.

NIO*2. [*Right* (p)不为线索]

WHILE *LThread*(q) = 0 DO $q \leftarrow$*Left*(q). ▌

- 算法NIO*的时间复杂度O(n)
- NIO*2可调用FIO实现

中序遍历线索二叉树

算法InOrder*(*t*)

/* *t*指向中序线索二叉树T*之根，中根遍历T*的全部结点 */

InOrder*1. [求中根序列首结点]

FIO(*t*. *q*) .

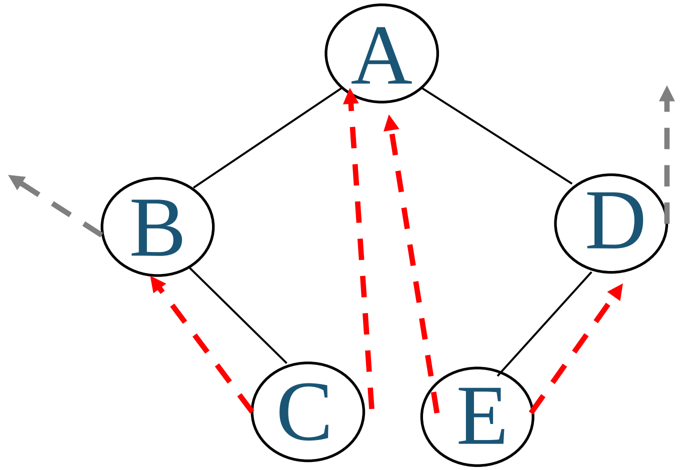
InOrder*2. [用NIO*求*q*之中序后继]

WHILE *q*≠Λ DO (

PRINT(*Data*(*q*)) .

NIO*(*t* , *q* . *q*) .) ▌

运行演示



线索二叉树的优点

- 时间复杂度：在算法**InOrder***中，**每个结点的Left链接和Right链接恰好都被检查一次**，因此，算法**InOrder***的时间复杂度为**O(n)**.
- 对于中序遍历操作，中序线索二叉树断然优于非线索二叉树。算法**InOrder***的时间复杂度为**O(n)**，而且不需要辅助的堆栈空间。
- 线索二叉树支持逆向遍历。从中根遍历序列的末结点出发，不断查找中序前驱，直到前驱为空。
  - ✓ 算法LIO和PIO;

其它说明

- 如果设置表头结点，中根序列最后一个结点的右线索指向**head**，**WHILE**语句的终止条件是**q = head.**（如果不设置表头结点，中根序列最后一个结点的右线索置空，**WHILE**语句的终止条件是**q = NULL**）

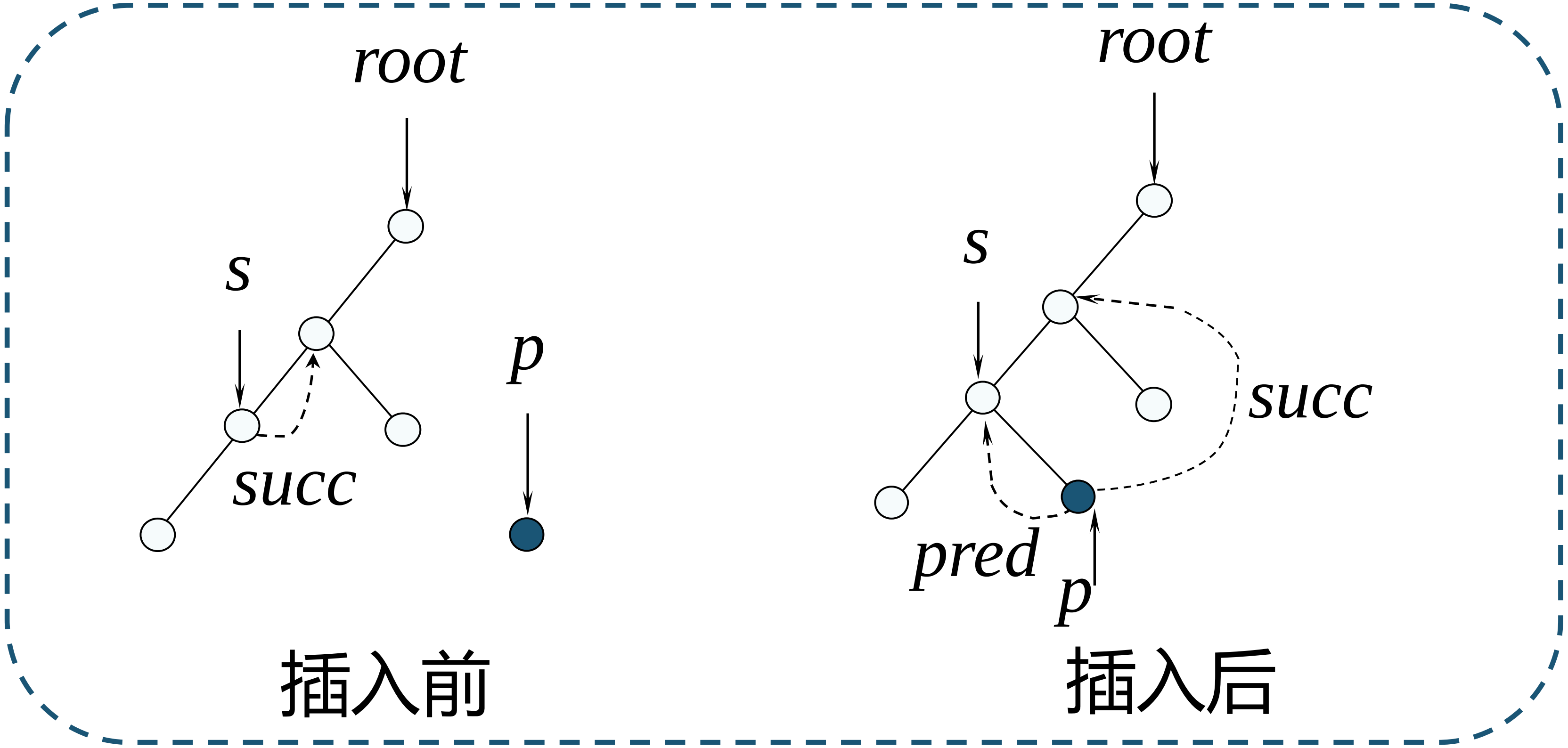
- 线索二叉树的构造也很容易。
  - 插入
  - 线索化

插入操作

- 插入操作：在线索二叉树 T* 中插入结点 p，作为T* 中某结点s的左子结点或右子结点。

- 插入右子结点为例演示（插入左子结点类似）

情况1： s无右子树

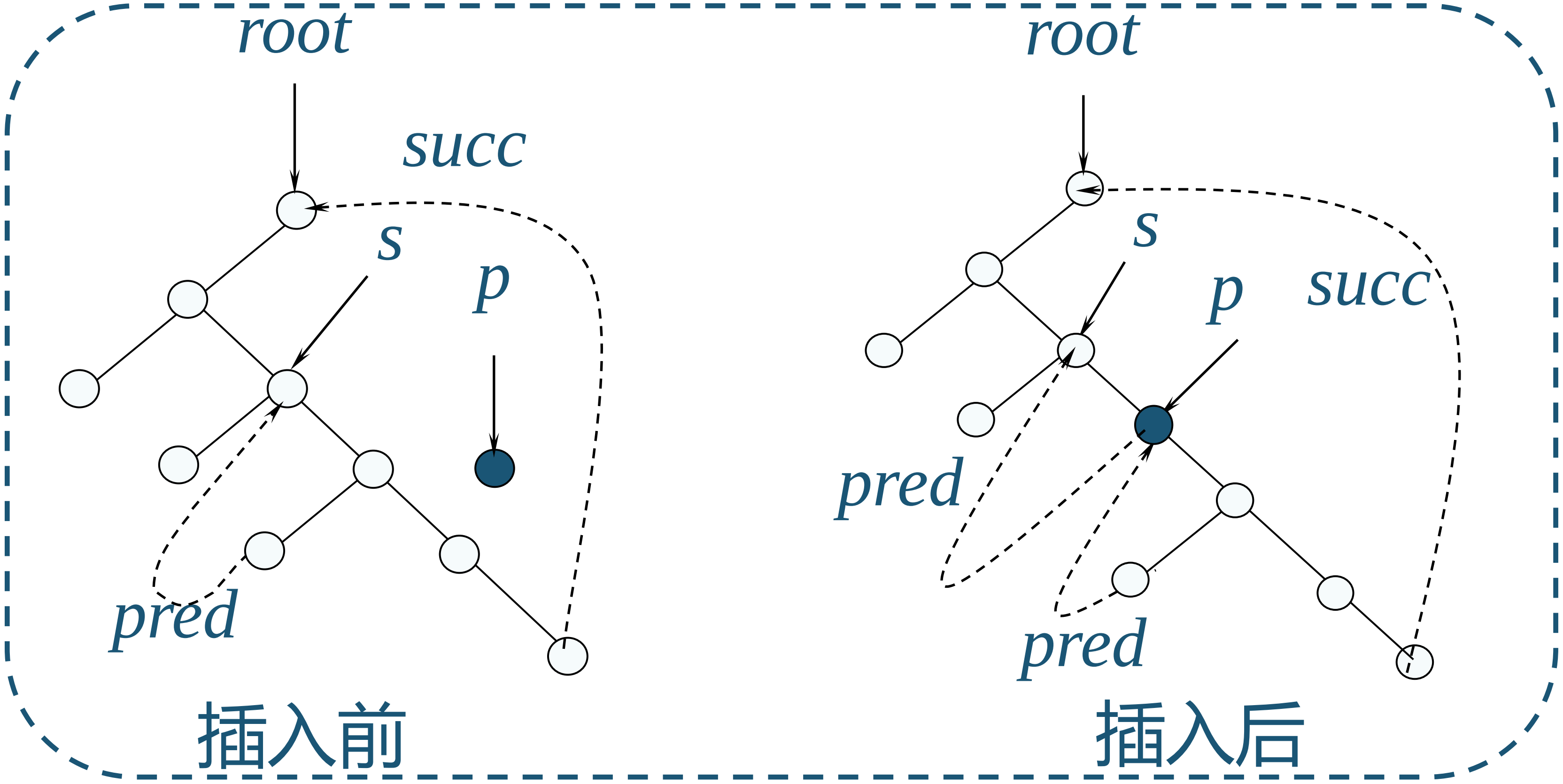


① $Right(p) \leftarrow Right(s)$. $RThread(p) \leftarrow RThread(s)$.

② $Left(p) \leftarrow s$. $LThread(p) \leftarrow 1$.

③ $Right(s) \leftarrow p$. $RThread(s) \leftarrow 0$.

情况2： s有右子树



①$Right(p) \leftarrow Right(s)$.$RThread(p) \leftarrow RThread(s)$.

②Left(p) ← s . LThread (p) ← 1 .

③$Right(s) \leftarrow p$

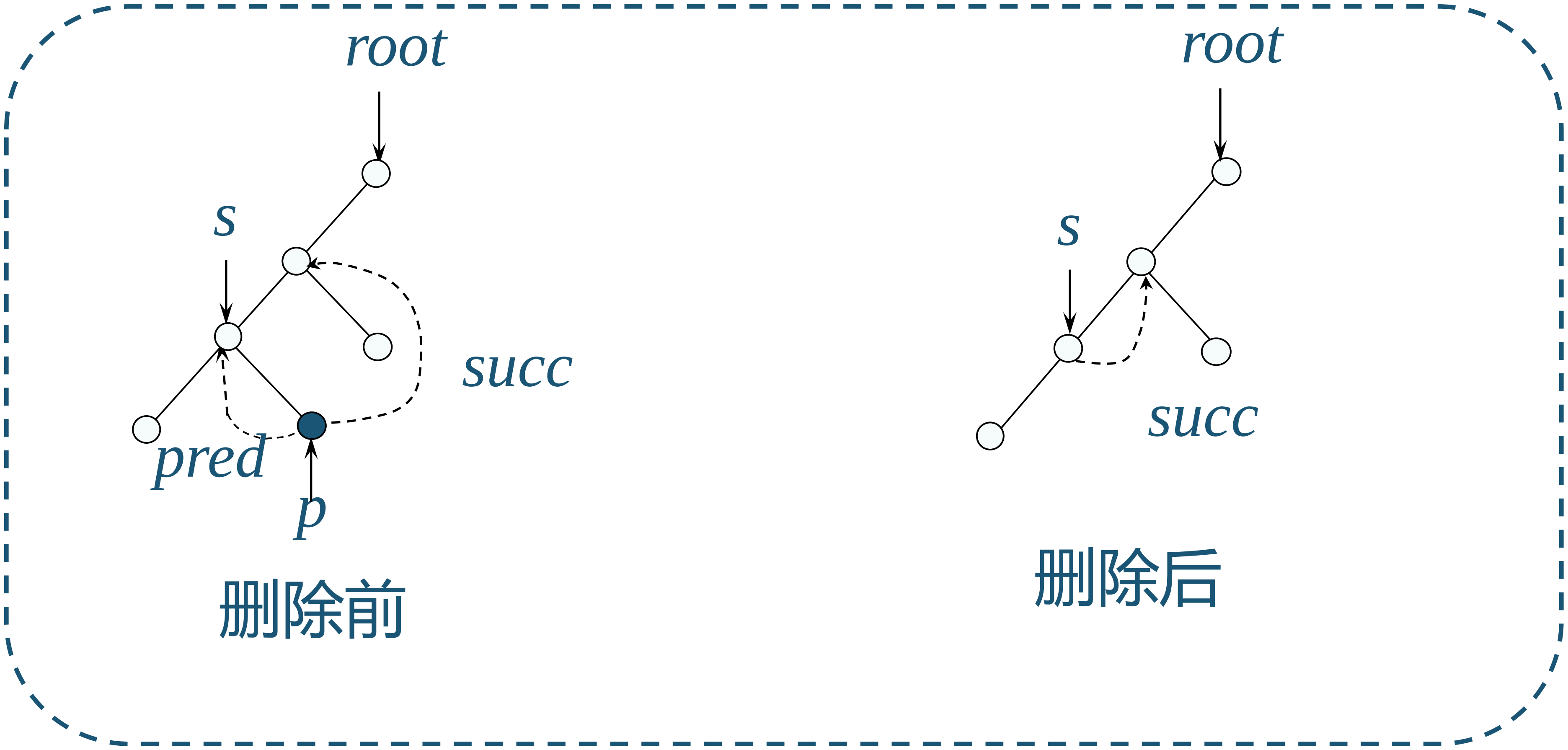
④$q \leftarrow Right(p)$. FIO(q .q). $Left(q) \leftarrow p$

删除操作

- 删除操作：在一棵线索二叉树中，可删除一个结点的左儿子，也可删除一个结点的右儿子

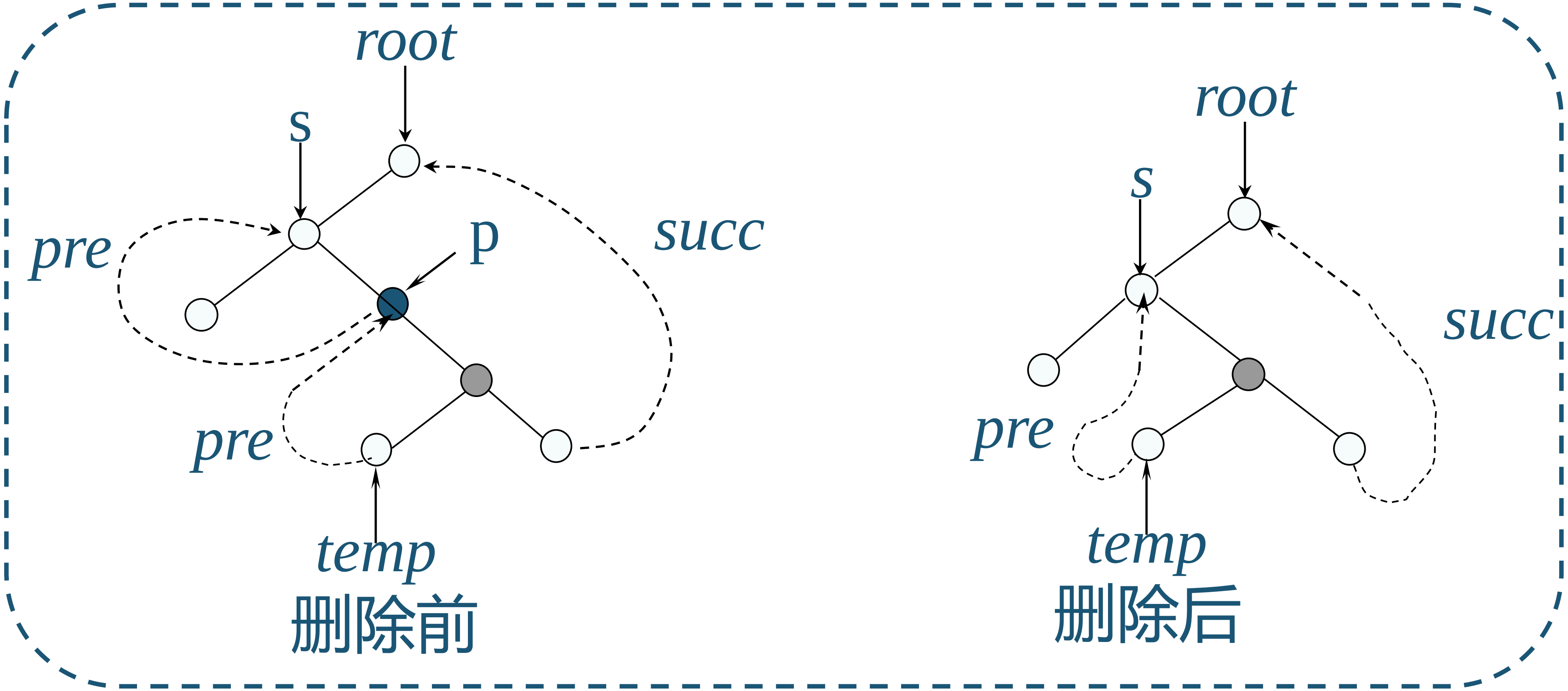
- 删除右子结点为例演示（删除左子结点类似），假定右子结点存在

情形1：p为叶结点.



$Right(\ s\ ) \leftarrow Right(\ p\ ).RThread(\ s\ ) \leftarrow 1.$

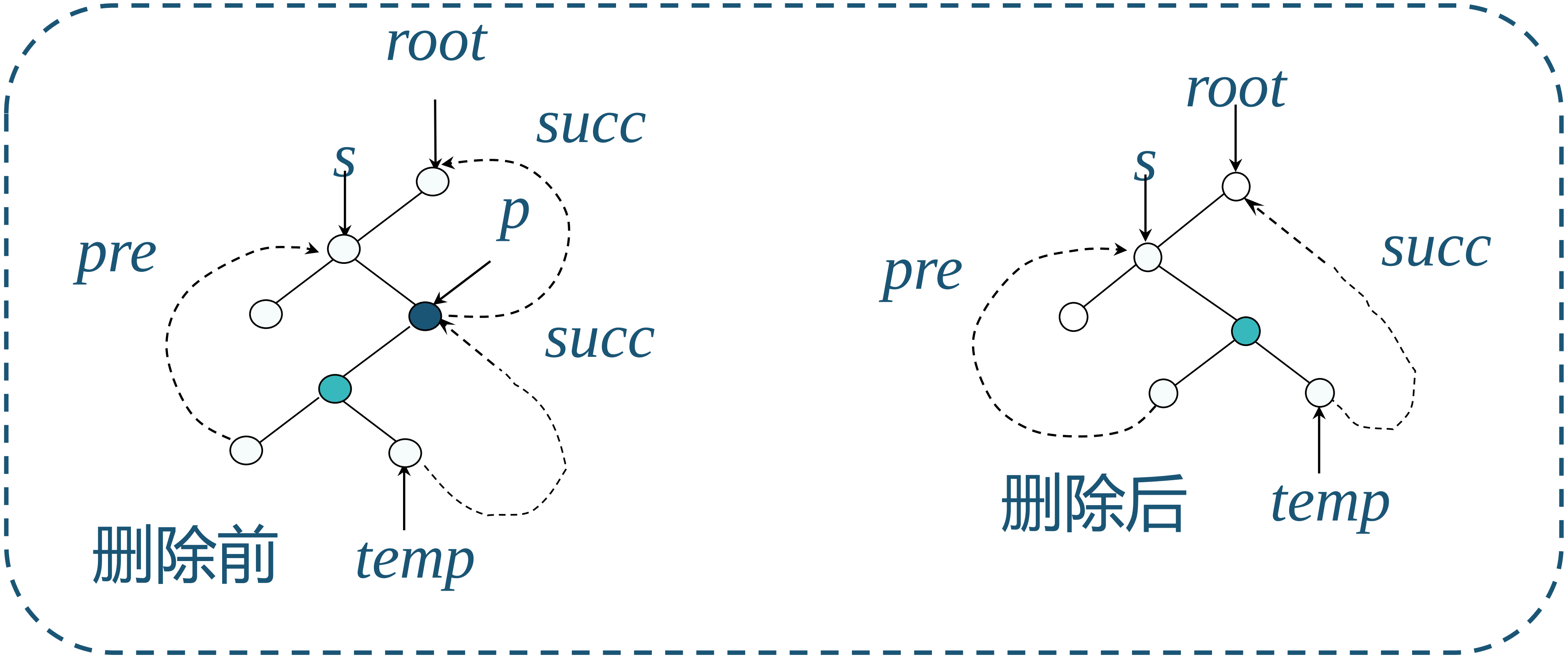
情形2：p有右子树，无左子树



设右子树之中根序列的第一个结点为temp，则

Right(s) ← Right(p).   Left(temp) ← s .

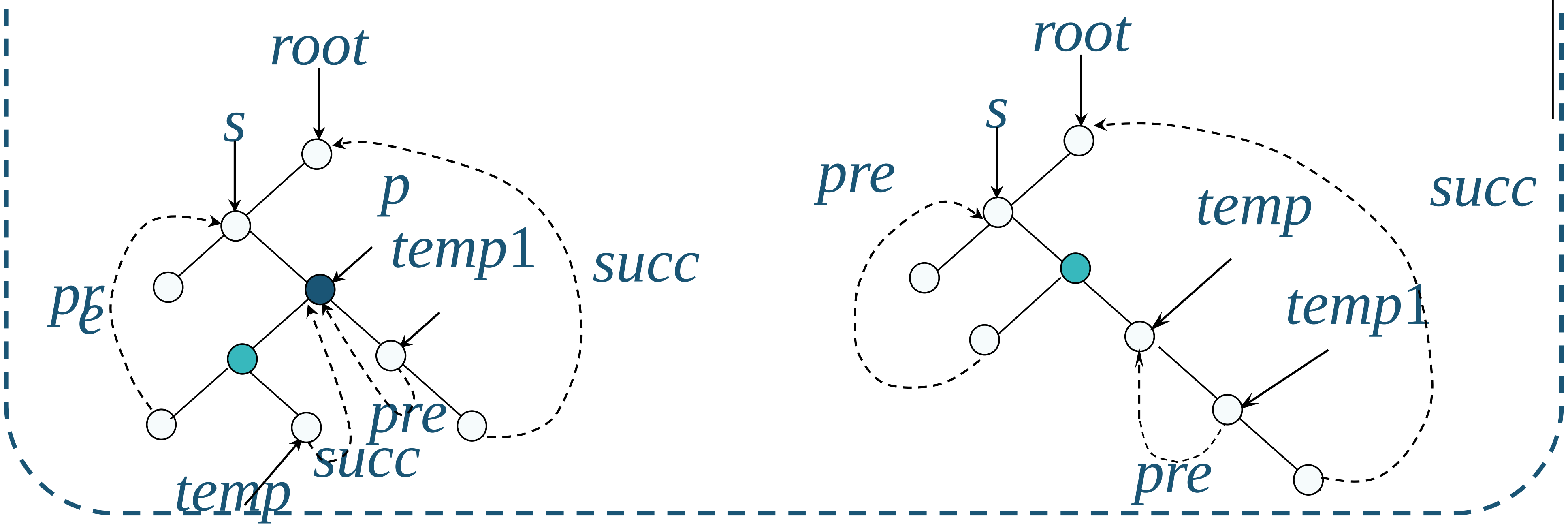
情形3：p无右子树，有左子树



设左子树之中根序列的最后一个结点为*temp*，则

Right (*s*)←*Left* (*p*). *Right* (*temp*)←*Right* (*p*).

情形4：p既有左子树又有右子树



删除前　　　　　　　　删除后

设*temp*1指向*p*之右子树的中根序列的第一个结点，*temp*指向*p*之左子树的中根序列的最后一个结点，则

Right(*temp*)←*Right*(*p*)　　*RThread*(*temp*)←0

Right(*s*) ←*Left* (*p*)　　*Left*(*temp*1)←*temp*

线索化（穿线，Threading）

- 线索化是构造线索二叉树的另一种方法，即将现有的非线索二叉树转化为线索二叉树。
- 线索化的前提是非线索二叉树的结点增加用于标识线索的LThread域和RThread域；或者，事先将非线索二叉树的数据和链接都复制到要生成的线索二叉树中。
- 线索化的实质是将二叉树中的空链接域填上相应的前驱和后继。

线索化算法思想

- 中序线索化就是在中根遍历的过程中生成中序线索的过程。算法思想如下：
  - ①递归为左子树增加线索；
  - ②访问操作就是**为当前的根结点增加线索**；
  - ③递归为右子树增加线索；
- 为根结点穿线（要保存其前驱结点）
  - 根结点的左线索指向前驱结点
  - 前驱结点的右线索指向根结点

中序线索化算法描述

算法InThread(*r*, *pre*. *pre*)

/*为二叉树*T* *增加中序线索,r指向*T**的根,T*中的结点线索标志为空*/

InThread1. [特判]

IF *r* =Λ THEN RETURN;

InThread2. [递归构造]

InThread(*Left*(*r*) , *pre*. *pre*).

IF *Left*(*r*)=ΛTHEN (*Left*(*r*)←*pre*. *LThread*(*r*)← 1.) //置前驱

IF *pre*≠Λ AND *Right*(*pre*)=Λ

THEN (*Right*(*pre*)←*r* . *RThread*(*pre*)← 1.) //置后继

***pre*←*r* .**

InThread(*Right*(*r*) , *pre*. *pre*). ▌

- 算法结束后，要调整最后一个结点右线索：**Right(pre) ← Λ; RThread(pre) ←1.**

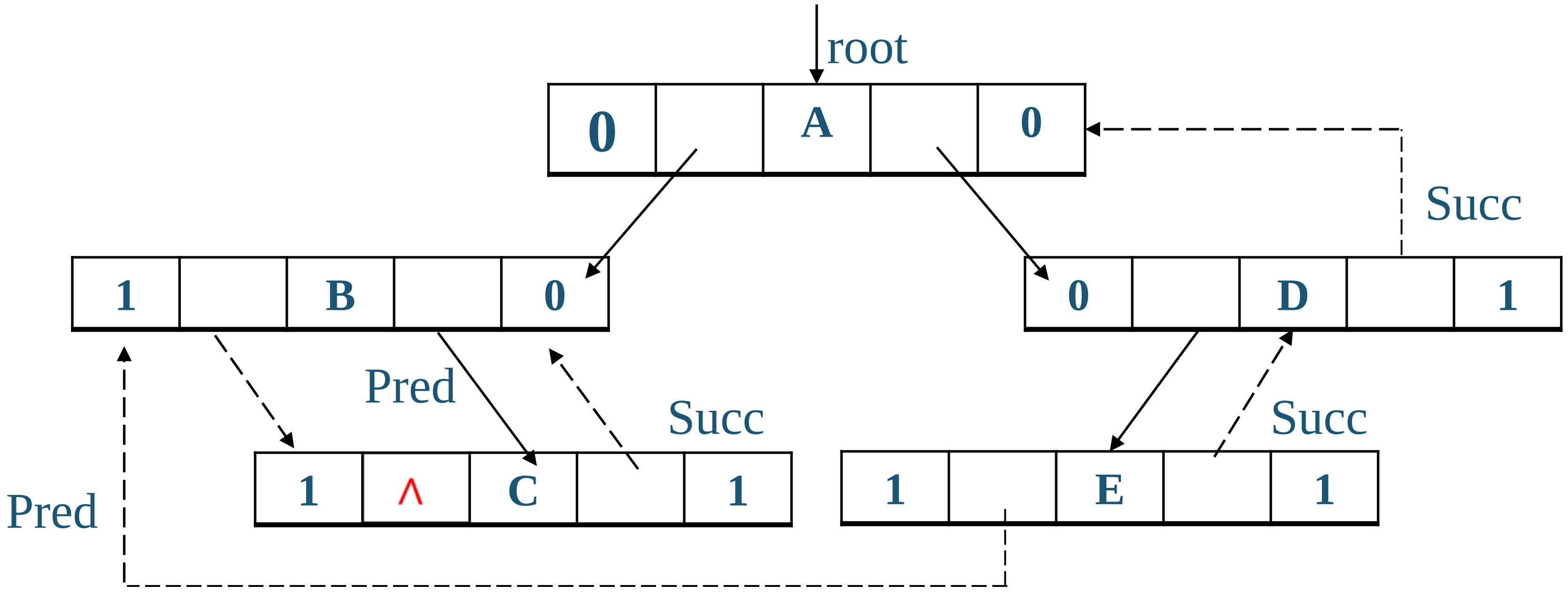
- 时间复杂度为**O(n)**

先序/后序线索二叉树

- 类似中根遍历对应的中序线索二叉树，也可定义先根遍历对应的先序线索二叉树和后根遍历对应的后序线索二叉树。

- 以后序线索二叉树为例讨论（先序线索二叉树与后序线索二叉树相对。）

后序线索二叉树



后序线索二叉树
原二叉树后根遍历序列为CBEDA;

- 后序线索二叉树可能只有**1**个空指针

- 后根序列最后一个结点就是根结点。

- 后序线索二叉树中，后根序列第一个结点就是二叉树的最左叶子结点；

后序线索二叉树
——查找结点p之后序前驱结点

- 算法思想：

若$LThread(p)=1$，则$Left(p)$指向p的后序前驱结点；

若$LThread(p)=0$，则：

若p有右子树，则p之右儿子是其后序前驱；

若p无右子树，则p之后序前驱是其左儿子.

- 后序线索二叉树中，查找结点的前驱代价O(1)

后序线索二叉树
——查找结点p之后序后继结点

- 算法思想

若*p*是根，则*p*无后序后继

若*p*非根，则：

若*RThread*(*p*)=1，则*Right*(*p*)指向*p*之后序后继结点

若*RThread*(*p*)= 0，则：

若*p*是其父亲的右儿子，则*p*之后序后继就是其父亲；

若*p*是其父亲的左儿子且*p*有右兄弟，则*p*的后序后继是其父亲之右子树中后序遍历到的首结点；

若*p*是其父亲的左儿子且*p*无右兄弟，则*p*的后序后继是其父亲.

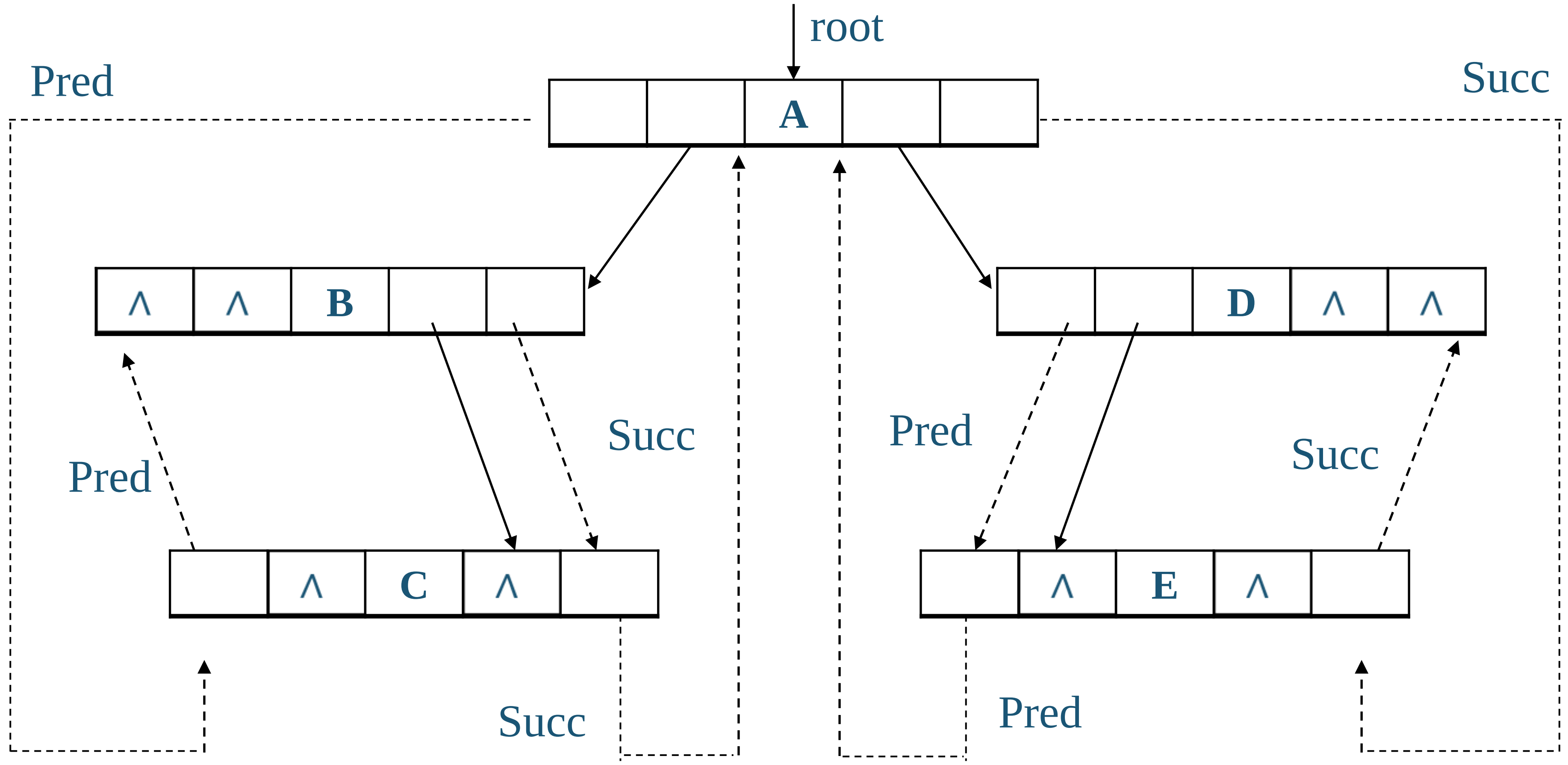
- 后序线索二叉树中，查找结点的后继结点的操并不总是有效的；操作的代价为O(n),

单边线索二叉树

- 以上介绍的线索二叉树是把线索同时运用到左边和右边两个方向，称为完全线索二叉树。
- 在完全线索二叉树和非线索二叉树的表示方法之间，有一个重要的中间地带，即只需左线索或右线索，建立单边的线索二叉树，分别称为左线索二叉树和右线索二叉树。
  - ✓ 例如：在右线索二叉树中，仍可通过Right建立线索，利用RThread标识线索；查找后继操作和遍历操作稍加修改即可工作。

二叉树的扩张（增加前驱后继）

Pred	Left	Data	Right	Succ



二叉树的扩展(用空间换时间)

总结

- 线索二叉树的提出
- 中序线索二叉树
  - 查找中序第一个结点；
  - 查找结点的中序后继
  - 中根遍历
  - 插入
  - 删除
  - 线索化
- 后序（先序）线索二叉树
  - 后序前驱、后续后继
- 单边线索二叉树