Algorithmics

Sebastian Iwanowski
FH Wedel

3. Solutions for the dictionary problem
3.3 Other optimal worst case methods for search trees

Algorithmics 3

3.3 Other optimal worst-case methods for search trees

A dictionary is a data structure for elements comparable by a key implementing the functions member (key), insert (key, newdata) and delete (key)

(a,b)-tree: Generalisation of (2,3)-trees (a ≥ 2, b ≥ 2a-1)

Data structure: search tree with the following properties:

i. All data is stored in the leaves which are all on equal level.
ii. All inner nodes (except for the root) have at least a and at most b children.
The root has at least 2 and at most b children.
iii. For directions, the inner nodes contain for each child the greatest key of the subtree rooted in the respective child.

An (a,b)-tree is height balanced (due to i.)

=> for all 3 dictionary operations: same run time for w.c. and a.c.

due to ii) (+ update of a node in constant time): run time Θ(log n) w.c. and a.c.

References:

Skript Alt S. 44 – 51 (Kap. 3.1.5) in German
Mehlhorn ch. 7

Algorithmics 3

3.3 Other optimal worst-case methods for search trees

A dictionary is a data structure for elements comparable by a key implementing the functions member (key), insert (key, newdata) and delete (key)

AVL tree

Data structure: binary search tree with the following property:

For each node, the height of the children subtrees differs by at most 1.

The maintenance of this invariant is guaranteed by rotations and double rotations of nodes
Which can be executed during insert and delete in constant time per node.

All 3 dictionary operations: same run time Θ(log n) w.c. and a.c.

Proof is complicated (via Fibonacci estimations).

References:

Skript Alt S. 41 – 44 (Kap. 3.1.4) contain both exact functionality and run time proof,
Knuth ch. 6.2.3 not relevant for exam

Algorithmics 3

3.3 Other optimal worst-case methods for search trees

A dictionary is a data structure for elements comparable by a key implementing the functions member (key), insert (key, newdata) and delete (key)

Red-black tree

Data structure: binary search tree, each node being red or black, with the following properties:

i. Each leaf is black.
ii. Red nodes have black children.
iii. Each path from root to any leaf contains the same number of black nodes.

The maintenance of this invariant is guaranteed by rotations and double rotations of nodes which can be executed during insert and delete in constant time per node.

All 3 dictionary operations: same run time $\Theta(\log n)$ w.c. and a.c.

Proof is shown in class.

References:

Skript Alt S. 54 (Kap. 3.1.7) in German
Cormen ch.13 (including algorithmic details and run time estimation)

Algorithmics 3

3.3 Other optimal worst-case methods for search trees

A dictionary is a data structure for elements comparable by a key implementing the functions member (key), insert (key, newdata) and delete (key)

Red-black tree

Prop.:: Full*) red-black trees may be easily transformed to (2,4)-trees and vice versa.

➔: Each black node is combined with its red children.
The children of the red children and the possibly existing other black child of the original node become the children of the combined node.

➔: i. Any node with 2 children becomes a black node with 2 black children.
ii. Any node with 3 children a_1, a_2, a_3 becomes a black node with a black child a_1 and a new red child having a_2, a_3 as black children.
iii. Any node with 4 children becomes a black node with 2 new red children, each having 2 of the former children as own black children.

*) A binary tree is called full (or saturated), if all inner nodes have exactly 2 children.
A binary tree is called complete, if all leaves are located on the lowest levels (completely filled from left to right)
A binary tree is called perfect, if all leaves are located on the same level.

References: Skript Alt S. 54 (Kap. 3.1.7) in German

Algorithmics 3

3.3 Other optimal worst-case methods for search trees

A dictionary is a data structure for elements comparable by a key implementing the functions member (key), insert (key, newdata) and delete (key)

B trees: Special (a,b)-trees

A B tree is equivalent to a (t,2t)-tree (t ≥ 2) "filled at least by half"

A B* tree (Knuth) is defined as ((2m-1)/3,m)-tree "filled at least by 2/3"
where the root may have between 2 and 2·(2m-1)/3 + 1 children.

Practical application: rapid disc access

All 3 dictionary operations: same run time Θ(log n) for w.c. and a.c.

References:

Cormen ch. 18 (B trees)
Knuth ch. 6.2.4

Algorithmics 3

3.3 Other optimal worst-case methods for search trees

A dictionary is a data structure for elements comparable by a key implementing the functions member (key), insert (key, newdata) and delete (key)

Trie tree(from re**trie**val) for strings over a k element alphabet

Data structure: k-ary tree having the following properties:

i. The root is empty.
ii. Each other node contains a letter being used in some string stored.
For each string using this node there is a child containing the next letter of the string.
iii. Each node has a flag indicating if this is the end of a string or not.

All 3 dictionary operations

Run time: Θ(length of string) w.c. and a.c. (clear)

Run time for a search in a tree storing n strings:
$\log_k(n)$ a.c. (Knuth, summary)

Space for a tree storing n strings: n/ln k a.c. (Knuth, summary)

References:

Skript Alt S. 54 – 56 (Kap. 3.1.8) in German
Knuth ch. 6.3 (Digital Searching) with several improvements, not relevant for exam