External Memory II

- Searching with Fast Updates
- Searching Strings

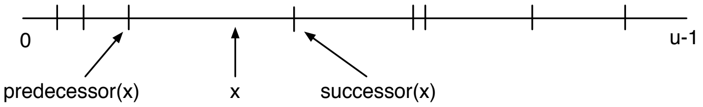
Philip Bille

External Memory II

- Searching with Fast Updates
- Searching Strings

Searching

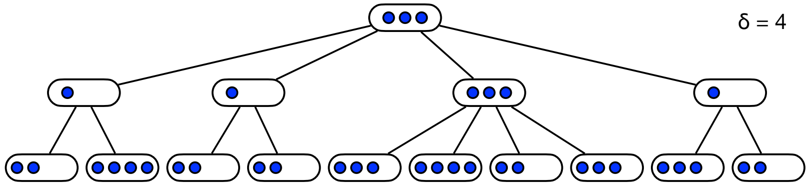
- Searching. Maintain a set S ⊆ U = {0, ..., u-1} supporting
  - member(x): determine if x ∈ S
  - predecessor(x): return largest element in S ≤ x.
  - successor(x): return smallest element in S ≥ x.
  - insert(x): set S = S ∪ {x}
  - delete(x): set S = S - {x}



Searching

- Applications.
  - Relational data bases.
  - File systems.

B-tree

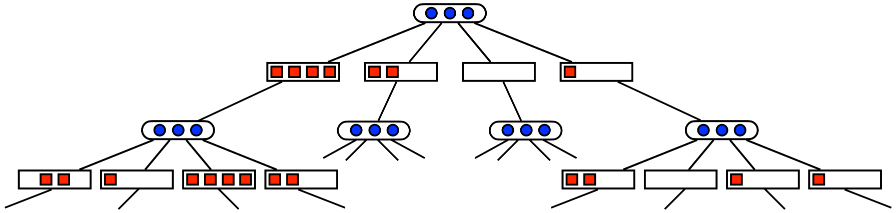


- B-tree of order δ = Θ(B) with N keys.
  - Keys in leaves. Routing elements in internal nodes.
  - Degree between δ/2 and δ.
  - Root degree between 2 and δ.
  - Leaves store between δ/2 and δ keys.
  - All leaves have the same depth.
- Height. $\Theta(\log_\delta (N/B)) = \Theta(\log_B N)$
- Search and update. $O(\log_B N)$ I/Os.

B^ε-tree

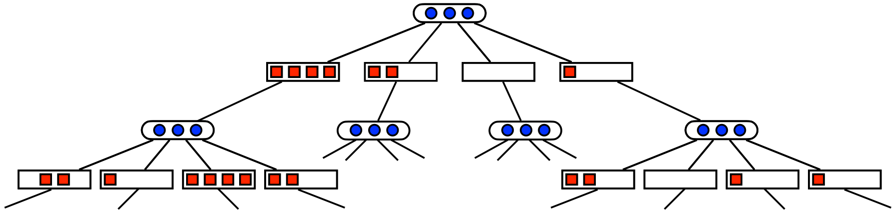
- Idea.
  - Speed up updates by buffering them at each node along the path to a leaf.
  - Move many updates together in each I/O.
  - Search (almost) as before.
  - $\varepsilon \in (0, 1]$ is a parameter.
- Solution in 2 steps.
  - Focus on $\sqrt{B}$-tree (ε = 1/2).
    - Searching in $O(\log_B N)$ I/Os.
    - Updates in $O((\log_B N)/\sqrt{B})$ amortized.
  - Generalize to any ε.

$\sqrt{B}$-tree



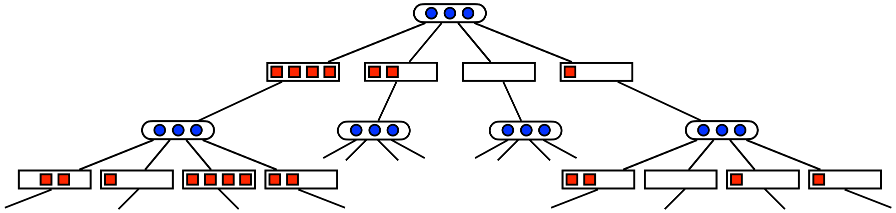
- $\sqrt{B}$-tree with N keys.
  - B-tree of degree $\Theta(\sqrt{B})$ with buffers of size $\Theta(\sqrt{B})$ at each edge.
  - Buffer stores delayed updates in subtree.
  - Nodes and child buffers stored together in O(1) blocks.
- Height. $\Theta(\log_{\sqrt{B}} N) = \Theta(\log_B N)$

$\sqrt{B}$-tree



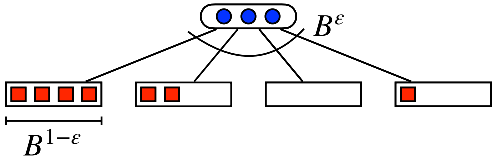
- Searching.
  - Find leaf using routing elements. Check buffers along path.
- I/Os. $O(\log_B N)$.

$\sqrt{B}$-tree



- Updates.
  - Insert update message into buffer at child.
  - If buffer full, flush and recurse at child.
  - If we fill leaf, rebalance tree as B-tree.
- I/O intuition. A flush moves $\sqrt{B}$ messages together ⇒ O((log$_B$ N)/$\sqrt{B}$) amortized I/Os.
- I/Os.
  - Assign (ch)/$\sqrt{B}$ credits to each update, where h = O(log$_B$ N) is height and c > 1 is appropiate constant.
  - Put c/$\sqrt{B}$ credits each node on path.
  - ⇒ We can pay for buffer overflows and rebalancing.

B^{ε}-tree



- B^{ε}-tree with N keys.
  - B-tree of degree Θ(B^{ε}) with buffers of size Θ($B^{1-\varepsilon}$) at each edge.
- Searching. $O\left(\frac{\log_B N}{\varepsilon}\right)$ I/Os.
- Updates. $O\left(\frac{\log_B N}{\varepsilon B^{1-\varepsilon}}\right)$ I/Os.

B^{ε}-tree

	Search	Update
B-tree	$O(\log_B N)$	$O(\log_B N)$
$\sqrt{B}$-tree	$O(\log_B N)$	$O\left(\frac{\log_B N}{\sqrt{B}}\right)$
B^{ε}-tree	$O\left(\frac{\log_B N}{\varepsilon}\right)$	$O\left(\frac{\log_B N}{\varepsilon B^{1-\varepsilon}}\right)$

External Memory II

- Searching with Fast Updates
- Searching Strings

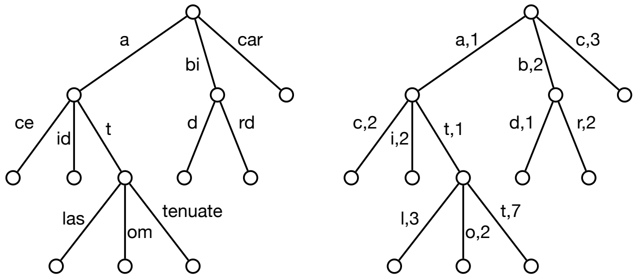
String Searching

- String searching. Maintain a set S = {S_1, S_2, ..., S_K} of K strings of total length N supporting the following operations:
  - search(P): return string in S with longest common prefix with P.
  - insert(P): set S = S ∪ {P}
  - delete(P): set S = S - {P}

String Searching

- Goal.
  - Searching in O(|P|/B + $\log_B$ K) I/Os.
  - Ignore insert and delete.
- Solution in 3 steps.
  - Blind tries.
  - String B-trees.
  - String B-trees with fast searches.

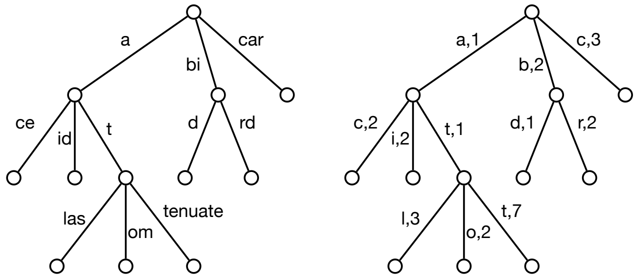
Blind Trie



1. ace
2. aid
3. atlas
4. atom
5. attenuate
6. bid
7. bird
8. car

- Data structure.
  - Sorted set of strings.
  - Compact trie for S. Edges store first char + string length.
- Space. Strings + trie: O(N + K) = O(N).

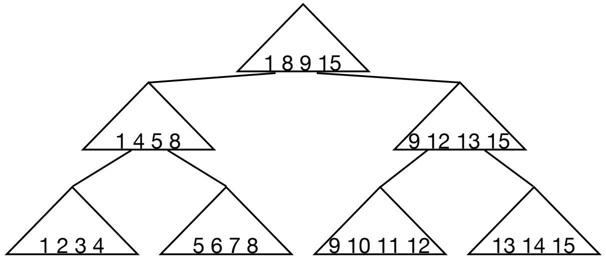
Blind Trie



1. ace
2. aid
3. atlas
4. atom
5. attenuate
6. bid
7. bird
8. car

- Search. Traverse and verify candidate.
- Time. O(|P|)

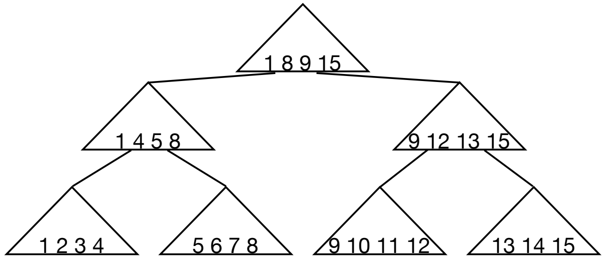
String B-tree



1. ace
2. aid
3. atlas
4. atom
5. attenuate
6. bid
7. bird
8. car
9. cod
10. dog
11. fit
12. lid
13. patent
14. sun
15. zoo

- Data structure. Combination of B-tree and blind tries.
  - Sorted set of strings.
  - Nodes store blind trie over B strings.
  - Leftmost and rightmost string in subtree stored for each child.
- Space. Strings + O(B) per node: O(N + N) = O(N)

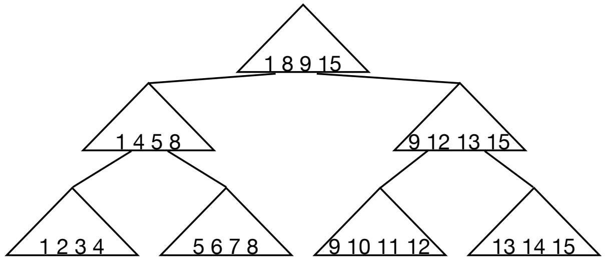
String B-tree



1. ace
2. aid
3. atlas
4. atom
5. attenuate
6. by
7. bye
8. car
9. cod
10. dog
11. fit
12. lid
13. patent
14. sun
15. zoo

- Searching.
  - Traverse and verify at each node.
- I/Os.
  - O(|P|/B + 1) I/Os at each node.
  - ⇒ $O((|P|/B)\log_B K)$ I/Os total.

String B-tree



1. ace
2. aid
3. atlas
4. atom
5. attenuate
6. by
7. bye
8. car
9. cod
10. dog
11. fit
12. lid
13. patent
14. sun
15. zoo

- Fast searching.
  - Traverse and verify at each node.
  - But: remember longest prefix matched at each node.
- I/Os.
  - O(|P|/B) I/Os in total for string verification. O(1) I/Os at each node.
  - ⇒ $O((|P|/B) + \log_B K)$ I/Os.

External Memory II

- Searching with Fast Updates
- Searching Strings