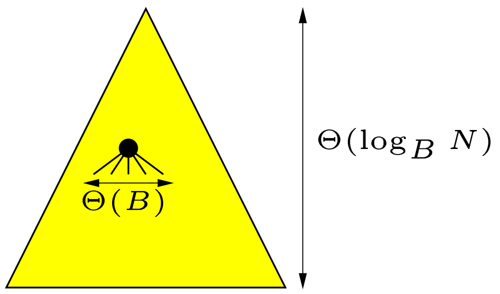
Cache-Oblivious B-trees

Michael A. Bender, Erik Demain, and Martin Farach-Colton. *Cache-Oblivious B-Trees*. In Proc. 41th Annual Symposium on Foundations of Computer Science, FOCS '00, November 12-14, 2000, Redondo Beach, California, USA.

B-trees

Definition

- All leaves have the same depth
- All internal nodes have degree at most B
- The root has degree at least two
- All internal nodes except the root have degree at least $B/2$



Properties

- Search tree
- Searches
- Insertions and deletions

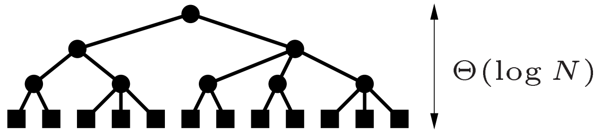
$\left.\right\}$ $O(\log_B N)$ I/Os

- Cache-aware, since degrees $\Theta(B)$

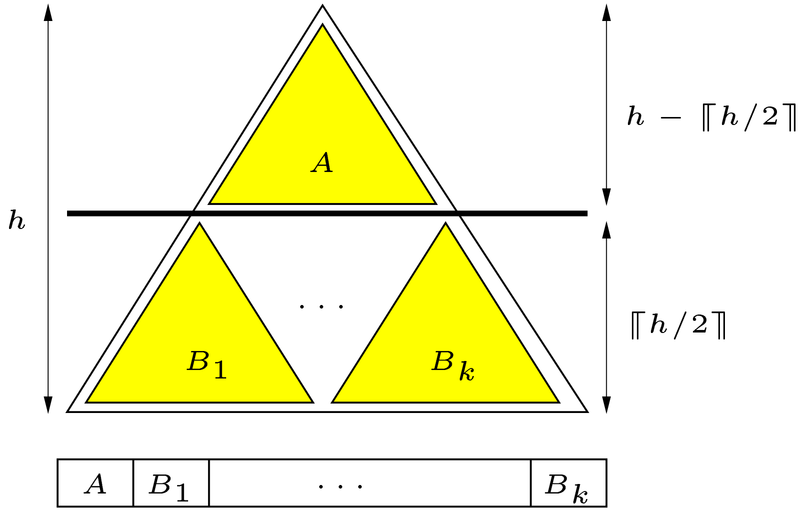
Layout of Static Trees

$O(1)$ **degree tree** $\longrightarrow$ **Recursive memory layout**

- All leaves equal depth
- Each node $O(1)$ children
- Height $\Theta(\log N)$



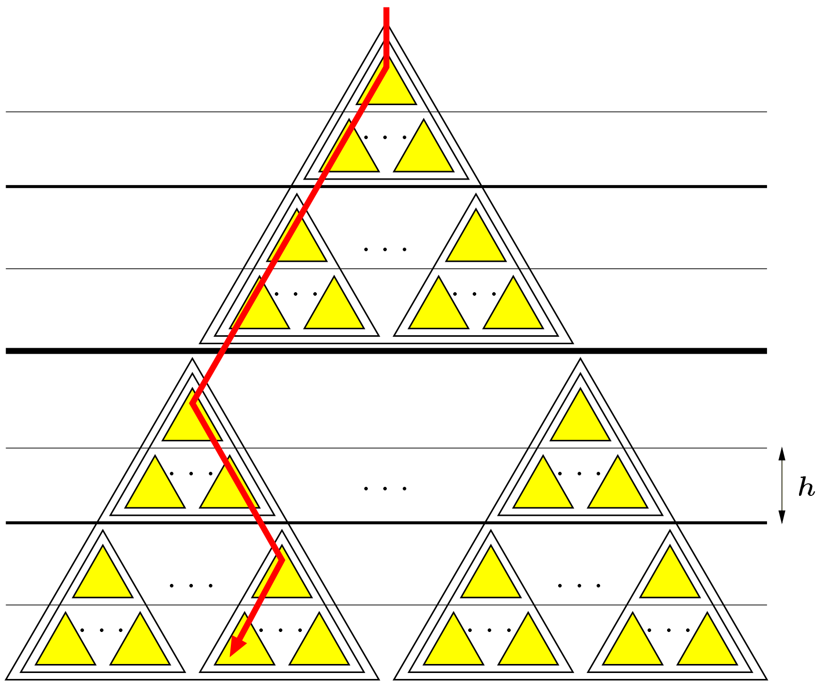
- van Emde Boas layout



$$\llbracket x \rrbracket = 2^{\lceil \log x \rceil}$$

Invariant Recursive structures are stored in contiguous blocks of memory

van Emde Boas Layout



If $\delta \leq \text{degree}(v) \leq \Delta$ for all v in T then #I/Os for a search

$$O\left(\frac{\log_\delta N}{\log_\Delta B}\right) = O(\log_\Delta \delta \cdot \log_B N) = O(\log_B N)$$

since for $h = \llbracket \log_\Delta B \rrbracket$ at most 2 I/Os required to read subtree

Strongly Weight Balanced B-trees

[Arge & Vitter, 1996]

B-trees where degree constraints are replaced by weight constraints

- $d \geq 5$
- All leaves have the same depth
- The root has degree at least two
- All internal nodes u have a weight $\omega(u) = \#$ leaves below u

$$\frac{1}{2} \cdot d^{h-1} \leq \omega(u) \leq 2 \cdot d^{h-1}$$

  where h is the height of u

Fact

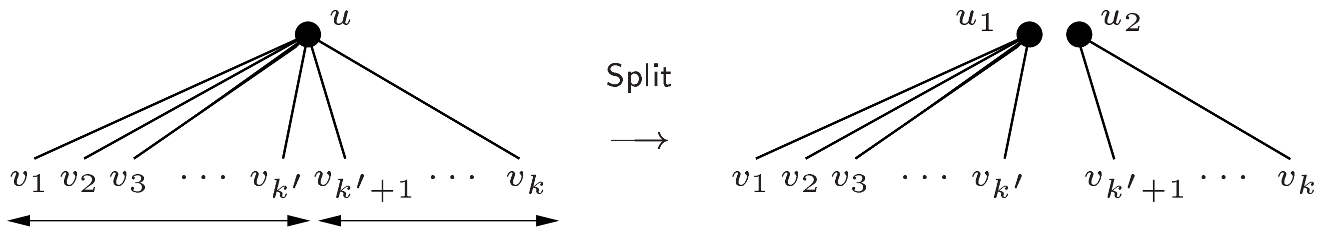
- $2 \leq$ degree of the root $\leq 4d$
- $d/4 \leq \mathrm{degree}(u) \leq 4d$ for all internal nodes u
- Height $O(\log_d N)$

Strongly Weight Balanced B-trees

Insertions

Like B-trees; rebalancing is done bottom-up by the following split

- If $\omega(u) > 2d^{h-1}$ then u is split into two nodes



where k' is max such that $\sum_i^{k'} \omega(v_i) \leq \lceil \omega(u)/2 \rceil$

Property

$$\frac{1}{2} \cdot d^{h-1} + \Theta(d^{h-1}) \leq \omega(u_j) \leq 2 \cdot d^{h-1} - \Theta(d^{h-1})$$

$\Omega(d^{h-1})$ updates below u_j before rebalancing at u_j necessary again

Deletions

Like B-trees; but rebalancing using weight ...

Strongly Weight Balanced B-trees

Lemma

When rebalancing a node all nodes in the subtree can be touched within the same amortized bounds

[Important property that e.g. is used in applications where each node has a secondary data structure holding information about all nodes in the subtree rooted at the node]

Lemma

Searching in a strongly weight balanced tree stored in the van Boas layout costs at most $O(\log_B N)$ I/Os

Packed Memory Management

Maintain a list $x_1, x_2, \ldots, x_N$ in an array of size $O(N)$ such that

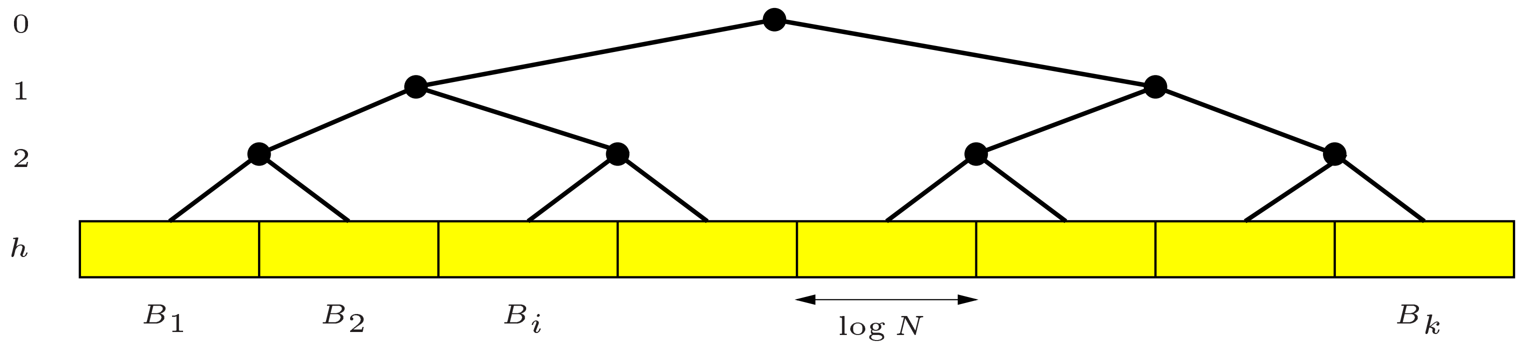
- Elements appear in correct order
- k consecutive elements $x_i, x_{i+1}, \ldots, x_{i+k-1}$ appear in a contiguous subarray of size $\Theta(k)$
- while the list is updated by insertions and deletions

Theorem

The packed memory problem can be solved in $O(1 + \frac{\log^2 N}{B})$ I/Os per insert and delete

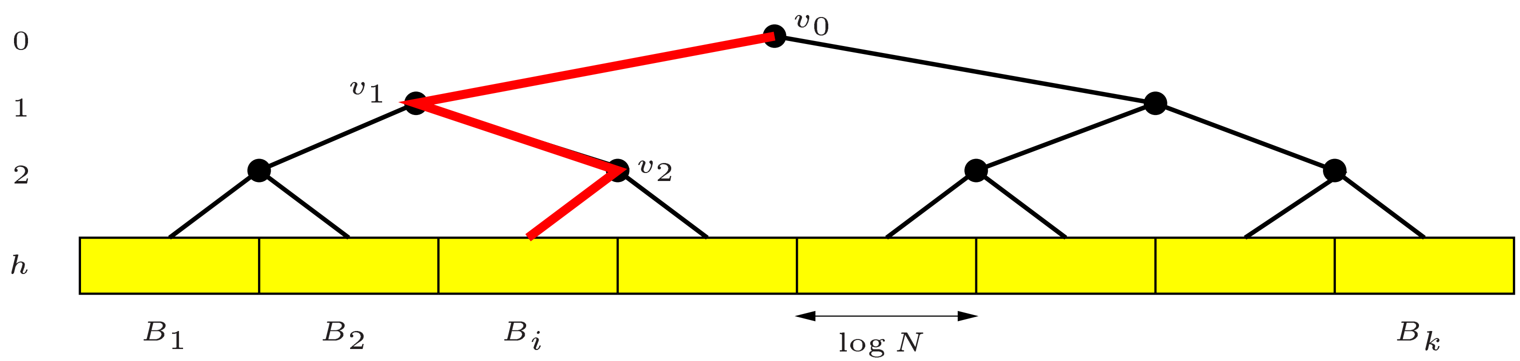
Packed Memory Management

[Itai et al., 1981]



- Partition array into blocks of size $\log n$
- For each level a threshold $\tau_i = \tau_L + i(\tau_U - \tau_L)/h$, such that $0 < \tau_L = \tau_0 < \tau_1 < \cdots < \tau_h = \tau_U < 1$
- For a node v_i on level i define the density $\rho(v_i) = \frac{s(v_i)}{m(v_i)}$, where $s(v_i) = \#$ elements below v_i and $m(v_i) =$length of array below v_i
- Invariant For each block $\rho(B_i) \leq \tau_h$

Packed Memory Management



Insertion

- Find block B to contain the new element
- Rearrange elements in B and insert new element in B
- If $\tau(B) > \tau_h$ then rebalance :
  - Find maximum i where ancestor v_i of B : $\rho(v_i) < \tau(v_i)$
  - Redistribute all elements below v_i

Theorem Insertions require amortized $O(\log^2 N)$ element moves

Packed Memory Management

Theorem Insertions require amortized $O(\log^2 N)$ element moves

Proof Consider two redistributions of v_i

- After the first redistribution $\rho(v_i) \leq \tau_i$
- Before second redistribution a child v_{i+1} of v_i has $\rho(v_{i+1}) > \tau_{i+1}$
- $\#$ insertions in $s(v_i)$ at least

$$m(v_{i+1}) \cdot (\tau_{i+1} - \tau_i) = m(v_{i+1}) \cdot (\tau_U - \tau_L)/h$$

- Redistribution of v costs $m(v_i)$, i.e. per insertion in $s(v_i)$

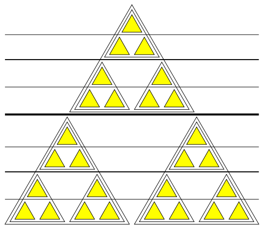
$$\frac{m(v_i)}{m(v_{i+1}) \cdot (\tau_U - \tau_L)/h} \leq \frac{2h}{\tau_U - \tau_L}$$

- Total insertion cost per element

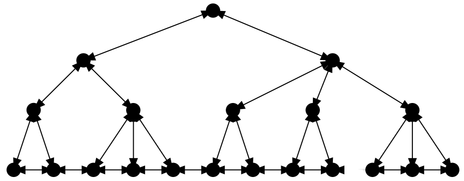
$$\sum_{i=0}^{h} \frac{2h}{\tau_U - \tau_L} = O(\log^2 N)$$

$\square$

Putting it together...



Strongly Weight Balanced B-Tree
+ van Emde Boas Layout
+ Packed Memory Management



- Strongly weight balanced B-tree
  - $d = 5$, i.e. degree $\in [2, 20]$
  - Leaves are doubly linked
  - Each node has a parent pointer and $O(1)$ child pointers
- van Emde Boas layout
  - Node splits/merges require changes in the layout
- Packed memory management handling an array of nodes
  - New node $\Rightarrow O(\log^2 N)$ moves $\Rightarrow$ update $O(\log^2 N)$ pointers

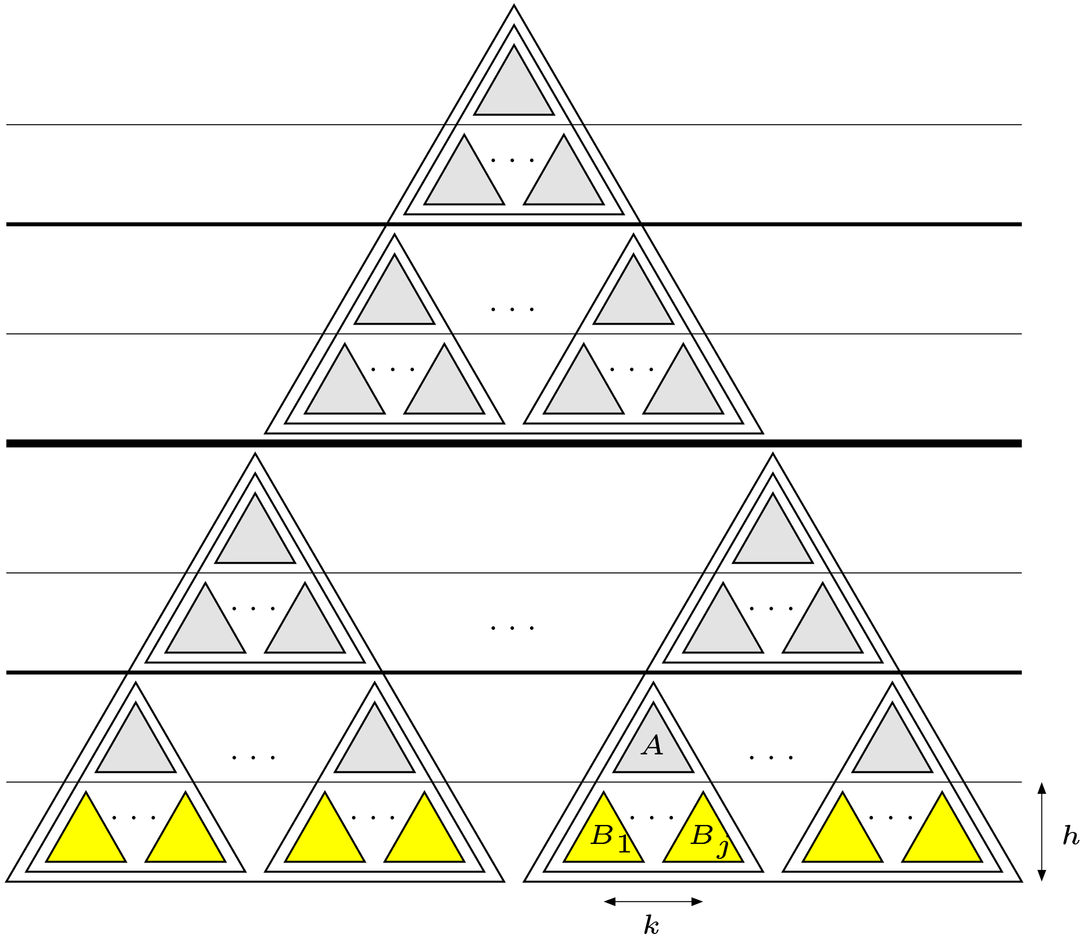
Scanning leaves

Lemma Scanning k consecutive leaves requires $O(1 + \frac{k}{B})$ I/Os

Proof

$$|B_i| \le B$$

$$\sum_{i=1}^{j} |B_i| = \Omega(B)$$



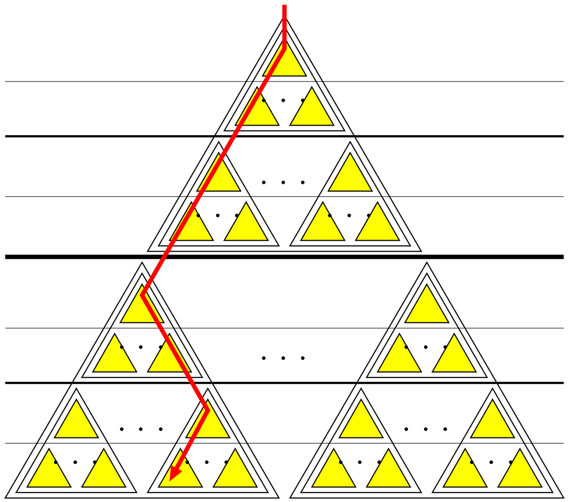
Scanning $B_{i_1}, \ldots, B_{i_2}$ requires $O(1 + (\sum_{i=i_1}^{i_2} |B_i|)/B)$ I/Os $\square$

Searching

Lemma Searching for a key requires $O(\log_B N)$ I/Os

Proof

- Strongly weight balanced B-tree
  $d = 5 \;\Rightarrow\;$ degree $O(1)$
- van Emde Boas layout
  $\Rightarrow\;$ $O(\log_B N)$ I/Os
- Packed memory management
  $\Rightarrow\;$ factor $O(1)$ more I/Os



$\square$

Updates

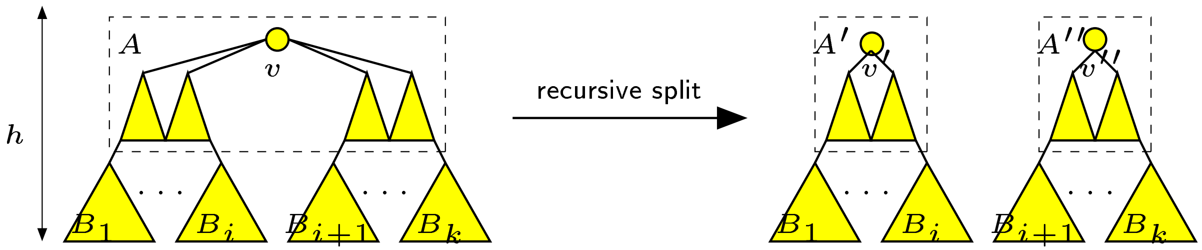
Lemma Updates can be done using $O(\log_B N + \log^2 N)$ I/Os

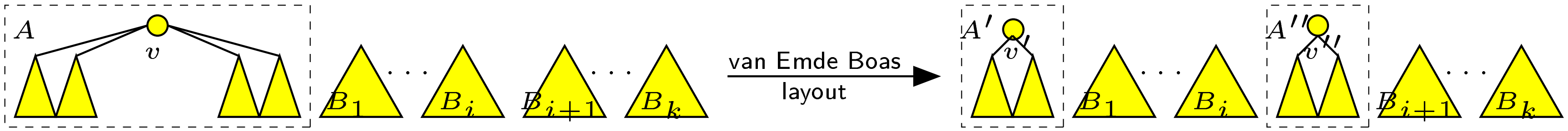
Proof $\#$ I/Os

1. Search for key	$O(\log_B N)$
2. Create/delete leaf	$O(1)$
3. Rearrange packed array of nodes	$O(1 + \frac{\log^2 N}{B})$
4. Update back pointers to moved nodes	$O(\log^2 N)$
5. Split/merge nodes	$O(1 + \frac{\log N}{B})$

Updates — Splitting

Consider splitting v and the first subtree in the van Emde Boas layout where v is the root





$$T(h) = T(\frac{h}{2}) + O(1 + \frac{d^h}{B}) = O(1 + \frac{d^h}{B}) \text{ I/Os}$$

which is payed by the last $\Theta(d^h)$ updates below v, i.e. each update requires amortized $O(\frac{1}{d^h} + \frac{1}{B})$ I/O for rebalancing v

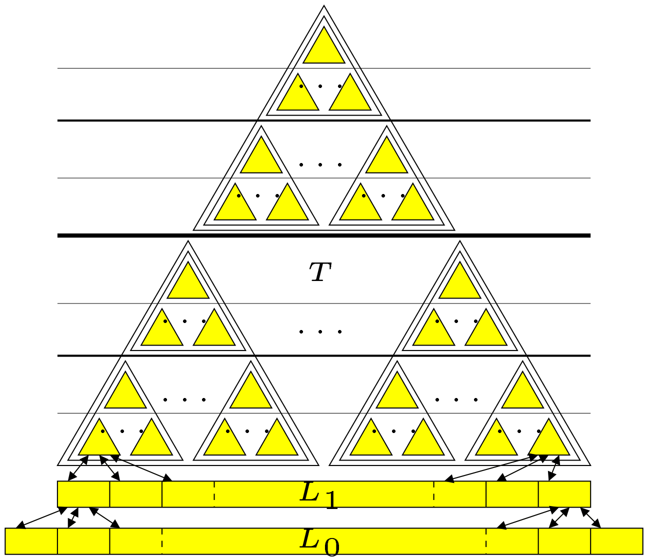
Totally each update requires amortized $O(1 + \frac{\log N}{B})$ I/O for rebalancing

First Result

Theorem There exist cache oblivious B-trees using linear space with

Operation	I/Os
Search	$O(\log_B N)$
Scan	$O(1 + \frac{k}{B})$
Updates	$O(\log_B N + \log^2 N)$

Two Levels of Indirection



L_0: packed array of all $\Theta(N)$ keys
partitioned into blocks of size $\Theta(\log N)$

L_1: packed array of pointers to L_0 blocks
partitioned into blocks of size $\Theta(\log N)$

T: cache-oblivious B-tree containing
pointers to the $O(\frac{N}{\log^2 N})$ blocks in L_1

- Updates in L_0 require $O(\log^2 N)$ movements and updates of pointers in L_1, using $O(1 + \frac{\log^2 N}{B})$ I/Os
- $O(N)$ updates require $O(\frac{N}{\log N})$ insertions/deletions in L_1, causing $O(N \log N)$ movements in L_1 and $O(N \log N)$ updates of pointers in L_0 and T, in total requiring $O(\frac{N}{\log N} + N \log N \frac{\log N}{B})$ I/Os
- $O(N)$ updates require $O(\frac{N}{\log^2 N})$ block insertions/deletions in T, in total requiring $O(N)$ I/Os
- Searches require $O(\log_B N + \frac{\log N}{B}) = O(\log_B N)$ I/Os

Final Result

Theorem There exist cache oblivious B-trees using linear space with

Operation	I/Os
Search	$O(\log_B N)$
Scan	$O(1 + \frac{k}{B})$
Updates	$O(\log_B N + \frac{\log^2 N}{B})$

Corollary For $B = \Omega(\log N \cdot \log\log N)$ this matches the B-tree bound