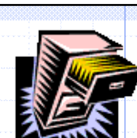
Presentation for use with the textbook Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014

B-Trees

Computer Memory



- In order to implement any data structure on an actual computer, we need to use computer memory.
- Computer memory is organized into a sequence of words, each of which typically consists of 4, 8, or 16 bytes (depending on the computer).
- These memory words are numbered from 0 to N −1, where N is the number of memory words available to the computer.
- The number associated with each memory word is known as its memory **address**.

0	1	2	3	4	5	6	7	8	9	10
	D		Z			C	Q			

Disk Blocks

- Consider the problem of maintaining a large collection of items that does not fit in main memory, such as a typical database.
- In this context, we refer to the external memory is divided into blocks, which we call **disk blocks**.
- The transfer of a block between external memory and primary memory is a **disk transfer** or **I/O**.
- There is a great time difference that exists between main memory accesses and disk accesses
- Thus, we want to minimize the number of disk transfers needed to perform a query or update. We refer to this count as the **I/O complexity** of the algorithm involved.

(a,b) Trees

- To reduce the number of external-memory accesses when searching, we can represent a map using a multiway search tree.
- This approach gives rise to a generalization of the **(2,4)** tree data structure known as the **(a,b) tree**.
- An (a,b) tree is a multiway search tree such that each node has between a and b children and stores between $a - 1$ and $b - 1$ entries.
- By setting the parameters a and b appropriately with respect to the size of disk blocks, we can derive a data structure that achieves good external-memory performance.

Definition

- An **(a,b) tree**, where parameters a and b are integers such that 2 ≤ a ≤ **(b+1)/**2, is a multiway search tree T with the following additional restrictions:
- **Size Property**: Each internal node has at least a children, unless it is the root, and has at most b children.
- **Depth Property**: All the external nodes have the same depth.

Height of an (a,b) Tree

Proposition 15.1: *The height of an* (a,b) *tree storing* n *entries is* $\Omega(\log n/\log b)$ *and* $O(\log n/\log a)$.

Justification: Let T be an (a,b) tree storing n entries, and let h be the height of T. We justify the proposition by establishing the following bounds on h:

$$\frac{1}{\log b}\log(n+1) \le h \le \frac{1}{\log a}\log\frac{n+1}{2}+1.$$

By the size and depth properties, the number n'' of external nodes of T is at least $2a^{h-1}$ and at most b^h. By Proposition 11.7, $n''=n+1$. Thus,

$$2a^{h-1} \le n+1 \le b^h.$$

Taking the logarithm in base 2 of each term, we get

$$(h-1)\log a + 1 \le \log(n+1) \le h\log b.$$

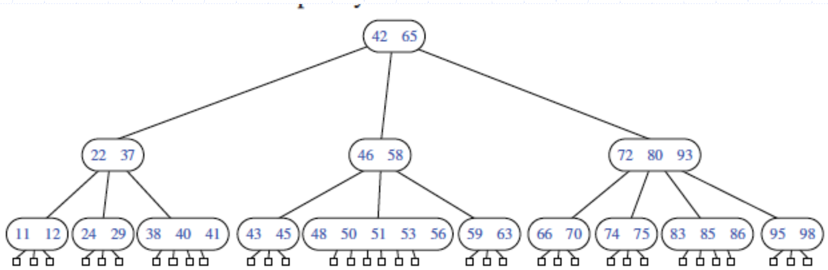
An algebraic manipulation of these inequalities completes the justification. ■

Searches and Updates

- The search algorithm in an **(a,b)** tree is exactly like the one for multiway search trees.
- The insertion algorithm for an **(a,b)** tree is similar to that for a **(**2,4**)** tree.
  - An overflow occurs when an entry is inserted into a **b**-node **w**, which becomes an illegal **(b+**1**)**-node.
  - To remedy an overflow, we split node w by moving the median entry of w into the parent of w and replacing w with a **(**b+1**)/**2-node w and a **(**b+1**)/**2-node w.
- Removing an entry from an **(a,b)** tree is similar to what was done for **(**2,4**)** trees.
  - An underflow occurs when a key is removed from an **a**-node **w**, distinct from the root, which causes **w** to become an **(a**–1**)**-node.
  - To remedy an underflow, we perform a transfer with a sibling of **w** that is not an **a**-node or we perform a fusion of **w** with a sibling that is an **a**-node.

B-Trees

- A version of the **(a,b)** tree data structure, which is the best-known method for maintaining a map in external memory, is a "**B-tree**."
- A **B-tree of order d** is an **(a,b)** tree with **a = d/**2 and **b = d**.



I/O Complexity

Proposition 15.2: *A B-tree with n entries has I/O complexity $O(\log_B n)$ for search or update operation, and uses $O(n/B)$ blocks, where B is the size of a block.*

- **Proof**:
  - Each time we access a node to perform a search or an update operation, we need only perform a single disk transfer.
  - Each search or update requires that we examine at most **O(**1**)** nodes for each level of the tree.

