CS 261: Data Structures

Week 5: Priority queues

Lecture 5b: Fibonacci heaps

David Eppstein
University of California, Irvine

Spring Quarter, 2023

Fibonacci heaps

Overview

A different priority queue structure optimized for Dijkstra:

- Create n-item heap takes time $O(n)$ (same as binary heap)
- Find-and-remove-min is $O(\log n)$ (same as binary tree)
- Insert takes time $O(1)$
- Decrease-priority takes time $O(1)$

$\Rightarrow$ Dijkstra + Fibonacci heaps takes time $O(m + n \log n)$

Can merge heaps (destructively) in time $O(1)$; occasionally useful

All time bounds are amortized, not worst case

In practice, high constant factors make it worse than k-ary heaps

Structure of a Fibonacci heap

It's just a heap-ordered forest!

Each node = an object that stores:

- An element in the priority queue, and its priority
- Pointer to parent node
- Doubly-linked list of children
- Its degree (number of children)
- "Mark bit" (see next slide)

Heap property: all non-roots have priority(parent) $\leq$ priority(self)

Whole heap = object with doubly-linked list of tree roots
and a pointer to the root with the smallest priority

No requirements limiting the shape or number of trees

(But we will see that operations cause their shapes to be limited)

Mark bits and potential function

Each node has a binary "mark bit"

- True: This node is a non-root, and after becoming a non-root it had one of its children removed
- False: It's a root, or has not had any children removed

(We do not allow the removal of more than one child from non-root nodes)

Potential function Φ:
$2 \times$ (number of marked nodes) $+ 1 \times$ (number of trees)

Some easy operations

Create new heap: Make each element its own tree root

Actual time: $O(n)$ $\Delta\Phi = +n$ Total: $O(n)$

Insert new element: Make it the root of its own tree

Actual time: $O(1)$ $\Delta\Phi = +1$ Total: $O(1)$

Merge two heaps: Concatenate their lists of roots

Actual time: $O(1)$ $\Delta\Phi = 0$ Total: $O(1)$

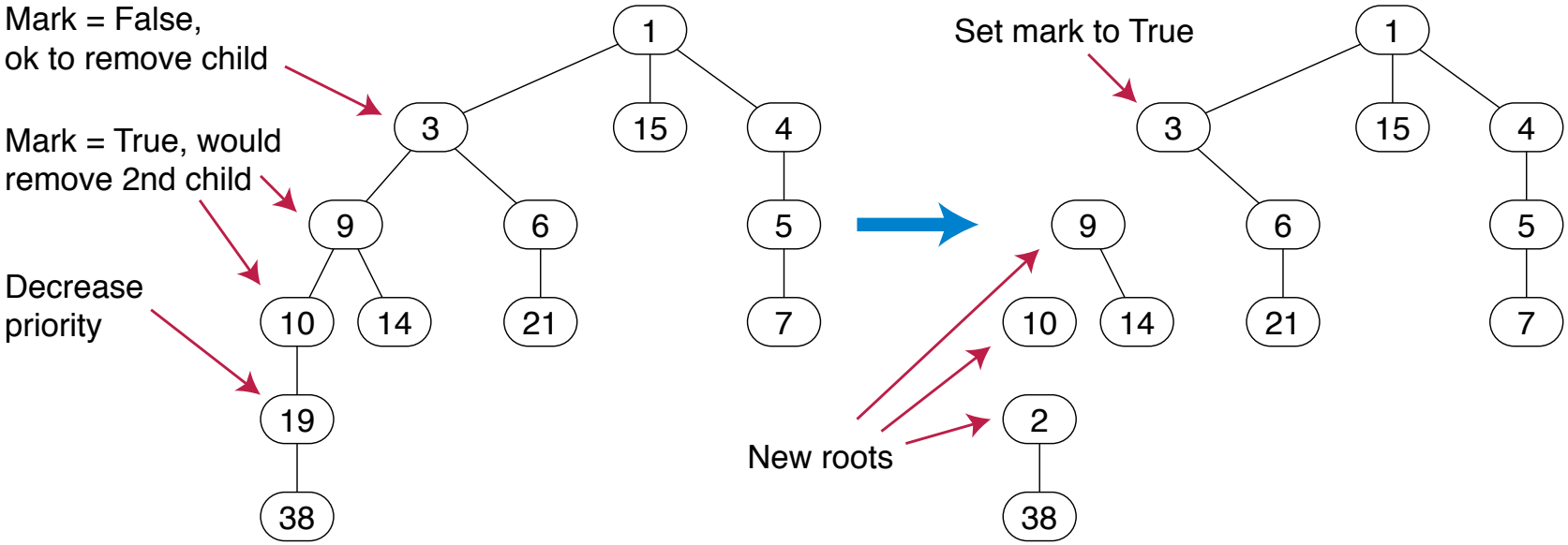
Find min-priority element: Follow pointer to minimum root

Actual time: $O(1)$ $\Delta\Phi = 0$ Total: $O(1)$

Decrease-priority operation, main idea

Cut edge from decreased node to its parent, making it a new root

If this removes 2nd child from any node, make it a new root too



Decrease-priority details

To decrease the priority of element at node x:

1. $y = \text{parent}(x)$
2. Cut link from x to y (making x a root)
3. Set $\text{mark}(x) = \text{False}$
4. While y is marked:
   - $z = \text{parent}(y)$
   - Cut link from y to z (making y a root)
   - Set $\text{mark}(y) = \text{False}$
   - Set $y = z$
5. If y is not a root, set $\text{mark}(y) = \text{True}$

If we cut k links, then the total actual time is $O(k)$. We unmark $k-1$ nodes, create k new trees, and mark at most one node, so $\Delta\Phi \leq -2(k-1) + k + 2 = 4 - k$. The $-k$ term in $\Delta\Phi$ cancels the $O(k)$ actual time leaving $O(1)$ amortized time.

Delete-min operation

The only operation that creates bigger trees from smaller ones...

1. Make all children of deleted node into new roots
2. Merge trees with equal degree until no two are equal

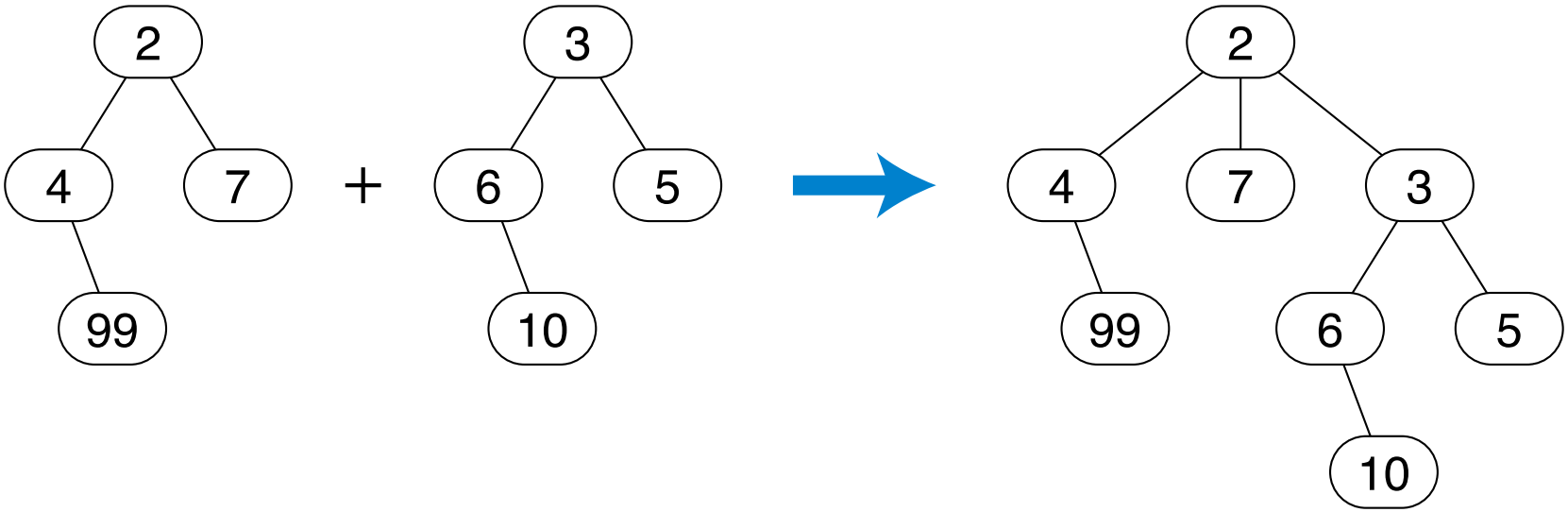
Actual time: $O(\#\text{ children} + \#\text{ merges} + \#\text{ remaining trees})$

$\Delta\Phi$: number of children - number of merges

Total amortized time:
$O(\#\text{ children} + \#\text{ remaining trees}) = O(\max\ \#\text{ children})$

Merging two trees

Make the larger root become the child of the smaller root!



We will only do this when both trees have equal degree (number of children)

$\Rightarrow$ Degree goes up by one

Merging until all trees have different degrees

1. Make a dictionary D, where $D[x]$ will be a tree of degree x
   x will be a small integer, so D can just be an array
2. Make a list L of trees needing to be processed
   Initially $L =$ all trees
3. While L is non-empty:
   - Let T be any tree in L; remove T from L
   - Let x be the degree of tree T
   - If key x is not already in D, set $D[x] = T$
   - Otherwise merge $D[x]$ and T, remove x from D, and add the merged tree to L
4. Return the collection of trees stored in D

Total times through loop = # merges + # additions to D
$=2 \times$ # merges + # remaining trees

Degrees of children

Lemma: Order the children of any node by the time they became children (from a merge operation)

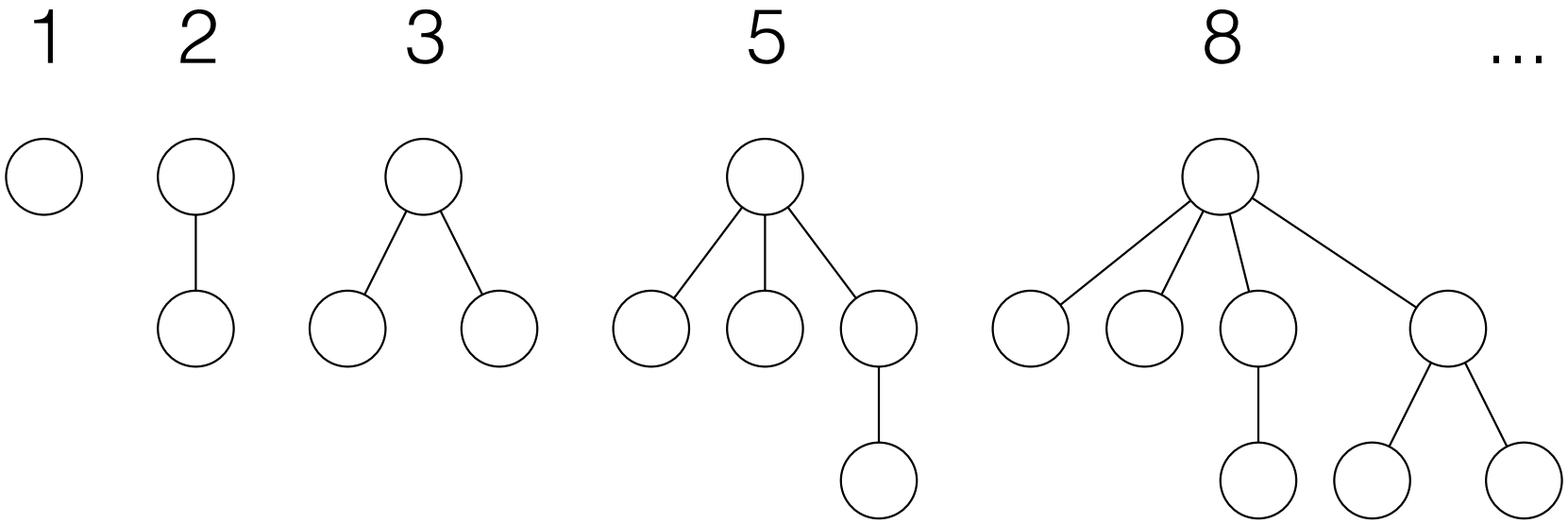
Then their degrees are $\geq 0, 0, 1, 2, 3, 4, 5, \ldots$

Proof: When ith child was merged in, earlier children were already there $\Rightarrow$ parent degree was $\geq i - 1$

At the time of the merge, it had same degree as parent, $\geq i - 1$

After taking away at most one child, its degree is $\geq i - 2$

The smallest trees obeying the degree lemma



Tree size is exponential in degree

Lemma: Subtree of degree x has # nodes $\geq$ Fibonacci(x)

Proof: By induction

Base cases $x = 0$ (one node) and $x = 1$ (two nodes)

For $x > 1$:

$$\begin{aligned}
\text{number of nodes} &\geq \text{root} + \text{numbers in each subtree} \\
&\geq 1 + \sum_{i=1}^{x} \max(1, F_{i-2}) \\
&\geq (\text{same sum for one fewer child}) + F_{x-2} \\
&= F_{x-1} + F_{x-2} = F_x
\end{aligned}$$

Consequences of the degree lemma

Corollary 1: Maximum degree in an n-node Fibonacci heap is $\leq \log_\varphi n$ where $\varphi = (1+\sqrt{5})/2$ is the golden ratio

Corollary 2: Amortized time for delete-min operation is $O(\log n)$

Ongoing research question

Can we achieve the same time bounds as Fibonacci (constant amortized decrease-priority, logarithmic delete-min) with a structure as practical as binary heaps?