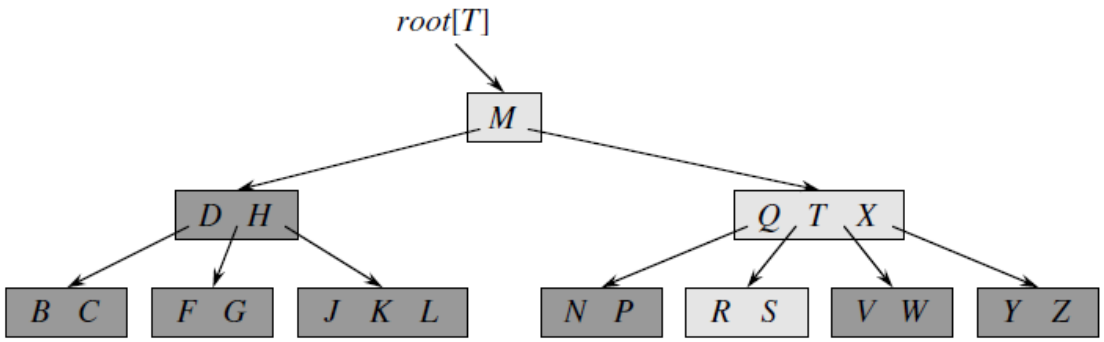
B-Trees

B-trees are balanced search trees designed to work well on magnetic disks or other direct-access secondary storage devices, because they are better at minimizing disk I/O operations.

Many database systems use B-trees (or variants of B-trees such as B^+-tree, B^*-tree) to store information.

Vs Red-Black tree:	
Difference	B-tree nodes may have many children, from a handful to thousands. That is, the "branching factor" of a B-tree can be quite large, although it is usually determined by characteristics of the disk unit used.
Similarity	Every n-node B-tree has height O(lg n), although the height of a B-tree can be considerably less than that of a red-black tree because its branching factor can be much larger. Therefore, B-trees can also be used to implement many dynamic-set operations in time O(lg n).

B-trees generalize binary search trees in a natural manner. Following diagram shows a simple B-tree.

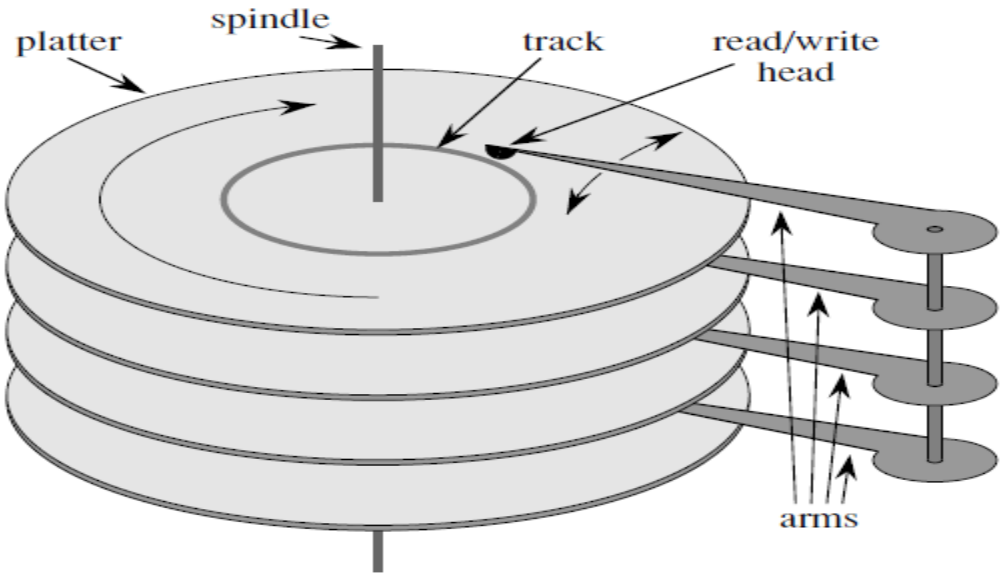


If an internal B-tree node x contains n[x] keys, then x has n[x] + 1 children. The keys in node x are used as dividing points separating the range of keys handled by x into n[x] + 1 subranges, each handled by one child of x. When searching for a key in a B-tree, we make an (n[x] + 1)-way decision based on comparisons with the n[x] keys stored at node x.

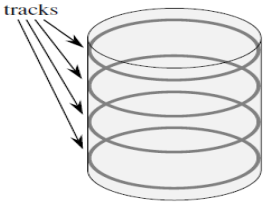
Data structures on secondary storage:

- There are many different technologies available for providing memory capacity in a computer system.
- The primary memory (or main memory) of a computer system normally consists of **silicon memory** chips. (It is more expensive than **magnetic storage**, such as tapes or disks)
- Most computer systems also have secondary storage based on magnetic disks. (It stores more data than MM)

Disk Drive:



- The drive consists of several **platters**, which rotate at a constant speed around a common **spindle**.
- The surface of each platter is covered with a **magnetizable material**.
- Each platter is read or written by a **head** at the end of an **arm**.
- The arms are physically attached, or "ganged" together, and they can move their heads toward or away from the spindle.
- When a given head is stationary, the surface that passes underneath it is called a **track**.
- The read/write heads are vertically aligned at all times, and therefore the set of tracks underneath them are accessed simultaneously, such a set of tracks, is called a **cylinder**.



Although disks are cheaper and have higher capacity than main memory, they are much, much slower because they have moving parts.

Eg: For example, commodity disks rotate at speeds of 5400–15,000 revolutions per minute (RPM), with 7200 RPM being the most common.

Although 7200 RPM may seem fast, one rotation takes 8.33 milliseconds, which is almost 5 orders of magnitude longer than the 100 nanosecond access times commonly found for silicon memory.

In other words, if we have to wait a full rotation for a particular item to come under the read/write head, we could access main memory almost 100,000 times during that span!

On average we have to wait for only half a rotation, but still, the difference in access times for silicon memory vs. disks is enormous. (Moving the arms also takes some time, because there are two components to the mechanical motion: platter rotation and arm movement.)
(Roughly, average access times for disks are: 3 to 9 milliseconds)

Definition of B-Tree:

A B-tree T is a rooted tree (whose root is root[T]) having the **following properties**:

(1) Every node x has the following fields:

a. n[x]: the number of keys currently stored in node x,

b. the n[x] keys themselves, stored in nondecreasing order, so that

$$key_1[x] \leq key_2[x] \leq \cdots \leq key_{n[x]}[x],$$

c. leaf [x]: a boolean value that is TRUE if x is a leaf and FALSE if x is an internal node.

(2) Each internal node x also contains n[x]+1 pointers $c_1[x], c_2[x], \ldots, c_{n[x]+1}[x]$ to its children. Leaf nodes have no children, so their c_i fields are undefined.

(3) The keys $key_i[x]$ separate the ranges of keys stored in each subtree: if k_i is any key stored in the subtree with root $c_i[x]$, then

$$k_1 \leq key_1[x] \leq k_2 \leq key_2[x] \leq \cdots \leq key_{n[x]}[x] \leq k_{n[x]+1}$$

(4) All leaves have the same depth, which is the tree's height h.

(5) There are lower and upper bounds on the number of keys a node can contain. These bounds can be expressed in terms of a fixed integer $t \geq 2$ called the **minimum degree of the B-tree**:

a. Every node other than the root must have at least **t − 1** keys.
(⇒ Every internal node other than the root thus has at least t children.)
If the tree is nonempty, the root must have at least one key.

b. Every node can contain at most $2t - 1$ keys.
(⇒Therefore, an internal node can have at most 2t children.)
We say that a node is full if it contains exactly $2t - 1$ keys.

Note: The simplest B-tree occurs when t = 2. Every internal node then has either 2, 3, or 4 children, and we have a 2-3-4 tree. (⇒ Each node can contain x keys where $1 \leq x \leq 3$)

In practice, however, much larger values of t are typically used.