Design and Analysis of Algorithms II
Lecture 3 Fibonacci Heaps
Decrease-Key

Gordon Lu (gl1589@nyu.edu)

September 21, 2023

1 Fibonacci Heaps

1.1 Fibonacci Heaps: Last Time

Recall that Fibonacci Heaps are **meldable heaps**.

- All operations take $\Theta(1)$ amortized with the exception of EXTRACT-MIN and DELETE, which are $O(\log(n))$ amortized.

1.2 Fibonacci Heaps: Structure

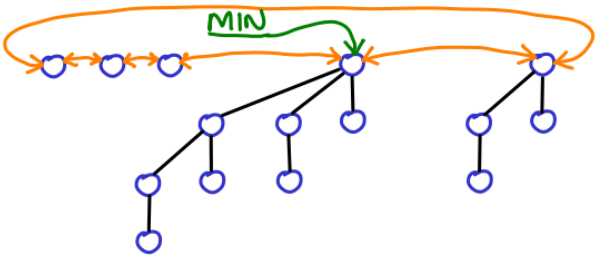
Recall that each node x contains:

1. A pointer $p[x]$ to its parent
2. A pointer $child[x]$ to any one of its children
3. The number of children in the child list $degree[x]$ of x
4. Whether x has lost a child since the last time x was made the child of another node, given by $mark[x]$.
    - Newly created nodes are unmarked, and x becomes unmarked whenever it is made the child of another node. This only comes into significance with **DECREASE-KEY**.

The children of x are linked together in a circular, doubly linked list, called the **child list** of x. Each child y in a child list has pointers **left[y]** and **right[y]** that point to y's left and right siblings.

- If y is an only child, $left[y] = right[y] = y$.

A Fibonacci Heap looks something like the following:



We keep track of the minimum using a pointer, **MIN[H]**.

1.3 Maximum Degree

The amortized analysis assumes there is a known upper bound, $D(n)$ on the maximum degree of any nodes in an n-node Fibonacci heap.

1. If **DECREASE-KEY** and **DELETE** are not supported, $D(n) \leq \lfloor lgn \rfloor$.
2. If **DECREASE-KEY** and **DELETE** are supported, $D(n) = O(lgn)$

1.4 Fibonacci Heaps without Decrease-Key and Delete

Without **DELETE** and **DECREASE-KEY**, the trees in a Fibonacci Heap are simply unordered Binomial trees.

- The degrees of the children are in arbitrary order instead.

2 Fibonacci Heap: Extract-Min

Up until this point, the operations on Fibonacci heaps have been fairly straightforward. Notice that unlike in Binomial Heaps where consolidation occurred frequently, all operations discussed so far have no mention of consolidation.

It is in the **EXTRACT-MIN** operation where consolidation will take place.

FIB-HEAP-EXTRACT-MIN will:

1. Make a root out of each of the minimum node's children and glue it to the root list (while removing the minimum node.
2. Then consolidate the root list by linking roots of equal degree until at most one root remains of each degree.
3. Update minimum pointer once we find the new minimum.

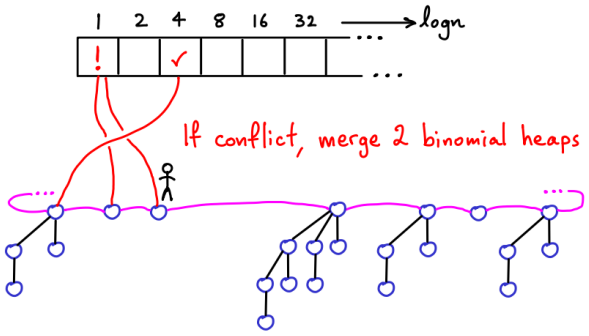
There are some considerations to think about:

1. Since we got rid of the minimum, what happens to **MIN[H]**?
2. How do we detect if roots have the same degree?

To address (1), before we delete the minimum, we can assign **MIN[H]** to be the root to the right of it. This does not need to be the new minimum, as we will reassign **MIN[H]** after we have consolidated the root list.

To address (2), we will allocate an array $A[0 \dots D(n)]$.

1. If $A[i] = y$, then y is currently a root with $degree[y] = i$.
2. We will start from **MIN[H]**, and look for entries in A that point to more multiple roots.



2.1 Amortized Analysis

The total actual work in consolidating the list is: $D(n) + t(H)$, since there are at most $D(n)$ children of the minimum node that are processed, and $t(H)$ is the number of roots in the root list before consolidate is called.
The potential before extracting the minimum node is: $t(H) + 2m(H)$, and the potential afterwards is at most $(D(n) + 1) + 2m(H)$, since at most $D(n) + 1$ roots remain and no nodes are marked during the operation.

The amortized cost is therefore at most:

$$O(D(n)) + t(H) + ((D(n) + 1) + 2m(H)) - (t(H) + 2m(H))$$
$$= O(D(n)) + O(t(H)) - t(H)$$
$$= O(D(n))$$

The crazy Fibonacci Heap stuff starts now!!

3 Fibonacci Heap: Decrease-Key

Assuming that the new key is valid, **FIB-HEAP-DECREASE-KEY** works as follows

1. Move the key's subtree to the root list

2. When a node moves to the root list, mark its parent.

Note that we do not cut a root, we can simply change its key. Also, when a node is moves to the root list, we unmark it.

3.1 Cascading Cut

There is a major problem with what was just described:

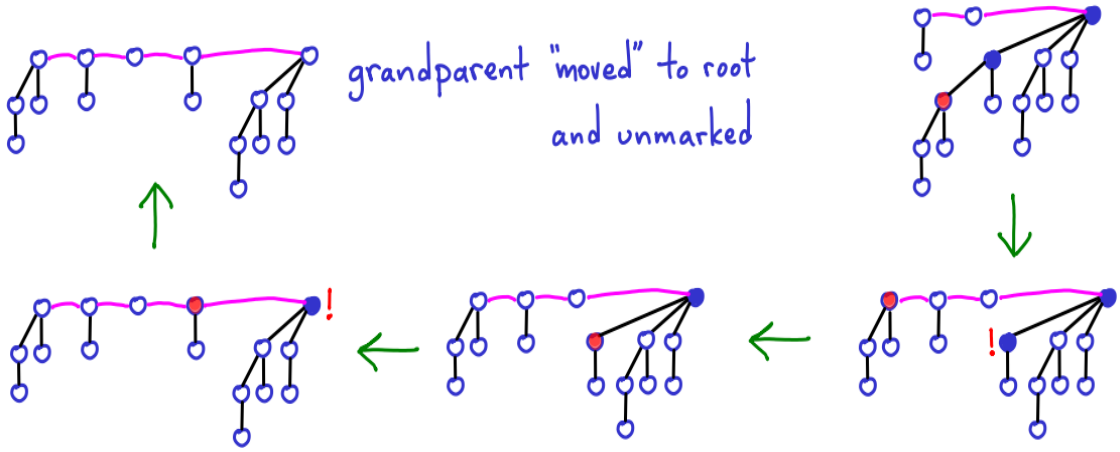
> Cut is fine **once**, but if it is done **many** times, we end up with very inbalanced and strange structures!! (Sausages)

We will prevent this by making use of the **mark** field.

Invariant: No node has > 1 child cut.
If we call decrease key, we set $mark[parent[x]] = true$.
If it happens again, we do the normal cut, then perform a **CASCADING-CUT** if $mark[parent[x]]$ is already true.

- We recursively cut the parents until we reach a node that is the root or has a parent that is not marked.

Below is an example of a **CASCADING-CUT**. This diagram starts from the tree on the right, the left tree is the final state.



3.2 Delete

A delete is simply the following:

1. Call DecreaseKey($-\infty$) on the node.
2. Call ExtractMin

3.3 Decrease Key Cost

The actual cost of **FIB-HEAP-DECREASE-KEY** is 1+ # of cascading cuts. If we let $c = \#$ of cascading cuts, then we have $1 + c$ as the actual cost.

We now compute the amortized cost by computing the change in potential.

The change in the number of trees is $(1 + c)$, since the first cut creates a new tree, and each cascading cut will create 1 more new tree.

, The number of marked nodes will change by $(1 - c)$, since we mark the parent, and for each cascading cut, we unmark at most c nodes.

The change in potential is therefore, $(1 + c) + (2(1 - c)) = 3 - c$. So the potential is: $(1 + c) + (3 - c) = 4$. Therefore, the amortized cost of **FIB-HEAP-DECREASE-KEY** is $O(1)$.

4 Bounding the maximum degree, $D(n)$

In order to prove that the amortized time of **FIB-HEAP-EXTRACT-MIN** and **FIB-HEAP-DELETE** is $O(\log n)$, we must show that upper bound $D(n)$ on the degree of any node of an n-node Fibonacci heap is $O(\log n)$.

1. When all trees in the Fibonacci heap are unordered binomial trees, $D(n) = \lfloor \log n \rfloor$.
2. The cuts that occur in **FIB-HEAP-DECREASE-KEY** may cause trees to violate the unordered binomial tree properties. We need to bound $D(n)$ so that we can guarantee it is $O(\log n)$.

4.1 Key Idea

For each node in the Fibonacci heap, define $size(x)$ to be the number of nodes including x, in the subtree rooted at x. We are going to show that $size(x)$ is exponential in $degree[x]$.

4.2 Lemma 1

Let x be a node in a tree that appears in a Fibonacci heap. Suppose that $degree[x] = k$.
Let $y_1, y_2, \ldots, y_k$ be the children of x in order in which they were linked to x (earliest to latest).
Claim:
$degree[y_1] \geq 0$ and $degree[y_i] \geq i - 2$ for $i = 2, 3, \ldots, k$.
Proof:
For $i \geq 2$, when y_i was linked to x, all of $y_1, y_2, \ldots, y_{i-1}$ were children of x, so we must have had $degree[x] \geq i - 1$. Node y_i is linked to x only if $degree[x] = degree[y_i]$, so $degree[y_i] \geq i - 1$ at that time. Since then, node y_i has lost at most one child, since it would have been cut from x if it had lost two

children. We end the proof by concluding the $degree[y_i] \geq i - 2$ (if it lost two children, it would be a root). ■

4.3 Fibonacci Sequence

Recall that the Fibonacci sequence is defined as follows:

Define $F_0 = 0, F_1 = 1$, we define F_k as the sum of the two preceding numbers:

$$F_k = F_{k-1} + F_{k-2}.$$

Fact 1: For $k \geq 0$,

$$F_{k+2} = 1 + \sum_{i=0}^{k} F_i$$

This is proved using induction.
Fact 2:

$$F_{k+2} \geq \Phi^k$$

where $\Phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio.

- They grow as powers a little greater than 1 and less than 2.

4.4 Using Fibonacci Numbers to Bound $D(n)$

Lemma:
Let x be a node in a Fibonacci heap, and let $deg[x] = k$. Then $size(x) \geq F_{k+2} \geq \Phi^k$.
Proof:
Let s_k be the minimum possible size of any node of degree k in any Fibonacci heap. Trivially, $s_0 = 0, s_1 = 2$. Since s_k is at most $size(x)$ and since adding children to a node cannot decrease the node's size, s_k increases monotonically with k. Since $s_k \leq size(x)$, we now claim the following:
Claim: $s_k \geq F_{k+2}$
We now show by induction on k, $s_k \geq F_{k+2}$ for all nonnegative integers k. In the inductive step, we have:

$$\begin{aligned} s_k &\geq 1 + 1 + (s_0 + s_1 + \cdots + s_{k-2}) \\ &= F_0 + F_1 + (F_2 + F_3 + \cdots + F_k) \\ &= 1 + \sum_{i=0}^{k} F_i \\ &= F_{k+2} \end{aligned}$$

Therefore, we have shown that $size(x) \geq s_k \geq F_{k+2} \geq \Phi^k$. ■
Corollary:

The maximum degree $D(n)$ of any node in an n-node Fibonacci heap is $O(\log n)$.

Proof: Let x be a node in an n-node Fibonacci heap, and let $deg[x] = k$. By the above lemma, we have $n \geq size(x) \geq \Phi^k$. Taking base-$\Phi$ logarithms yields $k \leq \log_\Phi n$ (Since $k \in \mathbb{Z}$, $k \leq \lfloor \log_\Phi n \rfloor$). The maximum degree $D(n)$ of any node is thus $O(\log n)$.