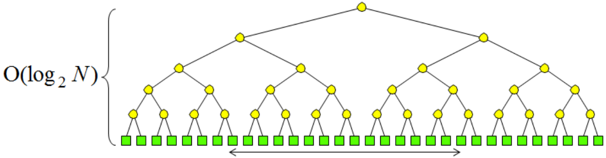
Massive Data Algorithmics

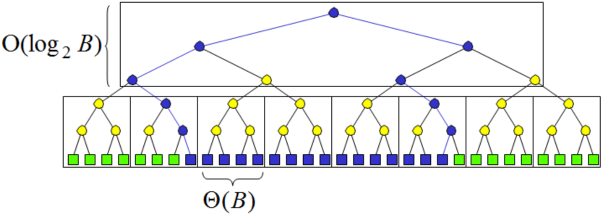
Lecture 3: External Search Trees

Binary search tree

- Standard method for search among N elements
- We assume elements in leaves
- Search traces at least one root-leaf path
- If nodes stored arbitrarily on disk
  $\Rightarrow$ Search in $O(\log_2 N)$ I/Os
  $\Rightarrow$ Range-search in $O(\log_2 N + T)$ I/Os



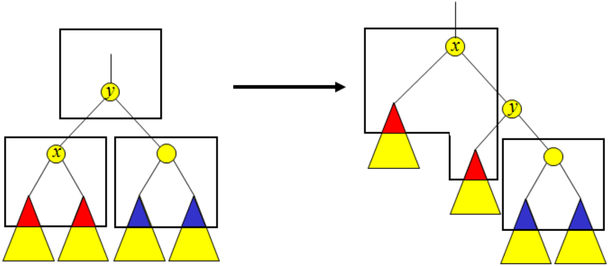
BFS Blocking



- Block height: $O(\log_2 N)/O(\log_2 B) = O(\log_B N)$
- Output elements blocked $\Rightarrow$ Range-search in $O(\log_B N + T/B)$ I/Os
- Optimal: $O(N/B)$ space and $O(\log_B N + T/B)$ query

Updating

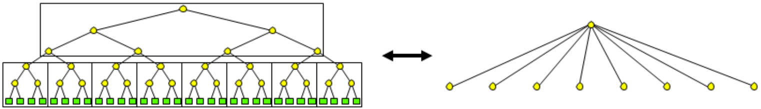
- Maintaining BFS blocking during updates?
  - Balance normally maintained in search trees using rotations



- Seems very difficult to maintain BFS blocking during rotation
  - Also need to make sure output (leaves) is blocked!

B-trees

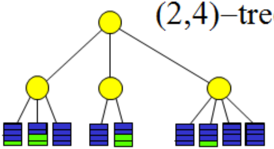
- BFS-blocking naturally corresponds to tree with fan-out $\theta(B)$



- B-trees balanced by allowing node degree to vary
  - Re-balancing performed by splitting and merging nodes

(a,b)-Trees

- T is an (a,b)-tree ($a \geq 2$ and $b \geq 2a-1$)
  - All leaves on the same level and contain between a and b elements
  - Except for the root, all nodes have degree between a and b
  - Root has degree between 2 and b

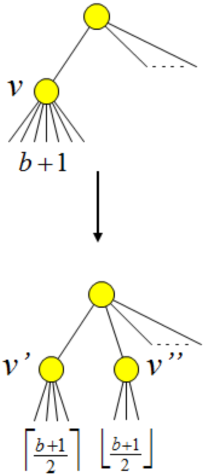


- (a,b)-tree uses linear space and has height $O(\log_a N)$
- Choosing $a,b = \Theta(B)$, each node/leaf stored in one disk block
- $O(N/B)$ space and $O(\log_B N + T/B)$ query

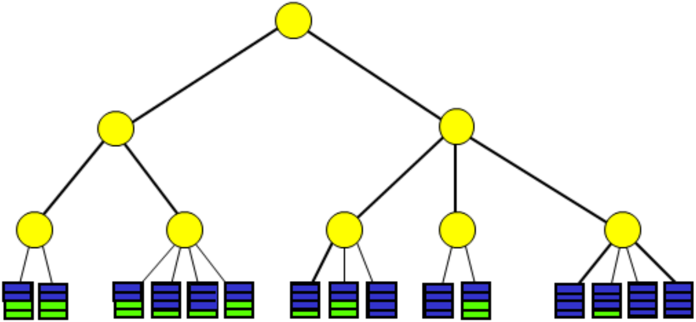
(a,b)-Trees Insert

- Search and insert element in leaf v
- DO v has $b+1$ elements/children
  - make nodes v' and v'' with $\lfloor (b+1)/2 \rfloor$ and $\lceil (b+1)/2 \rceil$ elements
  - insert element (ref) in parent(v) (make new root if necessary)
- $v = \text{parent}(v)$

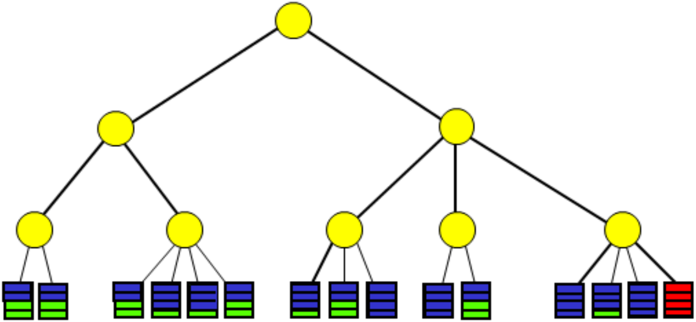
Insert touch $O(\log_a N)$ nodes



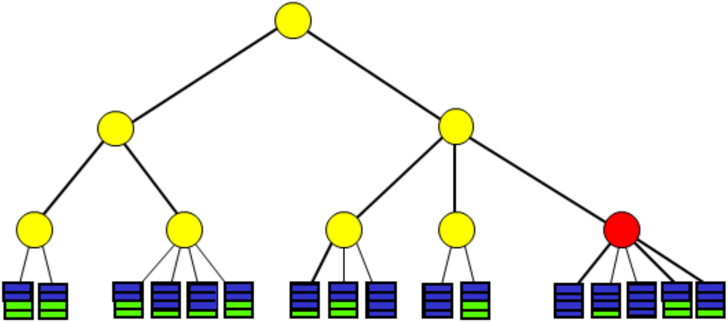
Example: $(2,4)$-Tree Insert



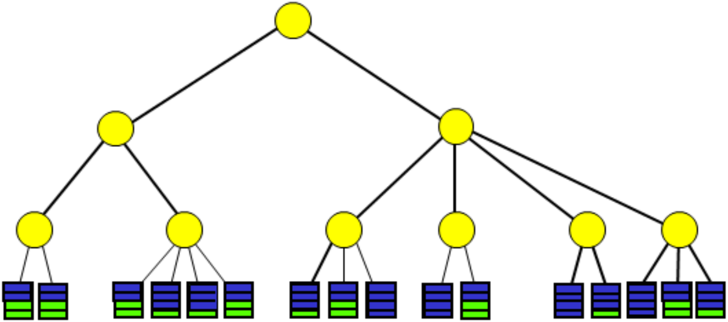
Example: $(2,4)$-Tree Insert



Example: $(2, 4)$-Tree Insert

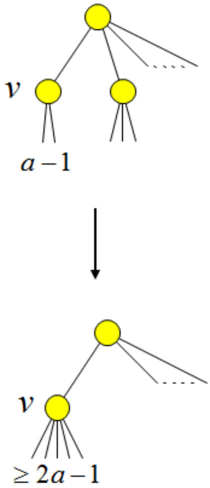


Example: $(2,4)$-Tree Insert



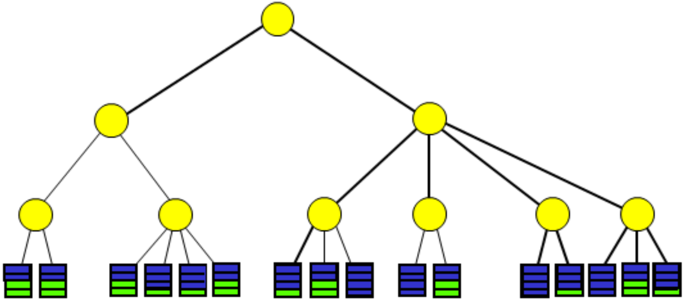
(a,b)-Trees Deletion

- Search and delete element from leaf v
- DO v has $a-1$ elements/children
  - Fuse v with sibling v':
    - move children of v' to v
    - delete element (ref) from parent(v) (delete root if necessary)
  - If v has $> b$ (and $\leq a+b-1 < 2b$) children split v
- $v = \text{parent}(v)$

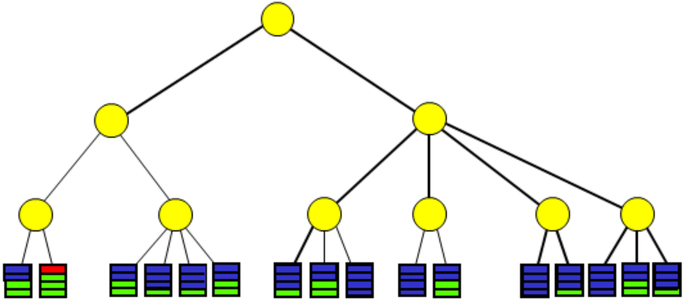


Delete touch $O(\log_a N)$ nodes

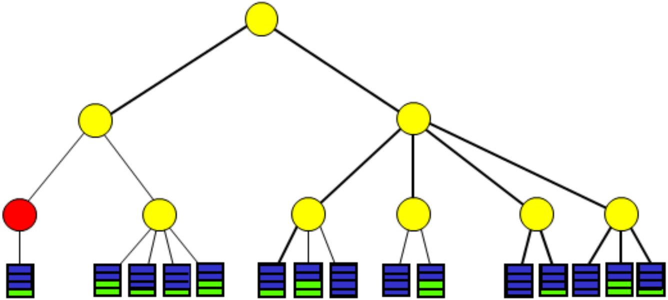
Example: $(2,4)$-Tree Delete



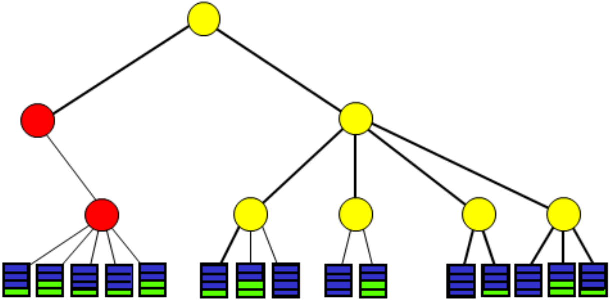
Example: $(2, 4)$-Tree Delete



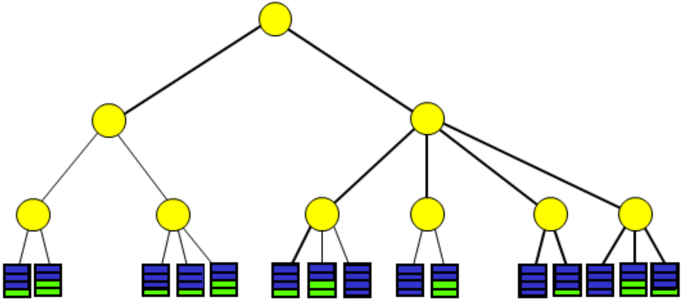
Example: $(2,4)$-Tree Delete



Example: $(2,4)$-Tree Delete

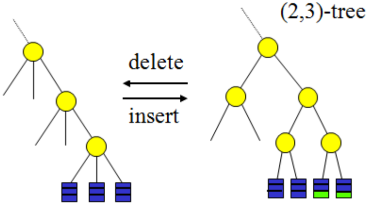


Example: $(2, 4)$-Tree Delete



(a,b)-Trees Properties

- If $b = 2a - 1$ every update can cause many re-balancing operations



- If $b \geq 2a$ update only cause $O(1)$ re-balancing operations amortized
- If $b > 2a$ only $O(1/(b/2-a)) = O(1/a)$ re-balancing operations amortized
  *Both somewhat hard to show
- If b=4a easy to show that update causes $O(1/a \log_a N)$ re-balance operations amortized
  * After split during insert a leaf contains $\cong 4a/2 = 2a$ elements
  * After fuse during delete a leaf contains between $\cong 2a$ and $\cong 5a$ elements (split if more than $3a \Rightarrow$ between $3/2a$ and $5/2a$)

Summary and Conclusion: B-trees

- B-trees: (a,b)-trees with $a,b=\Theta(B)$
  - $O(N/B)$ space
  - $O(\log_B N + T/B)$ query
  - $O(\log_B N)$ update
- B-trees with elements in the leaves sometimes called B^+-trees
- Construction in $O(N/B \log_{M/B} N/B)$ I/Os
  - Sort elements and construct leaves
  - Build tree level-by-level bottom-up

Summary and Conclusion: B-trees

- B-tree with branching parameter b and leaf parameter k ($b,k \geq 8$)
  - All leaves on same level and contain between $1/4k$ and k elements
  - Except for the root, all nodes have degree between $1/4b$ and b
  - Root has degree between 2 and b
- B-tree with leaf parameter $k = \Omega(B)$
  - $O(N/B)$ space
  - Height $O(\log_b N/B)$
  - $O(1/k)$ amortized leaf rebalance operations
  - $O(1/(bk)\log_b N/B)$ amortized internal node rebalance operations
- B-tree with branching parameter B^c, $0 < c \leq 1$, and leaf parameter B
  - Space $O(N/B)$, updates $O(\log_B N)$, queries $O(\log_B N + T/B)$

References

- **External Memory Geometric Data Structures**
  Lecture notes by Lars Arge.
  - Section 1-3