Fat heaps
(K & Tarjan 96)

Goal

Want to achieve the performance of Fibonnaci heaps but on the worst case.

Why ?

Theoretical curiosity and

some applications

Applications

Implement Dijkstra's algorithm in parallel on an EREW PRAM.

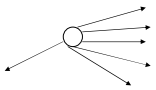
Where can we exploit paralelizm ?

When we scan a vertex v and there are many edges outgoing of v.

The algorithm

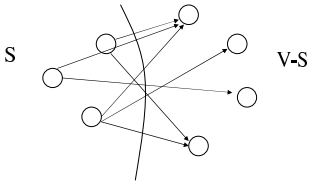
The $\delta(v)$ edges incident with every vertex v are divided evenly among the processors such that each processor is responsible for either

$$\left\lfloor \frac{\delta(v)}{p} \right\rfloor \quad \text{or} \quad \left\lfloor \frac{\delta(v)}{p} \right\rfloor + 1 \quad \text{edges}$$



The algorithm (cont.)

Each processor maintains a heap. At the start of each iteration, the heap of p_i contain every vertex of V-S with key $d_i(v) = \min\{d(u) + w(u,v) \mid u \in S \text{ and } (u,v) \text{ assigned to } p_i\}$



The algorithm (cont.)

Iteration of Dijkstra:

1) Each processor finds the minimum in its heap. (find-min).

2) Find the minimum among the results of the find-mins, and the vertex v which achieves it.

3) Broadcast v and d(v) to all processors.

4) Each processor does a delete(v)

5) Each processor scans the edges out of v assigned to it and make the appropriate decrease-key on its heap if needed. (using d(v))

PRAM basics

1) Can "Broadcast" a value v to p processors in O(log p) time

2) p processors, each holding a value can calculate the minimum in O(log p) time

7

The algorithm (cont.)

Iteration of Dijkstra:

1) Each processor finds the minimum in its heap. (find-min). O(1)

2) Find the minimum among the results of the find-mins, and the vertex v which achieves it. O(log p)

3) Broadcast v and d(v) to all processors. O(log p)

4) Each processor does a delete(v) O(log n)

5) Each processor scans the edges out of v assigned to it and make the appropriate decrease-key on its heap if needed. (using d(v))

$$O\left(\left\lfloor \frac{\delta(v)}{p} \right\rfloor + 1\right)$$

8

Summing up over all iterations we obtain that the time is O(m/p + nlog(n))
Space O(m + n*p)

If $m \geq n\log n$

take $p = m/n(\log n)$ and Dijkstra would run in O(nlog n) time and O(m) space

For $p \leq m/n(\log n)$ Dijkstra would run in O(m/p) time and O(m) space.

What is the problem with all this ?

Everything should be syncronized and time bounds are amortized

9

Redundant binary counters (Clancy & Knuth 77)

Go back to binomial heaps.

Lets get started from insert, How do we make insert O(1) on the worst case and keeping logarithmic worst case bounds for the other operations.

Insert is more or less like adding one to a binary number

So maybe we want first to be able to get a binary counter supporting increment in O(1) time on the worst case.

10

Redundant binary counters (cont)

We shall allow an additional digit.

Use 0,1, and 2, but keep the base to be 2 !

$012110 = 0*2^0 + 1*2^1 + 1*2^2 + 2*2^3 + 1*2^4 = 38$

The representation now is not unique

Can represent 38 also by 020110 or 100110

We shall exploit the redundancy.

11

Exploiting the redundancy

We are not going to use all possible representations

Define a representation to be regular if it does not contain the following patterns

…….211111112…...
……..……211111111

When you walk from left to right after you see a 2 you should see a 0 before hitting another 2 or running out of digits

12

Exploiting the redundancy (cont)

Note: every number has a regular representation.

Given a regular representation for x, how do you get a regular representation for x+1.

1) increment the rightmost digit.

We get a representation for x+1 which may not be regular

……..2101210 ⟹ ……..2101211 ⟹ ……..2102011

2) If indeed so, then replace the rightmost 2 with a 0 and increment the digit to its left. (Fix)

13

Exploiting the redundancy (cont)

Lemma: we end up with a regular representation for x+1.

Proof: Clear if we have not created a 2.

Suppose that we create a 2:

…….121111 ⟹ ……..20111

There is a zero between the new 2 and a 2 to its left because there was a zero between the old 2 and the 2 to its left.

□

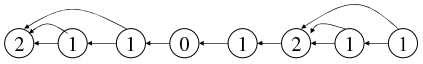
14

Exploiting the redundancy (cont)

We change only O(1) digits but can we implement it in O(1) time ?

Yes!

Represent the number as a sorted list with forward pointers:



If a 1 is followed by a sequence of 1's and a 2 then its forward pointer point to that 2.

Have to update forward pointer when you create a one.

15

Can incrementing an arbitrary digit

To increment d_i:

Fix d_i if $d_i = 2$

increment d_i

if $d_{i+1} = 1$ and the first d_j, $j>i+1$ such $d_j \neq 1$ is 2 then fix d_j

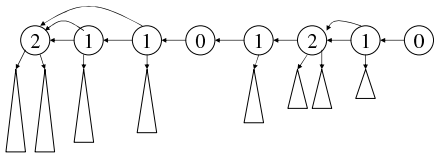
Fix d_i if $d_i = 2$

16

Back to heaps

Allow 0,1,2 binomial heaps of each rank.

Trees are hanging off a redundant counter. The i-th digit of the counter corresponds to the number of rank i trees



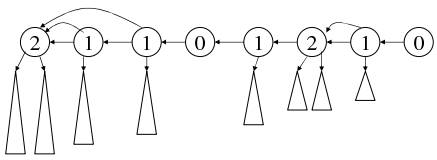
The representation in the counter must be regular.

17

Back to heaps (cont)

Insert now is like adding one to the redundant binary counter

While incrementing if we need to fix a 2 then it corresponds to linking two binomial trees of rank i to a binomial tree of rank i+1.



18

Back to heaps (cont)

Do other operations essentially as before.

Next, lets try to add a decrease key, in O(1) worst case time, without hurting other operations

19

Some additional facts about redundant counters

The idea extends from base 2 to any base.

For example in base 3 the forbidden configurations would be

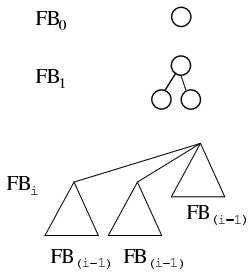
…….…..32222222223….

……..……..322222222

We can increment any digit in O(1) time if we get access to it.

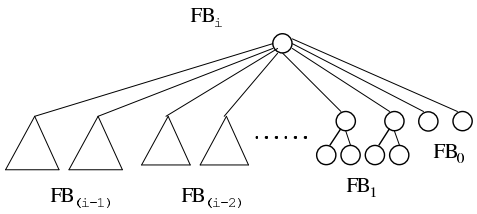
20

Fat binomial trees



21

Fat binomial trees



22

Properties of fat binomial trees

1) $|FB_k| = 3^k$
2) degree(root(FB_k)) = 2k
3) depth(FB_k) = k

==> The degree and depth of a binomial tree with at most n nodes is O(log(n)).

Define the rank of FB_k to be k

23

Fat heaps (def)

A collection of fat binomial trees at most 3 of every rank.
Items in the nodes.
Trees are hanging off of a redundant base 3 counter which is regular.

So far, the bounds for all operations are as before.

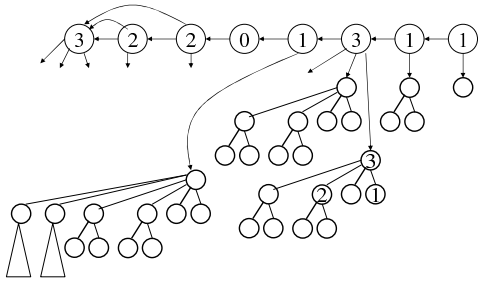
For efficient decrease key, allow violations to the heap order.

A violation is a node that is smaller than it parent.

Allow at most two such nodes of every rank in all trees

24

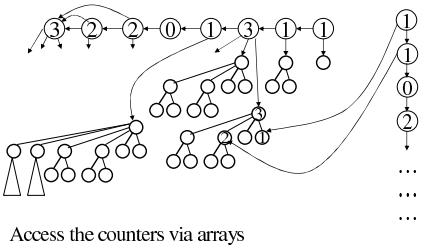
Fat heaps (example)



Organizing violations

Use another redundant binary counter.

The i-th digit corresponds to violations of rank i and stores pointers to these violating nodes. Counter must be regular.



Access the counters via arrays

Operations

Insert(x,h): essentially as before. O(1)

Deletemin(h): add the subtrees by performing increments to the root counter.

The new minimum is either at a root or at a violating node. We search for it and if it is in a violating node we swap it with one of the roots. O(log n)

Delete(x,h): decrease-key(x,h,∞), deletemin(h)

Decrease key(x,h,δ)

Decrease the key of x by δ. If x becomes the minimum swap it with the minimum item.

We may have created new rank i violation. So we need to increment digit i of the violation counter.

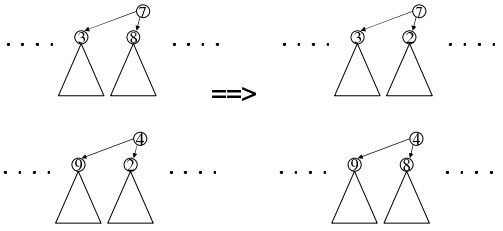
This may require a fix operation on some digits.

A fix operation corresponds to converting two violations of rank i to one violation of rank i+1.

How do we do that ?

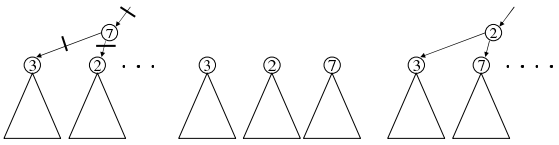
Decrease key(x,h,δ) (cont)

First arrange the two violations to have the same root. (the one with larger key)



Decrease key(x,h,δ) (cont)

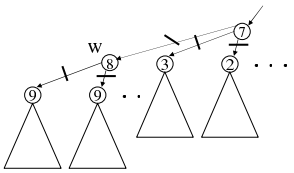
Case 1: The parent of the two rank i violations is of rank i+1.



Cut the two violating nodes and their parent. That gives 3 rank i trees. Link them to one rank i+1 tree which replaces the parent of the two violations.

Decrease key(x,h,δ) (cont)

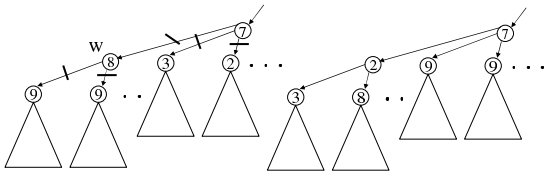
Case 2: The parent of the two rank i violations is of rank at least i+2.



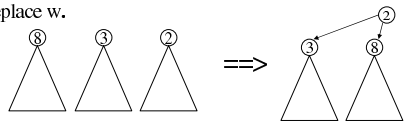
This parent must have a non-violating child w of rank i+1.
This child must have two non violating rank i children.

31

Decrease key(x,h,δ) (cont)

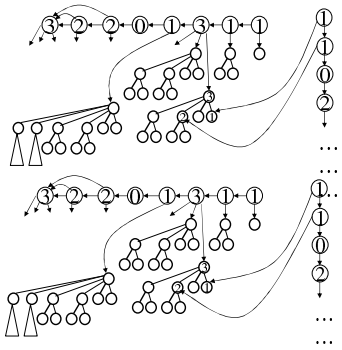


Replace the violating subtrees with by the rank i subtrees of w. Cut w (without its leftmost subtrees). Link the subtree rooted by w with the violating subtrees. Use the resulting subtree to replace w.



32

Meld(h1,h2)



Fix all violations in h1 and then insert the subtrees of h1 into the counter of h2 by doing increments

33