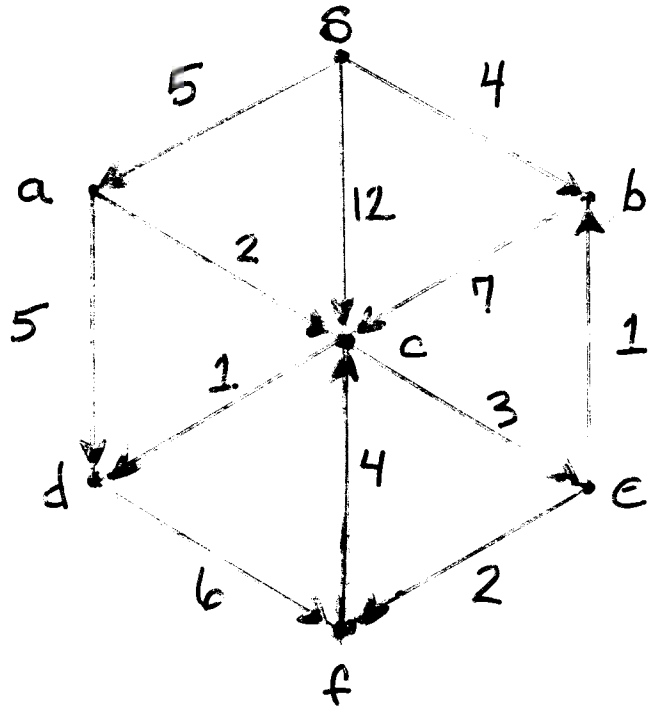
Shortest Path Problem

Given a directed graph with non-negative edge lengths, find shortest paths from a start vertex s to all other vertices (or to a single goal vertex t).



n = # vertices = 7

m = # edges = 12

Dijkstra's Algorithm

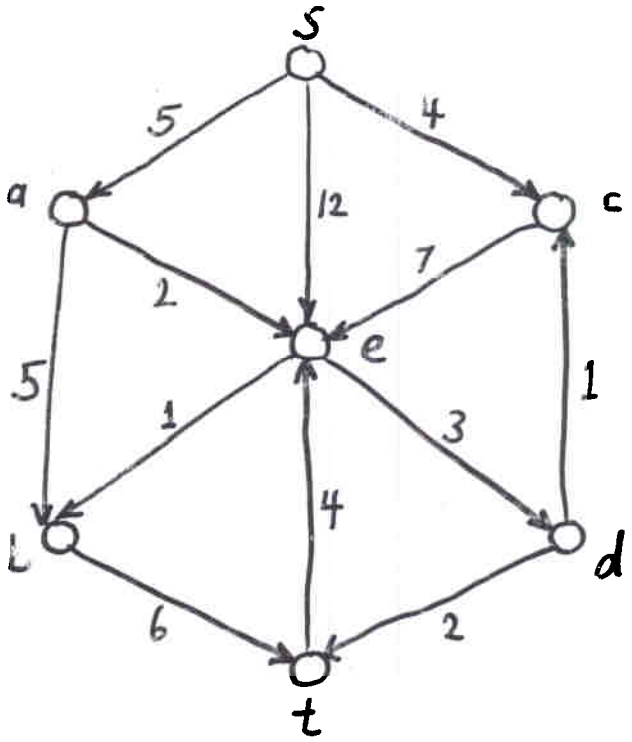
Maintain a set F of frontier vertices v, each labeled with a tentative distance $d(v)$.

Initially $F = \{s\}$, $d(s) = 0$, $d(v) = \infty$ for $v \neq s$.

General Step. Select $v \in F$, with $d(v)$ minimum (Now $d(v)$ is actual shortest distance from s.) For each edge $v \to w$ such that $d(v) + \ell(v,w) < d(w)$, replace $d(w)$ by $d(v) + \ell(v,w)$ and if $w \notin F$, add w to F.

Repeat general step until t selected.

Can be augmented to compute shortest paths as well as shortest distances.



distance

frontier set

s

a

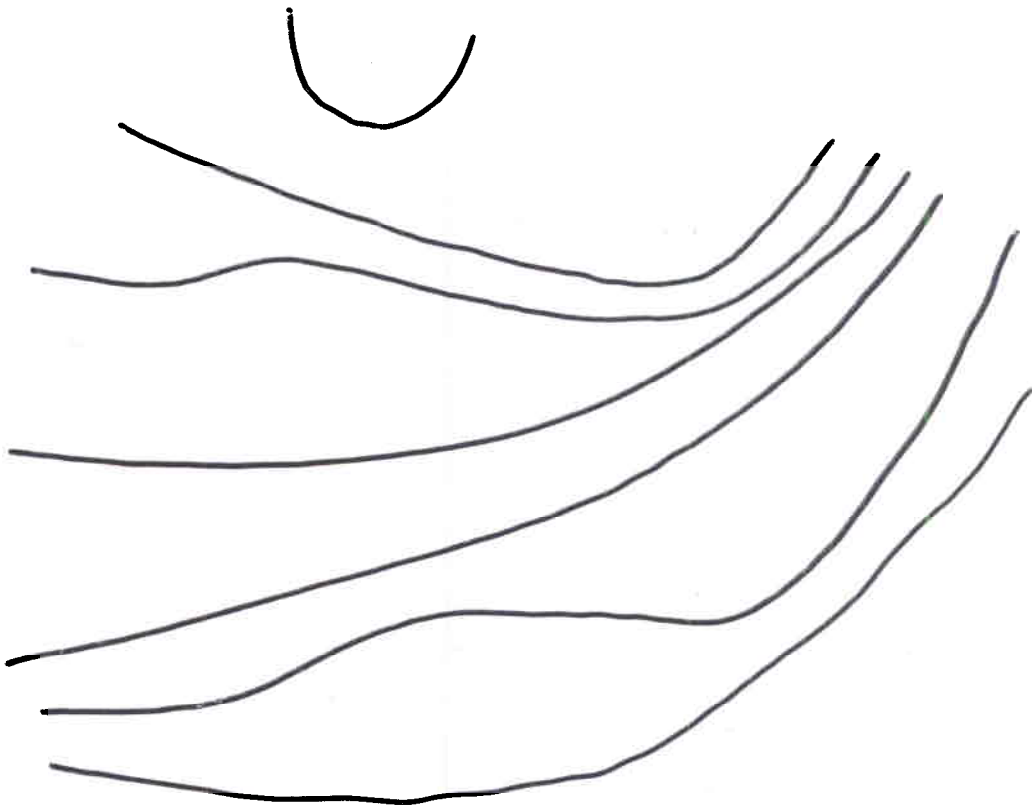
b

c

d

e

t



O	S
∞ 5	a, c, e
∞ 10 8	a, e
∞ 4	b, e
∞ 10	b, d
∞ 12 11 7	d, t
∞ 14 12	t

Operations on the Frontier Set

Initialize to empty (make)

Remove a vertex of smallest distance. (delete min)

Insert a vertex with a predefined distance (insert)

Reduce the distance of a vertex (decrease)

A data structure with these four operations is a heap (priority queue).

The shortest path problem needs 1 make, n delete min, n insert, m decrease operations. heap size = n.

Known Heap Implementations

Pointer-based

leftist trees (Crane)

binomial queues (Vuillemin)

$O(\log n)$ per operation

Implicit (array-based)

2-heaps (Floyd) $O(\log n)$ per operation

$\frac{m}{n}$-heaps (E. Johnson, D.B. Johnson)

$O(\log_{m/n} n)$ per insert or decrease

$O(\frac{m}{n} \log_{m/n} n)$ per delete min

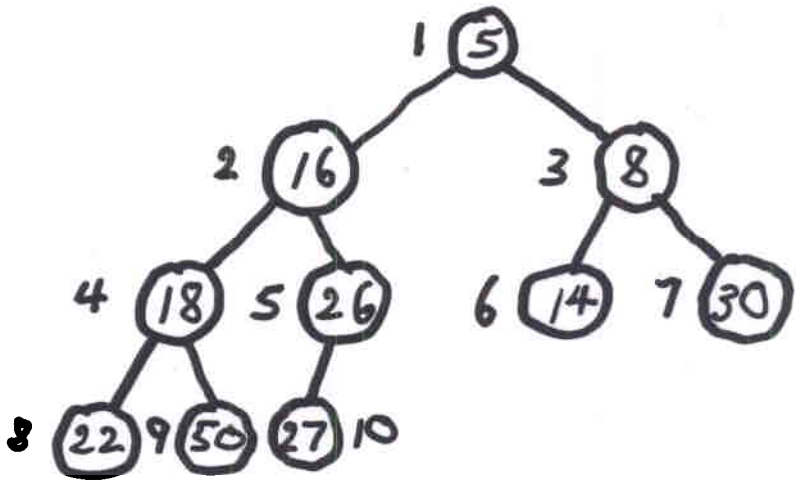
Shortest path time:

Dijkstra: $O(n^2)$

$\frac{m}{n}$-heaps: $O(m \log_{m/n} n)$

others: $O(m \log n)$

2-heap

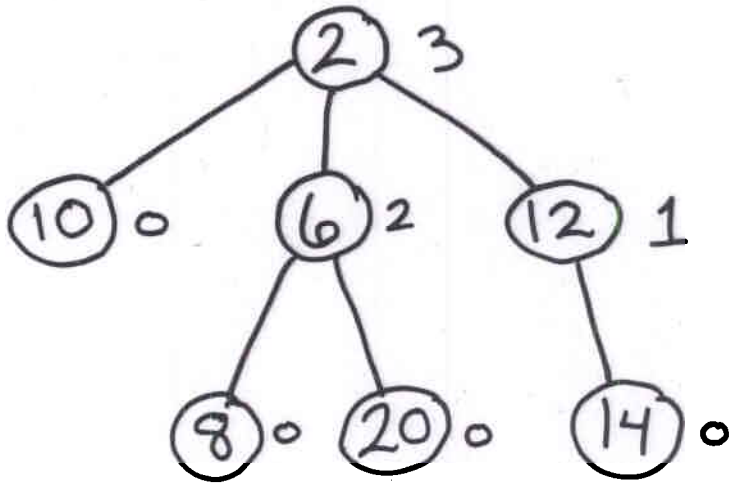


$p(x) = \lfloor x/2 \rfloor$

children $(x) = 2x, 2x+1$

(Lazy) Binomial Heaps

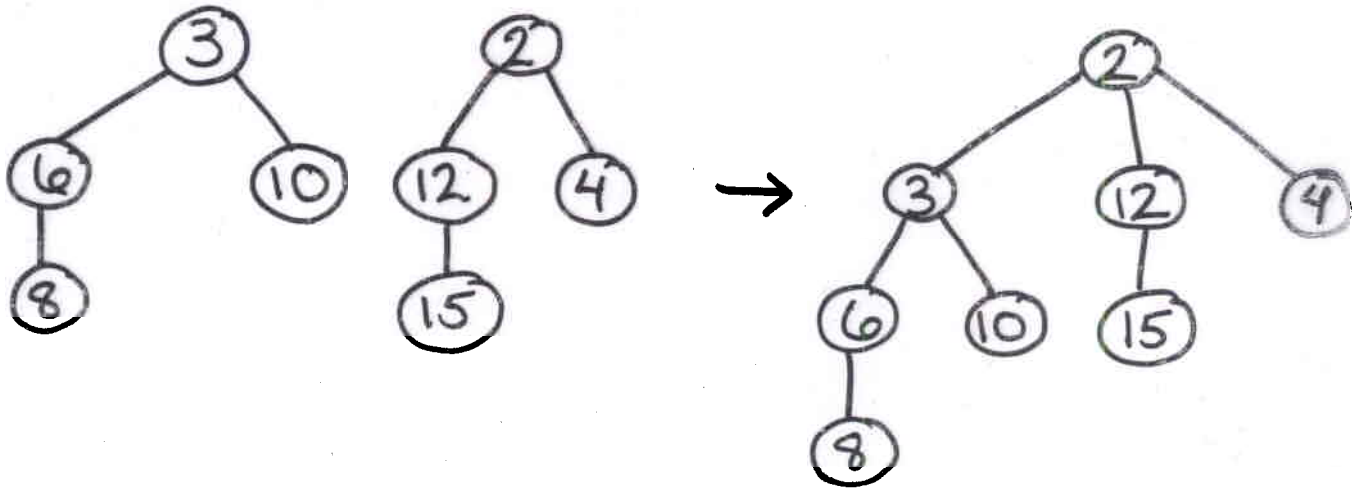
Heap-ordered tree: rooted tree, one item per node, arranged in heap order: item in parent no larger than smallest item in children (root has minimum item).



Rank of a node: number of children

Linking of trees: compare items in roots, make node with smaller item the parent of the node with the larger.

Linking rule: Only link trees with roots of equal rank.



Linking takes $O(1)$ time.

Represent a heap by a forest of heap-ordered trees.

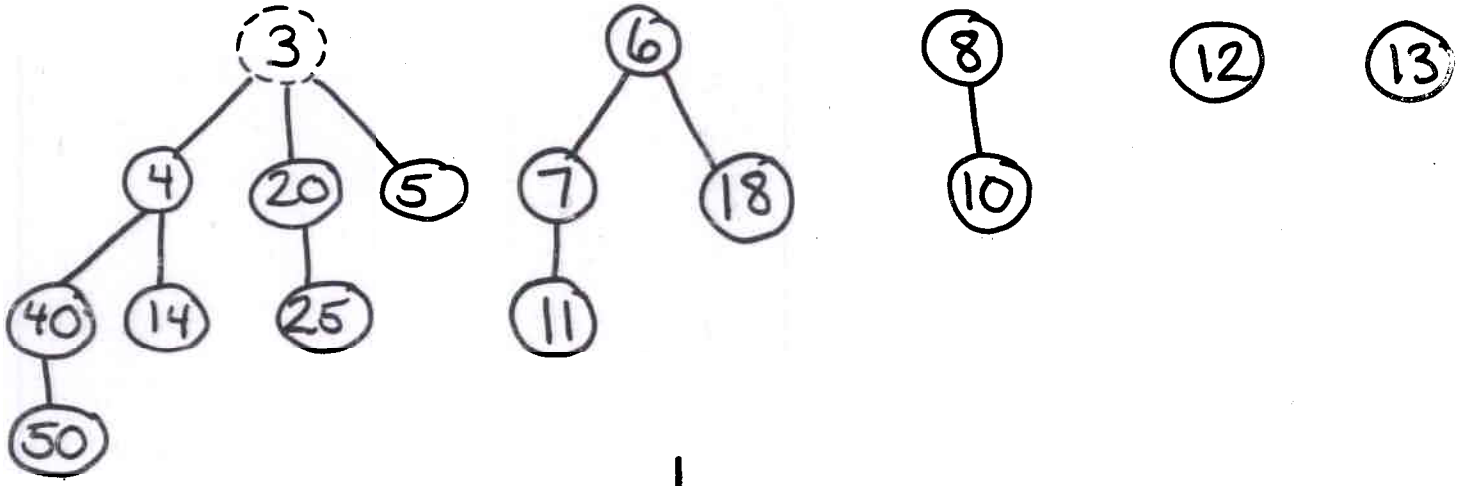
make: create a new, empty forest.

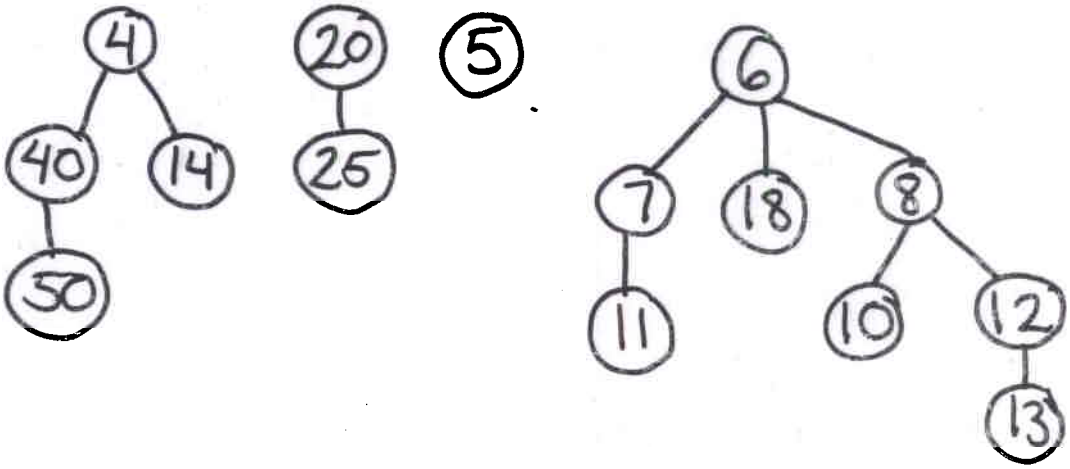
insert: create a one-node tree out of new item and add it to the forest.

delete min:

1. Find root with minimum item.
2. Remove this root and save item.
3. Repeatedly link trees with roots of equal rank, until no two roots have equal rank.

delete min

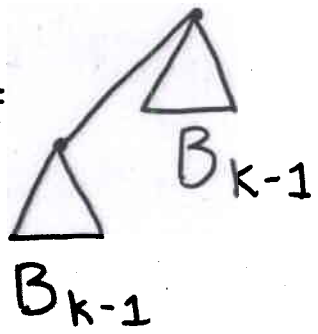




To do linking, use an array of trees with one position per rank.

Facts Needed for Analysis

1. All trees are binomial:

- B_0 $\quad B_k =$ 

B_k has 2^k nodes, root of rank k.

2. Each link reduces number of trees by one.

<u>insert</u> creates one new tree.

<u>delete min</u> creates at most log n new trees, *ends with* at most log n trees.

Φ = # trees

Amortization

Amortized time of an operation ≡ actual time plus net increase in number of trees.

Sum of amortized times of operations = sum of actual times + final number of trees

$\Rightarrow$ Total actual time $\leq$ total amortized time.

Amortized time of make, insert is $O(1)$.

Amortized time of delete min is $O(\log n)$:

k initial trees, at most $\log n$ new trees,

$k + \log n$ actual time, at most $\log n$ final tree

$$A \leq k + \log n + (\log n - k) \leq 2 \log n$$

Implementation Details

Children are ordered from largest rank to smallest.

Each node stores rank, first child, next sibling.

Eager (worst-case) version:

Maintain at most one tree per rank.

Insert becomes $O(\log n)$ worst-case.

Our Goal

Implement a heap with the following time bounds:

$O(\log n)$ for delete min

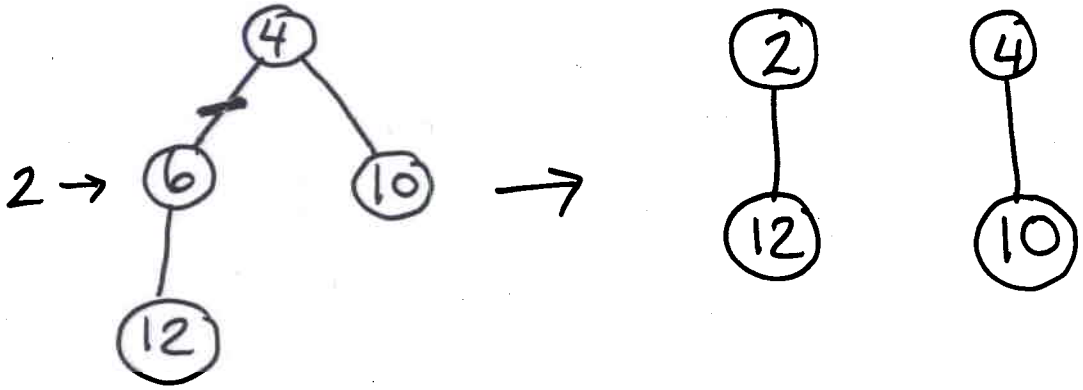
$O(1)$ for other operations

To give $O(n \log n + m)$ for shortest paths.

Our structure achieves these bounds in an amortized (time-averaged) sense.

Extension to decrease (Fibonacci heaps)

decrease: cut edge joining *decreased item's node* to its parent (creates a new tree).



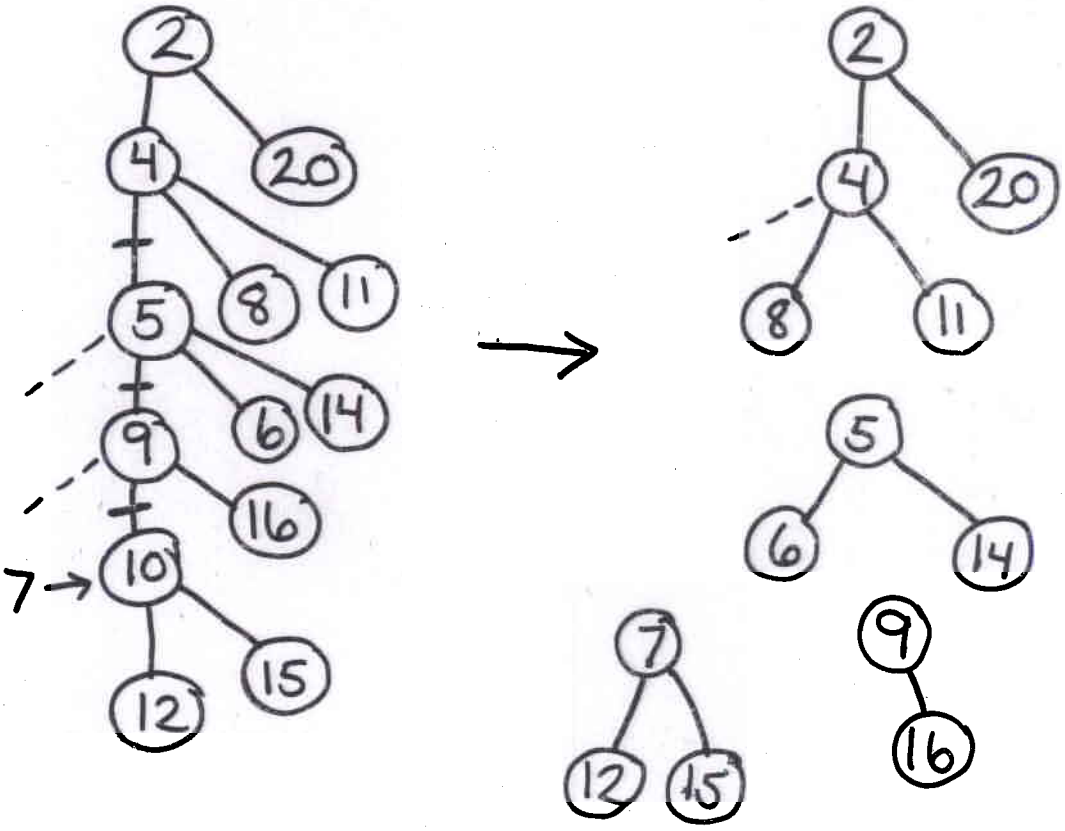
Problem: Trees are no longer binomial ⇒ rank of root is no longer logarithmic in size ⇒

$O(\log n)$ bound for delete min is no longer valid.

Solution: Cascading Cuts

When a node loses a second child (via a cut), cut it from its parent.

Count of lost children restarted when a node becomes a non-root via a link.



Amortized Analysis

Φ = #trees + 2x#marked nodes

$\Rightarrow$ a non-terminating cut

decreases Φ by 1: +1 tree,
-1 marked node

$\Rightarrow$ decrease has $O(1)$ amortized time

insert, delete min analysis as before:

amortized time of delete min =

O(maximum rank)

Facts for Analysis

Marked node: has lost a child.

1. Number of cuts is at most twice the number of decrease operations.

   Each cut in a decrease except the first unmarks a node. The last cut can mark a node.

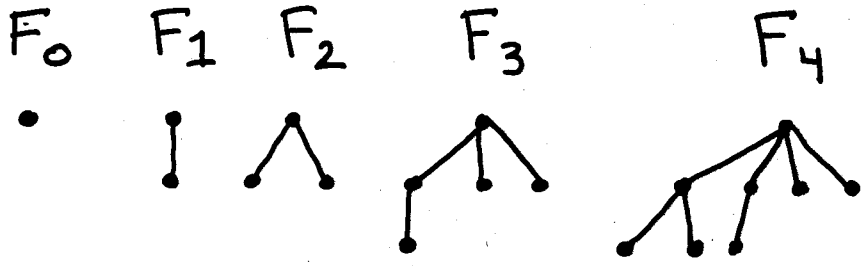
2. A node of rank k has at least $f_{k+2} \geq \phi^k$ descendants.

   $f_0 = 0, f_1 = 1, f_k = f_{k-2} + f_{k-1}$ are the Fibonacci numbers $0, 1, 1, 2, 3, 5, 8, 13, \ldots$

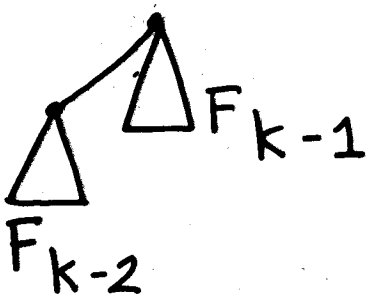
   $\phi = (1 + \sqrt{5})/2$

Worst-Case Trees

(Highest rank for smallest size)



$F_k =$



Lemma. The youngest child of a node of rank k has rank at least $k-2$.

Analysis of delete min is as before with $\log_\phi n$ in place of $\log_2 n$.

Rank Bound

Order the children of a node by time of linking

When k^{th} child linked, parent must have rank $\geq k-1$ $\Rightarrow$ child has rank $\geq k-1$

Child must remain of rank $\geq k-2$, or cut

Let F_k = min # of nodes in a subtree whose root has rank k.

$$F_0 = 1,\ F_1 = 2,\ F_k \geq \sum_{i=0}^{k-2} F_i + 2$$

The Fibonacci #s obey this recurrence with equality (why?), hence

$F_k \geq f_k$ (actually $=$), hence

max rank $= O(\log n)$

Implementation Details

Each node stores its rank, whether it is marked, first child, parent, next sibling, previous sibling

Too many pointers! Bad time, space constants

Extensions of Fibonacci Heaps

meld: combine two disjoint heaps

$O(1)$ time

delete or increase any item

$O(\log n)$ time

add a constant to all items in a heap

$O(1)$ time

Thin heaps: Alternative to Fibonacci heaps

1) Each node has a rank = degree (thick node)
   or 1 + degree (thin node)

2) Children of a node of degree k have ranks

   $k-1, k-2, \ldots, 0$

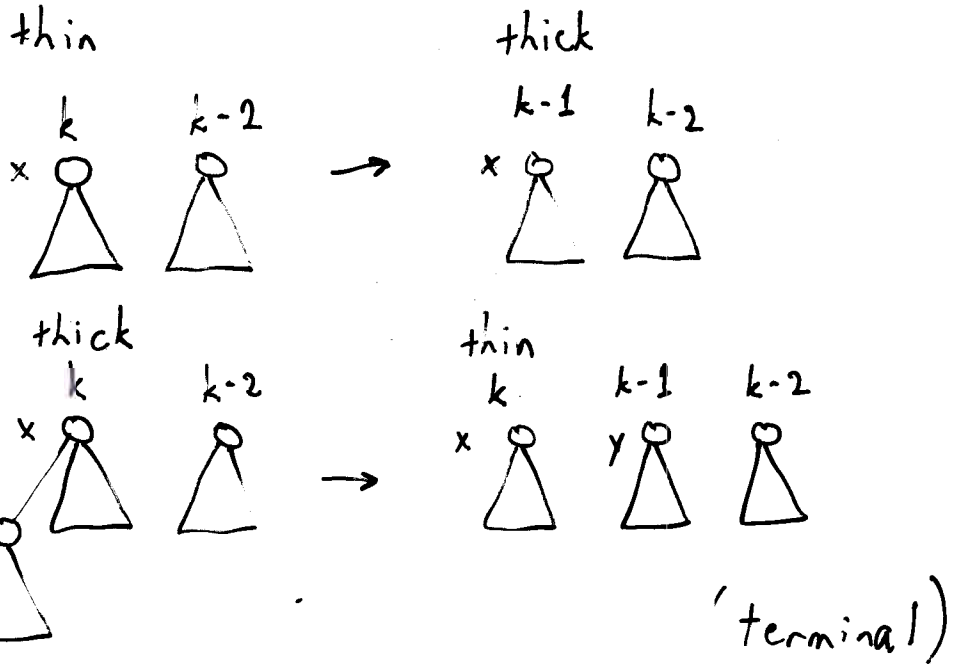
Decrease: removes subtree at decreased node,

creates new tree

May also create a sibling violation (of 2) or

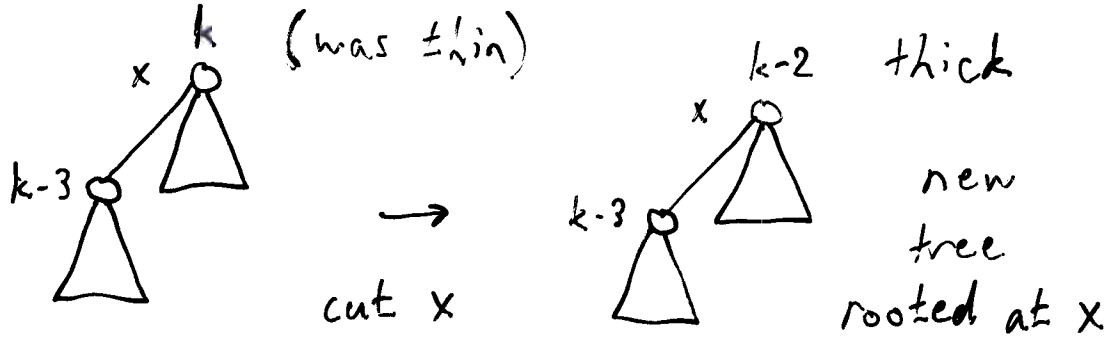
a parent violation (of 1)

Sibling violation



(terminal)

Parent violation



Every non-terminating case consumes a thin node: $\Phi = 2\#\text{thin nodes} + \#\text{roots}$

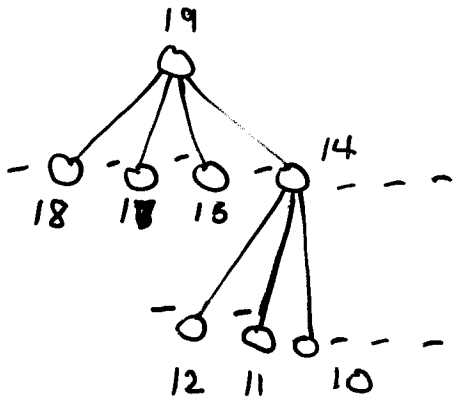
Repairs after a decrease proceed

right-to-left through siblings and

up through leftmost children:

Need parent pointer only for leftmost child:

three pointers per node, instead of four



Size of thin heaps

$$S_k \geq S_{k-1} + S_{k-2} \quad \Rightarrow \quad S_k \geq F_k$$