```
{
   if (c == ' ') return 0;
   if ('a' <= c && c <= 'z') return c - 'a' + 1;
   if ('A' <= c && c <= 'Z') return c - 'A' + 1;
   return 27;
}
```

The Key class implementation is identical to that used in Radix sorting from Chapter 9.

11.3 EXTERNAL SEARCHING: B-TREES

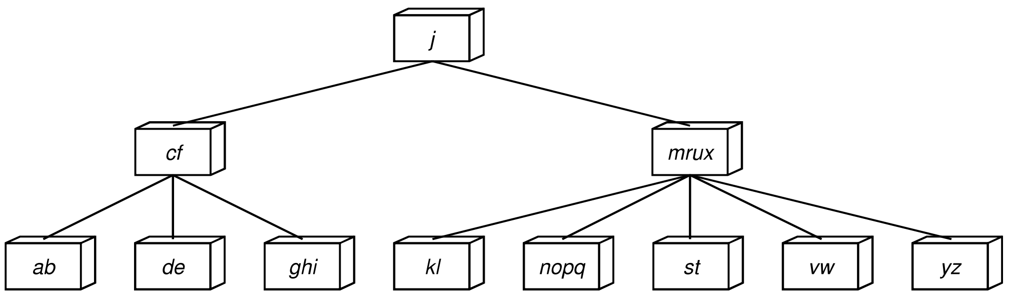
Exercises 11.3

E1. *Insert the six remaining letters of the alphabet in the order*

z, v, o, q, w, y

into the final B-tree of Figure 11.10 (page 539).

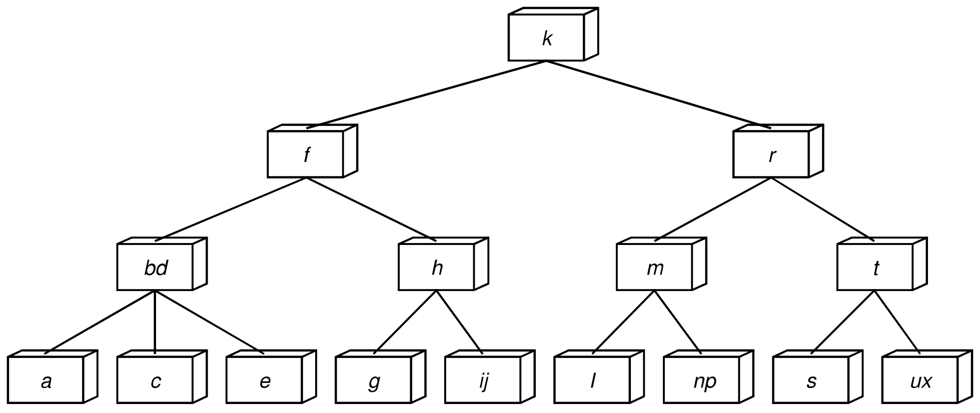
Answer



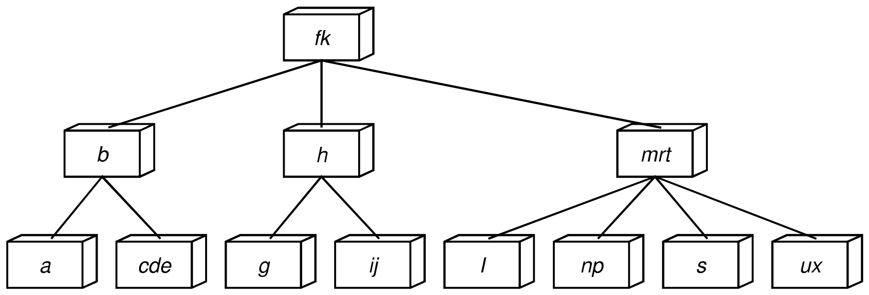
E2. *Insert the following entries, in the order stated, into an initially empty B-tree of order* **(a)** *3,* **(b)** *4,* **(c)** *7:*

a g f b k d h m j e s i r x c l n t u p

Answer

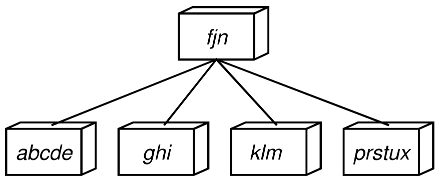


(a)



(b)

TOC
Index
Help



(c)

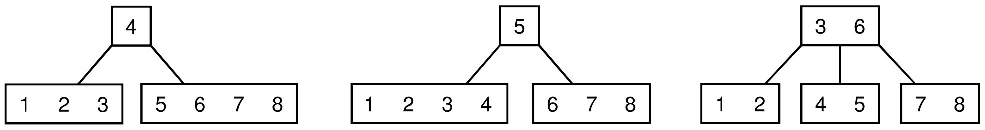
E3. *What is the smallest number of entries that, when inserted in an appropriate order, will force a B-tree of order 5 to have height 3 (that is, 3 levels)?*

Answer The sparsest possible B-tree of order 5 with three levels will have one entry in the root, two nodes on the next level down, each with two entries and each of which must have three children on the lowest level, each of which must have at least two entries. Hence the total number of entries is

$$1 \times 1 + 2 \times 2 + 6 \times 2 = 17.$$

E4. *Draw all the B-trees of order 5 (between 2 and 4 keys per node) that can be constructed from the keys 1, 2, 3, 4, 5, 6, 7, and 8.*

Answer There are too many keys to fit in one node (the root) and too few to make up a tree with three levels. Hence all the trees have two levels. They are:



E5. *If a key in a B-tree is not in a leaf, prove that both its immediate predecessor and immediate successor (under the natural order) are in leaves.*

Answer The immediate successor is found by taking the right child (assuming it is not in a leaf) and then taking the leftmost child of each node as long as it is not NULL. This process ends in a leaf, and the immediate successor of the given key is the first key in that leaf. Similarly, the immediate predecessor is found by taking the left child and then right children as long as possible, and the predecessor is then the rightmost key in a leaf.

E6. *Suppose that disk hardware allows us to choose the size of a disk record any way we wish, but that the time it takes to read a record from the disk is $a + bd$, where a and b are constants and d is the order of the B-tree. (One node in the B-tree is stored as one record on the disk.) Let n be the number of entries in the B-tree. Assume for simplicity that all the nodes in the B-tree are full (each node contains $d - 1$ entries).*

disk accesses

(a) *Explain why the time needed to do a typical B-tree operation (searching or insertion, for example) is approximately $(a + bd)\log_d n$.*

Answer The height of the B-tree is about $\log_d n$ and so the operation will require about this many disk reads, each of which takes time $a + bd$, giving the total indicated. The time required for computation in high-speed memory will likely be trivial compared to the time needed for disk reads. Hence the total time required is essentially that needed for the disk reads.

(b) *Show that the time needed is minimized when the value of d satisfies $d(\ln d - 1) = a/b$. (Note that the answer does not depend on the number n of entries in the B-tree.) [Hint: For fixed a, b, and n, the time is a function of d: $f(d) = (a + bd)\log_d n$. Note that $\log_d n = (\ln n)/(\ln d)$. To find the minimum, calculate the derivative $f'(d)$ and set it to 0.]*

Answer Since

$$f(d) = \frac{(a + bd)\ln n}{\ln d},$$