External memory II: B-trees

02282 Algorithms for Massive Data Sets

Patrick Hagge Cording

DTU Compute

Department of Applied Mathematics and Computer Science

Search trees

- Store a set of integer keys
- B-tree is a classic search tree used in theory and practice

Why bother?

- Searching is used everywhere
- Pointer-based structures incur many cache misses
- Even implicit tree structures can be bad

Outline

B-trees in external memory

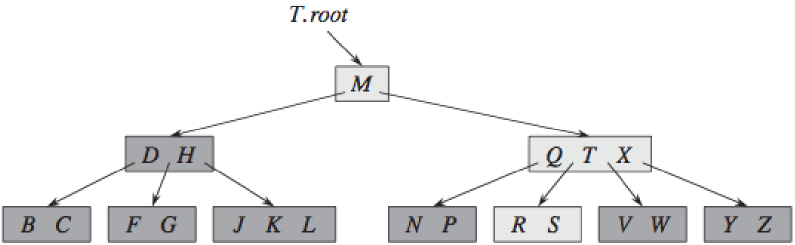
- Static and dynamic

Cache-oblivious B-trees

- Static
- Ordered-file maintenance
- Dynamic

B-trees in external memory

- Internal nodes have $\Theta(B)$ children
- Tree has height $O(\log_B N)$

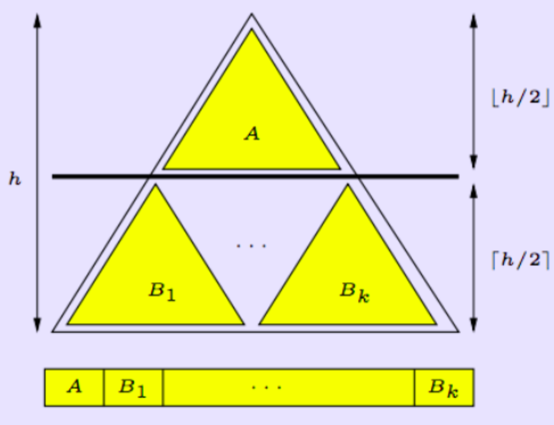


- Search and update requires $O(\log_B N)$ IOs (see CLRS)

(Image from CLRS)

Static cache-oblivious B-trees: Layout

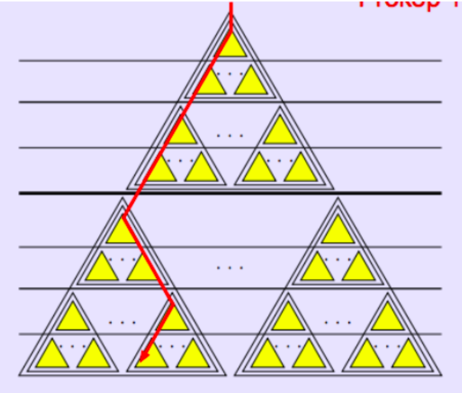
- van Emde Boas layout of binary tree



- $O(N)$ space

(Image due to Gerth Stølting Brodal)

Static cache-oblivious B-trees: Searching



- Yellow substrees have size $\sqrt{B} \leq x \leq B$
- Path will visit $\leq \frac{\log N+1}{\log \sqrt{B}} = O(\log_B N)$ yellow subtrees
- $\Rightarrow O(\log_B N)$ IOs

(Image due to Gerth Stølting Brodal)

Outline

B-trees in external memory

- Static and dynamic

Cache-oblivious B-trees

- Static
- **Ordered-file maintenance**
- Dynamic

Problem definition

Store an ordered sequence of N elements (integers) in an array with constant sized gaps subject to insertions and deletions.

- $\textsc{Insert}(k, p)$: Insert the key p after the element with key k
- $\textsc{Delete}(k)$: Delete element with key k

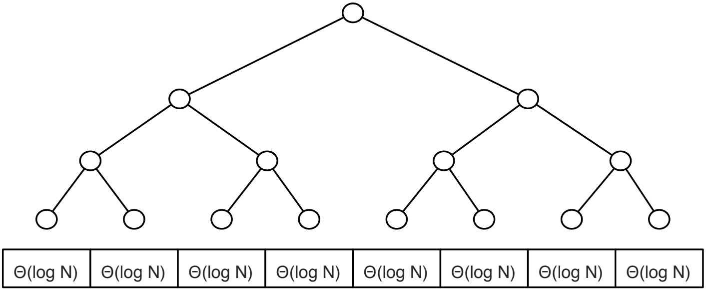
$O(N)$ space and amortized $O(\frac{\log^2 N}{B})$ IOs for $\textsc{Insert}$ and $\textsc{Delete}$.

Simple solution

- Rewrite entire array for INSERT and DELETE
- Double/half array when full/below threshold
- $O(N)$ space and $O(N/B)$ IOs

Data structure

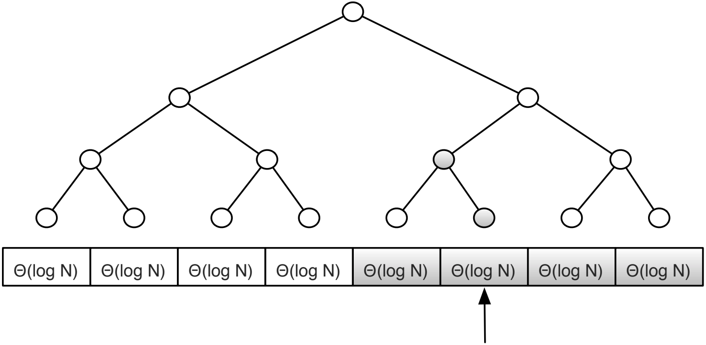
- A node v stores its density $D(v) = \frac{\text{number of elements}}{\text{capacity}}$
- Initially, $\frac{1}{2} - \frac{1}{4}d/h \leq D(v) \leq \frac{3}{4} + \frac{1}{4}d/h$
- *Threshold* range increase with depth



Operations

INSERT:

- Redistribute block
- If block is full
    - Find deepest node within threshold
    - Evenly redistribute entire subrange
    - ⇒ All descendants are within range



Analysis

- Claim: Suppose v has capacity K and has just been redistributed. At least $\Theta(K/\log N)$ elements must be inserted before v's range is redistributed

- Accounting method:
  - Pay $O(\log^2 N)$ to insert
  - $O(\log N)$ is payed for redistributing the leaf
  - Each node on path to root gets $O(\log N)$ credit
  - When internal node with capacity K is redistributed it has at least $O(K)$ credit

- INSERT requires redistribution of amortized $O(\log^2 N)$ elements
  $\Rightarrow$ amortized $O(\frac{\log^2 N}{B})$ IOs (because we use scanning)
- Similar analysis for DELETE

Outline

B-trees in external memory

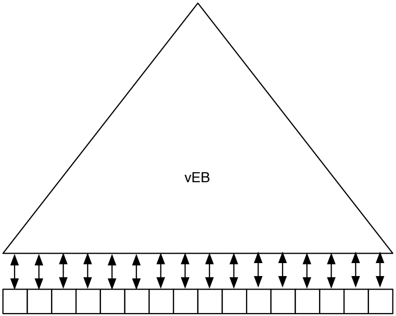
- Static and dynamic

Cache-oblivious B-trees

- Static
- Ordered-file maintenance
- **Dynamic**

Dynamic cache-oblivious B-trees

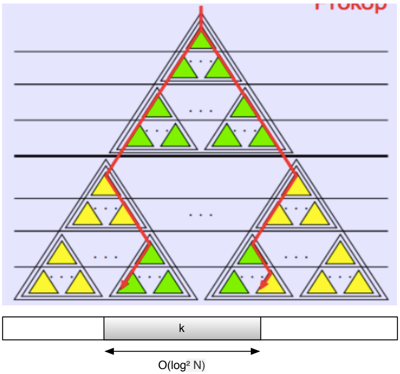
- Put elements in ordered-file maintenance data structure
- Static search tree with vEB layout on top



- Size of array: $O(N)$
- Size of tree: $2 \cdot O(N) - 1 = O(N)$
- Searching: $O(\log_B N)$ IOs

Dynamic cache-oblivious B-trees: Updates

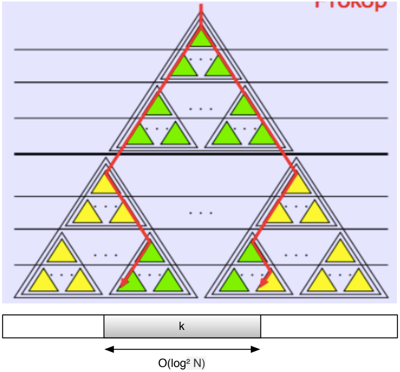
- Search for location
- Update ordered-file maintenance-array
- Propagate changes to tree (post-order)



- Rebuild entire structure if it grows/shrinks too big/small

Dynamic cache-oblivious B-trees: Updates

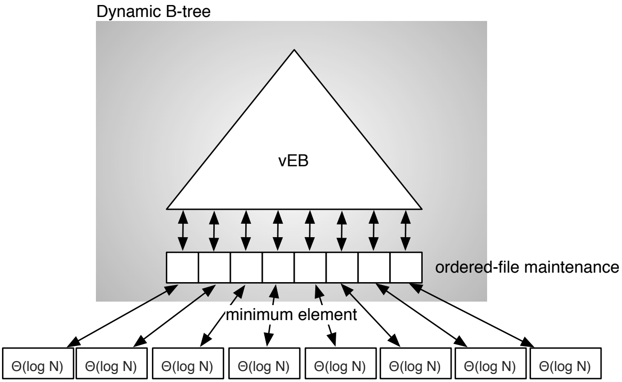
- Search for location $O(\log_B N)$ IOs
- Update ordered-file maintenance-array amortized $O(\frac{\log^2 N}{B})$ IOs
- Propagate changes to tree (post-order) $O(\frac{\log^2 N}{B})$ IOs



- Rebuild entire structure if it grows/shrinks too big/small

Dynamic cache-oblivious B-trees: Final tweaks

- So far: $O(\log_B N)$ IOs for search and $O(\log_B N + \frac{\log^2 N}{B})$ IOs for updates
- Add a layer of indirection!



- Changes to tree only occurs for every $\Omega(\log N)$ updates $\Rightarrow$ amortized $O(\log_B N + \frac{\log N}{B}) = O(\log_B N)$ IOs for updates

Outline

B-trees in external memory

- Static and dynamic — $O(\log_B N)$ IOs

Cache-oblivious B-trees

- Static — $O(\log_B N)$ IOs
- Ordered-file maintenance — amortized $O(\frac{\log^2 N}{B})$ IOs
- Dynamic — $O(\log_B N)$ IOs for search and amortized $O(\log_B N)$ IOs for updates