External Priority Queue

[Fadel et al., 1999 (1997)]

Goal:

INSERT $O(\frac{1}{B} \cdot \log_{M/B}(N/M))$ amortized.

DELETEMIN $O(\frac{1}{B} \cdot \log_{M/B}(N/M))$ amortized.

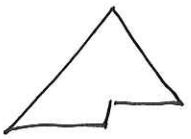
$\Rightarrow$ External HEAPSORT in $O(\frac{N}{B} \cdot \log_{M/B}(N/M))$ I/O's via $N \times$ INSERT plus $N \times$ DELETEMIN.

(We also desire linear space, ie. $O(N/B)$ blocks and $O(\log_2(N))$ amortized comparisons per operation, which will make the HEAPSORT optimal wrt. CPU time.)

The structure: [Let M' be $M/3$. Note: B is called P in the article, and min is max (it describes a max-heap).]

On disk:

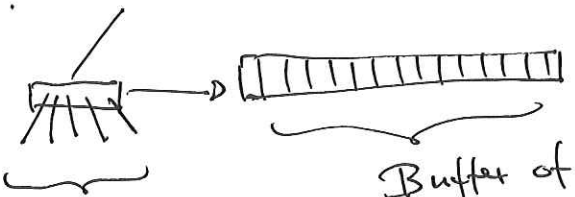
Tree with fan-out $\Theta(M/B)$, heap-shape:



} 1 level

and heaporderd: all elements in a node (in a node's buffer) are stronger than all elements in the subtree of the node.

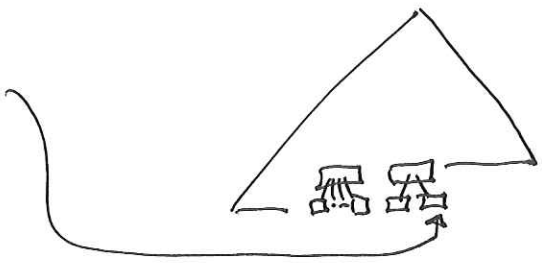
Each node:



$\Theta(M/B)$ children.

Buffer of size M', contains elements, contents sorted.

Def.:
Last node



Last node is special: its parent may have fan-out $< M/B$ and its buffer (not shown above) may contain $< M'/2$ elements.

All other nodes: fan-out $= M/B$ and buffer (not shown above) has between $M'/2$ and M' elements.

In RAM: An insert buffer (structured as an internal/normal heap) of up to M' elements.

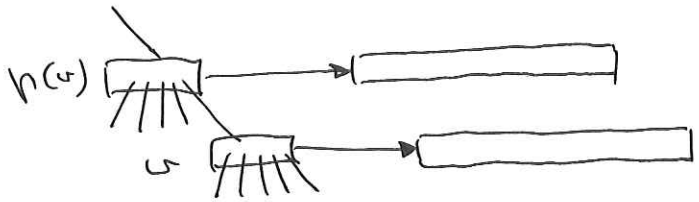
The Operations:

INSERT: Insert new element in insert buffer
IF this buffer gets size M':
Make a new Last leaf
Sort elements in insert buffer
Move these elements to node buffer of new last leaf.

Swap new node with previous last leaf.

SIFTUP on both these nodes.

SIFTUP: Works on a node v where some elements in v's node buffer are in heap-order conflict with elements in node buffer of parent $p(v)$, tree else OK.



Note: conflict can be checked by comparing strongest element in v with weakest in $p(v)$, since buffers are sorted.

Actions:

1) Merge the two sorted node buffers of v and $p(v)$ into one sorted sequence

2) Split into buffers of same sizes as before. First part of sequence becomes $p(v)$'s new node buffer, last part becomes v's new node buffer.

   Observe: i) Heaporder restored between v and $p(v)$ [and between v and its children] ii) Heaporder may now be violated between $p(v)$ and its parent.

3) IF ii) holds, run SIFTUP recursively on $p(v)$.

NB: this detail is not so clearly stated in article.

DELETEMIN :

Check root of tree [keep first block of its buffer in RAM] vs. insert buffer.

IF min element is in insert buffer:
Remove it [standard DeleteMin on this internal heap] and return.
ELSE: Remove it from root's buffer and return.

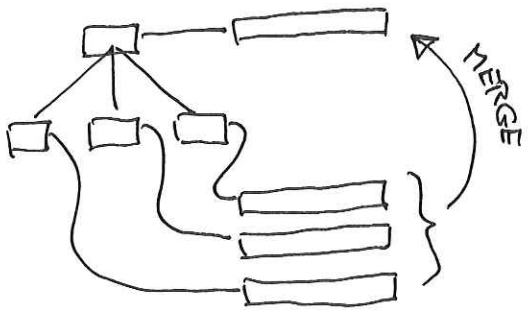
IF |root buffer| < $M'/2$:
REFILL(root)
CLEAN_UP_LEAVES

REFILL(v) :

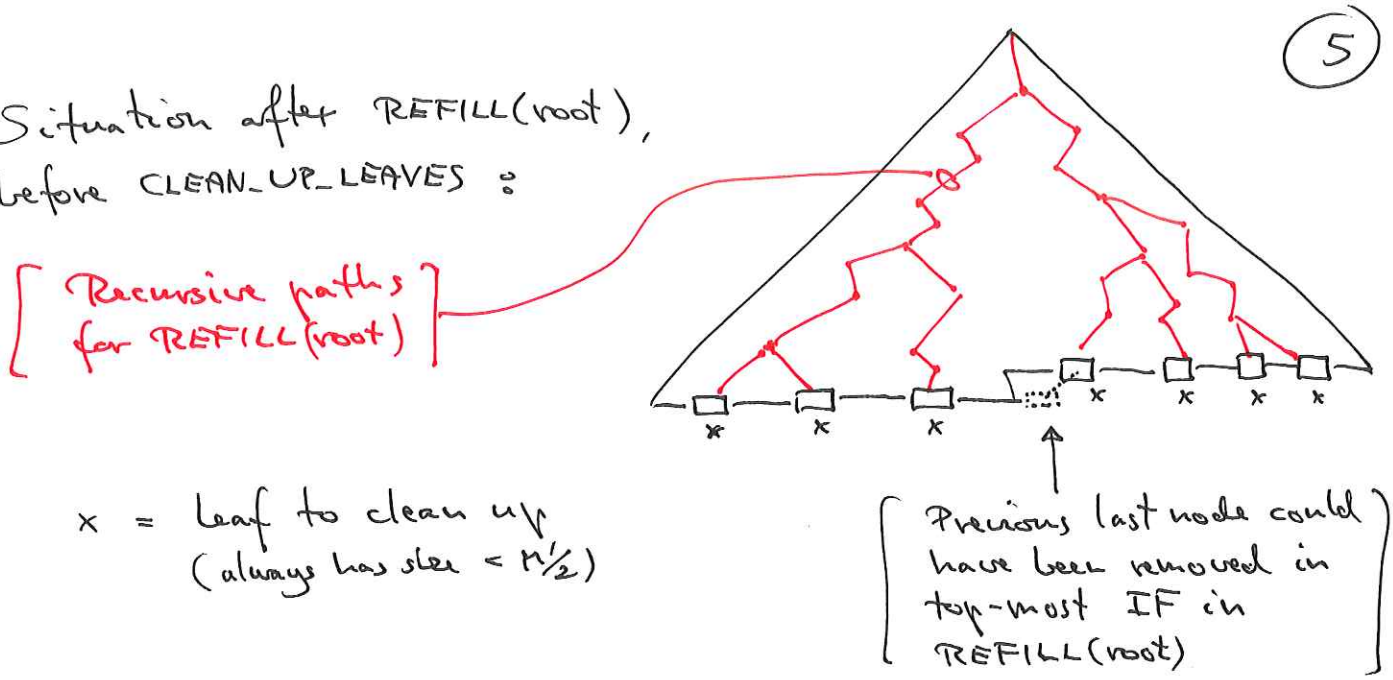
IF v has last node as its single child and buffer of last node has < $M'/2$ elements :
Append buffer of last node to buffer of v.
Remove last node.
Add v to list of leaves to be cleaned up if |v| still < $M'/2$.

ELSE :
Do $M'/2$ steps of a fan-out M/B merge from children's buffers to v's buffer

FOR each child w of v :
IF size of w's buffer < $M'/2$:
IF w is not a leaf :
REFILL(v)
ELSE :
Add w to list of leaves to be cleaned up.



Situation after REFILL(root), before CLEAN_UP_LEAVES :



CLEAN_UP_LEAVES :

WHILE there is a leaf v to be cleaned up ≠ current last node w :

i) IF $|v| + |w| > M'$:

Merge the $M' - |v|$ ($> |w|$) strongest elements in w's buffer into v's buffer.
Remove v from list to be cleaned up.
SIFTUP(v).

ii) ELSIF $M'/2 \leq |v| + |w| \leq M'$:

Merge all of w's buffer into v's buffer.
Delete w (updating what is current last node).
Remove v from list to be cleaned up.
SIFTUP(v).

iii) ELSE :

Merge all of w's buffer into v's buffer.
Delete w (updating what is current last node).
SIFTUP(v).

Analysis

N elm. stored $\Rightarrow$ max $\frac{N}{M'/2} + 1$ nodes

($\frac{N}{M'/2}$: all nodes ($\neq$ last leaf) stores $M'/2$ or more; $+1$: last leaf)

$$\text{Height} \leq \lceil \log_{\text{fanout}} (\#\text{nodes}) \rceil$$

$$= O(\log_{M/B} (N/M))$$

There are $\Omega(M)$ insert between each emptying of input buffer. Charge everything to these inserts:

1) $O(1)$ SiftUps just after input buffer emptying.

Price: Height $\times$ Scan(M)

$$= O\left(\frac{M}{B} \cdot \log_{M/B}\left(\frac{N}{M}\right)\right). \text{ I.e. } O\left(\frac{1}{B}\log_{M/B}(N/M)\right) \text{ per inserted element.}$$

[Binary merge of two buffers in nodes.]

2) Lifting of elements upwards in tree during Refills: $M'/2$ mergesteps (lifting $M'/2$ elms one level) cost in I/O's:

(Initiate Merge = load first block of children buffers)

$$\text{Fanout} + \text{Scan}\left(\frac{M'}{2}\right) = O\left(\frac{M}{B}\right) + O\left(\frac{M}{B}\right) = O\left(\frac{M}{B}\right).$$

So if each of the $M'/2$ lifted elements gets charged $O(\frac{1}{B})$, then the price of the lift is covered.

Therefore we define the invariant that a node buffer with m elements, residing in a node at depth d, has

$$d \cdot m \cdot \frac{1}{B}$$

credits, then the lift will release credits covering the charging [as $M'/2$ elements moves from depth d to depth $d-1$].

So $O\left(\frac{M}{B} \cdot \log_{M/B}(N/M)\right)$ credits for this must be supplied when the insertion buffer overflows during an insertion

Note: here we use that siftups do not change sizes of buffers [p. ③, middle], so they don't change credits in buffers.

3) Each step in CleanUp Leaves does (zero or) one binary merge of (parts of) node buffers, and one SiftUp. This costs $O(M/B) + O(\frac{M}{B} \log_{M/B}(N/M))$.

Each step involves the current last leaf. Since $|v| < M/2$ (v is underfull), then in case i) [on page ⑤] we move $M' - |v| > M'/2$

elements from current last leaf's buffer. Hence, for a given current last leaf, case i) can only happen once. Next step will be case ii) or iii), which removes the node.

Therefore we when creating a node gives it $2 \times O(\frac{M}{B} \cdot \log_{M/B}(N/M))$ credits, to cover the cost of the (up to) two steps of CleanUpLeaves where it is the current last leaf.

[Note: once a node becomes last leaf, it will stay that way until deleted, also in between CleanUpLeaves.]

Summing up: if we charge the creation of a new node [during an insert overflowing the insert buffer] $O(\frac{M}{B} \log_{M/B}(N/M))$, then all the cost/credits of 1) + 2) + 3) are covered.

(for new node and its buffer)

This is $O(\frac{1}{B} \log_{M/B}(N/M))$ per element of the overflowing insert buffer, and all of these elements were inserted since last such overflow. Thus charging each Insert operation $O(\frac{1}{B} \log_{M/B}(N/M))$ I/Os covers all costs of the structure. □