# External Memory Algorithms

*IT-C Course on Advanced Algorithms*

Gerth Stølting Brodal

BRICS
Department of Computer Science
University of Aarhus

gerth@brics.dk



# Lectures on External Memory Algorithms

**March 20, 2001**

- External memory model – parameters $N$, $M$, $B$, $D$
- Algorithms: scanning, merging, sorting, permutation
- Lower bounds: sorting, permutation

**March 27, 2001**

- B-trees
- Analysis of B-trees
- Cache-Oblivious B-trees

**April 3, 2001**

- Minimum spanning trees
- Functional approach
- Superphases and blocking values



# Literature

Sorting

**The Input/Output Complexity of Sorting and Related Problems**
Alok Aggarwal and Jeffrey Scott Vitter
*Communications of the ACM*, 31(9):1116–1127, 1988

B-trees

**Amortized Analysis of $(a, b)$-Trees**
Gerth Stølting Brodal and Rolf Fagerberg
Note, 4 pages, 2000

**Cache-Oblivious B-Trees**
Michael A. Bender, Erik Demain, and Martin Farach-Colton
In *Proc. 41th Annual Symposium on Foundations of Computer Science*, 399–409, 2000

MST

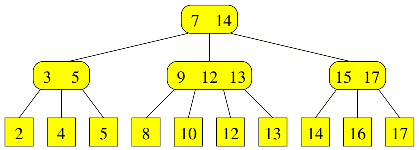
**A Functional Approach to External Graph Algorithms**
James Abello, Adam L. Buchsbaum, and Jeffery R. Westbrook
In *Proc. 6th Annual European Symposium on Algorithms*, LNCS 1461, 332–343, 1998

**On External Memory MST, SSSP and Multi-way Planar Graph Separation**
Lars Arge, Gerth Stølting Brodal, and Laura Toma
In *Proc. 7th Scandinavian Workshop on Algorithm Theory*, LNCS 1851, 433–447, 2000



# $(a, b)$-Trees and B-trees

[Bayer & McCreight, 1972]

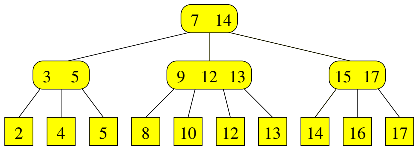



**Definition** A tree is an $(a, b)$-tree if $a \geq 2$, $b \geq 2a - 1$ and

- All leaves have the same depth.
- All internal nodes have degree at most $b$.
- All internal nodes except the root have degree at least $a$.
- The root has degree at least two.

If $b = 2a - 1$ then the trees are also denoted B-trees

# Properties of $(a, b)$-Trees




**Lemma** $N$ leaves implies $\lceil \frac{\log n}{\log b} \rceil \leq$ height $\leq \lfloor \frac{(\log n)-1}{\log a} \rfloor + 1$

**Lemma** Searches require $O(\log_a n)$ I/Os if $b = O(B)$



# An Application of B-Trees

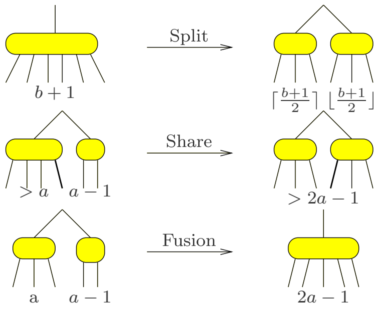
Core indexing data structure in many database management systems

*TELSTRA, an Australian telecommunications company, maintains a customer database with 51.000.000.000 rows and 4.2 terabytes of data*



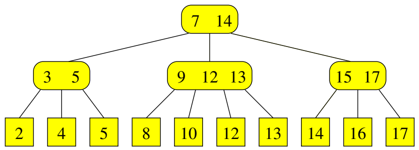
# Updates in $(a, b)$-Trees

- Search for location to insert or delete a leaf
- Create/delete leaf and search key at the parent node
- Rebalance using the following transformations

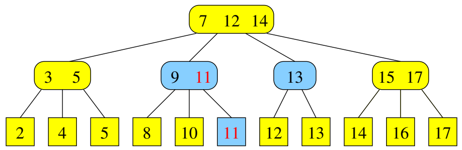





# Example : Insert into a (2,4)-Tree




$\Downarrow$ Insert(11)



# Analysis of $(a, b)$-Trees – Insertions Only

**Theorem**

$n$ insertions imply $n/\lfloor (b+1)/2 \rfloor^h$ splits at height $h$

i.e. in total at most $O(n/b)$ splits

*Proof*

- Nodes are created due to splits
- All nodes except the root has degree at least $\lfloor (b+1)/2 \rfloor^h$
- The number of nodes in the lowest level dominates all other levels

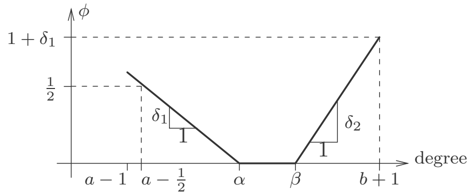
$\square$



# Analysis of $(a, b)$-Trees

**Theorem** If $b \geq 2a$, then $i$ insertions and $d$ deletions perform at most $O(\delta^h(i+d))$ splits and fusions at height $h$, where $\delta < 1$ depends on $a$ and $b$, and # sharings $\leq$ # fusions + # deletions

*Proof (sketch)* Amortization argument, each node has a potential $\phi$




$\square$

**Theorem** If $b \geq 2a$, then the total # splits and # fusions is $O(i+d)$. If $b \geq (2+\varepsilon)a$, for some $\varepsilon > 0$, the number of node splittings and node fusions is $O(\frac{1}{a}(i+d))$



# Analysis of $(a, b)$-Trees

**Theorem**

$(B/3, B)$-trees perform $\Theta(1/B)$ rebalancing per update

**Theorem**

$(\lfloor B/2 \rfloor, B)$-trees perform $\Theta(1)$ rebalancing per update

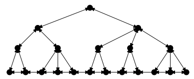
**Theorem**

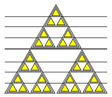
$(\lceil B/2 \rceil, B)$-trees perform $\Theta(\log_B N)$ rebalancing per update if $B$ odd



# Cache-Oblivious B-trees

Strongly Weight Balanced B-Tree





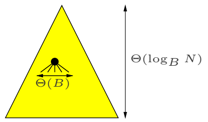
+ van Emde Boas Layout

+ Packed Memory Management

# B-tree Characteristics

**Definition**

- All leaves have the same depth
- All 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

$\Big\}$ $O(\log_B N)$ I/Os

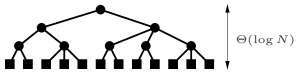
- Cache-aware, since degrees $\Theta(B)$

Cache awareness essential ?



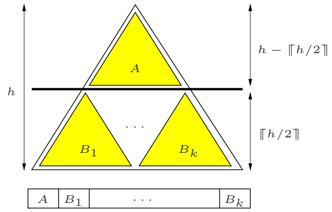
# Layout of Static Trees

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




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

van Emde Boas layout



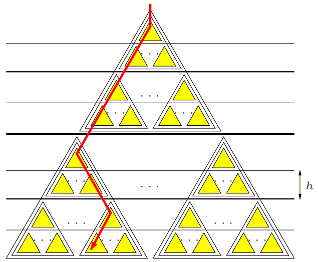

$$[\![x]\!] = 2^{\lceil \log x \rceil}$$

Invariant

Recursive structures are stored in contiguous blocks of memory



# van Emde Boas Layout




If $\delta \leq \text{degree}(v) \leq \Delta$ for all nodes $vb$ 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 $h = [\![\log_\Delta B]\!]$ at most 2 I/Os required to read subtree

Dynamic tree ?



# 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 $u$

**Properties**

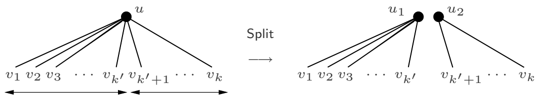
- $2 \leq$ degree of the root $\leq 4d$
- $d/4 \leq \text{degree}(u) \leq 4d$ for all internal nodes $u$
- Height $O(\log_d N)$

# 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 based on weights ...



# Strongly Weight Balanced B-trees

**Lemma**

If the rebalancing of a node (e.g. splitting the node) touches all nodes in the subtree then the amortized # I/Os per update is $O(\log_B N)$

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**

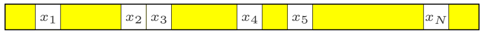
Searching in a strongly weight balanced tree stored using the van Boas layout costs at most $O(\log_B N)$ I/Os

Contiguous recursive structures
v.s. Dynamic structures ?



# 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

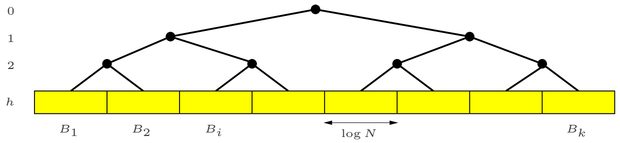
**Theorem**

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



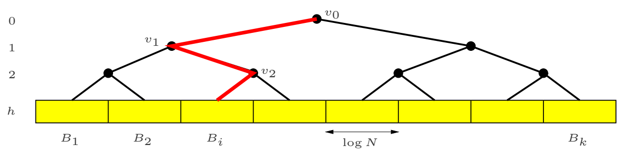
# Packed Memory Management

[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 $\rho(B_i) \leq \tau_h$ for each block $B_i$ on level $h$

# 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

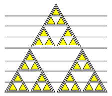
*Proof* Consider two redistributions of $v_i$

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

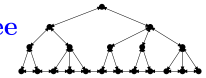
□



# 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



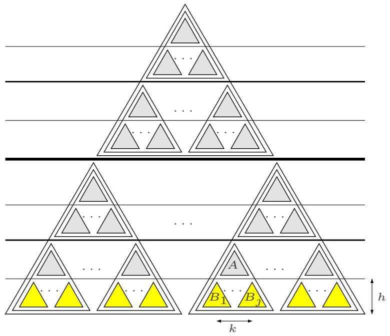
# Scanning Leaves

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

*Proof*

$|B_i| \leq B$

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



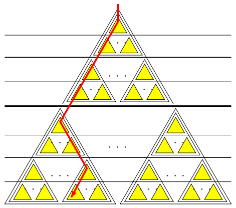

Scanning $B_1, \ldots, B_j$ requires $O(1 + (\sum_{i=1}^{j} |B_i|)/B)$ I/Os □

# 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



□



# Updates

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

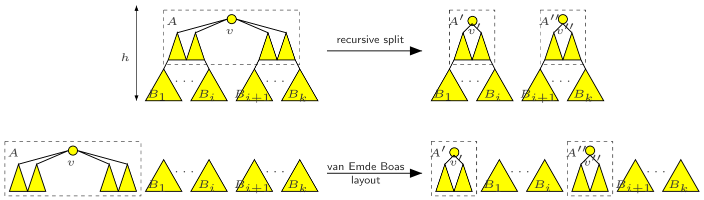
| *Proof (sketch)* | # I/Os |
|---|---|
| • Search for key | $O(\log_B N)$ |
| • Create/delete leaf | $O(1)$ |
| • Rearrange packed array of nodes | $O(1 + \frac{\log^2 N}{B})$ |
| • Update back pointers to moved nodes | $O(\log^2 N)$ |
| • Split/merge nodes | $O(1 + \frac{\log N}{B})$ |

□



# Updates – Splitting

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




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

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



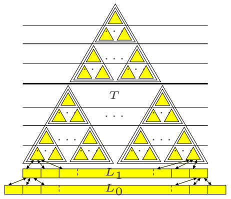
# First Result

**Theorem**

There exist cache-oblivious B-trees using linear space and # 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 and # 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 $B = \Omega(\log N \cdot \log \log N)$ updates use optimal $O(\log_B N)$ I/Os