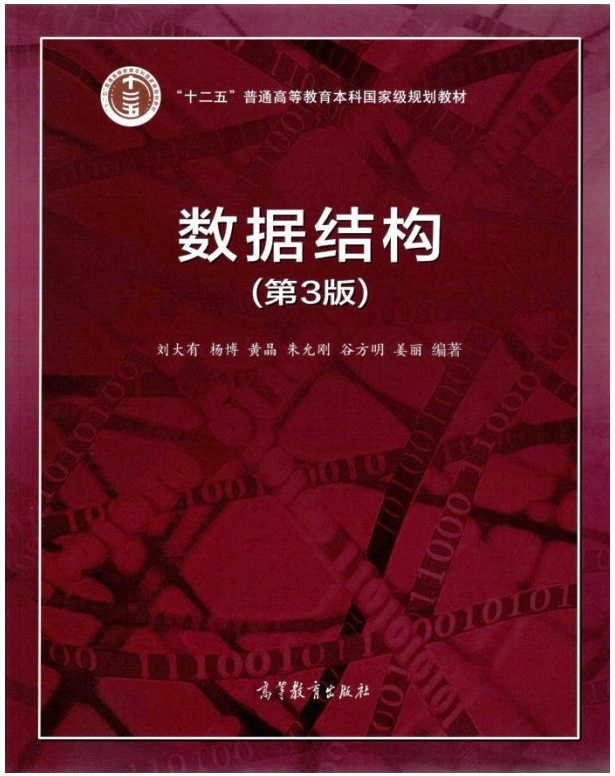


线索二叉树

动机和基本概念

中序线索二叉树的基本操作

实现方案

关于先序/后序线索二叉树

数据之法

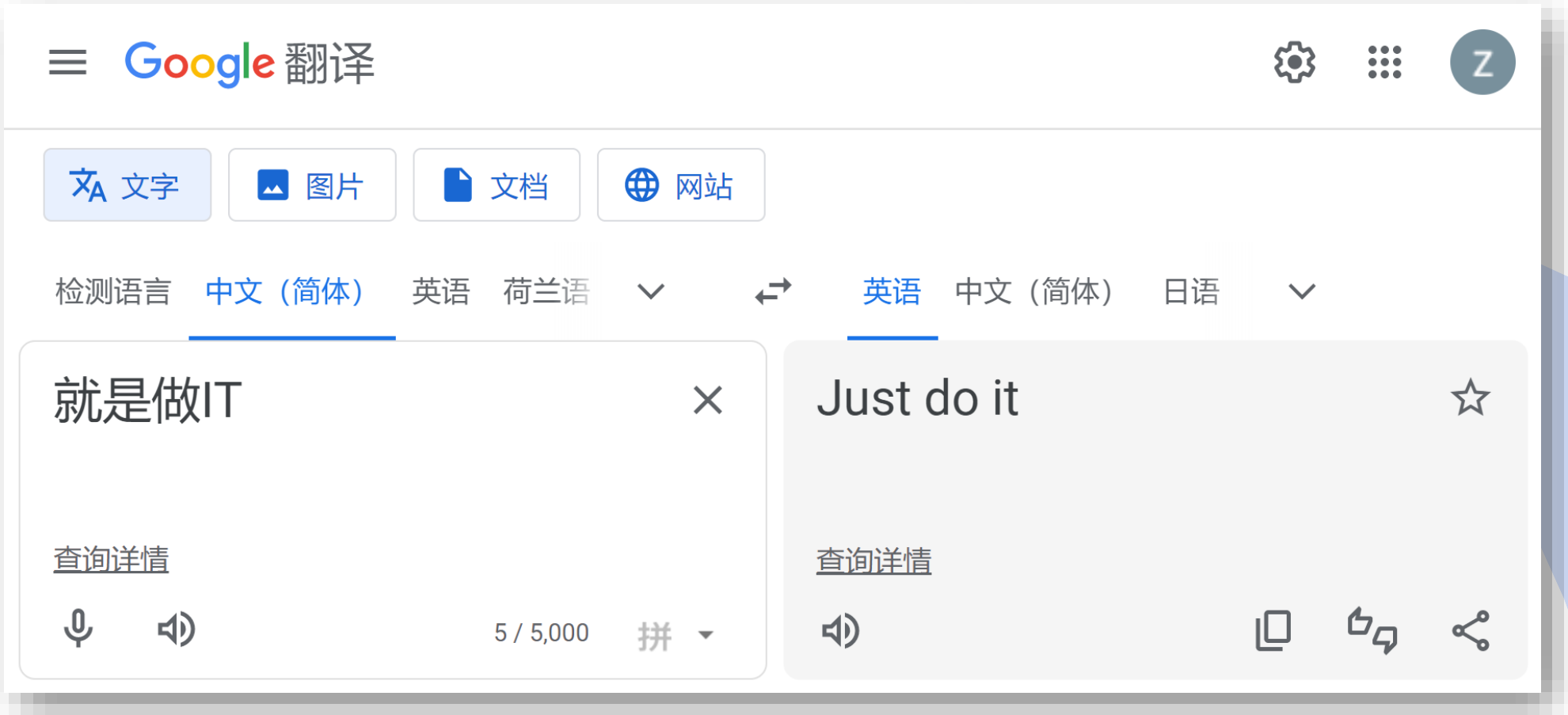
结构之美

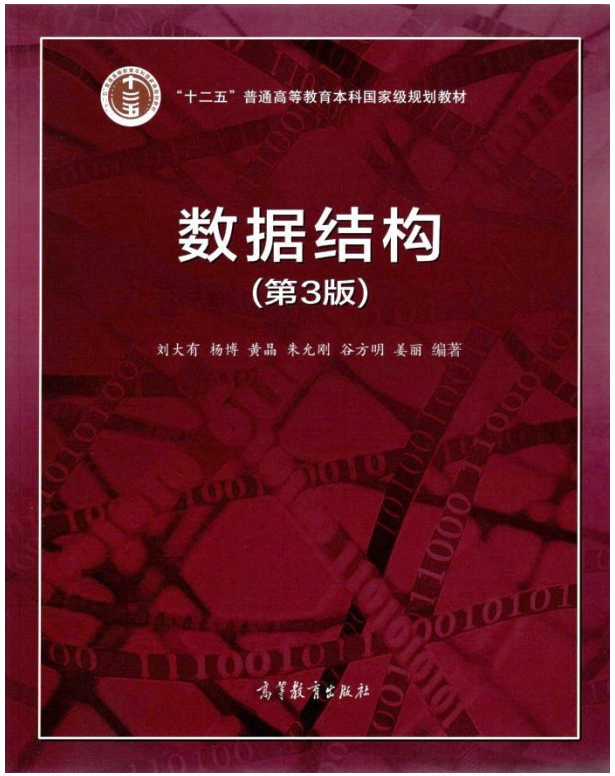
算法之道

zhuyungang@jlu.edu.cn

Last updated on 2023.10

谷歌权威翻译认证


Google 翻译
Z
文字
图片
文档
网站
检测语言
中文（简体）
英语
荷兰语
英语
中文（简体）
日语
就是做IT
查询详情
5 / 5,000
拼
Just do it
查询详情



线索二叉树

动机和基本概念

中序线索二叉树的基本操作

实现方案

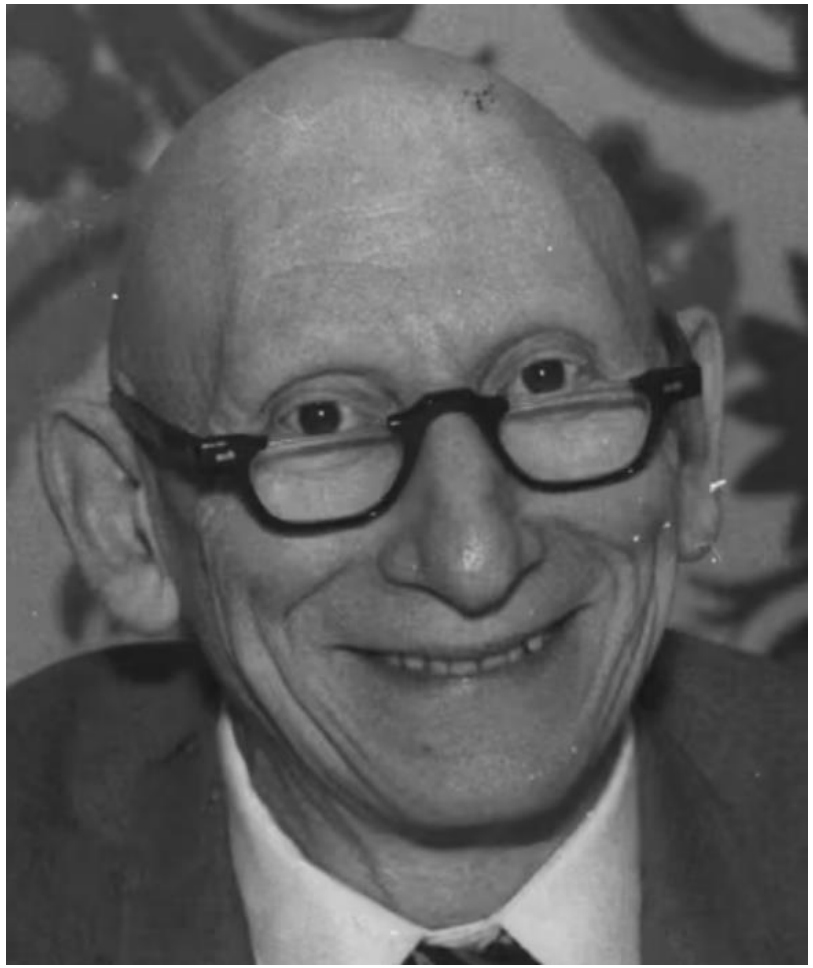
关于先序/后序线索二叉树

数据之法
结构之美
算法之道

zhuyungang@jlu.edu.cn

Last updated on 2023.10

线索二叉树的提出者



Alan J. Perlis

(1922 - 1990)

首届图灵奖获得者

美国工程院院士

卡内基梅隆大学计算机系创始人

卡内基梅隆大学教授

耶鲁大学教授

美国计算机学会理事长

李凯

普林斯顿大学教授

美国工程院院士

中国工程院外籍院士

1954年生于吉林省长春市

1977年本科毕业于吉林大学计算机系

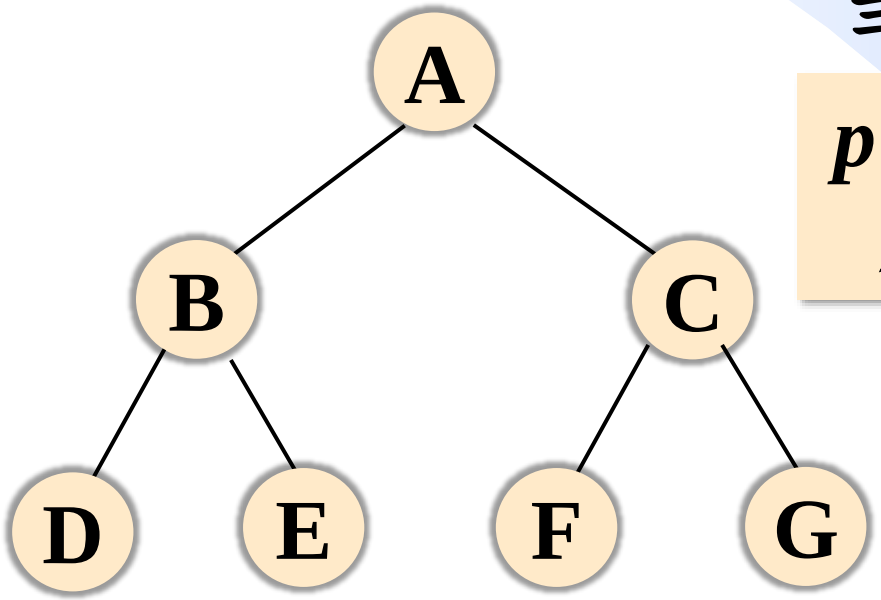
耶鲁大学博士（师从**Alan Perlis**）

吉林大学计算机学科发展咨询委员会委员

练习

已知二叉树结点结构如下，给定二叉树和其中一个结点*p*，找出*p*的中根后继结点。【腾讯面试题】

```
struct TreeNode{
   int data;
   TreeNode *parent;
   TreeNode *left;
   TreeNode *right;
};
```



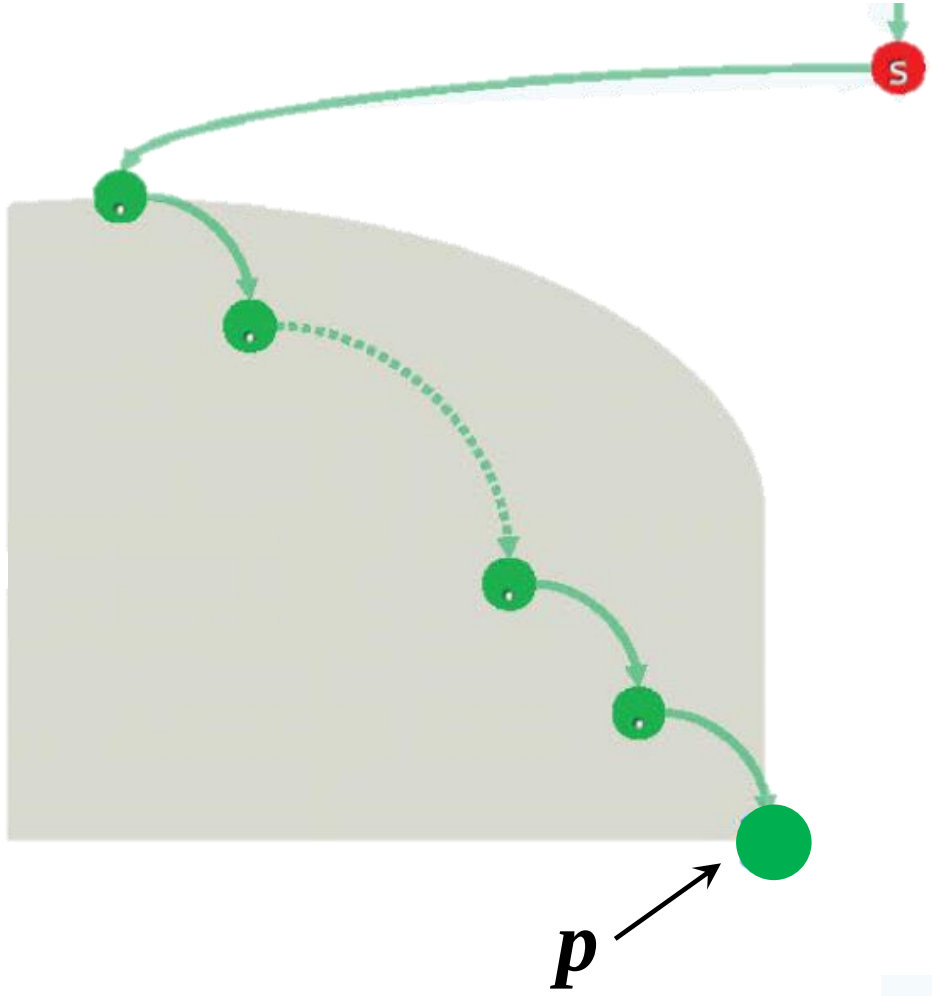
中根序列：D B E A F C G

当*p*有右孩子时：

*p*的右子树的中根序列第1个结点

中根序列中结点的前驱称作中根前驱，结点的后继称作中根后继

练习



当p无右孩子时：

将p包含于其左子树的最低祖先

时间复杂度

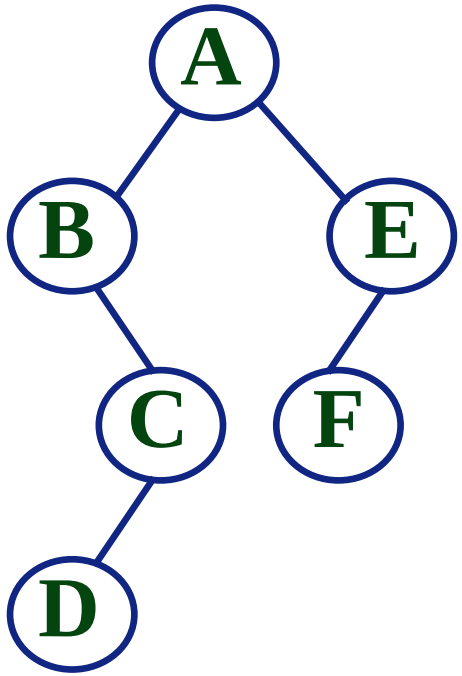
有父指针：$O(h)$

无父指针：中根遍历$O(n)$

对于完全二叉树$h=\lfloor \log n \rfloor$

线索二叉树——动机

- 在二叉树（结点无父指针域）上只能找到结点的左孩子、右孩子，找结点的中（先、后）根前驱和后继只能通过遍历。
- 能否更快速的找到给定结点的中（先、后）根前驱和后继？
- 二叉树的结点中有很多空指针，造成存储空间的浪费。

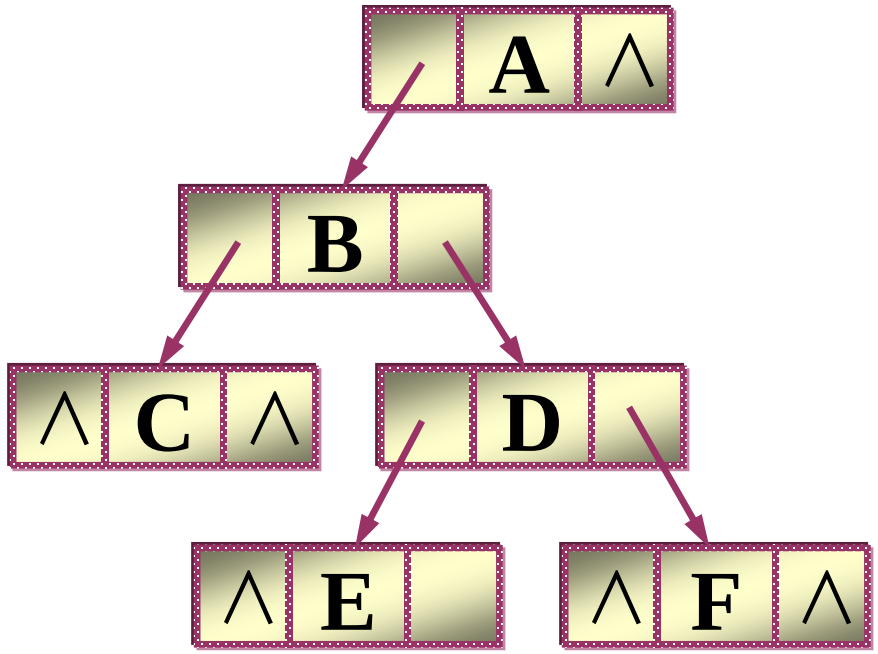


中根遍历序列：B D C A F E

中根序列中结点的前驱称作中根前驱，结点的后继称作中根后继。

线索二叉树——动机

- 包含n个结点的二叉树，在其$2n$个指针域中仅有$n-1$个被使用。
- 可以把这些空指针利用起来：指向结点的中根前驱或后继。



每个非空指针域都对应一条边

线索二叉树

Thread=0

Thread=1

- 如果某结点
  - ✓有子结点，则其Left/Right指向子结点
  - ✓无子结点，则其Left/Right指向其遍历序列的前驱/后继
- 增加“标志位”标识指针到底是指向子结点还是前驱/后继
- 指向某结点中根前驱和后继的指针称为**线索**；
- 按中根遍历得到的线索二叉树称为**中序线索二叉树**；
- 按先根遍历得到的线索二叉树称为**先序线索二叉树**；
- 按后根遍历得到的线索二叉树称为**后序线索二叉树**。

线索二叉树的结点结构

LThread	Left	Data	Right	RThread

- 增加域 *LThread* 和 *RThread*，为二进制位（占1比特），表示该结点的*Left*和*Right*是否为线索。
- 若结点 *t* 有左孩子，则*Left*指向 *t* 的左孩子，且*LThread* 值为0；若 *t* 无左孩子，则*Left* 指向 *t* 的某一遍历序的前驱结点，且 *LThread* 值为1，此时称 *Left* 为线索。
- 若结点 *t* 有右孩子，则*Right*指向 *t* 的右孩子，且*RThread* 值为0；若 *t* 无右孩子，则 *Right*指向*t*的后继结点，且 *RThread* 值为1，此时称 *Right* 为线索。

中根线索二叉树为例

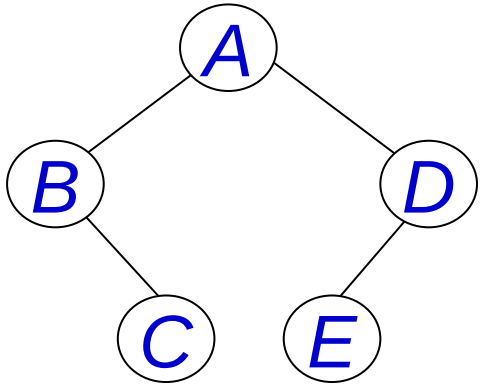
LThread	Left	Data	Right	RThread

$$LThread = \begin{cases} 0, Left\text{ 指向该结点的左孩子} \\ 1, Left\text{ 指向该结点的中根前驱} \end{cases}$$

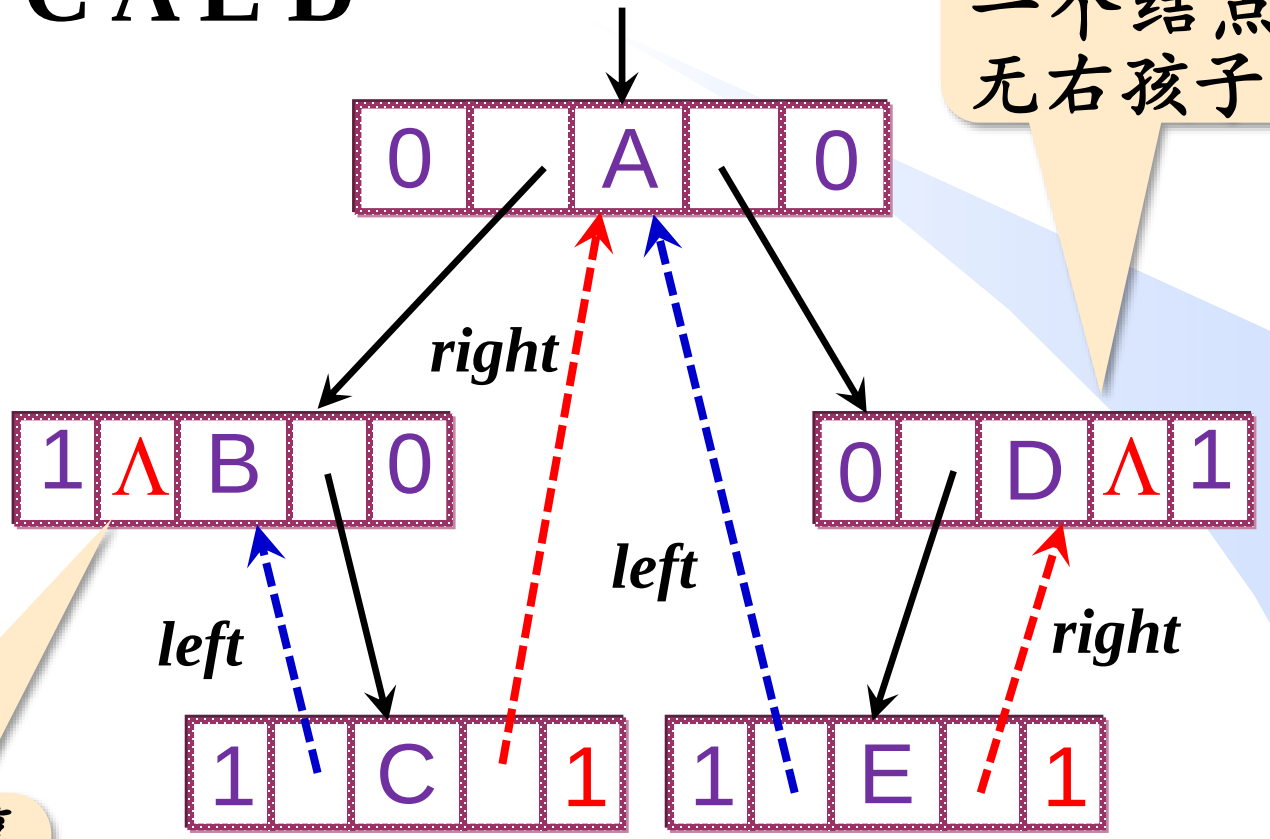
$$RThread = \begin{cases} 0, Right\text{ 指向该结点的右孩子} \\ 1, Right\text{ 指向该结点的中根后继} \end{cases}$$

[例] 中序线索二叉树。

中根遍历序列： **B C A E D**



(a)二叉树



(b) 中序线索二叉树

中根序列最后一个结点一定无右孩子

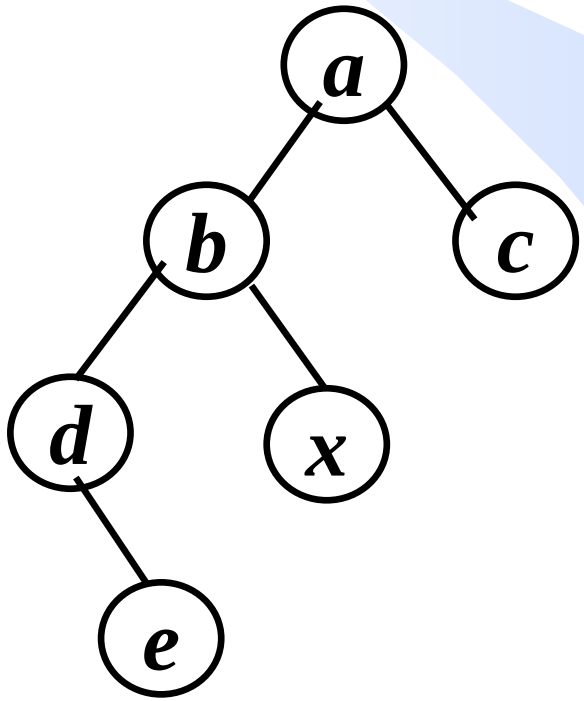
中根序列第一个结点一定无左孩子

课下思考

若对图中的二叉树进行中序线索化，则结点 x 的左右线索指向的结点分别是(　D　)【考研题全国卷】

A. e, c　　B. e, a

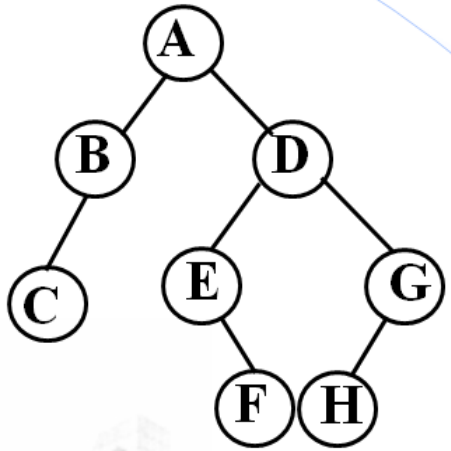
C. d, c　　D. b, a

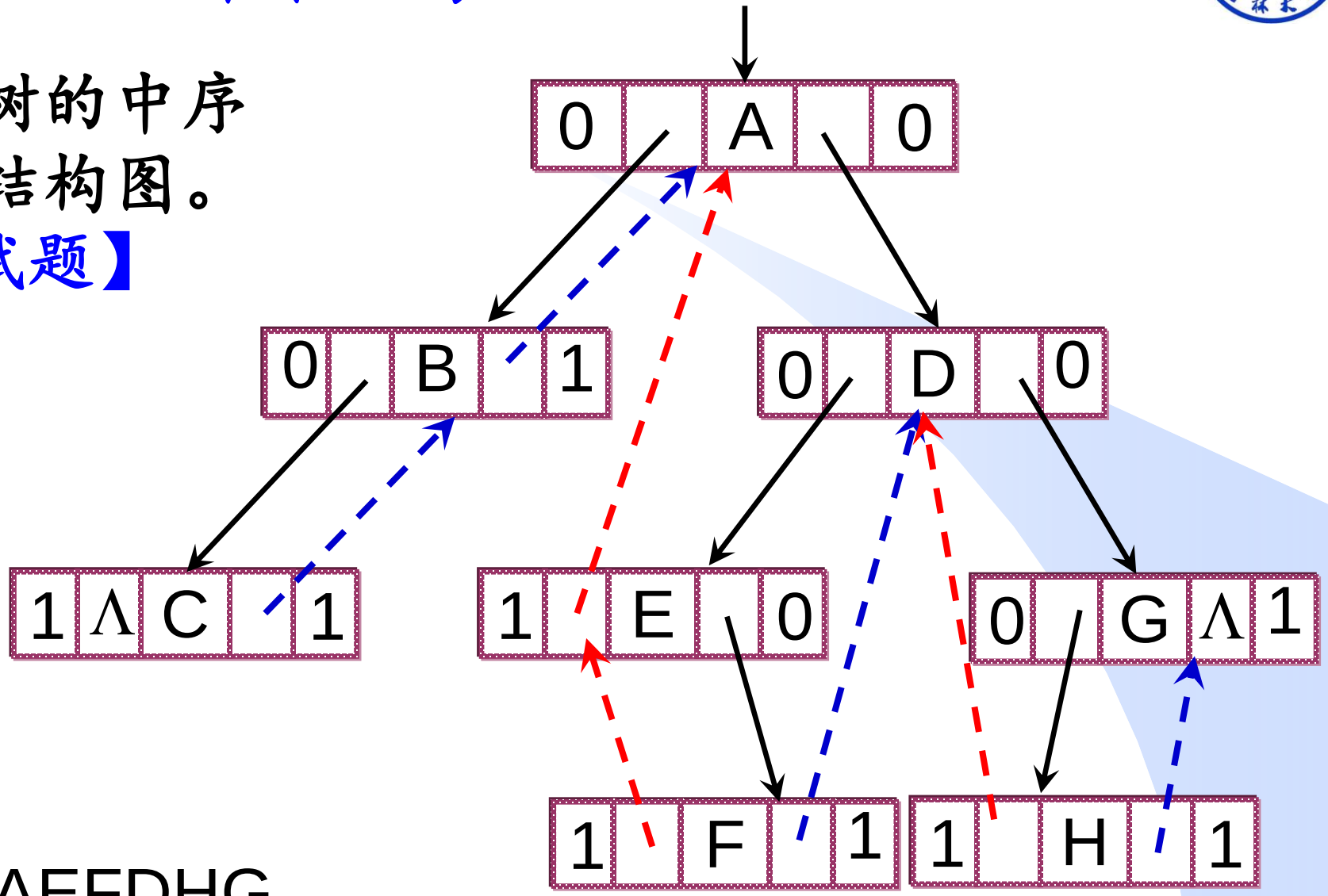


课下思考

画出左图所示二叉树的中序线索二叉树的链接结构图。

【吉林大学期末考试题】



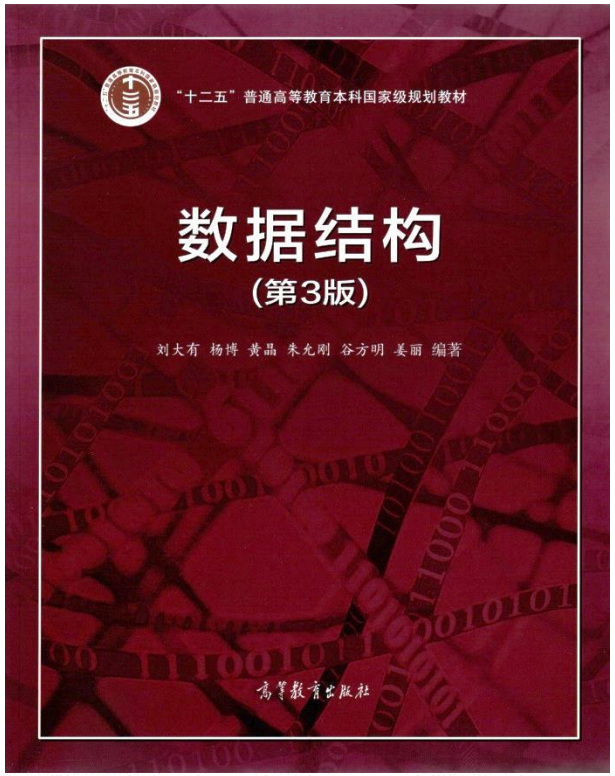


中根遍历序列： CBAEFDHG

总结

线索二叉树的目的：
在中序线索二叉树中可以方便的找到给定结点的中序前驱和中序后继结点，并且不需要太多额外的空间。

线索二叉树中，一个结点是叶结点的充要条件为：左、右标志(*LThread*、*RThread*)均是1。



线索二叉树

动机和基本概念

中序线索二叉树的基本操作

实现方案

关于先序/后序线索二叉树

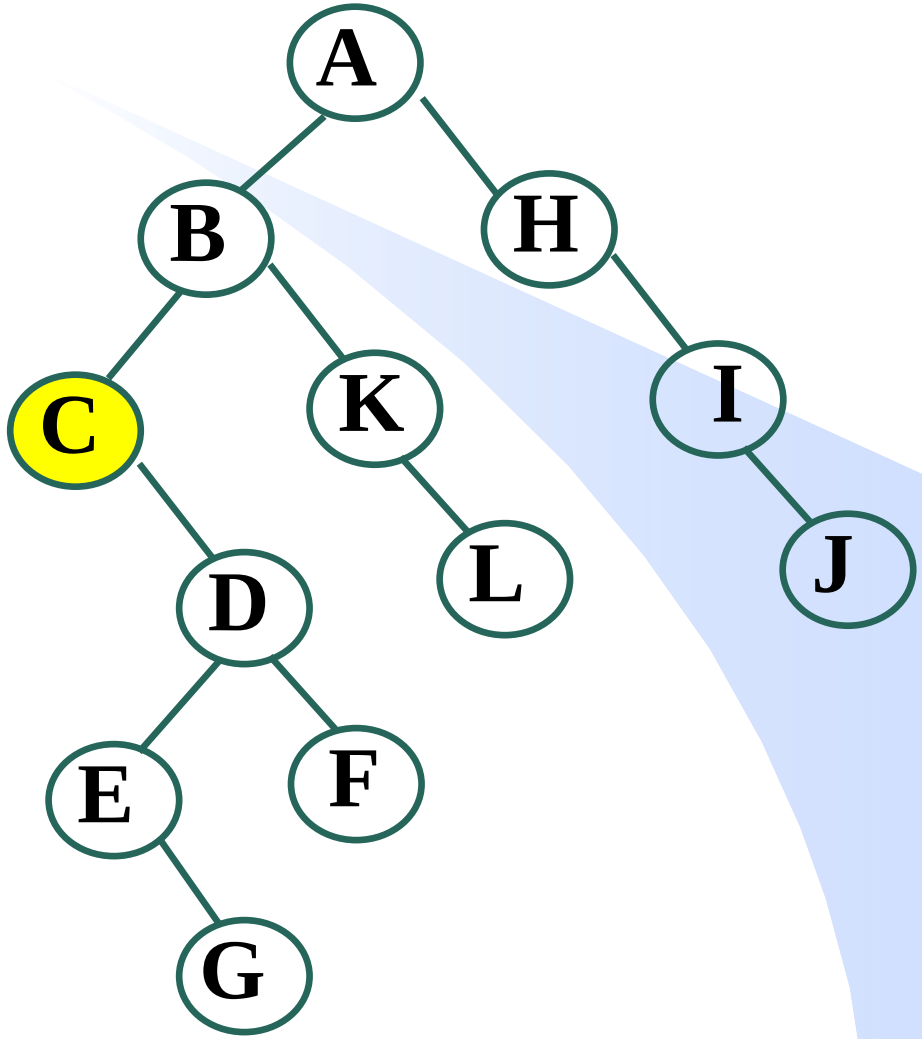
数据之法
结构之美
算法之道

zhuyungang@jlu.edu.cn

Last updated on 2023.10

找线索二叉树的中根序列的第一个结点

从根结点出发，沿左分支下行，直到最深的结点，该结点是中根序列第一个结点。

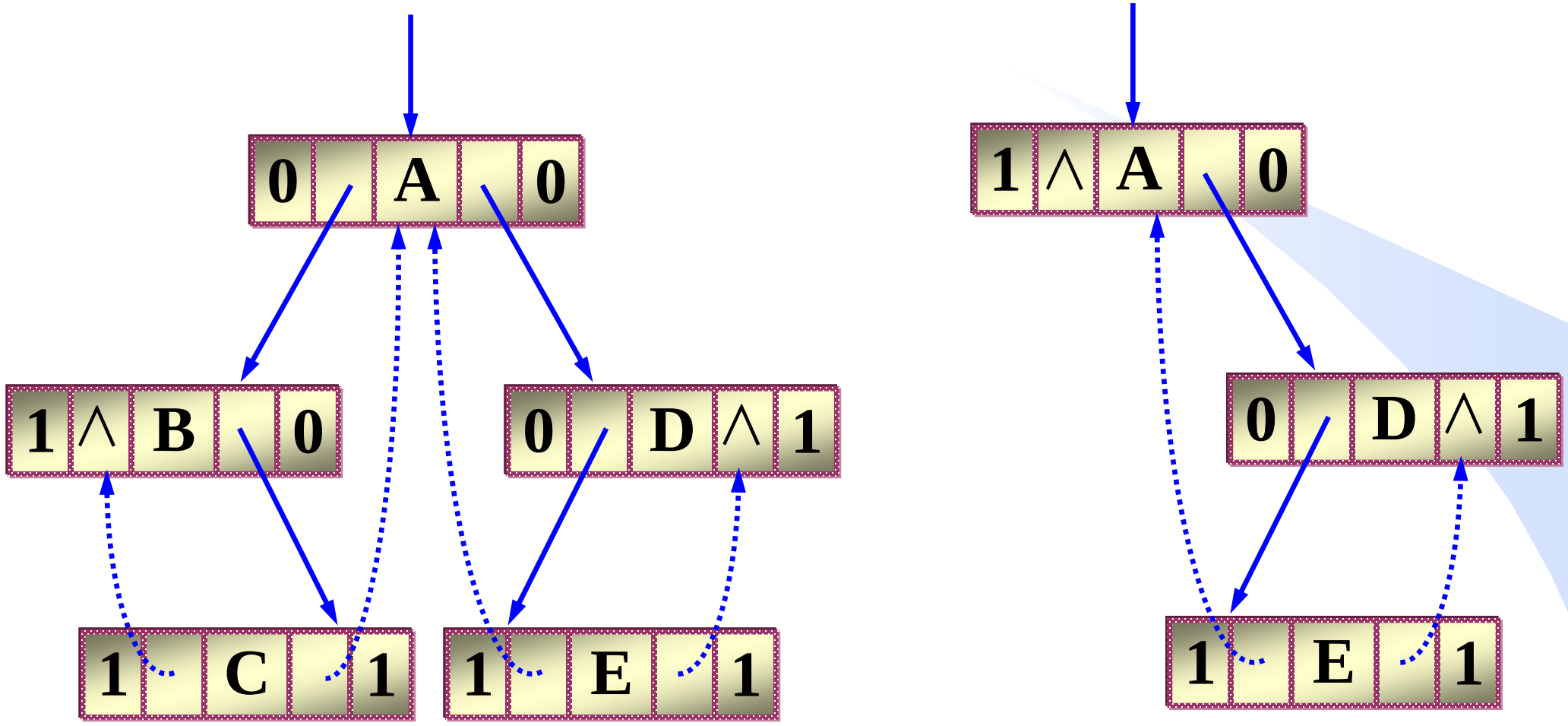


搜索以t为根的线索二叉树的中根序列的第一个结点

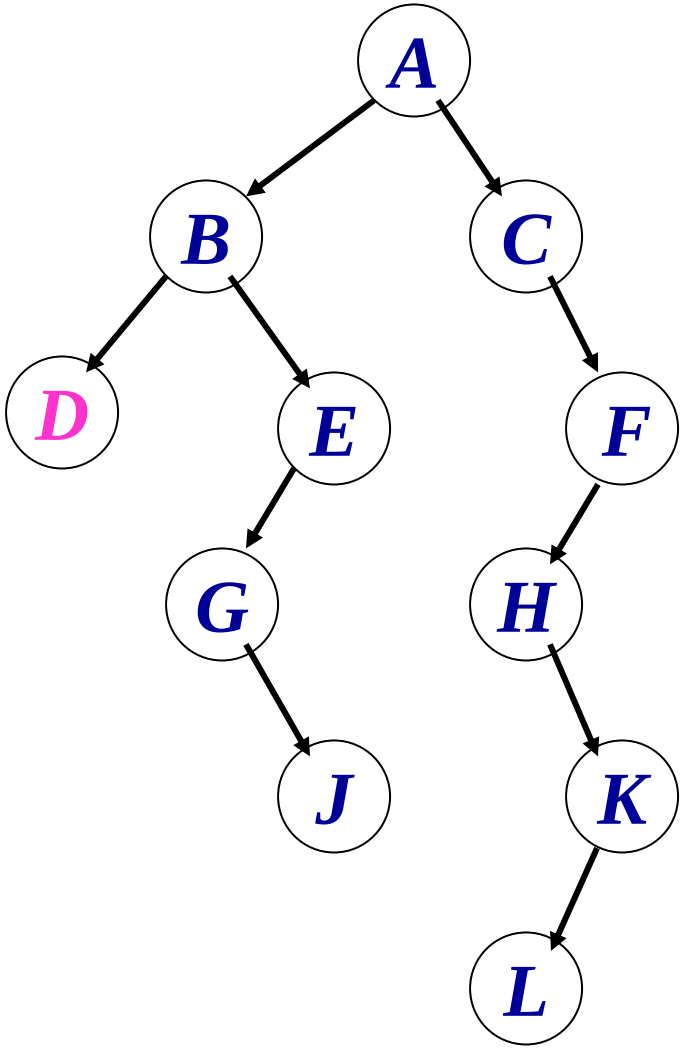
```
TreeNode* FirstInOrder(TreeNode* t){/*在以t为根的中序线索二叉树中找中根序列的首结点，并返回指针*/
    Node* p = t;
    while(p->LThread == 0)    //p有左孩子
        p = p->left;
    return p;
}
```

时间复杂度O(*h*)
***h*为二叉树高度**

```
while(p->LThread == 0) p = p->left;
```



找线索二叉树的中根序列的最后一个结点

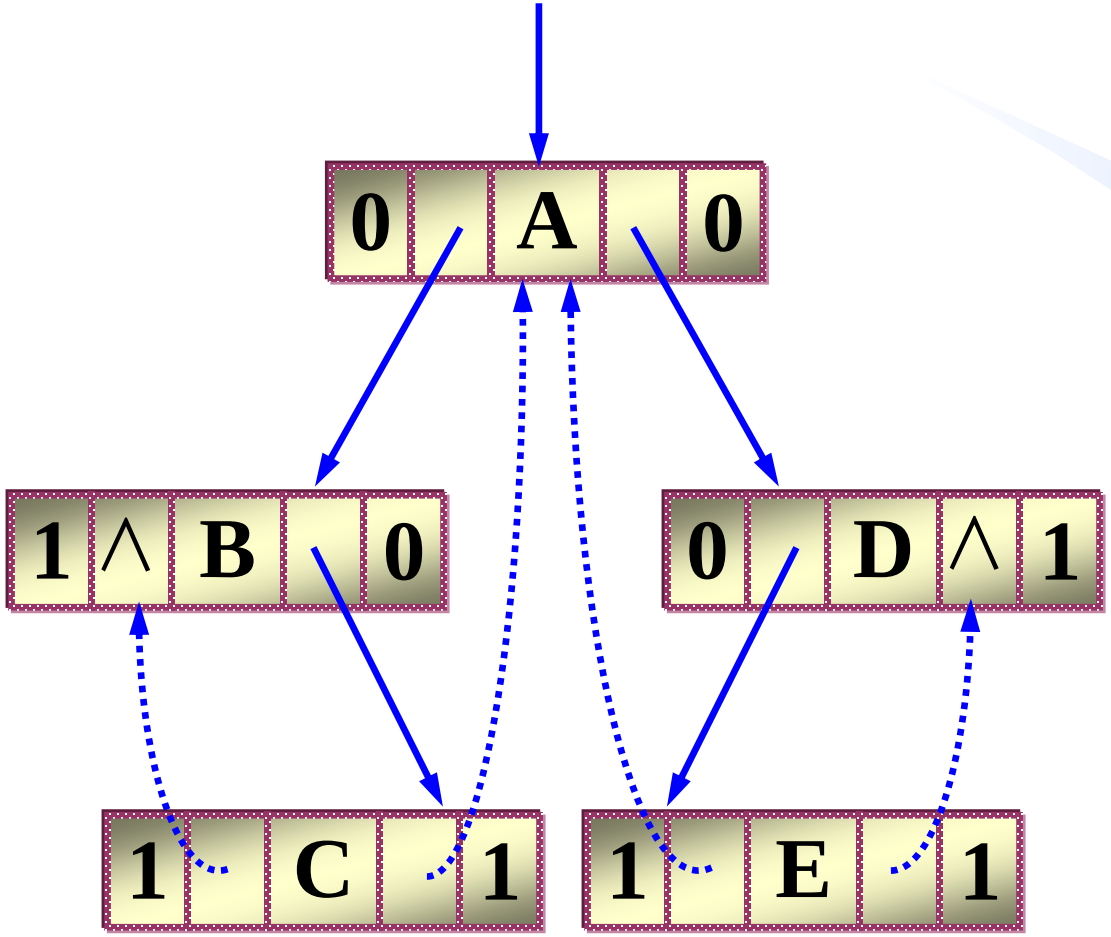


从根结点开始沿右分支下行，找第一个无右孩子的结点。

```
TreeNode* LastInOrder(TreeNode* t){
/*在以t为根的中序线索二叉树中找中根序列的末结点，并返回指向它的指针*/
    TreeNode* p = t;
    while(p->RThread == 0) //p有右孩子
        p = p->right;
    return p;
}
```

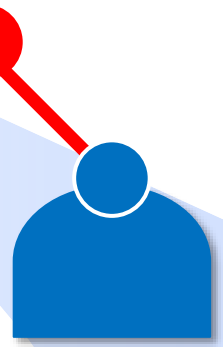
时间复杂度O(*h*)
***h*为二叉树高度**

```
while(p->RThread == 0) p = p->right;
```



在中序线索二叉树中，查找结点p的中根后继结点

- 若p->RThread为1，则p->right指向p的中根后继；
- 若p->RThread为0，则p的中根后继为p的右子树的中根序列的首结点。

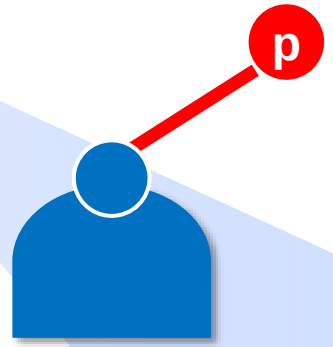



```
TreeNode* NextInOrder(TreeNode* p){
    if(p->RThread==1) return p->right;
    return FirstInOrder(p->right);
}
```

时间复杂度O(*h*)
*h*为二叉树高度

在中序线索二叉树中，查找结点p的中根前驱结点

- 若p->LThread为1，则p->left指向p的中根前驱；
- 若p->LThread为0，p的中根前驱结点是p的左子树的中根序列的最后一个结点。




```
TreeNode* PreInOrder(TreeNode* p){
    if(p->LThread==1) return p->left;
    return LastInOrder(p->left);
}
```

时间复杂度O(*h*)
*h*为二叉树高度

回顾中序线索二叉树的操作

- 找中根序列的首结点　　FirstInOrder
- 找中根序列的末结点　　LastInOrder
- 找中序前驱结点　　PreInOrder
- 找中序后继结点　　NextInOrder

中序线索二叉树的中根遍历

先访问中根序列中的第一个结点，然后依次访问结点的中根后继，直至其后继为空为止。

```
void InOrder(TreeNode *t) {
 for(TreeNode *p=FirstInOrder(t); p!=NULL; p=NextInOrder(p))
     visit(p->data);
}
```

时间复杂度O(*n*)

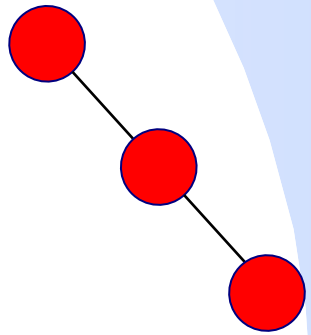
无需递归或栈
空间复杂度O(1)

二叉树的中序线索化

- 使二叉树变为线索二叉树的过程称为线索化。【大厂笔试题】
- 将二叉树中根遍历算法中的“访问结点”操作具体化为“建立当前访问的结点与其中根前驱结点的线索关系”。
- 算法中指针*p*指向当前正在访问的结点，同时设置一个指针*pre*（作为全局变量）始终指向当前访问结点*p*的中根前驱结点，即中根遍历过程中在*p*之前访问的结点，*pre*的初值为*NULL*。

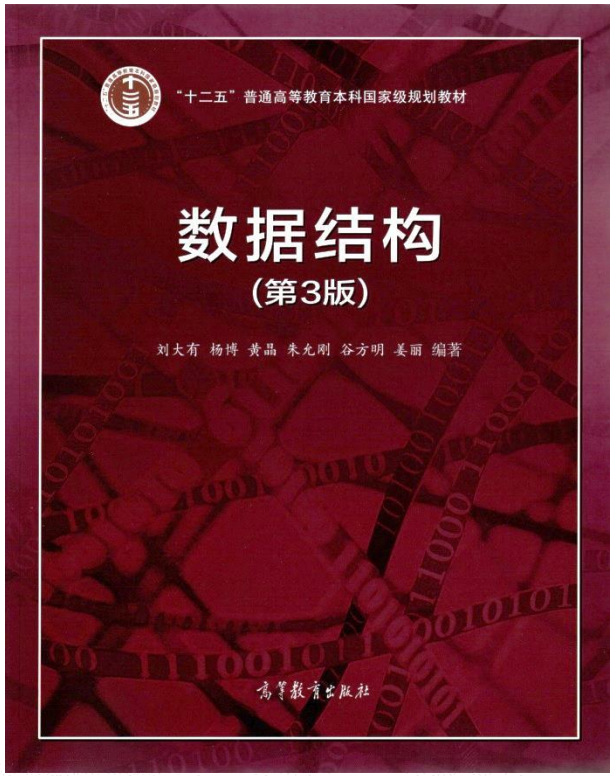
二叉树的中序线索化

```
void Inorder_threading(TreeNode *p){ //中序线索化以p为根的二叉树
    if (p==NULL) return;
    Inorder_threading(p->left); //中序线索化p的左子树
    //如果p没有左孩子,p的左指针是线索域,指向中根前驱pre
    if(p->left==NULL){ p->LThread=1;  p->left=pre; }
    else p->LThread=0;
    //如果pre没有右孩子, pre的右指针是线索域,指向其中根后继p
    if(pre!=NULL && pre->right==NULL){pre->RThread=1;pre->right=p;}
    else if(pre!=NULL) pre->RThread=0;
    pre=p;   //pre指向刚访问完的点
    Inorder_threading(p->right); //中序线索化p的右子树
}
```



算法结束后要调整中序最后一个结点（pre指向）的右线索：

```
pre->RThread=1;
```



线索二叉树

动机和基本概念

中序线索二叉树的基本操作

实现方案

关于先序/后序线索二叉树

数据之法
结构之美
算法之道

zhuyungang@jlu.edu.cn

Last updated on 2023.10

中序线索二叉树——实现方案1

int?
bool?

LThread	*Left*	*Data*	*Right*	*RThread*

- *LThread*和*RThread*理论上为1比特（bit），但是包括C/C++语言在内的众多程序设计语言无法定义1 bit的变量。
- 变通做法：用1字节（8 bit）的变量表示4种标志状态。

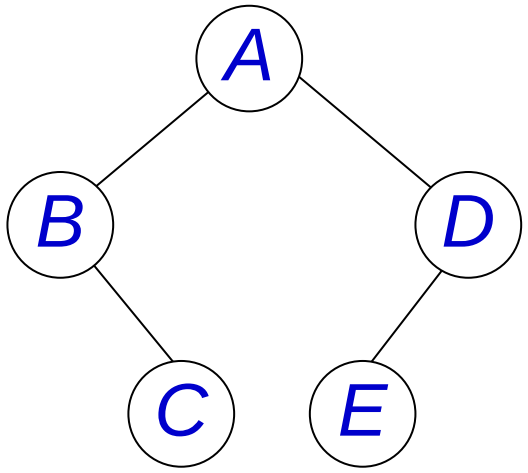
Left	*Data*	*Right*	*Tag*

LThread	*RThread*	*Tag*
0	0	**0**
0	1	**1**
1	0	**2**
1	1	**3**

中序线索二叉树——实现方案2

- 空间换时间，把*LThread* 和 *Rthread*换成指针域，*Pred*指向中根前驱，*Succ*指向中根后继。虽然存储空间增加，但使找中根前驱/后继的最坏时间复杂度由O(*h*)降为O(1)。
- 此种结构亦称为“扩展二叉树”。

Pred	*Left*	*Data*	*Right*	*Succ*

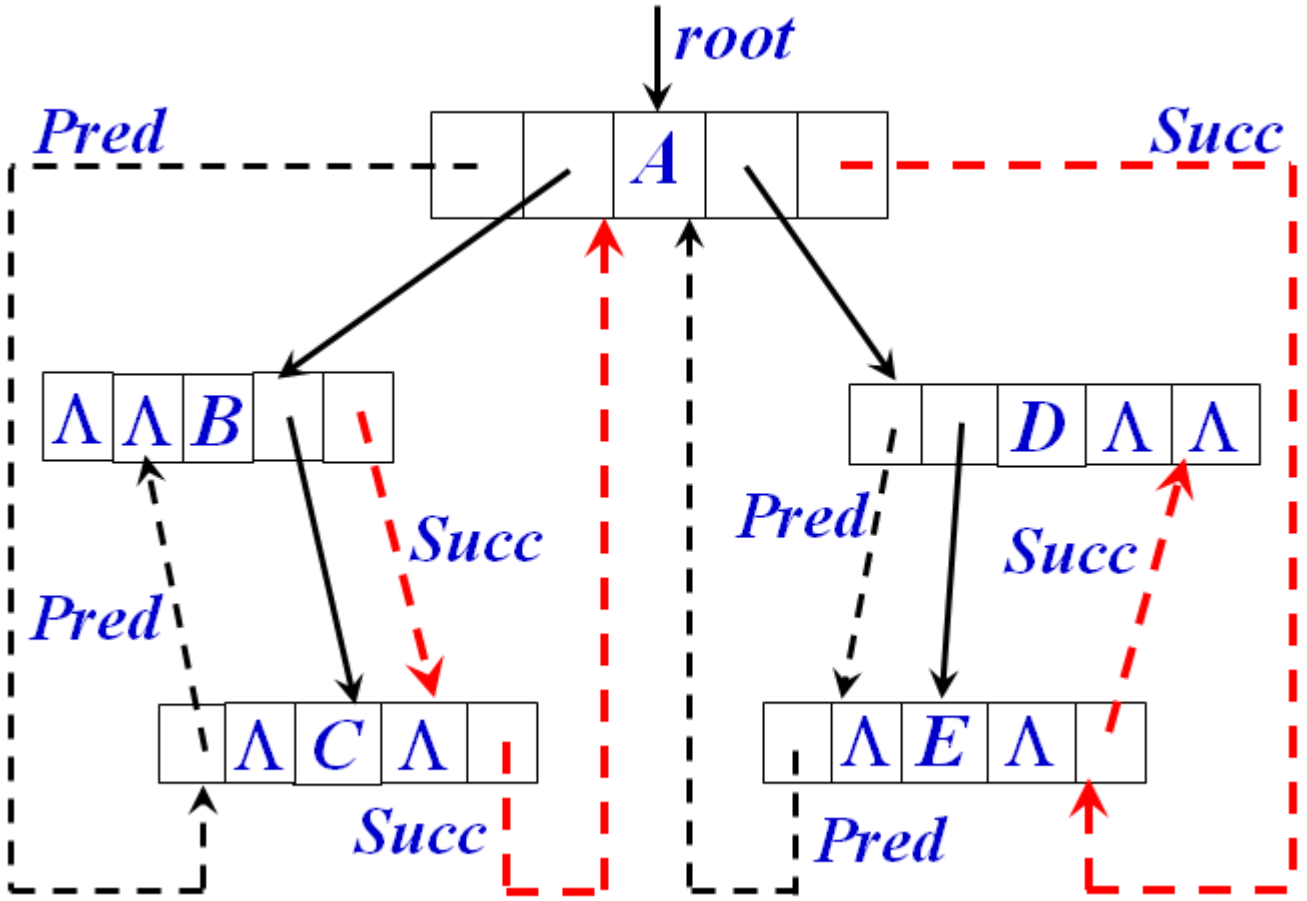


二叉树

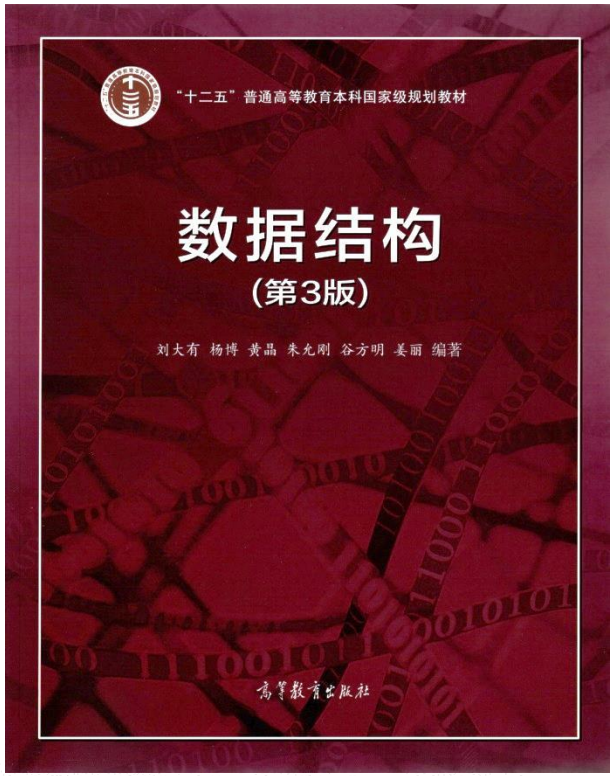
中序扩展二叉树

中根序列：BCAED

Pred	Left	Data	Right	Succ



图中虚线箭头表示线索



线索二叉树

动机和基本概念

中序线索二叉树的基本操作

实现方案

关于先序/后序线索二叉树

数据之法

结构之美

算法之道

zhuyungang@jlu.edu.cn

Last updated on 2023.10

前序/后序线索二叉树

LThread	*Left*	*Data*	*Right*	*RThread*

若结点结构中没有父指针：

✓前序线索二叉树不能解决高效查找结点的先根前驱的问题。

✓后序线索二叉树不能解决高效查找结点的后根后继的问题。