Data Structures and Algorithms in Java™

Sixth Edition

Michael T. Goodrich
Department of Computer Science
University of California, Irvine

Roberto Tamassia
Department of Computer Science
Brown University

Michael H. Goldwasser
Department of Mathematics and Computer Science
Saint Louis University

Instructor's Solutions Manual

WILEY

Chapter 15 Memory Management and B-Trees

Hints and Solutions

Reinforcement

R-15.1) Hint Perform an Internet search to determine a good estimate on the number of atoms on earth.

R-15.2) Hint Draw the memory cache and manually process the requests using a pencil with a good eraser.

R-15.2) Solution This sequence causes 5 page misses, on the first 5, second 3, second 4, third 2, and last 3.

R-15.3) Hint Draw the memory cache and manually process the requests using a pencil with a good eraser.

R-15.3) Solution This sequence causes 3 page misses, on the first 5, the last 2, and the last 3.

R-15.4) Hint Draw the memory cache and manually process the requests using a pencil with a good eraser.

R-15.5) Hint Start with the description provided in the book.

R-15.6) Hint Revisit the definition of an (a,b) tree.

R-15.6) Solution T is a valid (a,b) tree for $2 \leq a \leq \min\{5,(b+1)/2\}$ and $b \geq 8$. For example, T could be a $(4,8)$ tree, a $(5,9)$ tree, but not a $(5,8)$ tree.

R-15.7) Hint The definition of an order-d deals with the minimum and maximum number of children an internal node can have. Please see the book for details.

R-15.7) Solution T could be an order-8, order-9, or order-10 B-tree.

R-15.8) Hint Use a pencil with a good eraser.

Creativity

C-15.9) Hint Review the external-memory sorting algorithm.

C-15.10) Hint Keep the top one or two blocks of the stack in main memory.

C-15.11) Hint Keep queue runs in blocks.

C-15.11) Solution Use a linked list where each node is a block of size B. We add elements to the end, using $O(1)$ disk transfers (creating a new block when we have filled the last block). We dequeue elements from the front, returning the block to the free memory heap when it becomes empty.

C-15.12) Hint Consider an alternate linked list implementation that uses "fat" nodes.

C-15.12) Solution Consider a linked list implementation of the ADT where every node is a block of size B, holding at least $B/4$ entries. When an element is added to a node with a full block, that node can be split into two blocks having $B/2$ entries, and then the new element can be inserted. Likewise, if an element is deleted from a block bringing its number of elements below $B/4$, an element can either be borrowed from a neighboring block, or else two neighboring blocks can be combined into one. Because of the requirement that a node hold at least $B/4$ entries, there will be at most $4n/B$ blocks and therefore a complete scan uses $O(n/B)$ transfers.

C-15.13) Hint Note that each valid node v and its children in a (2,4) tree correspond to a red-black subtree of height 2. In a (4,8) tree, you will need bigger subtrees.

C-15.14) Hint Consider the extreme cases.

C-15.15) Hint Try to block order-B sized sub "trees" in the skip list.

C-15.16) Hint Start from sequence solution for the union-find problem.

C-15.17) Hint A single scan suffices.

C-15.17) Solution Sort the elements of S using an external-memory sorting algorithm. This brings together the elements with the same keys. One more scan through this sorted order matches up the red and blue elements. The time for this algorithm is dominated by the time to sort, which can be done using $O((n/B)\log(n/B)/\log(M/B))$ block transfers.

C-15.18) Hint Each request can "see into the future" to see when is the next time existing blocks will be accessed next.

C-15.19) Hint In an initial scan, keep track of the best candidate majority value, x, and a counter that keeps track of the number of times you have seen a copy of x versus some other integer.

C-15.20) Hint Consider what happens to a page that is accessed a lot and then never accessed again.

C-15.20) Solution Consider a sequence where page 1 is accessed m times and then never accessed again, and then pages 2 through $m+1$ are each accessed once and then this same access sequence of pages 2 through $m+1$ is repeated. The LFU algorithm will never through out page 1; hence, will have a page miss on every access, starting with page $m+1$. But an optimal algorithm will throw out page 1 and then never have any more page misses after the misses for the first time pages 2 through $m+1$ are accessed.

C-15.21) Hint The answer just uses some simple logarithm identities.

C-15.21) Solution $O(\log n)$ time.

Projects

P-15.22) Hint Make sure to use typical memory sizes and do a long simulation.

P-15.23) Hint Let a and b be definable parameters or constants. And let insertion be the first update method you program.

P-15.24) Hint Start with insertion as the first update operation you code up, and use a simple uniform distribution of keys to perform the experiments.