CS 240: Data Structures and Data Management
Module 11 Study Guide

Taylor J. Smith — Spring 2017

Key Concepts

- Memory exists in a **hierarchy**; the closer to the CPU memory is, the faster the access time.
- Since we can't store large dictionaries in register/cache memory, we must keep it in external memory instead. But external memory is slow.
- Since data is read in disk blocks/pages, we can take parts of our dictionary to store it in register/cache memory as needed.
- **(2,3)-trees** are like BSTs, but every internal node contains either one key and two children, or two keys and three children.
- **(a,b)-trees** are a general version of $(2,3)$-trees, where each internal node (except the root) has at least a children and at most b children. If a node has k children, then it stores $k-1$ keys.
- **B-trees** are a special type of (a,b)-tree where $a = \lceil M/2 \rceil$ and $b = M$ for some "order" value M.
- Searching in these trees is identical to searching in BSTs. Insertion and deletion may require "splitting" or "merging" of nodes.
- B-trees require us to load $\Theta(\log(n)/\log(M))$ pages to main memory, compared to AVL trees and $(2,3)$-trees which require $\Theta(\log(n))$ pages. Depending on our choice of M, this could result in huge savings!
- **Extendible hashing** is a hashing technique that is similar to using a B-tree of height 1 with a maximum leaf size of S, where S is the block size.
- The hash function hashes keys to the set $\{0, 1, \dots, 2^L - 1\}$. The extendible hashing **directory** is of size 2^d, where $d \leq L$ is the **order** of the directory.
- Each directory entry points to a **block**, which contains at most S items. The hash values in each block agree on the leading k_B bits, where k_B is the **local depth** of the block.
- If the hash table fills up, we may either perform a **block split** or a **directory grow** depending on whether a block or the directory is full.
- Extendible hashing allows us to store the directory in main memory (easy to search), while each block is stored in external memory (reduces number of unnecessary external memory accesses).

Suggested Readings

- **Sedgewick:** 13.3 (Top-Down 2–3–4 Trees), 16.3 (B Trees), 16.4 (Extendible Hashing)
- **CLRS:** Chapter 18 (B-Trees)
- **Goodrich/Tamassia:** 14.1 (External-Memory Algorithms)

Practice Questions

Sedgewick

13.39. Draw the balanced $(2,3)$-tree that results when you insert items with the keys $E\ A\ S\ Y\ Q\ U\ T\ I\ O\ N$ in that order into an initially empty tree.

13.41. What are the minimum and maximum heights possible for balanced $(2,3)$-trees with n nodes?

16.8. Draw the B-tree that results when you insert 16 equal keys into an initially empty tree, with $M = 4$.

16.9. Explain why $(1,2)$-trees are not of practical interest as balanced trees.

16.28. Draw figures to illustrate the process of inserting the keys $\{562, 221, 240, 771, 274, 233, 401, 273, 201\}$ in that order into an initially empty extendible hash table, with a block local depth of 5.

16.30. Assume that you are given an array of items in sorted order. Describe how you would determine the directory size of the extendible hash table corresponding to that set of items.

CLRS

18.1-3. Show all legal B-trees of minimum degree 2 that represent $\{1, 2, 3, 4, 5\}$.

18.1-4. As a function of the minimum degree t, what is the maximum number of keys that can be stored in a B-tree of height h?

18.2-1. Show the results of inserting the keys

$$\{F, S, Q, K, C, L, H, T, V, W, M, R, N, P, A, B, X, Y, D, Z, E\}$$

in order into an empty B-tree with minimum degree 2.

18.2-3. Explain how to find the minimum key stored in a B-tree and how to find the predecessor of a given key stored in a B-tree.

Goodrich/Tamassia

R-14.1. Describe, in detail, the insertion and removal algorithms for an (a, b)-tree.

R-14.2. Suppose T is a multi-way tree in which each internal node has at least five and at most eight children. For what values of a and b is T a valid (a, b)-tree?

R-14.3. For what values of d is the tree T of the previous exercise an order-d B-tree?

R-14.4. Draw the order-7 B-tree resulting from inserting the following keys (in this order) into an initially empty tree T:

$$\{4, 40, 23, 50, 11, 34, 62, 78, 66, 22, 90, 59, 25, 72, 64, 77, 39, 12\}.$$

Additional Practice Questions

1. Starting with an empty tree, construct a $(2,3)$-tree by inserting the following keys in the order given: $\{2, 3, 5, 6, 9, 8, 7, 4, 1\}$.
2. Show the B-tree of order 3 that results when inserting the keys $\{R, Y, F, X, A, M, C, D, E, T, H, V, L, W, G\}$ in that order.
3. Suppose you have an application in which you want to use B-trees. Suppose that the computer you will be using has disk blocks holding 4096 bytes, the key is 4 bytes long, each child pointer is 4 bytes, the parent is 4 bytes, and the data record reference is 8 bytes. You want to store 1 000 000 items in your B-tree.
   (a) What value would you select for M? Justify your choice.
   (b) What is the maximum number of disk blocks that will be brought into main memory during a search, given your chosen value of M?