Set 12: Ordered Dictionary Abstract Data Types: $(2, 4)$ Trees, B Trees

CS240: Data Structures and Data Management

Jérémy Barbay

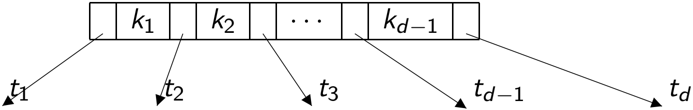
Outline

Multi-way search trees

Definition

A d-node is an internal node with

- d children, $t_1, \ldots, t_d$, and
- $d - 1$ keys such that $k_1 < k_2 < \ldots < k_{d-1}$.

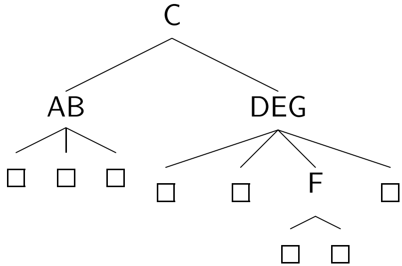


This generalizes binary search trees to larger degrees.

Multi-Way Search Trees

Definition

A Multi-Way Search Tree is an ordered search tree consisting of linked d-nodes, where each node may have a different value for d:



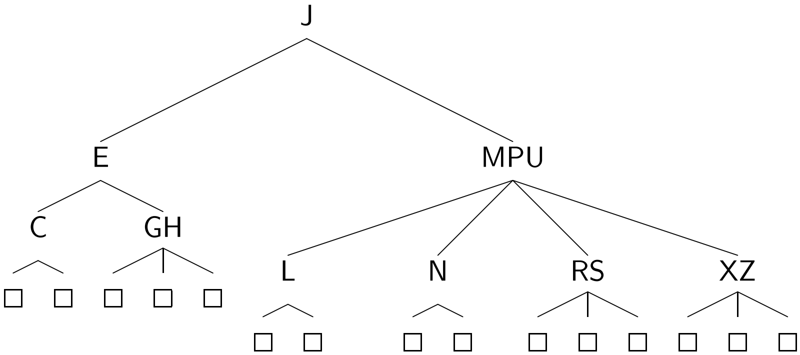
It is searched almost like a binary tree.

$(2,4)$-Trees

Definition

A (2,4)-tree is a multi-way search tree such that

- Every node has between 2 and 4 children
- All external nodes have the same depth

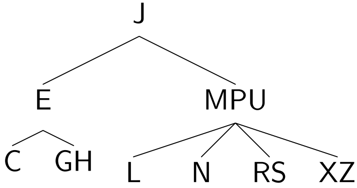


(2, 4)-Trees

Definition

A (2,4)-tree is a multi-way search tree such that

- Every node has between 2 and 4 children
- All external nodes have the same depth



Note: As all external nodes have the same depth, placeholders don't carry much information anymore, and can be omitted.

Properties

Theorem

Consider a $(2,4)$-tree with n internal keys:

1. *The number of external placeholders is $|E| = n + 1$.*
2. *The height h is $\Theta(\log n)$.*

Proof: Exercise.

Insertion

- Find deepest node where the key belongs, and insert.
- If **overflow**, perform a **node split**:

BCD

`Insert( A )`

C

AB D

OR

BCD

`Insert( E )`

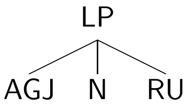
D

BC E

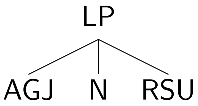
- The third element (counting the new element) moves up to parent, possibly causing a new overflow.

Insertion

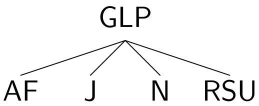
Example



`Insert( S )`

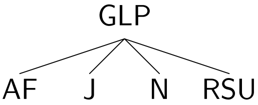


`Insert( F )`

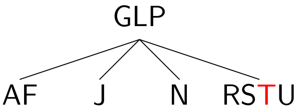


Insertion

Example (cont)

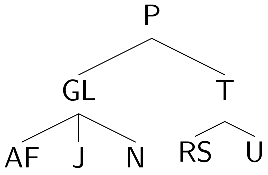


`Insert( T )`



Insertion

Example (end)

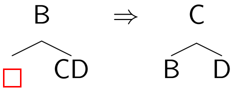


Theorem

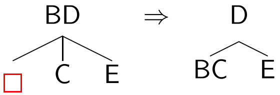
Insertion requires at most operations.

Deletion

- Search for the key
- As in an AVL:
  - if it has no children, remove it.
  - if it has children, replace it with in-order predecessor or successor.
- If too few keys (**underflow**), then
  - **Transfer** a node from an *immediate sibling* if possible

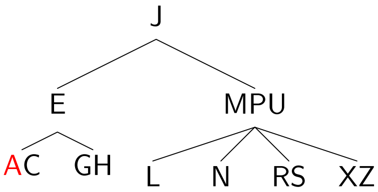


  - Otherwise, **fuse** with an *immediate sibling* and parent element



Deletion

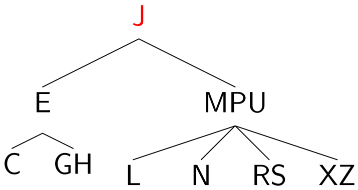
Example




```
Delete( A )
```

Deletion

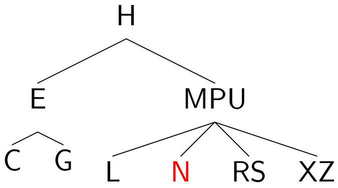
Example (cont)



`Delete( J )`

Deletion

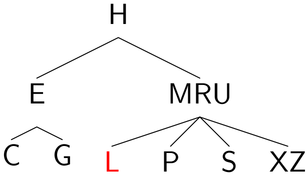
Example (cont)




```
Delete( N )
```

Deletion

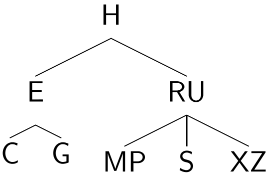
Example (cont)



`Delete( L )`

Deletion

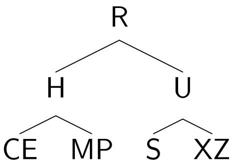
Example (cont)




```
Delete( G )
```

Deletion

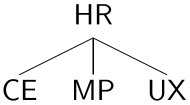
Example (cont)




```
Delete( Z )
Delete( S )
```

Deletion

Example (end)



Theorem

Deletion requires at most operations.

Summary for $(2,4)$-trees

- search easy
- insert may involve several splittings.
- deletion may involve one transfer, or several fusions.
- h increases only if root is split.
- h decreases only if root's sibling's fuse,
  and root becomes empty.
- $O(\log n)$ since constant amount of work at each node.

Outline

B-Trees

Definition

- A generalization of $(2, 4)$-trees
- A B-Tree of order d ($d \geq 3$) is a multi-way search such that
  - every node has $\leq d$ children
  - every *non-root* node has $\geq \lceil \frac{d}{2} \rceil$ children
  - all the external nodes have the same depth
- Often called an (a, b)-tree where $a = \lceil \frac{d}{2} \rceil$ and $b = d$
- The operations are performed the same as before
  - For overflow we promote element $\lceil \frac{d+1}{2} \rceil$
    (counting the new element)

B-Tree of Order 6

Example

Also known as a $(3, 6)$-tree:

$$BCDEF$$

- `Insert( A )`
- `Delete ( F )`
- `Delete ( B )`

Motivations

External Searching

- What if the dictionary cannot fit in main memory?
- Need to store data in persistent memory (e.g. on disks)
- A single access to the data structure takes much longer
  - RAM Seek – 100 000 memory accesses
  - Disk Seek – for 1 disk access
- A disk access brings in a whole page of data
- Assume we can fit B dictionary elements per page:
  How many disk accesses would binary search (or an AVL tree) require?

B-Trees

- **Problem**: One disk access only cuts range of keys in half
- **Solution**: Use a B-Tree of order d
  - Choose d such that one d-node fills exactly one disk page
  - One disk access narrows search a lot more
- Searching the d-node is still far less expensive than bringing it into memory
- Running time is proportional to the number of blocks read
  - Insert and delete designed to reduce the number of d-nodes examined

Performance

- Suppose $d = 256$
- Minimum and maximum number of keys *found at each depth*:

Depth	Minimum # Keys	Maximum # Keys
0	1	255
1	254	65,280
2	32,512	16,711,680
3	4,161,536	4,278,190,080
4	532,676,608	1.1×10^{12}

Property

Theorem

The height h of a B-tree of order d is

- $\Omega(\ \log_d(n)\ )$
- $O(\ log_{\lceil \frac{d}{2} \rceil}(n)\ )$

Proof.

We know $2\lceil \frac{d}{2} \rceil^{h-1} \leq |E| \leq d^h$ combined with $|E| = n + 1$ gives the result. $\square$

One-Pass Update

- Insert and delete require two passes
  - First pass down tree finds the bottom level node
  - Second pass up tree performs splitting or fusing
- Algorithm can be reworked to perform preemptive splits and fusions on the way down
- See CLRS textbook for details
  - You will never need to perform this in our class

B*-Tree

- Each non-root d-node could have as low as 50% utilization
- On average a node is 69% filled
- We could insist that a non-root node is at least $\frac{2}{3}$ filled
- More difficult to do a node split or fusion
- A B^*-Tree node is 90% filled on average

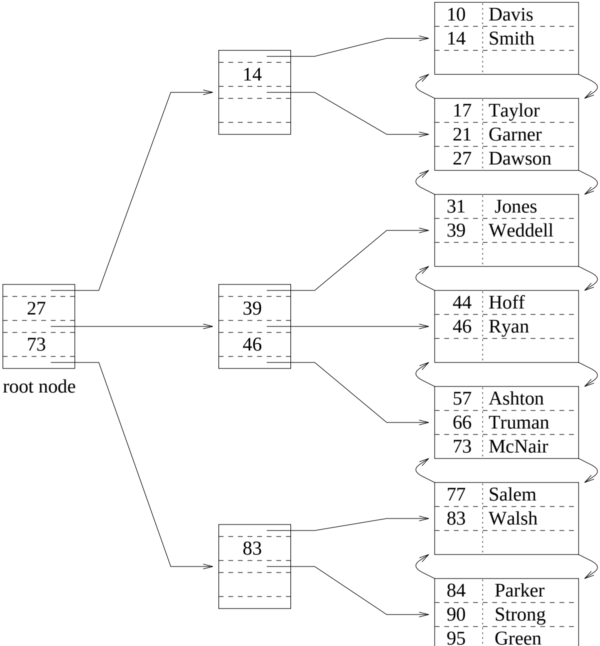
Improvements

B^+-Tree

$\mathbf{B^+}$-Tree

- Desire greatest branching possible
- Internal nodes contain only keys (not the corresponding satellite data)
- Bottom level of tree contains the real key-data pair
- We also wish to perform **Range Queries**
  - List all professors with id-numbers between 39 and 75
- Each node has a pointer to the next and previous bottom level page

B^+-Tree


27
73
root node
14
39
46
83
10 Davis
14 Smith
17 Taylor
21 Garner
27 Dawson
31 Jones
39 Weddell
44 Hoff
46 Ryan
57 Ashton
66 Truman
73 McNair
77 Salem
83 Walsh
84 Parker
90 Strong
95 Green

Summary for B-trees

- Generalisation of $(2, 4)$-trees
- Usefull for large dictionaries,
  which does not fit in memory.
- Several variants, corresponding to various needs.
- Very important in practical Databases.

Outline

Ordered Dictionary ADTs and their DS

- Array
- Binary Search Tree (BST)
- Sequence (Skip Lists)
- AVL
- $(2, 4)$ Trees
- B-Trees

Diferent solutions to different problems...

References

	GT	CLRS
Arrays	pp. 140-151	pp. 253-264
BST		
Skiplists	pp. 195-202	Not covered.
AVL	pp. 152-158	pp. 296 (poorly covered)
$(2, 4)$-trees	pp. 159-169	pp, 434-452 (indirectly)
B-Trees	pp. 649-653	pp. 434-452