参考资料（感兴趣者自行阅读）

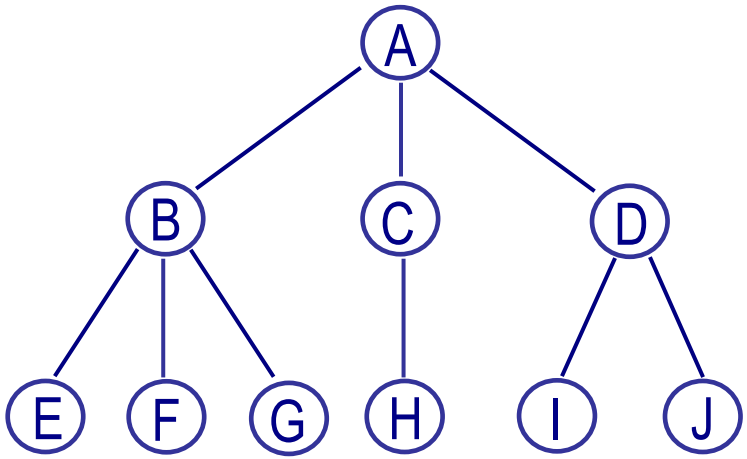
计算机科学中的树

二叉树	二叉树	二叉查找树	笛卡尔树	Top tree
	T树			
自平衡二叉查找树	AA树	AVL树	红黑树	伸展树
	树堆	节点大小平衡树		
B树	B树	B+树	B*树	Bx树
	UB树	2-3树	2-3-4树	(a,b)-树
	Dancing tree	H树		
Trie	前缀树	后缀树	基数树	
空间划分树	四叉树	八叉树	k-d树	vp-树
	R树	R*树	R+树	X树
	M树	线段树	希尔伯特R树	优先R树
非二叉树	Exponential tree	Fusion tree	区间树	PQ tree
	Range tree	SPQR tree	Van Emde Boas tree	
其他类型	堆	散列树	Finger tree	Metric tree
	Cover tree	BK-tree	Doubly-chained tree	iDistance
	Link-cut tree	树状数组		

11 多叉树及其应用

11.1 多叉树的基本概念

每个树节点可以有两个以上的子节点，称为**m阶多叉树**，或称为**m叉树**。

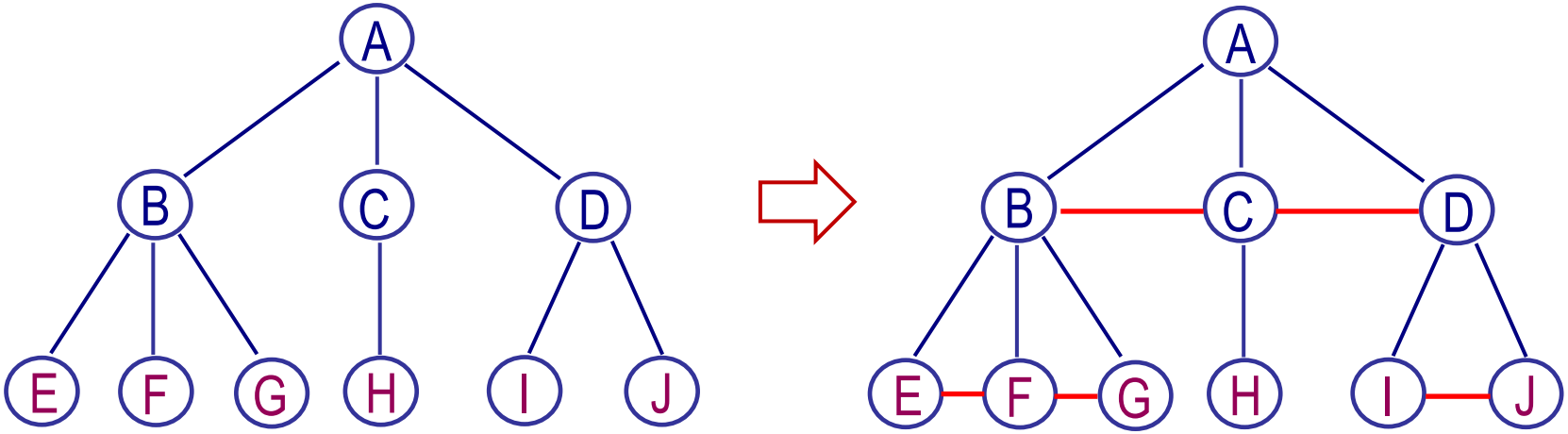


11.2 树、树林与二叉树之间的转换*

1. 树与二叉树的转换*

步骤

(1) 在所有相邻的兄弟结点之间分别加一条连线；

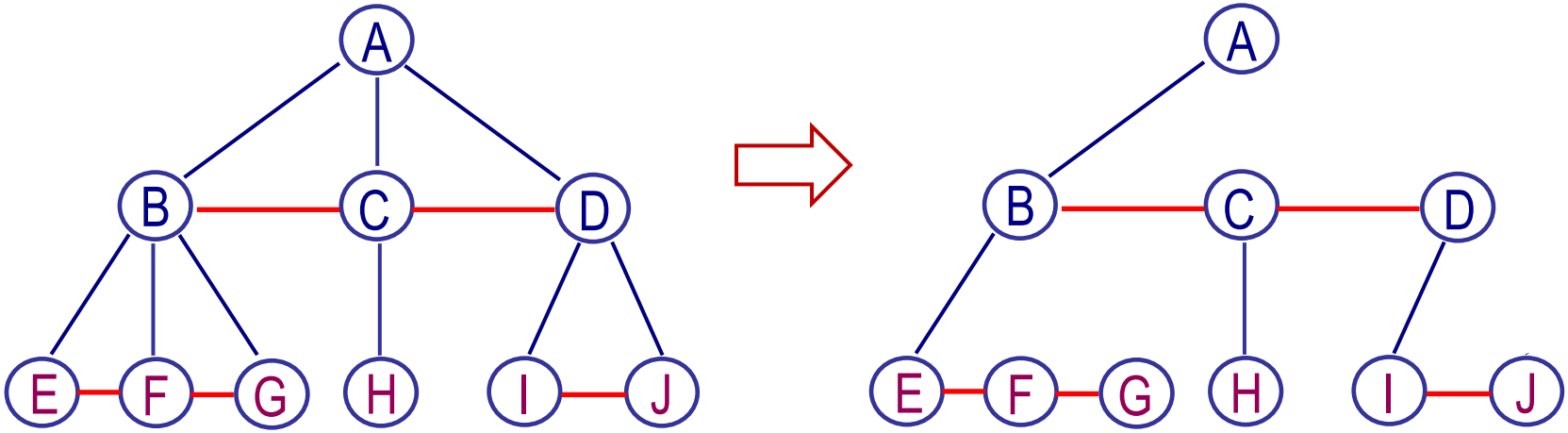


1. 树与二叉树的转换*

步骤

(1) 在所有相邻的兄弟结点之间分别加一条连线；

(2) 对于每一个分支结点，除了其最左孩子外，删除该结点与其他孩子结点之间的连线；



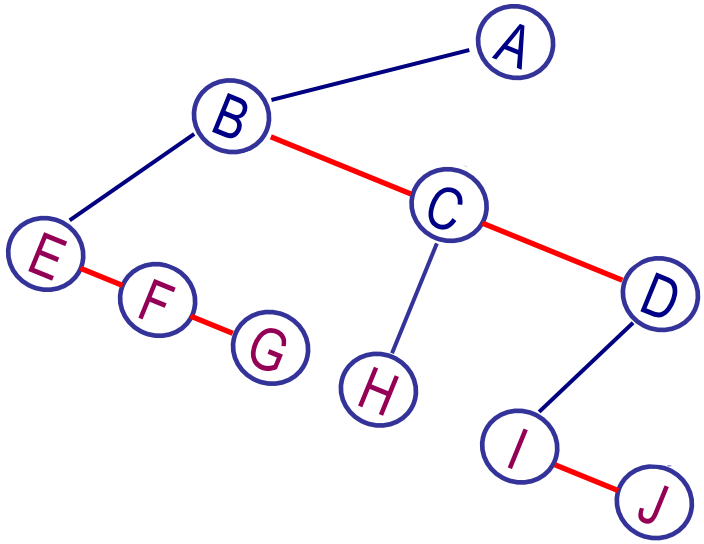
1. 树与二叉树的转换*

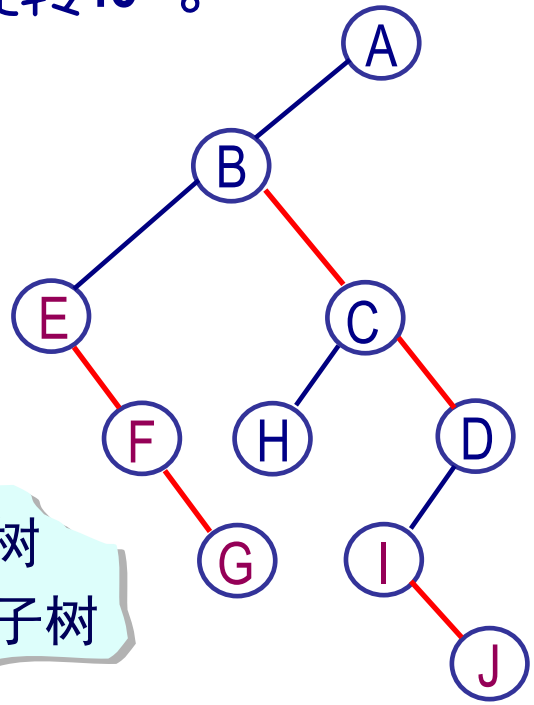
步骤

(1) 在所有相邻的兄弟结点之间分别加一条连线；

(2) 对于每一个分支结点，除了其最左孩子外，删除该结点与其他孩子结点之间的连线；

(3) 以根结点为轴心，顺时针旋转45°。

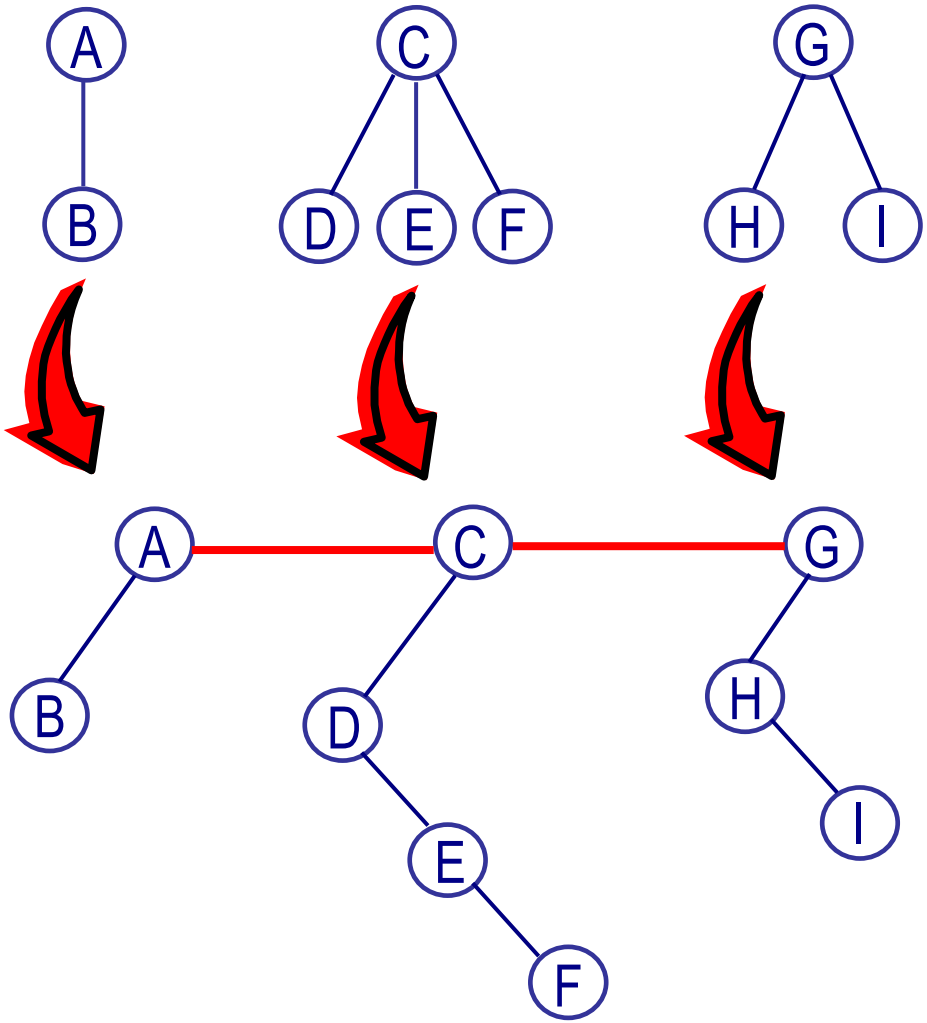




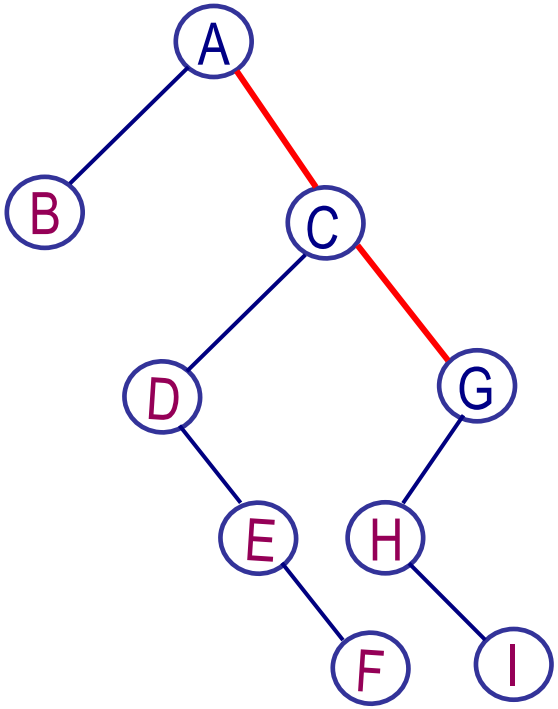
特点

根结点无右子树
原叶结点无左子树

2. 森林与二叉树的转换*

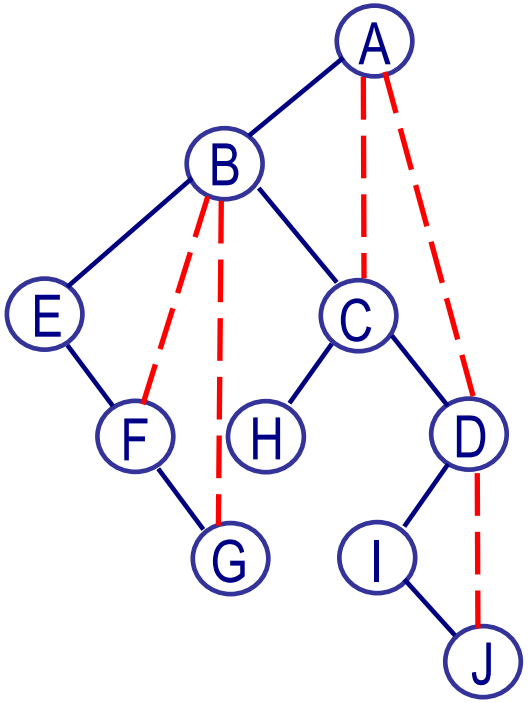


分别将每一棵树转换为二叉树



转换后的二叉树

3. 二叉树还原为树*



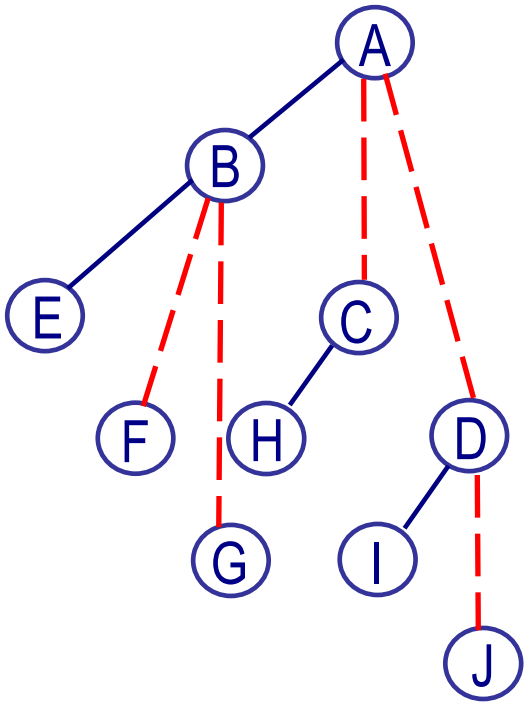
步骤

前提

由一棵树转换而来

1. **若某结点是其双亲结点的左孩子，则将该结点的右孩子以及当且仅当连续地沿此右孩子的右子树方向的所有结点都分别与该结点的双亲结点用一根虚线连接；**

3. 二叉树还原为树*



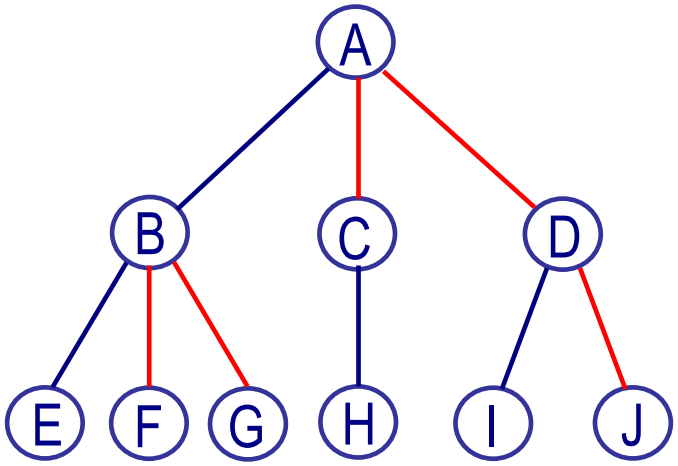
步骤

前提

由一棵树转换而来

1. **若某结点是其双亲结点的左孩子，则将该结点的右孩子以及当且仅当连续地沿此右孩子的右子树方向的所有结点都分别与该结点的双亲结点用一根虚线连接；**
2. **去掉二叉树中所有双亲结点与其右孩子的连线；**

3. 二叉树还原为树*



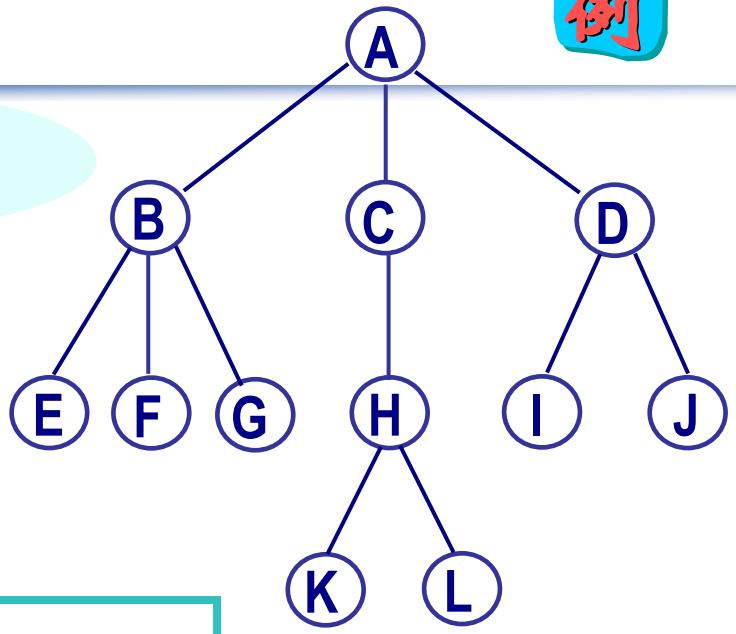
步骤

前提

由一棵树转换而来

1. **若某结点是其双亲结点的左孩子，则将该结点的右孩子以及当且仅当连续地沿此右孩子的右子树方向的所有结点都分别与该结点的双亲结点用一根虚线连接；**
2. **去掉二叉树中所有双亲结点与其右孩子的连线；**
3. **规整图形(即使各结点按照层次排列),并将虚线改成实线。**

11.3 多叉树的遍历



(1) 前序遍历

类似转换后的二叉树的前序遍历

若被遍历的树非空,则

① 访问根结点;

② 依次按前序遍历方式遍历根结点每一棵子树。

前序遍历序列：A B E F G C H K L D I J

(2) 后序遍历

类似转换后的二叉树的中序遍历

若被遍历的树非空,则

① 依次按后序遍历方式遍历根结点每一棵子树;

② 访问根结点。

后序遍历序列：E F G B K L H C I J D A

深度优先遍历算法

```
void  DFStree(TNodeptr t){
    int i;
    if(t!=NULL){
        VISIT(t);      /* 访问t指向结点 */
        for(i=0;i<MAXD; i++)
            if(t->next[i] != NULL)
                DFStree(t->next[i]);
    }
}
```

```
#define MAXD  3   //树的度

struct  node{
        Datatype   data;
        struct  node   *next[MAXD];
 };
typedef struct node  TNode;
typedef struct node  *TNodeptr;
```

广度优先遍历算法

```
void  BFStree(TNodeptr t){
    TNodeptr p;  int i;
    if(t!=NULL){
        enQueue(t);
        while(!isEmpty()){
            p= deQueue();      VISIT(p);
            for(i=0; i<MAXD; i++) //依次访问p指向的子结点
                if( p->next[i] != NULL)    enQueue(p);
        }
    }
}
```

11.4 多叉树的主要应用

多叉树通常用于大数据的快速检索和信息更新。典型多叉树：

- B-树，B+树
- trie树

12 B-树——多路(平衡)查找树

B-树(B-Tree)由R.Bayer和E.MacCreight于1970年提出，是一种平衡的多路树。为什么叫B-树，有人认为是由“平衡(Balanced)”而来，而更多认为是因为他们是在Boeing科学研究实验发明的此概念并以此命名的。

B-树针对大块数据的读写操作做了优化。

B-树多用于文件系统或数据库系统常用的索引结构。

注：网络上经常混用B树，B-树，B_树。其中“B_”是没有的，“B减树”的叫法是错误的。

12 B-树——多路(平衡)查找树

二叉查找树的问题：

- 二叉查找树：当数据规模比较大时，二叉查找树深度高（查找树的深度主要由数据规模决定），查找效率随之降低；极端情况会退化成线性表，查找效率更差。
- AVL树降低了树的高度，一定程度上提高了查找效率，但是由于数据规模大，树的深度任然较高。
- 当数据存储在外存时并且以二叉树/AVL树管理数据，由于深度过大，造成磁盘IO次数多，访问效率显著降低。

12.1 B-树的定义

B-树是平衡二叉查找树的泛化!

一个m阶的B-树为满足下列条件的m叉树:

(1) 每个分支结点最多有m棵子树;

(2) 除根结点外，每个分支结点最少有$\lceil m/2 \rceil$棵子树;

(3) 根结点最少有两棵子树(除非根为叶结点);

(4) 所有“叶结点”都在同一层上;

降低树的高度

(5) 每个节点都存有关键字索引和数据，包含下列信息:

$$n,\ p_0,\ key_1,\ p_1,\ key_2,\ p_2,\ \ldots,\ key_n,\ p_n$$

关键字大小关系……

其中,n为结点中关键字值的个数, $n \le m-1$

最多m−1个关键字

key_i为关键字，且满足 $key_i < key_{i+1}$, $1 \le i < n$

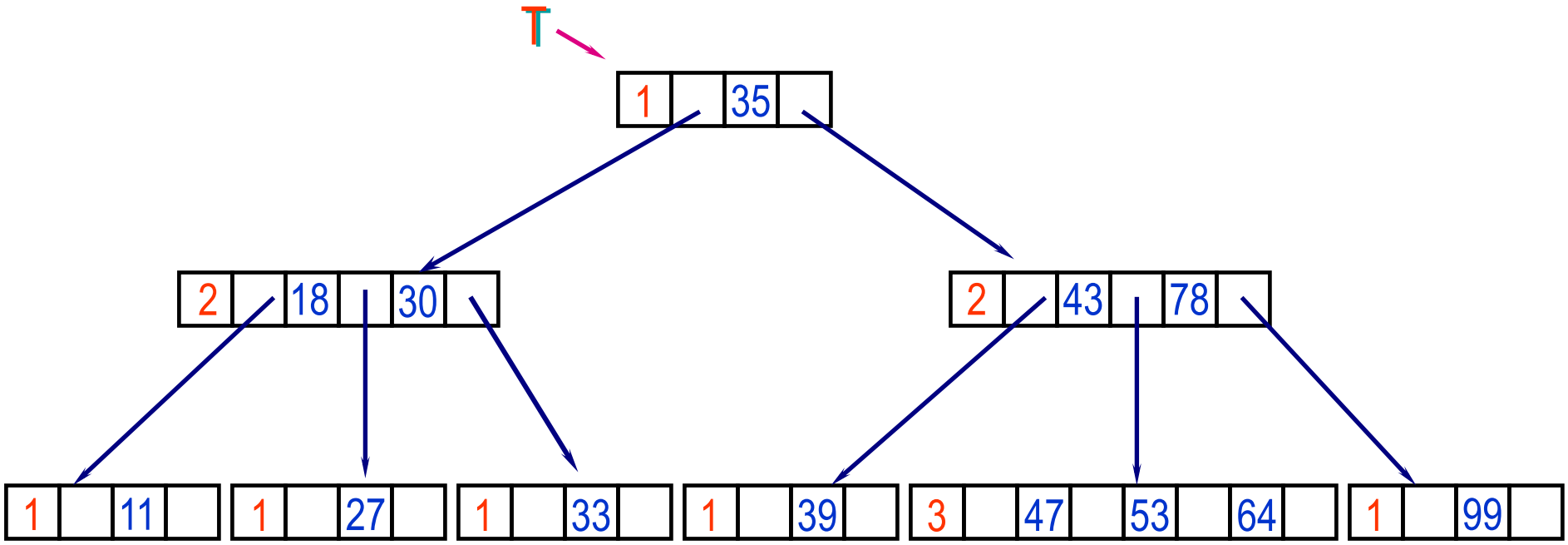
p_i为指向该结点的第i+1棵子树根结点的指针.

注：不同文献对B-树的定义可能有些微的差异

一棵4阶B-树

也称为2-3-4 树

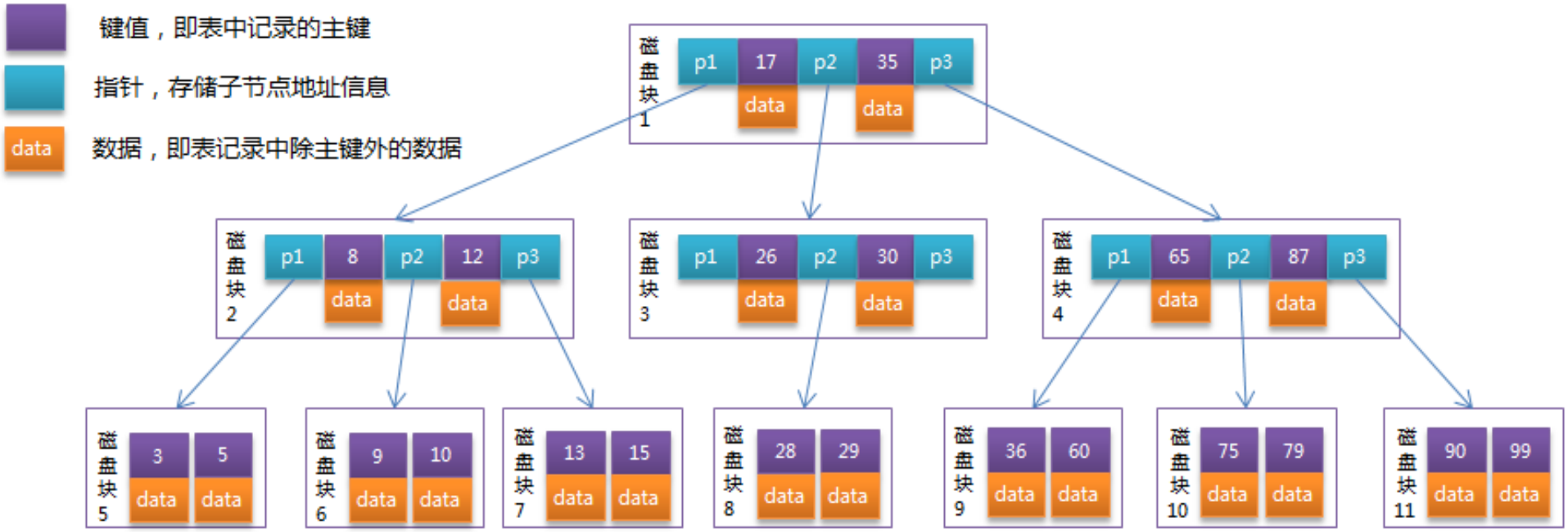
- 每个分支结点最多有4棵子树(即最多有3个关键字值)
- 每个分支结点最少有2棵子树
- 根结点最少有2棵子树
- 所有“叶结点”都在同一层上



B-树用于管理外存数据*

B-Tree是为磁盘等外存储设备设计的一种多路平衡查找树

文件系统从磁盘读取数据到内存时是以磁盘块（block）为基本单位，大小一般为4K, 8K或16K

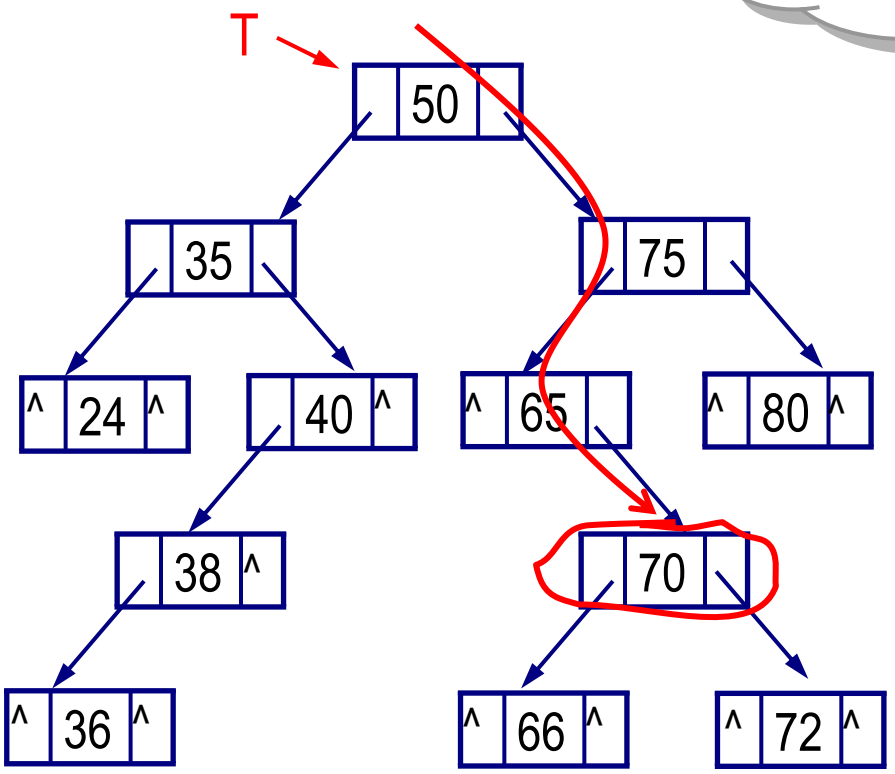


图片来源：BTree和B+Tree详解_b+tree 结构_-CSDN博客
(https://blog.csdn.net/yin767833376/article/details/81511377)

12.2 B-树的查找

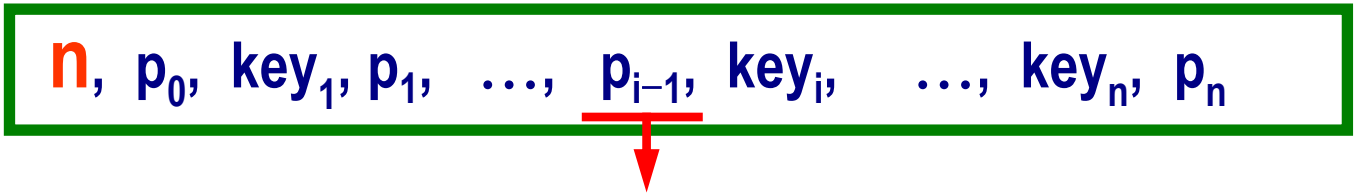
查找 key=70

类似于二叉排序树的查找

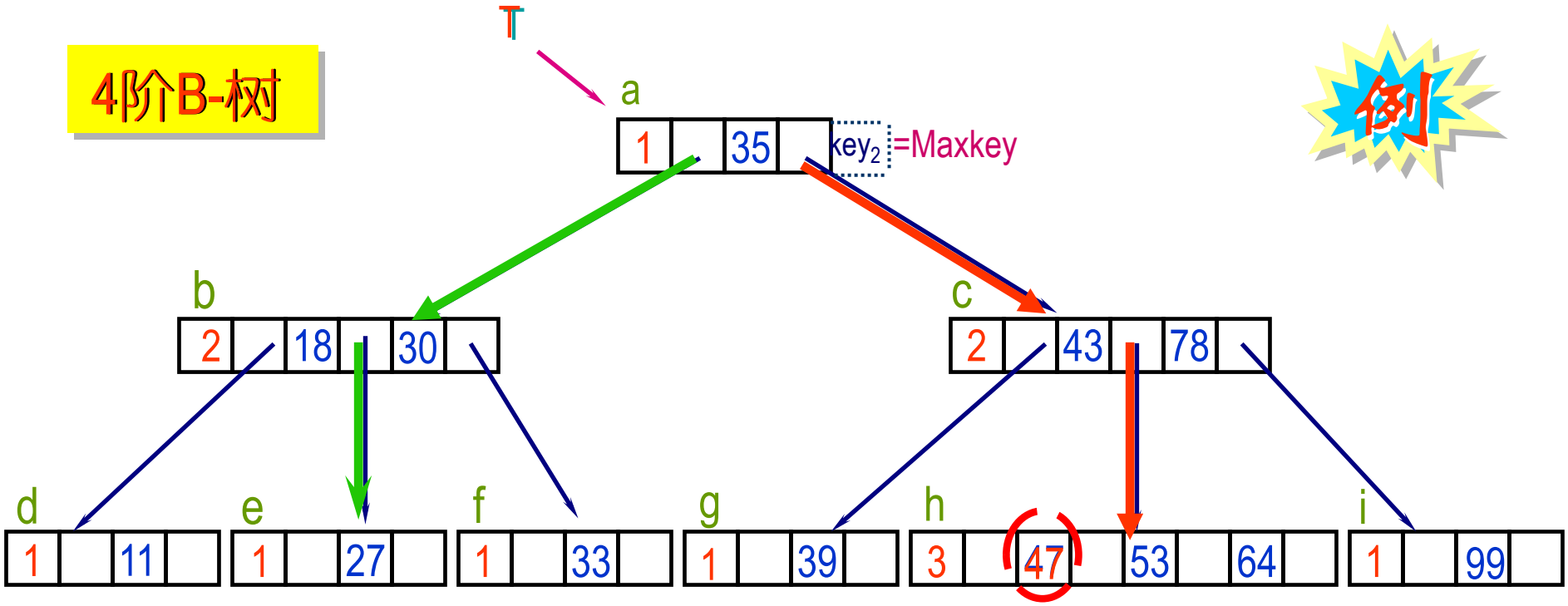


首先将给定的关键字k在B-树根结点的关键字集合中采用顺序查找法 或者折半查找法 进行查找，

- 若$k=key_i$, 则查找成功，根据相应的指针取得记录.否则，
- 若$k<key_i$,则在指针p_{i-1}所指的结点中重复上述查找过程,直到查找成功，或者有$p_{i-1}=NULL$，查找失败。



4阶B-树



例

例如，查找关键字值k=47　　查找成功 ！

例如，查找关键字值k=23　　查找失败 ！

原则

(1) $k=key_i$　查找成功

(2) $k<key_i$　在p_{i-1}所指的结点中查找

类型定义

```
#define M     1000
typedef struct node {
      int  keynum;
      keytype  key[M+1];        //多一个Key用来记录Maxkey
      struct node  *ptr[M+1];
      rectype  *recptr[M+1];    //指向数据记录
} BTNode;
```

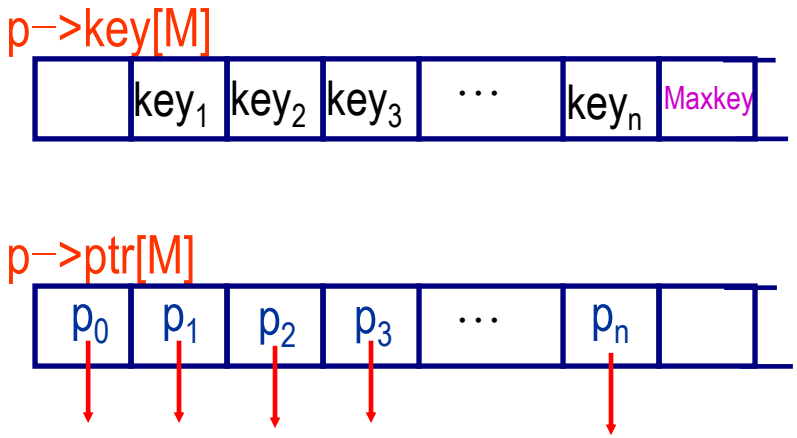
```
keytype  searchBTree(BTNode *t,keytype k){
    int i,n;
    BTNode *p=t;

    while(p!=NULL){
        n=p->keynum;
        p->key[n+1]=Maxkey;

        i=1;
        while(k>p->key[i])
            i++;

        if(p->key[i]==k)
            return  p->key[i];
        else
            p=p->ptr[i-1];
    }
    return  -1;
}
```

在p指结点的关键字集合中查找k

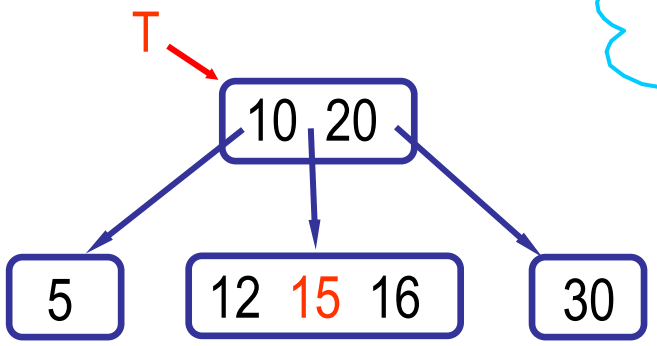


12.3 B-树的插入

B-树的生成从空树开始，即逐个在叶结点中插入结点(关键字)而得到

一棵3阶B-树

插入 k=15



一棵m阶B-树的结点中最多有m-1个关键字值

结点分裂

基本思想

若将k插入到某结点后使得该结点中关键字值数目超过m-1时，则要以该结点**位置居中**的那个关键字值为界将该结点**一分为二**，产生一个新结点，并把位置居中的那个关键字值插入到双亲结点中；

如双亲结点也出现上述情况，则需要再次进行分裂. 最坏情况下，需要一直分裂到根结点，以致于使得B-树的深度加1.

一般情况下

若某结点已有m−1个关键字值，在该结点中插入一个新的关键字值，使得该结点内容为

q

m	key_1	key_2	key_3	…	key_i	key_{i+1}	…	key_{m-1}	key_m

则需要将该结点分解为两个结点q与q′，即

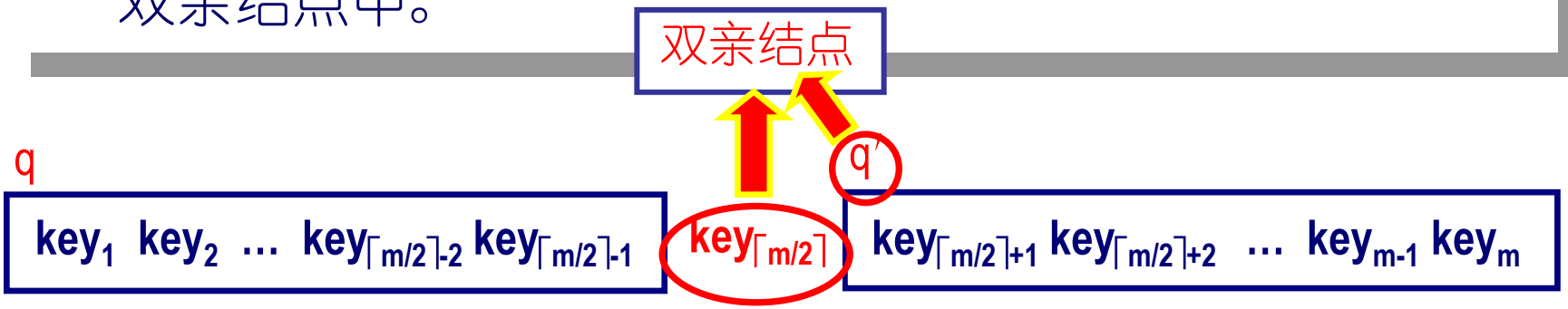
q

$\lceil m/2 \rceil - 1$	key_1	key_2	…	$key_{\lceil m/2 \rceil - 2}$	$key_{\lceil m/2 \rceil - 1}$

q′

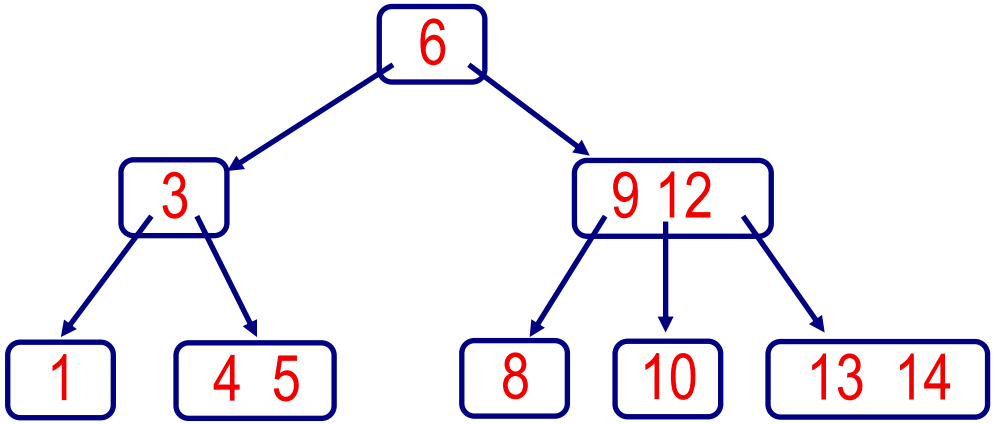
$m - \lceil m/2 \rceil$	$key_{\lceil m/2 \rceil + 1}$	$key_{\lceil m/2 \rceil + 2}$	…	key_{m-1}	key_m

并且将关键字值$key_{\lceil m/2 \rceil}$与一个指向q′的指针插入到q的双亲结点中。



练习

请画出依次插入关键字序列(5, 6, 9, 13, 8, 1, 12, 14, 10, 4, 3)中各关键字值以后的4阶B-树。



原则

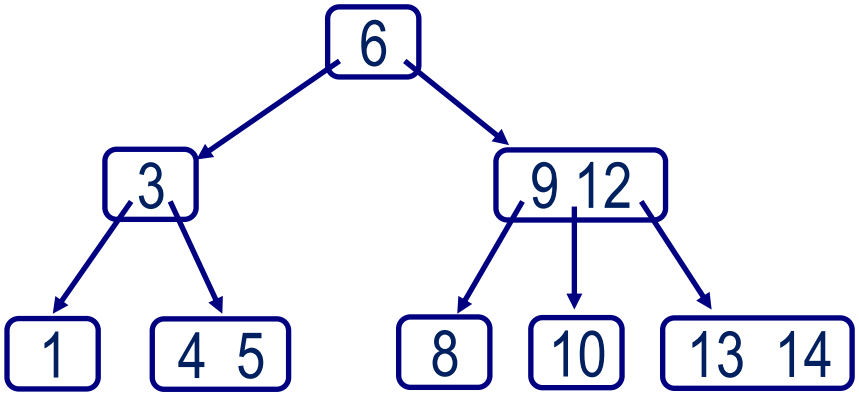
1. 4阶B-树的每个分支结点中关键字个数不能超过3；
2. 生成B-树从空树开始，逐个插入关键字而得到的；
3. 每次在最下面一层的某个分支结点中添加一个关键字;若添加后该分支结点中关键字个数不超过3,则本次插入成功，否则，进行结点分裂。

B-树与平衡二叉树

- **B-树是平衡二叉树的泛化。**当数据规模大需要存储在外存时，平衡二叉树由于深度过大，造成磁盘IO次数多，效率低；而B树通过降低树的高度，可以减少IO次数
- 设计上可以让**B-树结点的大小**和磁盘扇区/块的大小相适应，一次IO读取整个块，提高访问效率
- B-树所有叶子结点在同一层
- B-树节点的关键字数量在一定范围之内，因此不需要象平衡二叉树那样经常的重新平衡
- 插入结点时，B-树通过分裂保持树的平衡，而平衡二叉树通过和旋转(4种情况)保持平衡

B-树的不足

- Deletion process is complicated.
  - Eg., delete 3 or 9



- The search process may take a longer time because all the keys are not obtainable at the leaf.
  - Eg., search keys in [4, 9]

13 B+树

一个m阶的B+树为满足下列条件的m叉树：

同B-树

(1) 每个分支结点最多有m 棵子树；

(2) 除根结点外，每个分支结点最少有$\lceil m/2 \rceil$棵子树；

(3) 根结点最少有2棵子树(除非根为叶结点结点,此时B+树只有一个结点)；

结点内容(和功能)有不同

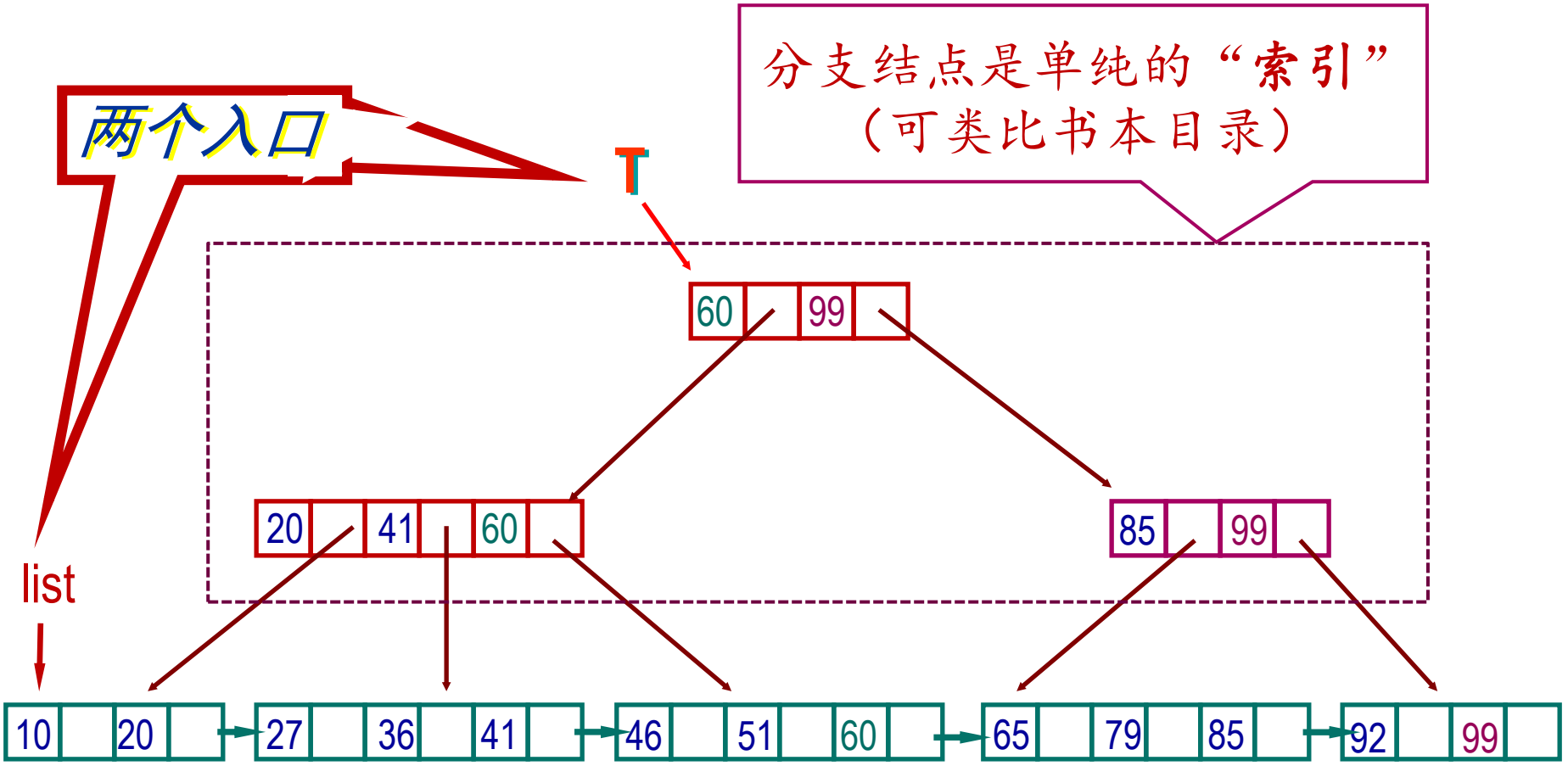
(4) 叶结点中存放记录的关键字以及指向记录的指针,或数据分块后每块的最大关键字值及指向该块的指针，并且叶结点按关键字值的大小顺序链接成线性链表。

(5) 所有分支结点可以看成是索引的索引，结点中仅包含它的各个孩子结点中最大(或最小)关键字值和指向孩子结点的指针。

(6) 结点中有 n 个关键字：

key_1	p_1	key_2	p_2	……	key_n	p_n

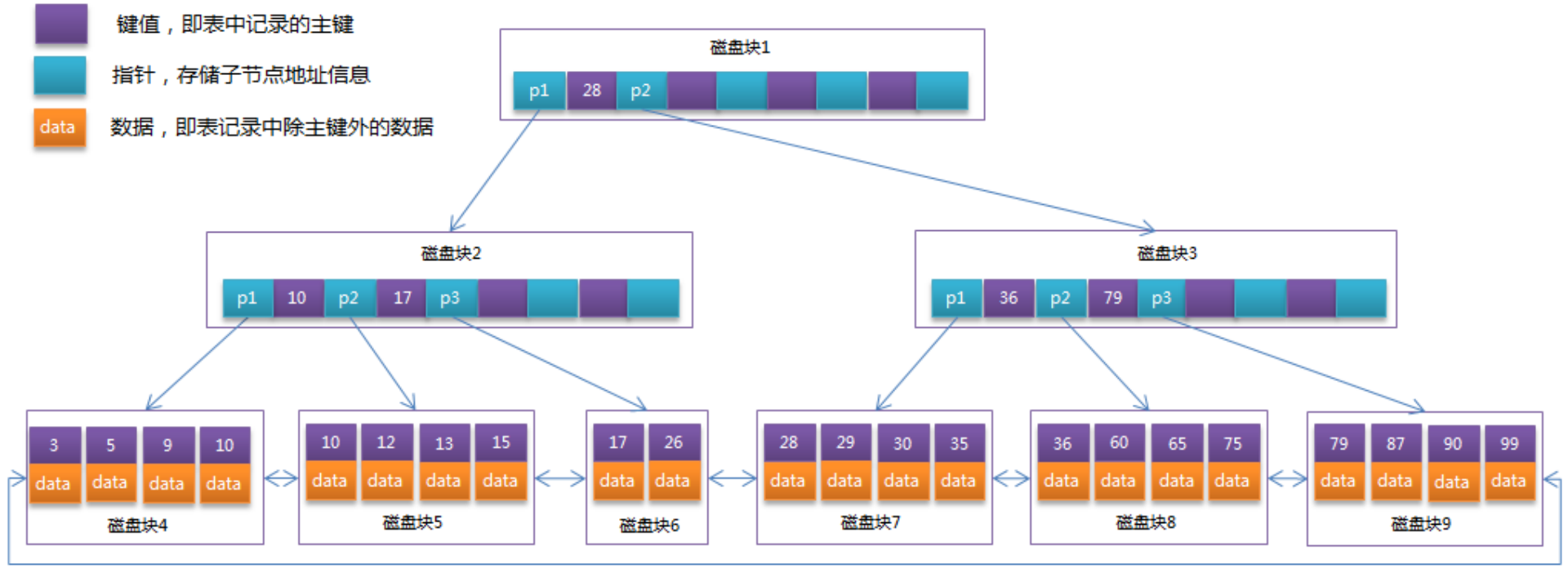
一棵3阶B+树



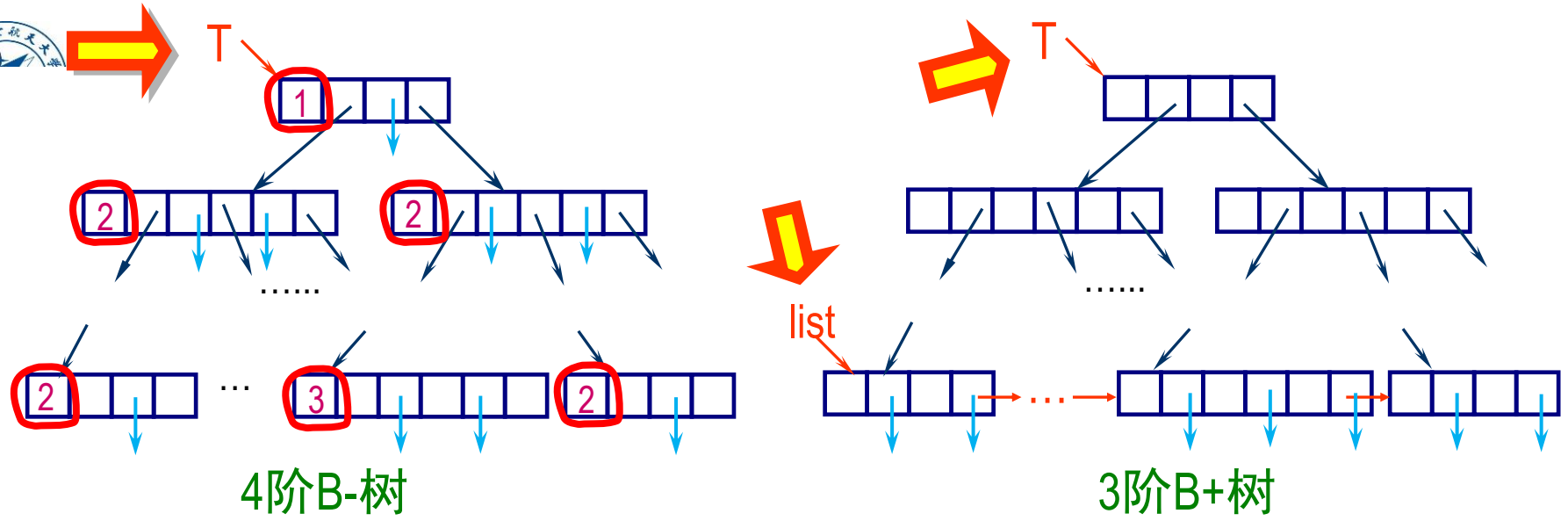
只有叶结点包含数据记录或指向相应记录的指针

B+树用于磁盘数据管理*

文件系统从磁盘读取数据到内存时是以磁盘块（block）为基本单位，大小一般为4K, 8K或16K



图片来源：BTree和B+Tree详解_b+tree 结构_-CSDN博客
(https://blog.csdn.net/yin767833376/article/details/81511377)



-分支可以存更多Key;
-删除发生在叶子结点

B-树与B+树的区别（从结构上看）

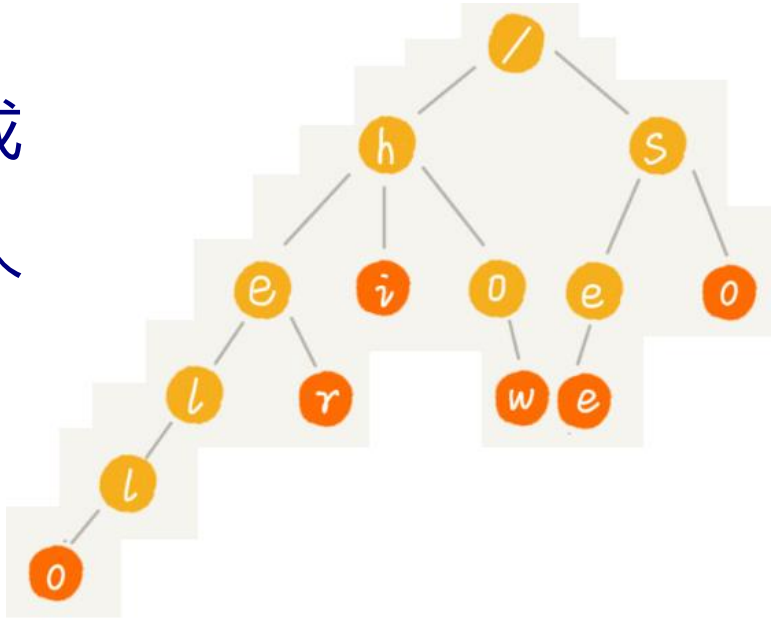
1. B-树的每个分支结点中含有该结点中关键字值的个数,B+树没有;
2. B-树的每个分支结点中含有指向关键字值对应记录的指针,而B+树只有叶结点有指向关键字值对应记录的指针;
3. B-树只有一个指向根结点的入口，而B+树的叶结点被链接成为一个不等长的链表，因此，B+树有两个入口，一个指向根结点，另一个指向最左边的叶结点(即最小关键字所在的叶结点)。

容易实现高效的范围查询(Range Query)

14 Trie树及查找

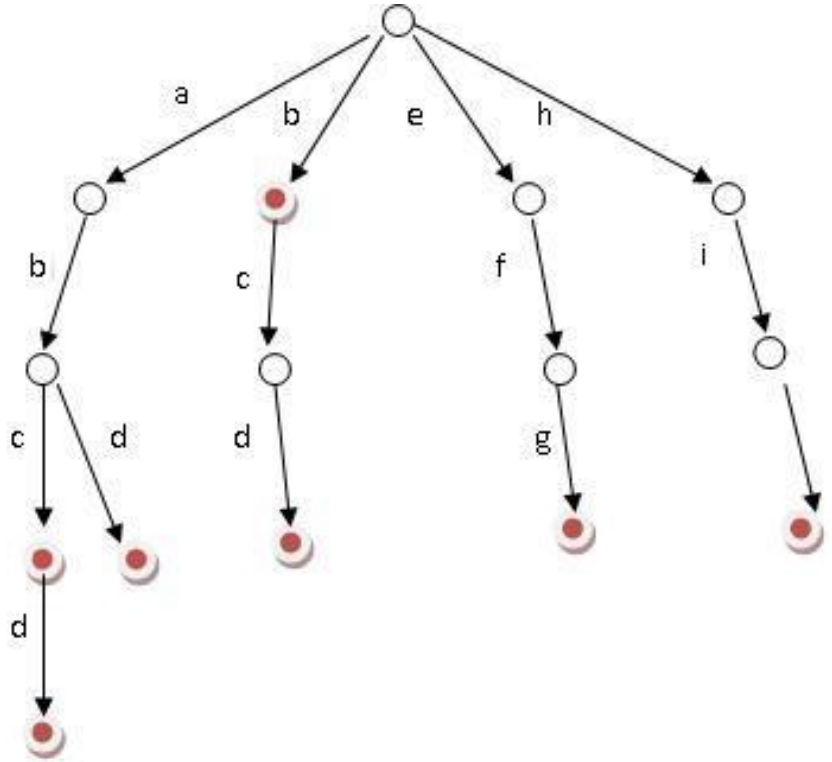
- 在树的遍历中通常是通过比较**整个键值**来进行的，即每个节点包含一个键值，该键值与要查找的键值进行比较来在树中寻找正确的路径。而用**键值的一部分**来确定查找路径的树称为**trie**树（它来源于re**trie**val，*为了在发音上区别tree，可读作try*）

- Trie结构典型应用“字典树”：
  英文单词仅由26个字母组成（不考虑大小写）
  - 字典树每个内部结点都有26个子结点 – 多叉树
  - 树的高度为最长单词长度



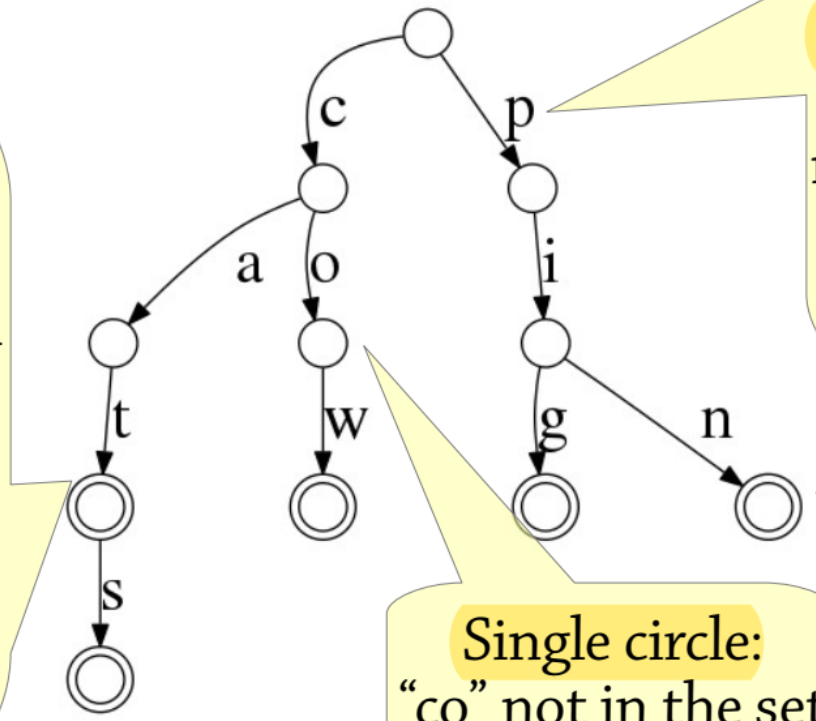
Trie结构的适用情况

- Trie结构主要基于两个原则：
  - 键值由固定的字符序列组成（如数字或字母），如Huffman码(只由0,1组成)、英文单词（只由26个字母组成）；
  - 对应结点的分层标记；
- 主要应用
  - 信息检索（information retrieval）
  - 用来存储英文字符串，特别是大规模的英文词典（如词频统计、拼写检查）
  - 字符串排序



Trie树扩展知识及示例*

This trie represents the set {“cat”, “cats”, “cow”, “pig”, “pin”}:



Edges labelled with characters.
Invariant: no node has two edges with the same label

Double circle: cat is in the set (Concatenate all the characters on the path from the root to this node – c-a-t)

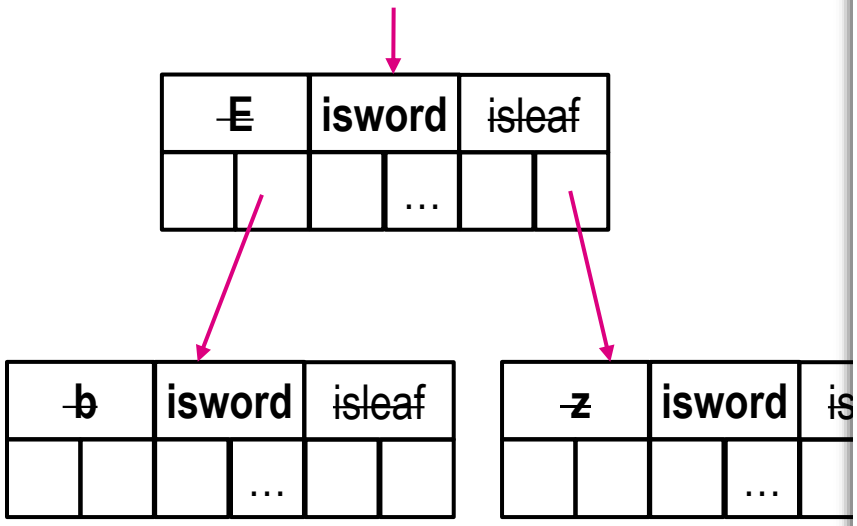
Single circle: “co” not in the set

Invariant: all leaves are “double circled”

Trie结构构造示例*

一种字典树结构定义（参考）

```
struct tnode {  // word tree
    char ch;       // the character
    char isword;  // is or not a word
    char isleaf;  // is or not a leaf node
    struct tnode *ptr[26];
};
```



基于trie结构的单词树的构造

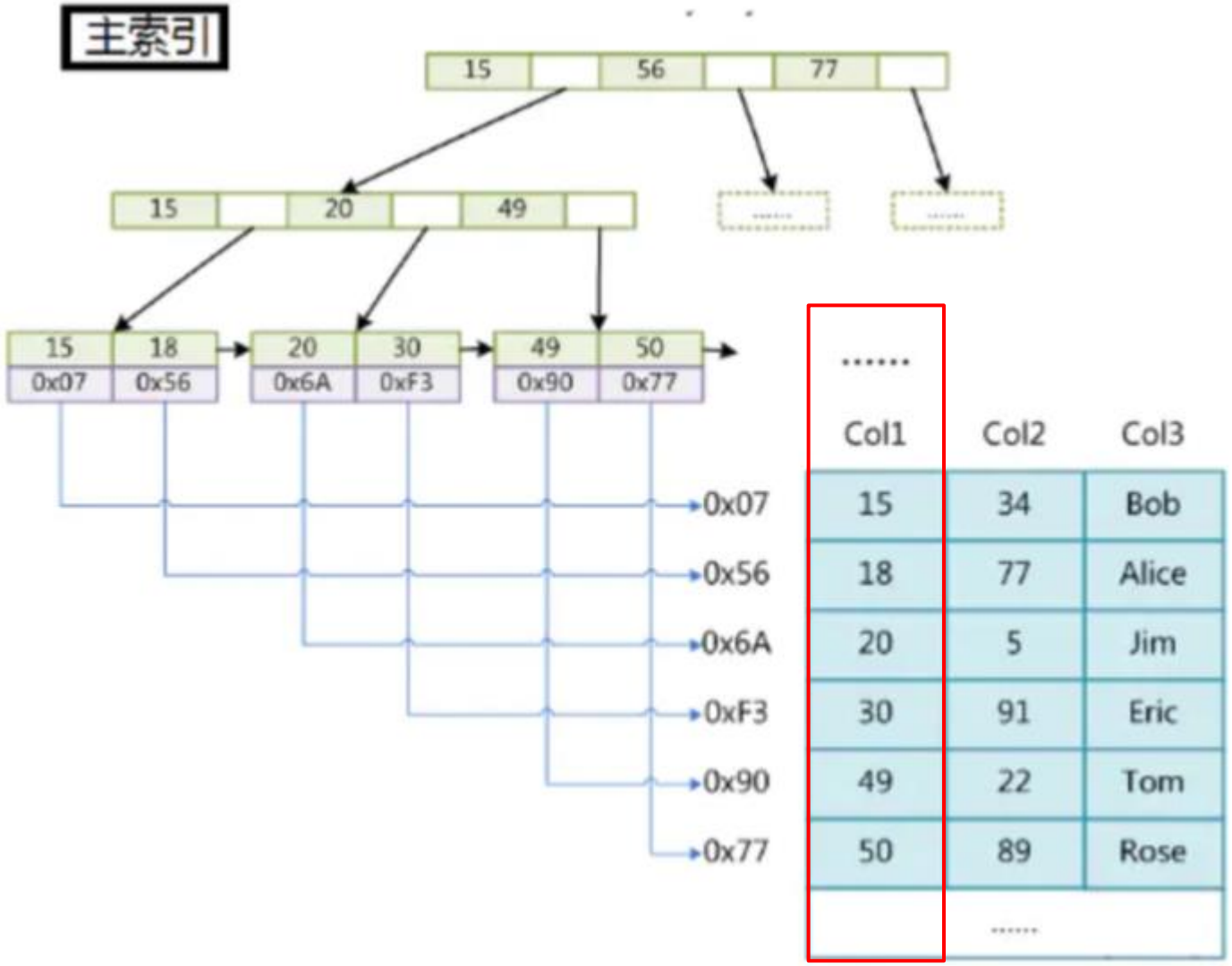
```
/* install w at or below p */
void wordTree(struct tnode *root,char *w) {
    struct tnode *p;
    for(p=root; *w != '\0'; w++){
        if(p->ptr[*w-'a'] == NULL) {
            p->ptr[*w-'a'] = talloc();
            p->isleaf = 0;
        }
        p = p->ptr[*w-'a'];
    }
    p->isword = 1;
}
struct tnode *talloc(){
   int i;
    struct tnode *p;
    p = (struct tnode *)malloc(sizeof(struct tnode));
    isword = 0; isleaf = 1;
    for(i=0; i<26; i++)
        ptr[i] = NULL;
    return p;
}
```

Trie结构性能分析*

- 对英文单词来说，Trie树的高度取决于最长的单词长度。绝大多数常用单词通常都不是很长，一般访问几个节点（很可能是5~7个）就可以解决问题。
- 而采用（最理想的）平衡二叉查找树，假设有10000个单词，则树的高度为14（lg10000）。由于大多数的单词都存储在树的最低层，因此平均查找单词需要访问13个节点，是trie树的两倍。
- 在二叉搜索树(BST)中，查找过程需要比较整个单词，而在trie结构中，每次比较只需要比较一个字母。
- 因此，**在访问速度要求很高的系统中，如拼写检查、词频统计中，trie结构是一个非常好的选择。**

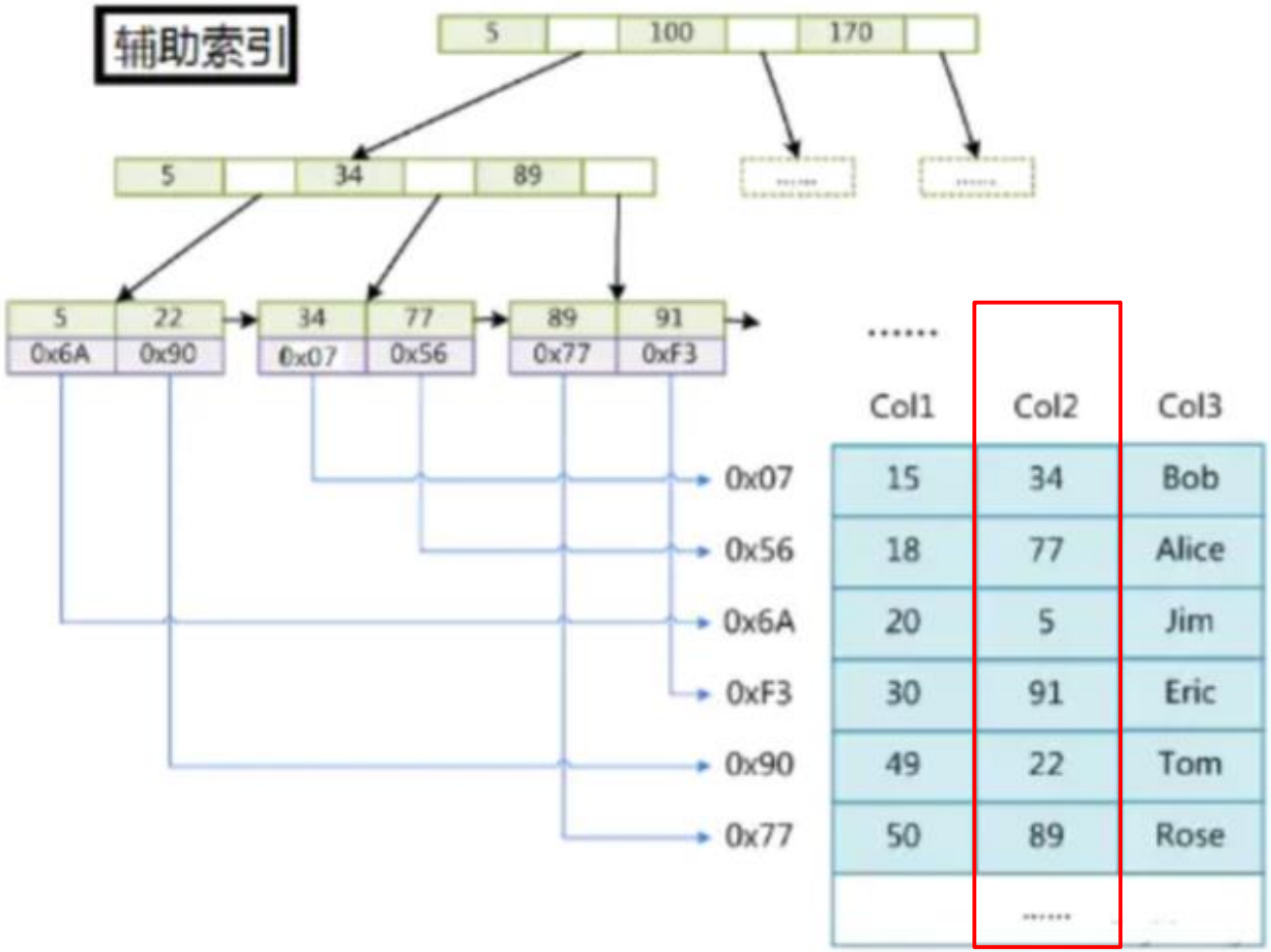
本章结束！

B+树的应用*： MyISAM数据管理系统，分别存储数据文件和索引文件



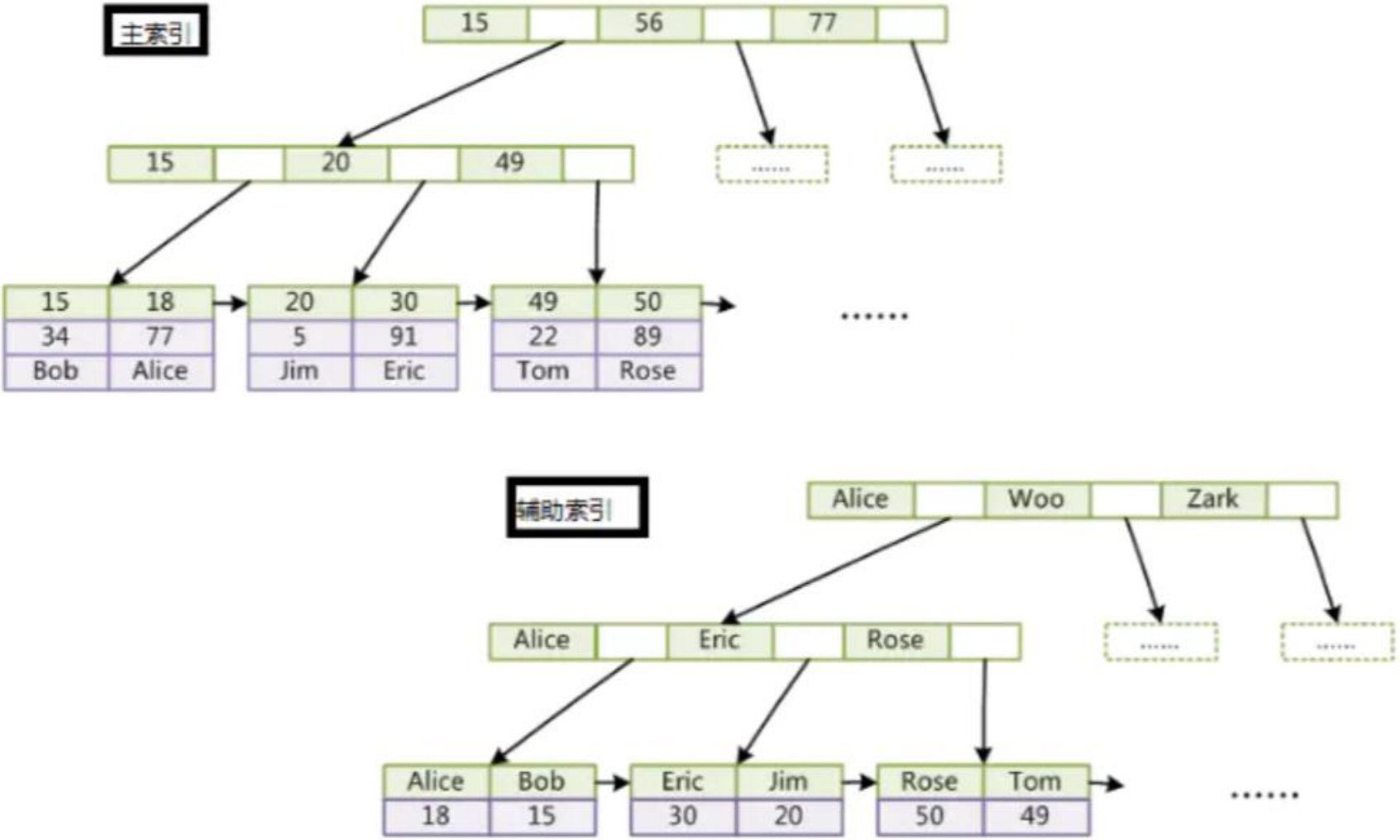
图片来源：B树、B+树原理及其应用. https://www.jianshu.com/p/8d03c3677472

B+树的应用*： MyISAM是MySQL的默认数据库引擎（5.5版之前），它分别存储数据文件和索引文件



图片来源：B树、B+树原理及其应用. https://www.jianshu.com/p/8d03c3677472

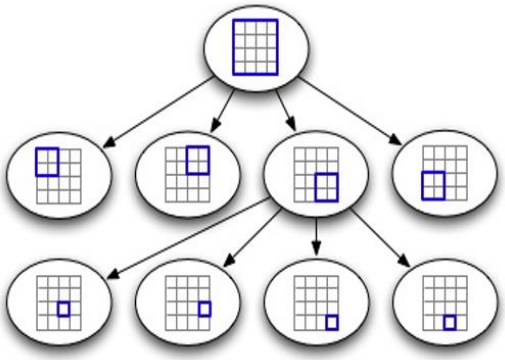
B+树的应用*： 在InnoDB中，数据按B+Tree组织，主索引叶节点数据域保存了完整的数据记录



图片来源：B树、B+树原理及其应用. https://www.jianshu.com/p/8d03c3677472

扩展知识*：空间划分树

- ***k*-d树**：每个节点都为k维点的二叉树。所有非叶子节点可以视作用一个超平面把空间分割成两个半空间。
- **四叉树和八叉树**：针对二（三）维空间，在每一个节点上有四（八）个子区块
- **R树(R-Tree，空间索引树)**：是B树向多维空间发展的另一种形式，也是平衡树。它将对象空间按范围划分，每个结点都对应一个区域和一个磁盘页，非叶结点的磁盘页中存储其所有子结点的区域范围，非叶结点的 所有子结点的区域都落在它的区域范围之内；叶结点的磁盘页中存储其区域范围之内的所有空间对象的外接矩形。R树的扩展有R+, R*, QR, RT树等



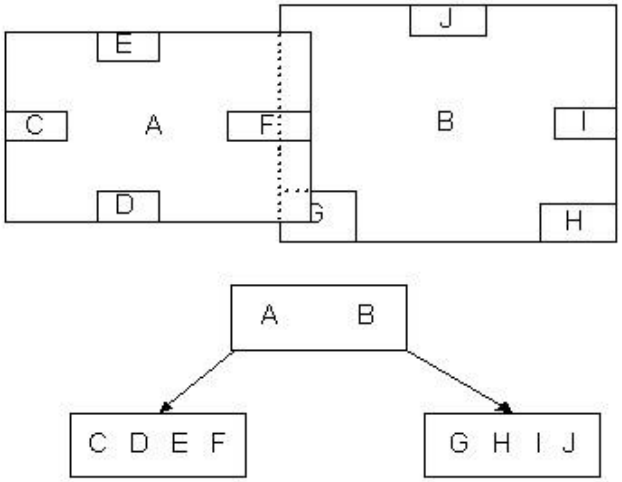
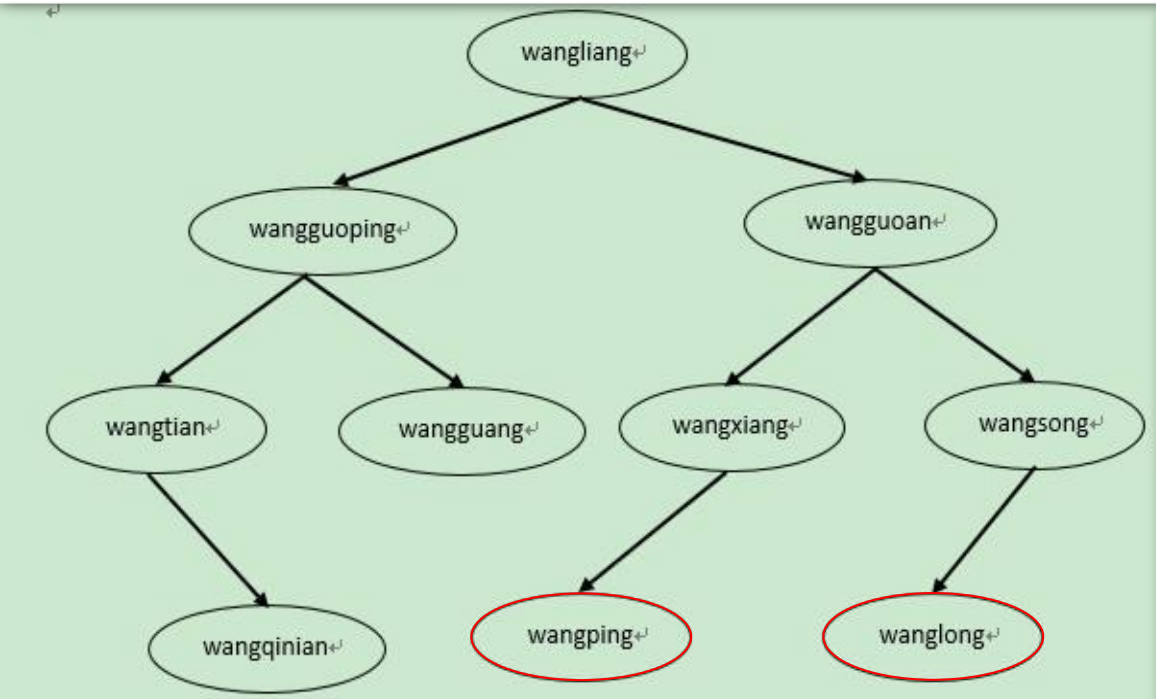


图1：一个R树实例

问题5.3*-查家谱（2016级期末考试）

同姓氏中国人见面常说的一句话是“我们五百年前可能是一家”。从当前目录下的文件in.txt中读入一家谱，从标准输入读入两个人的名字（两人的名字肯定会在家谱中出现），编程查找判断这两个人相差几辈，若同辈，还要查找两个人共同的最近祖先以及与他（她）们的关系远近。假设输入的家谱中每人最多有两个孩子，例如下图是根据输入形成的一个简单家谱。



若要查找的两个人是wangping和wanglong，可以看出两人共同的最近祖先是wangguoan，和两人相差两辈。
若要查找的两个人是wangqinian和wangguoan，从家谱中可以看出两人相差两辈；

【输入示例】
假设家谱文件中内容为：
6
wangliang wangguoping wangguoan
wangguoping wangtian wangguang
wangguoan wangxiang wangsong
wangtian wangqinian NULL
wangxiang wangping NULL
wangsong wanglong NULL
从标准输入读取：
wangping wanglong
【输出示例】
wangguoan wangping 2
wangguoan wanglong 2
【说明】
wangping和wanglong共同的最近祖先是wangguoan，该祖先与两人相差两辈。

问题5.3*：问题分析与设计

- 构造家谱（树）：如何利用结点之间的（父子）关系构造树（家谱）。一个简单直接的方法是：

① **结点插入法构造。**利用前序遍历找到相应的父结点，然后将子结点插入。该方法简单，对结点顺序要求不高（但父结点要在子节结前输入）；该方法的核心就是结点**查找。**

```
...
root = NULL;
for(i=0;i<n;i++){      //create a family tree
      fscanf(in,"%s%s%s",name0,name1,name2);
      root = insert(root,name0,name1,name2);
}
```

问题5.3*：问题分析与设计

- 查家谱：实际上就是查找相应节点。如果能得到节点至根的路径信息，就很容易计算出两个节点关系（如是否同辈、相差几辈、共同的祖先等）。

  如何在查找一个结点时得到其（从根结点至该结点的）路径信息：

  ① **在前序查找过程中设置一个栈来保存路径信息；**

  ② **一个简单的方法：为每个结点增加一个指向父结点的指针信息，这样在找到结点的同时，也就获得了相应的路径。**