External Priority Queue

[Fadel et al, 99]

Goal: Insert : $O_A\left(\frac{1}{B} \cdot \log_{M/B} N/B\right)$ (sz)
DelMin :

Space: $O(N/B)$

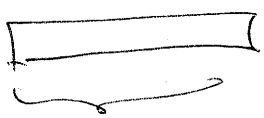
[CPU : $O(\log_2 N)$ for ops.]

$\Longrightarrow$ Efficient external HeapSort ($= N \cdot$ Insert + $N \cdot$ DelMin)

Note for this article : $B = P$

Min = Max (max heap)

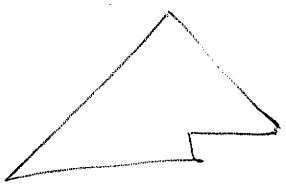
RAM : Insert buffer (an internal heap)



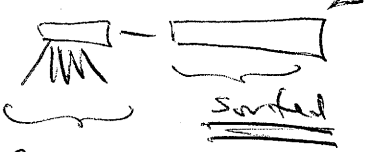
$M' = \Theta(M)$ elms. (Eg $M' = M/2$)

Heap : Tree of fanout $\Theta(M/B)$

Shape:



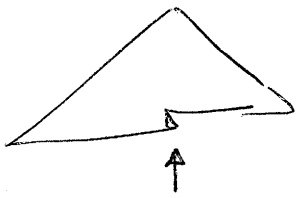
Nodes :



$\Theta(M/B)$ children

Heap ordered : Elms in node buffer dominates any elms in subtree.

Last node :



is the

- Parent only internal node of parent $< M'/B$
- Node buffer may contain $\leq M'/2$ elms.

All other nodes : Buffer contains at least $M'/2$ elms and at most M elms.

Operations

Insert : Insert new elm. into insert buffer (heap in RAM)

If this buffer gets size $\geq M'$:

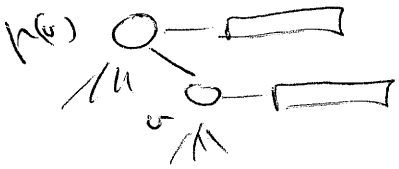
Make a new last leaf

Move input buffer contents to new node's buffer (first sort elm.s)

Swap new node with previous last leaf.

SiftUp on both nodes

SiftUp



← Some elements in conflicts (heap-order-wise) with ancestors.
Else tree OK.

[Note: can be checked by comparing strongest elm. in v with weakest in r(v)]

Merge the two (sorted) node buffers.

Unclear (wrong?) stated in paper → Split into same sized buffers as before.
↑ sorted outcome

Think/Consider

Note: Heap order restored between v and r(v).
Heap order between r(v) and r(r(v)) may be violated. ⇒ recurse SiftUp on r(v).

DeleteBest :

Check root of tree [best block of its buffer kept in RAM].

and insert buffer (a heap).

If best is in insertbuffer, remove & return.

Else remove from root node buffer

If |rootbuffer| < $M'/2$:

Refill (root).

CleanUpLeaves

Refill (v)

Do $M'/2$ mergesteps on children buffers

If not possible (has last leaf as only child),

Add to list of leaves to be cleaned up. (remove child)

Else for each child w :

If buffer size < $M'/2$

If not leaf :

Refill (w)

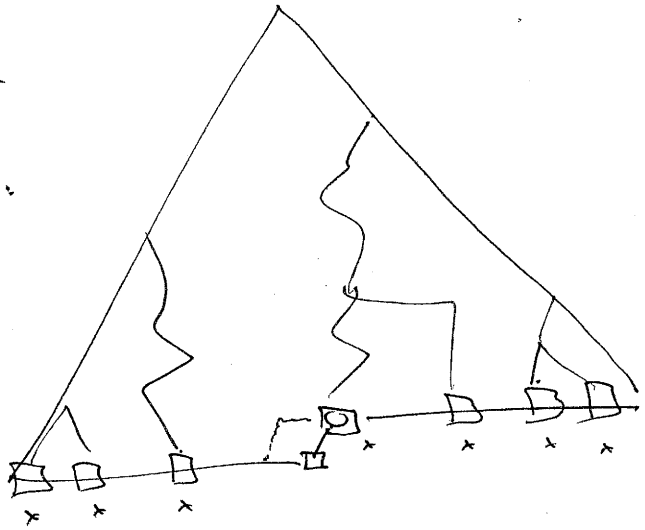
Else

Add to list of leaves to be cleaned up

do as many steps as possible (⇒ becomes a leaf)

Note : If there exists just one child ≠ last leaf, then these steps can be performed (due to size of childrens buffers).

~~After~~ recursive
~~Rebuild~~, before
CleanUpLeaves:



x = leaves to clean up. (= all leaves with $< \frac{M'}{2}$ elms. in buffers)

{ "underfull" }

CleanUpLeaves

current
v
While ∃ underfull leaf $v \neq$ last leaf w:

{ may change during CleanUpLeaves }

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

Move $M' - |v|$ strongest elm in w's buffer to v's buffer (merge)

SiftUp(v)

ii) Elsif $\frac{M'}{2} \leq |v| + |w| \leq M'$:

Merge w's buffer into v
Delete(w)
SiftUp(v)

iii) Else: // $|v| + |w| < M'/2$

| Same as

// v is still underfull

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. □