计算机问题求解 – 论题3-4 -B树

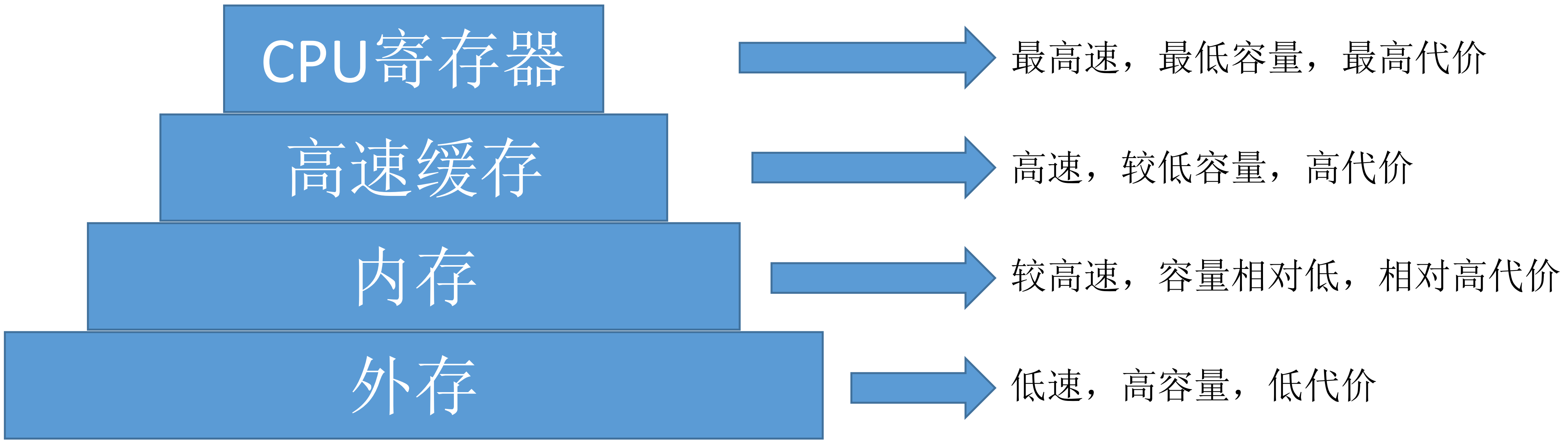
2016年9月22日

陶先平

问题1：我们为什么要为动态集合设计不同的数据结构？你能说出哪几种？

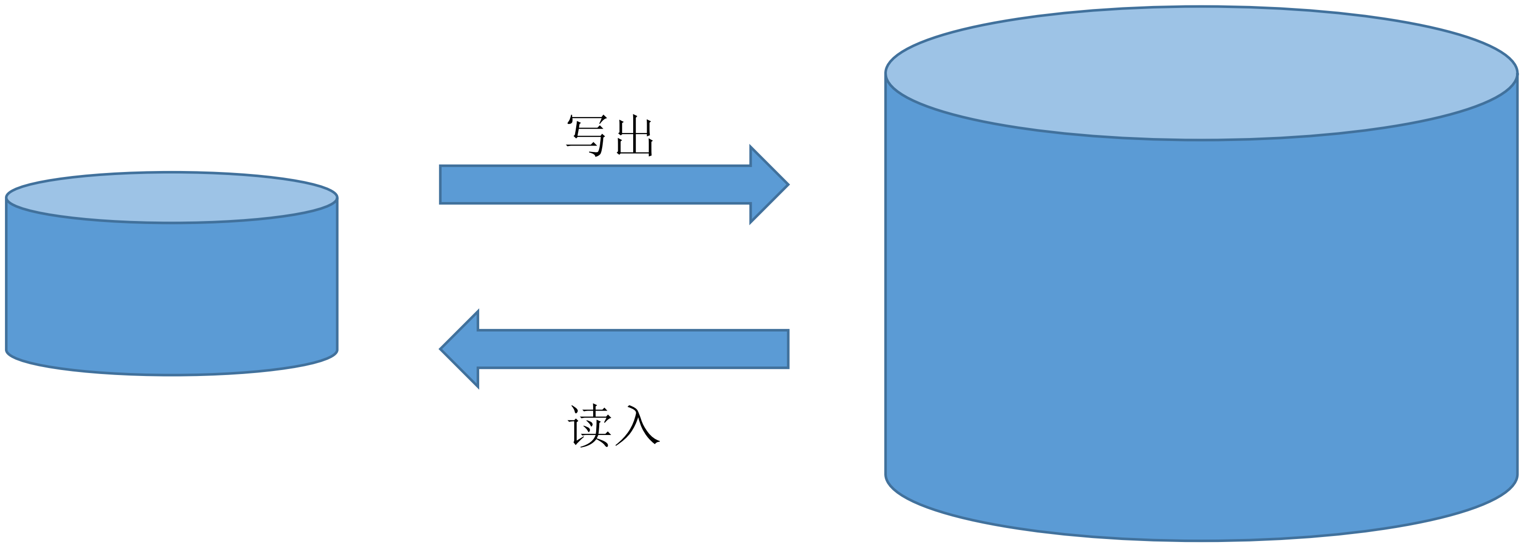
问题2：我们考察一个数据结构的某个操作的性能时，为什么没有考虑数据读写的时间开销？

计算机存储体系结构


CPU寄存器
最高速，最低容量，最高代价
高速缓存
高速，较低容量，高代价
内存
较高速，容量相对低，相对高代价
外存
低速，高容量，低代价

实际上：

- 当处理很大的文件（或者难以将所有数据都一次性载入内存再计算）时，我们总是根据需要从外存读取数据进入内存，总是从内存中将更新的数据写到外存



问题3：

假定我们需要存储10亿个键值。检索是作用在该数据集上的重要操作。请问，你该如何为此类应用设计外存上的数据结构？

你能想到的最好的数据结构是什么？

计算机软件技术研发的基本方法论

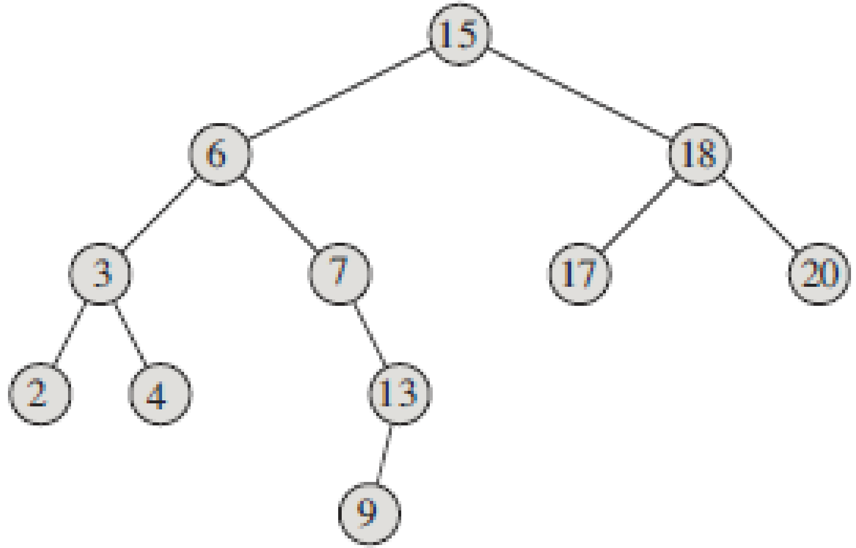
应用需求及特征

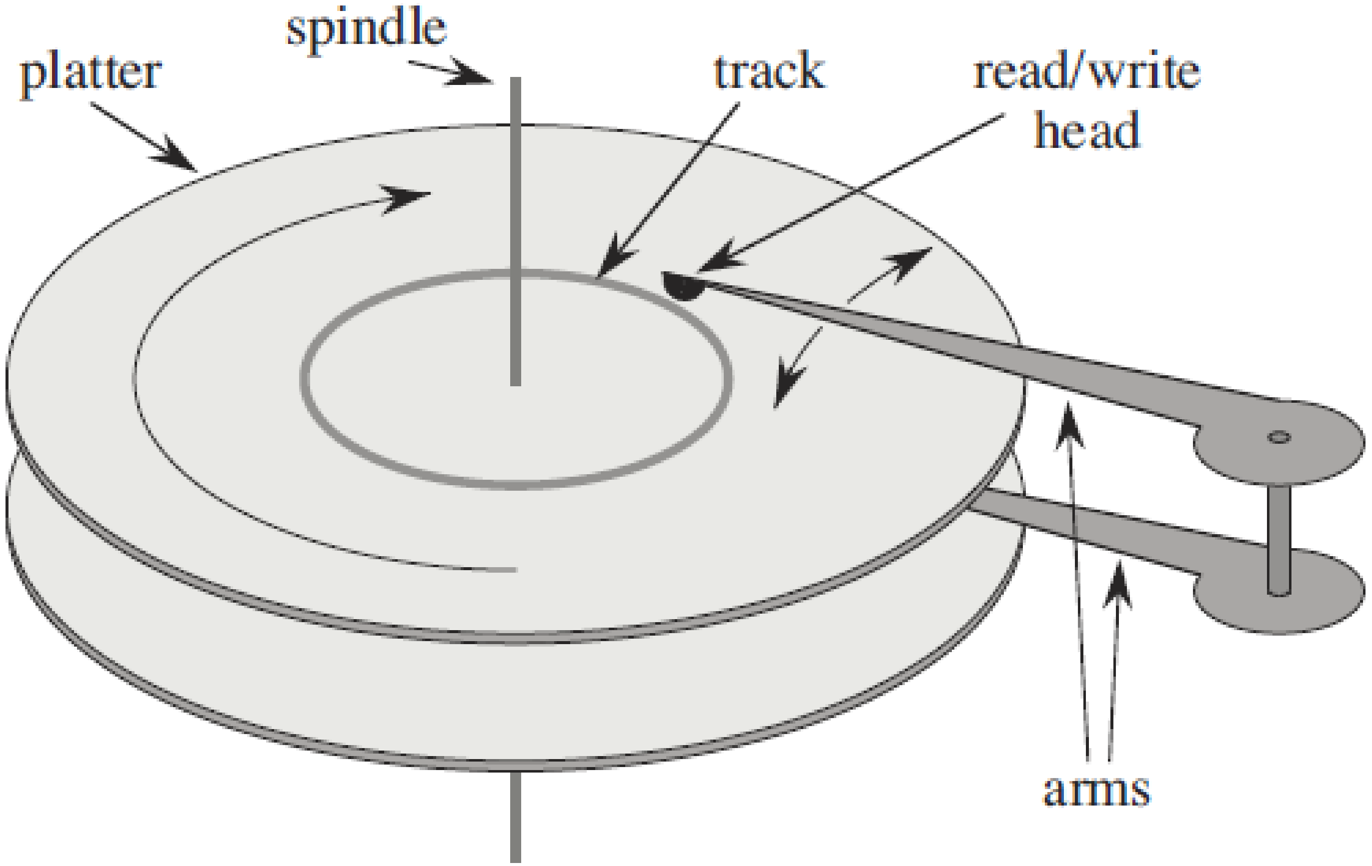
软件技术

基础硬件平台

look separately at the two principal components of the running time:

- the number of disk accesses, and
- the CPU (computing) time.

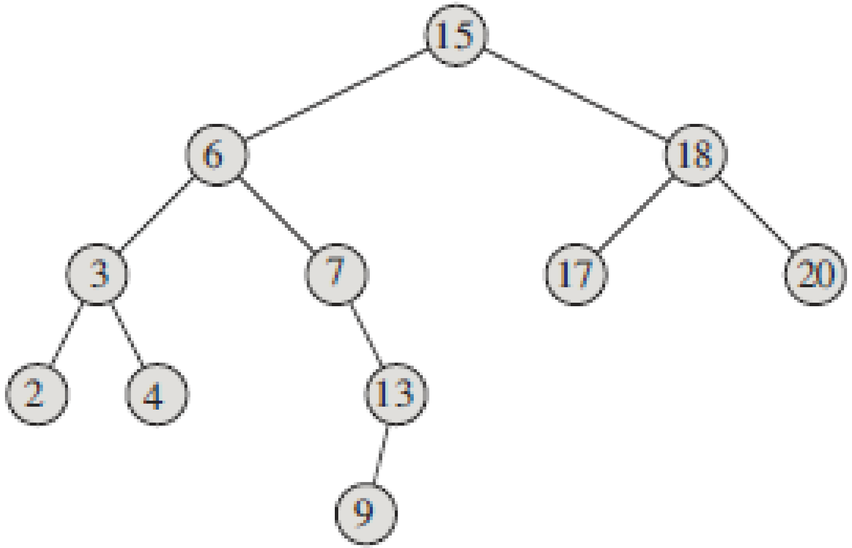




当我们受限于（受惠于）现实的物理世界时，我们该如何思考？

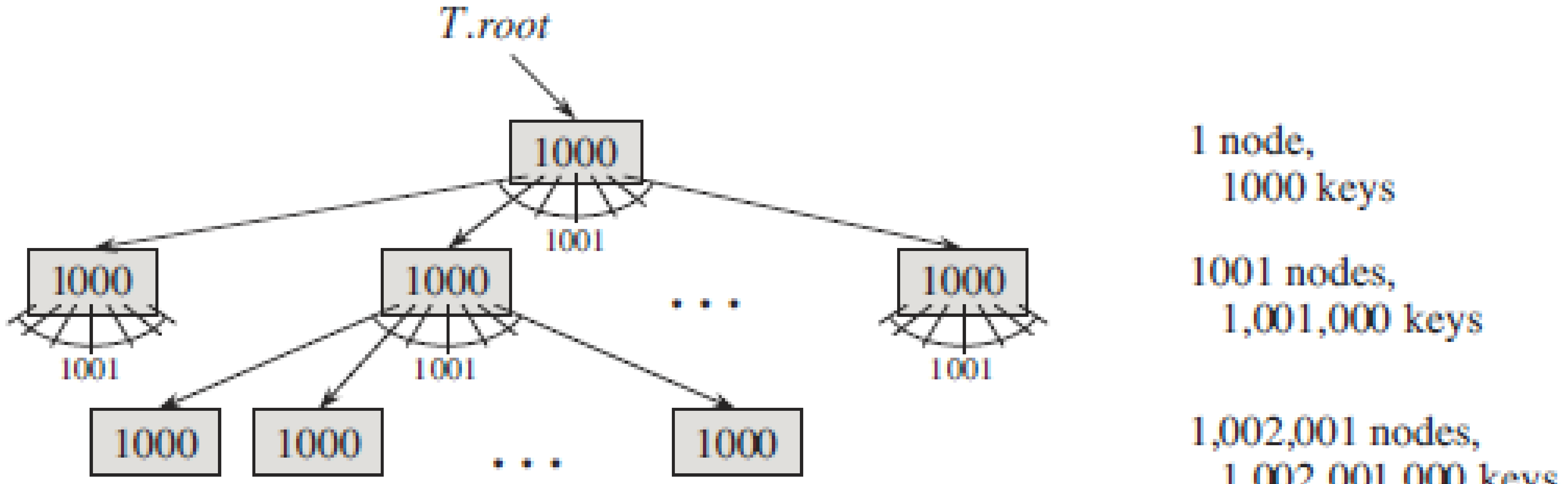
如果仅仅是BST

- 如果键值所需存储空间远小于页面大小

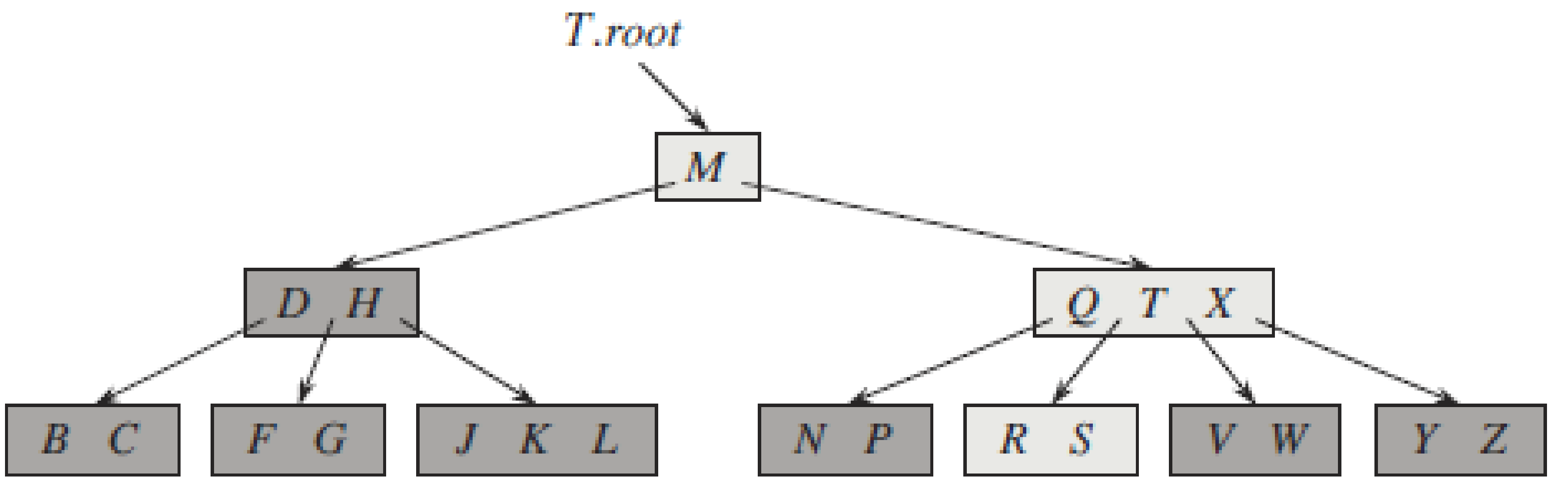


lgn次的磁盘访问 VS 和每次访问预取内容的浪费

如果我们在外存这样组织这10亿个键值：



问题4：这样的数据结构应该具有什么特性？



多子树：一个节点存储n个递增的键值，该节点有n+1个子树

分割：节点x的n个键值均匀分割以x为根的子树中存储的键值

问题5：为B树设计“度”有何用意？这个度为什么叫“最小度”？为什么又叫上下限？

Theorem 18.1

If $n \geq 1$, then for any n-key B-tree T of height h and minimum degree $t \geq 2$,

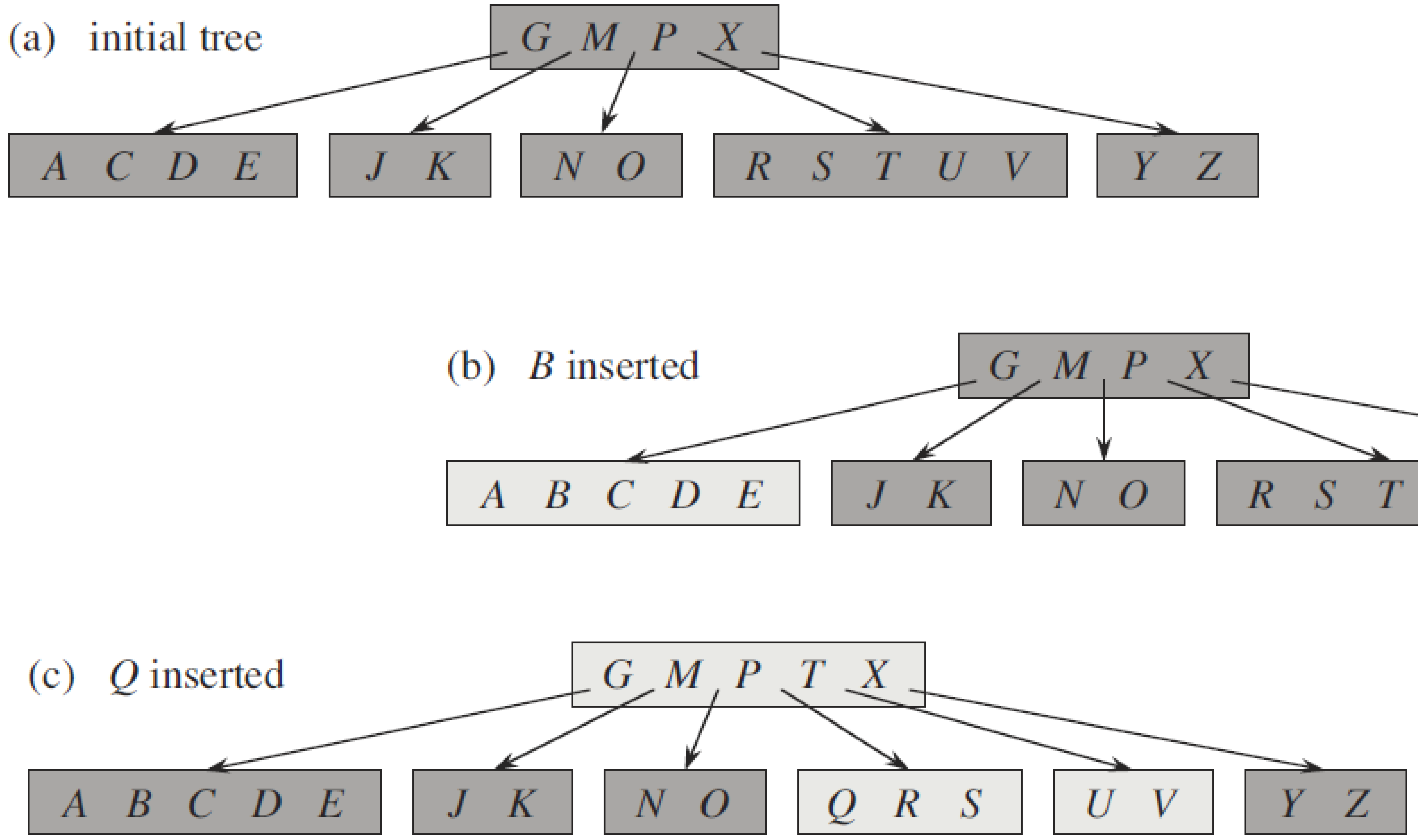
$$h \leq \log_t \frac{n+1}{2} .$$

B树上的搜索操作

```
B-TREE-SEARCH(x, k)
i = 1
while i ≤ x.n and k > x.key_i
    i = i + 1
if i ≤ x.n and k == x.key_i
    return (x, i)
elseif x.leaf
    return NIL
```

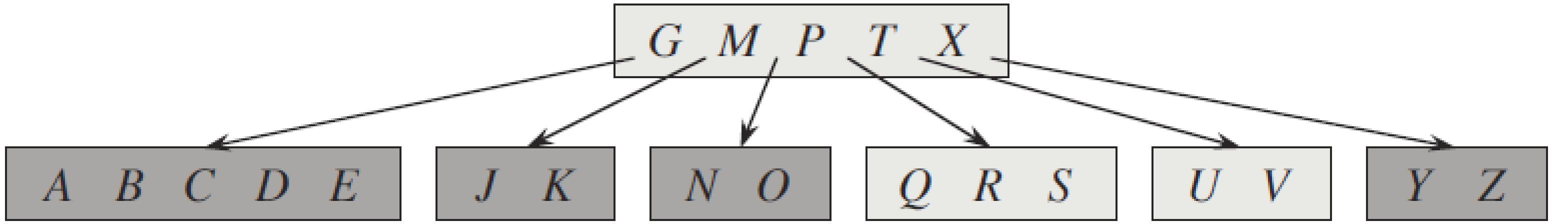
X和k分别是什么？这个操作返回的是什么？

插入一个键值，必须保证B树性质

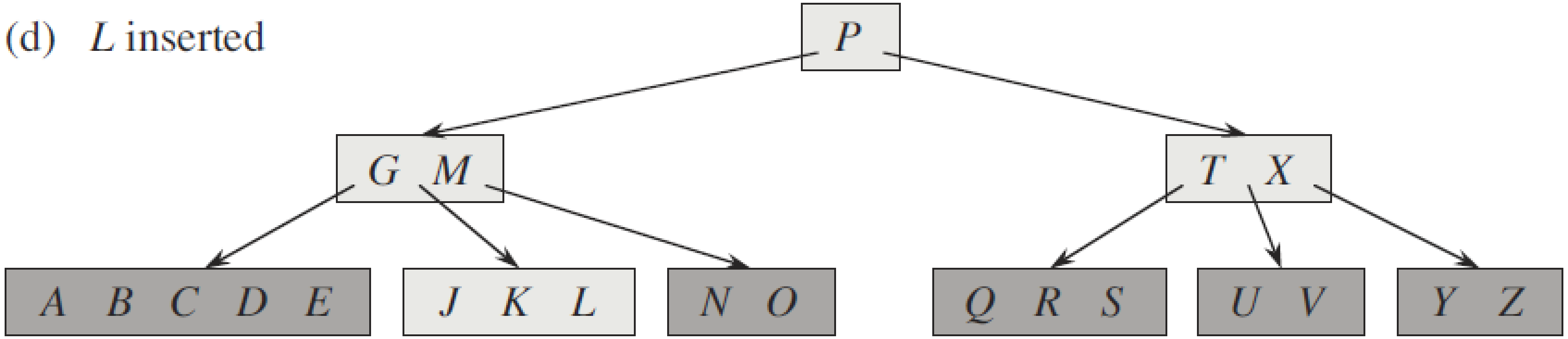


节点的分裂！

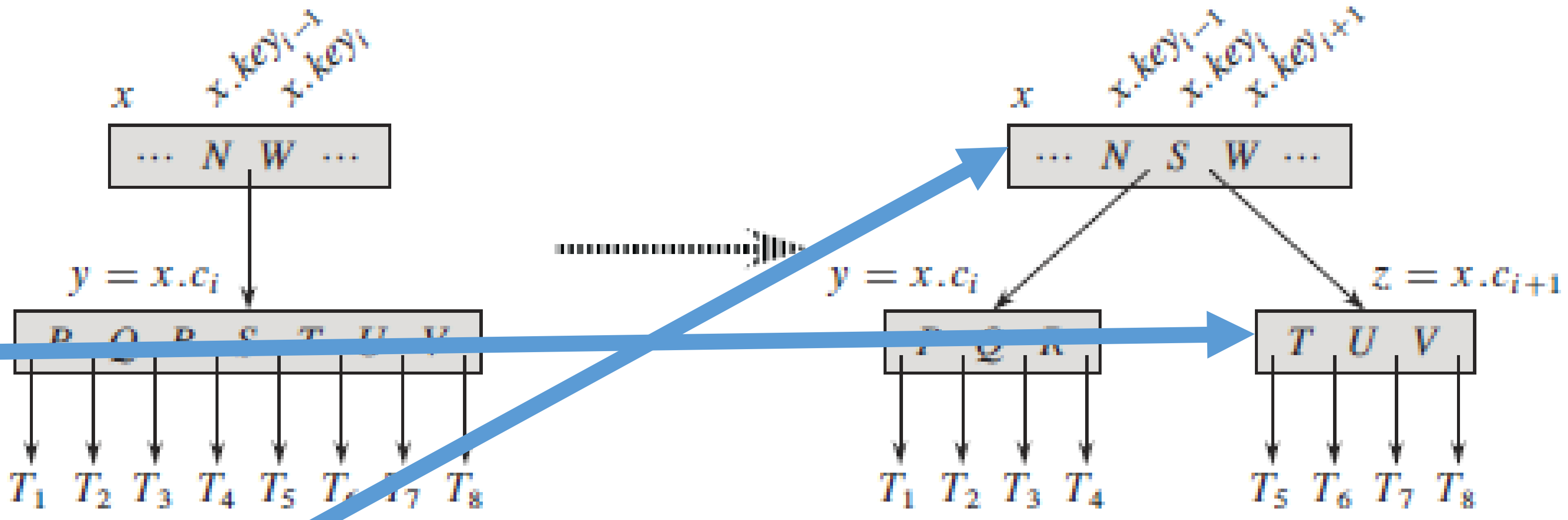
当L插入时，为什么必须引起分裂？



(d) *L* inserted



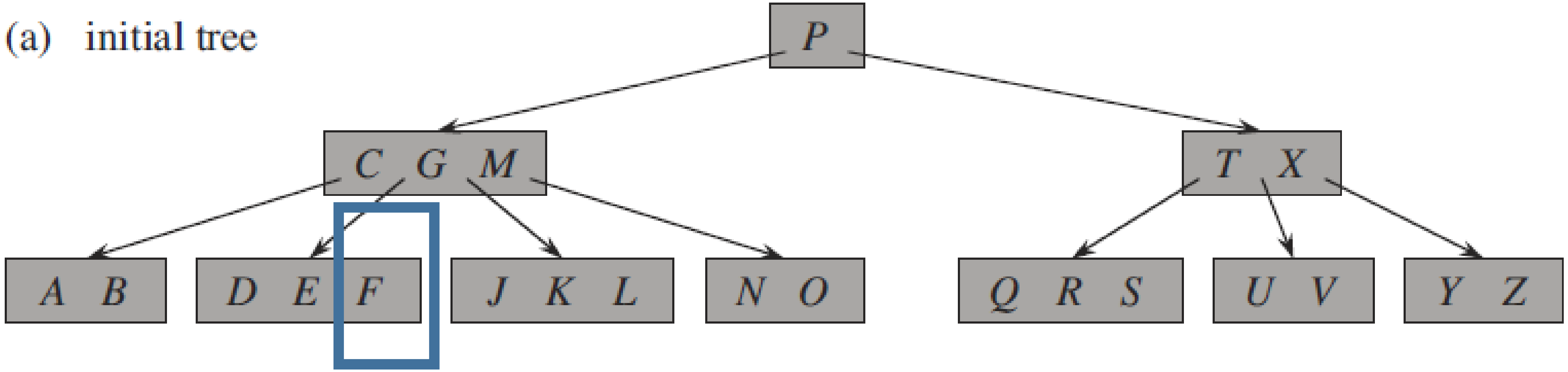
```
B-TREE-SPLIT-CHILD(x, i)
z = ALLOCATE-NODE()
y = x.c_i
z.leaf = y.leaf
z.n = t - 1
for j = 1 to t - 1
    z.key_j = y.key_{j+t}
if not y.leaf
    for j = 1 to t
        z.c_j = y.c_{j+t}
y.n = t - 1
for j = x.n + 1 downto i + 1
    x.c_{j+1} = x.c_j
x.c_{i+1} = z
for j = x.n downto i
    x.key_{j+1} = x.key_j
x.key_i = y.key_t
x.n = x.n + 1
DISK-WRITE(y)
DISK-WRITE(z)
DISK-WRITE(x)
```



Y节点的处理代码是什么？

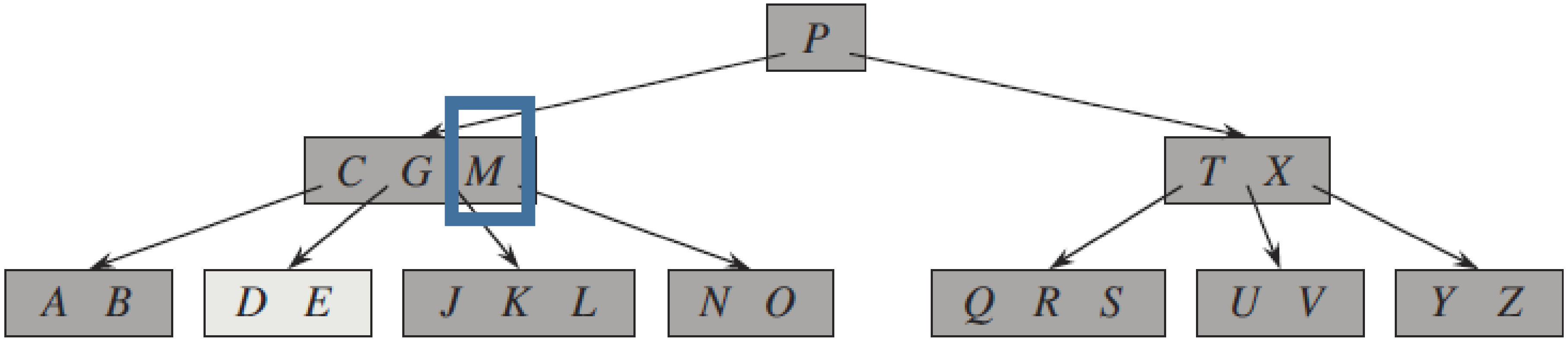
为什么要有这三条语句？

在删除B树中某个节点时，最根本的关注点是什么？



1. If the key k is in node x and x is a leaf, delete the key k from x.

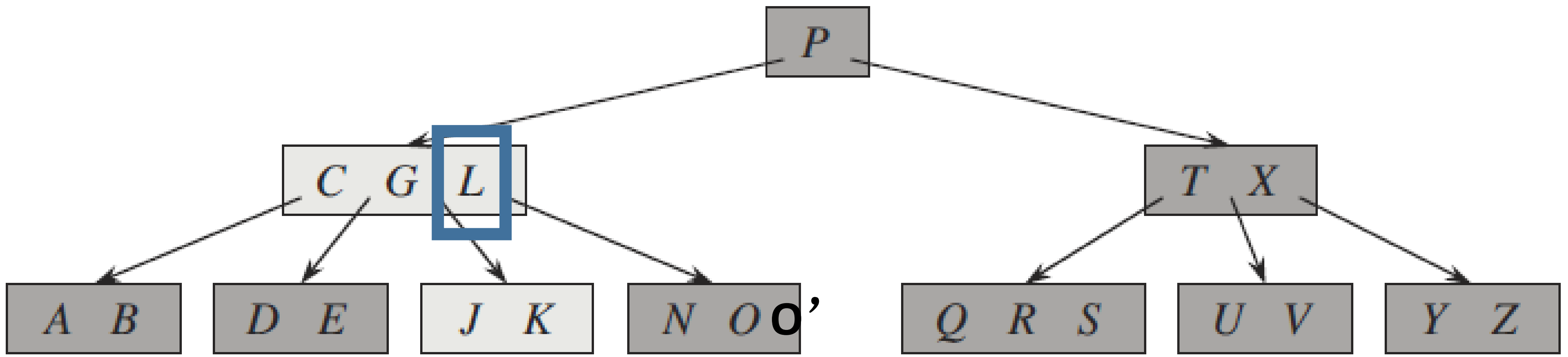
且x的键值数>t



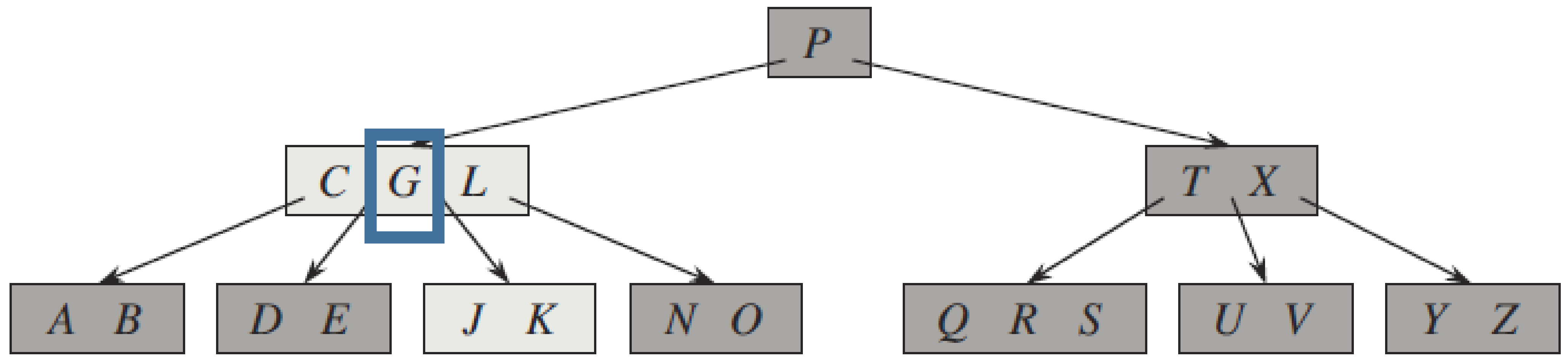
2. If the key k is in node x and x is an internal node, do the following:

   a. If the child y that precedes k in node x has at least t keys, then find the predecessor k' of k in the subtree rooted at y. Recursively delete k', and replace k by k' in x. (We can find k' and delete it in a single downward pass.)

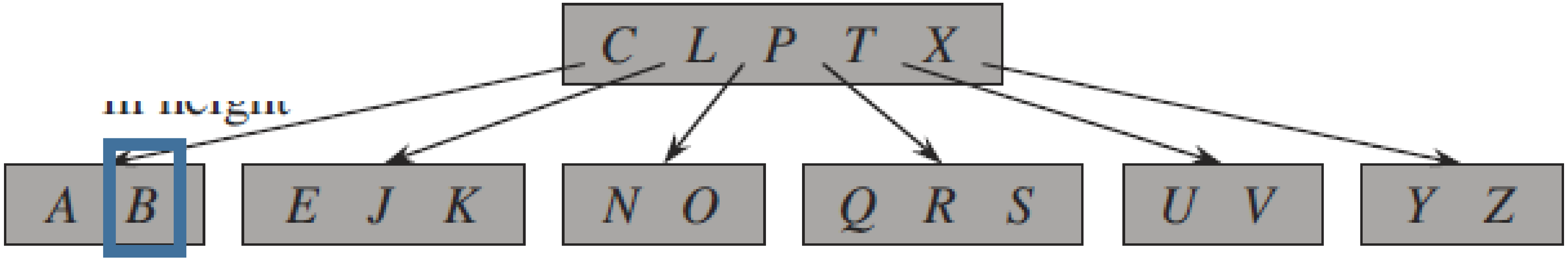
怎么去找到这个k’?



b. If y has fewer than t keys, then, symmetrically, examine the child z that follows k in node x. If z has at least t keys, then find the successor k' of k in the subtree rooted at z. Recursively delete k', and replace k by k' in x. (We can find k' and delete it in a single downward pass.)

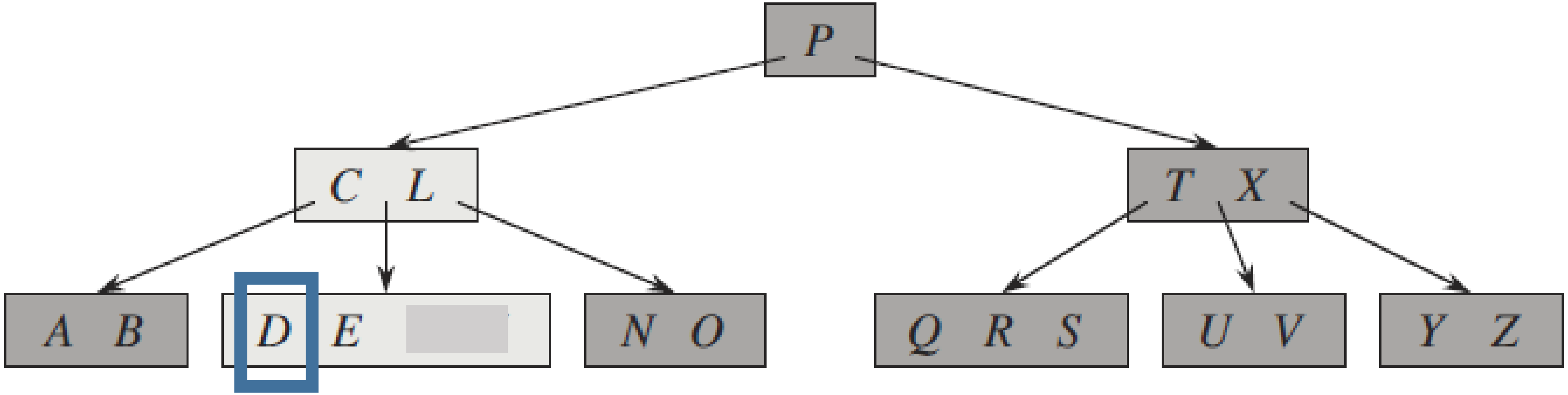


c. Otherwise, if both y and z have only $t-1$ keys, merge k and all of z into y, so that x loses both k and the pointer to z, and y now contains $2t-1$ keys. Then free z and recursively delete k from y.



3. If the key k is not present in internal node x, determine the root $x.c_i$ of the appropriate subtree that must contain k, if k is in the tree at all. If $x.c_i$ has only $t-1$ keys, execute step 3a or 3b as necessary to guarantee that we descend to a node containing at least t keys. Then finish by recursing on the appropriate child of x.

   a. If $x.c_i$ has only $t-1$ keys but has an immediate sibling with at least t keys, give $x.c_i$ an extra key by moving a key from x down into $x.c_i$, moving a key from $x.c_i$'s immediate left or right sibling up into x, and moving the appropriate child pointer from the sibling into $x.c_i$.



3. If the key k is not present in internal node x, determine the root $x.c_i$ of the appropriate subtree that must contain k, if k is in the tree at all. If $x.c_i$ has only $t-1$ keys, execute step 3a or 3b as necessary to guarantee that we descend to a node containing at least t keys. Then finish by recursing on the appropriate child of x.

   b. If $x.c_i$ and both of $x.c_i$'s immediate siblings have $t-1$ keys, merge $x.c_i$ with one sibling, which involves moving a key from x down into the new merged node to become the median key for that node.

Open topics

- 请证明：我们使用的插入节点的算法，不会使得叶节点的高度不一致

- 请写出在B树中删除一个节点的算法。算法原型为：
  - B_Tree_Delete(x,k)

作业：

- 18.1.1；18.1.4
- 18.2.3；18.2.4
- 18.3.1