

# **The Design and Analysis of Algorithms**

**Dexter C. Kozen**

With 72 Illustrations



**Springer-Verlag**

New York Berlin Heidelberg London Paris  
Tokyo Hong Kong Barcelona Budapest

## Lecture 9 Fibonacci Heaps

*Fibonacci heaps* were developed by Fredman and Tarjan in 1984 [35] as a generalization of binomial heaps. The main intent was to improve Dijkstra's single-source shortest path algorithm to  $O(m + n \log n)$ , but they have many other applications as well. In addition to the binomial heap operations, Fibonacci heaps admit two additional operations:

<b>decrement</b> ( $h, i, \Delta$ )	decrease the value of $i$ by $\Delta$
<b>delete</b> ( $h, i$ )	remove $i$ from heap $h$

These operations assume that a pointer to the element  $i$  in the heap  $h$  is given.

In this lecture we describe how to modify binomial heaps to admit **delete** and **decrement**. The resulting data structure is called a *Fibonacci heap*. The trees in Fibonacci heaps are no longer binomial trees, because we will be cutting subtrees out of them in a controlled way. We will still be doing links and melds as in binomial heaps. The *rank* of a tree is still defined in the same way, namely the number of children of the root, and as with binomial heaps we only link two trees if they have the same rank.

To perform a **delete**( $i$ ), we might cut out the subtree rooted at  $i$ , remove  $i$ , and **meld** in its newly freed subtrees. We must also search these newly freed subtrees for the minimum root value; this requires  $O(\log n)$  time. In **decrement**( $i, \Delta$ ), we decrement the value of  $i$  by  $\Delta$ . The new value of  $i$  might violate the heap order, since it might now be less than the value of  $i$ 's parent. If so, we might simply cut out the subtree rooted at  $i$  and **meld** it into the heap.

The problem here is that the  $O(\log n)$  time bound on **deletemin** described in the last lecture was highly dependent on the fact that the size of  $B_k$  is exponential in  $k$ , *i.e.* the trees are bushy. With **delete** and **decrement** as described above, cutting out a lot of subtrees might make the tree scraggly, so that the analysis is no longer valid.

## 9.1 Cascading Cuts

The way around this problem is to limit the number of cuts among the children of any vertex to two. Although the trees will no longer be binomial trees, they will still be bushy in that their size will be exponential in their rank.

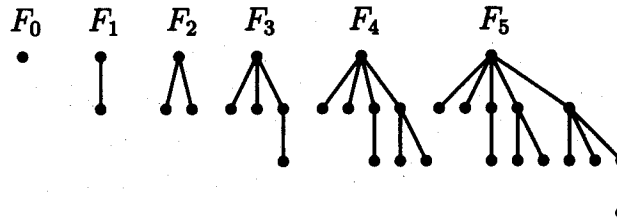
For this analysis, we will set up a savings account for every vertex. The first time a child is cut from vertex  $p$ , charge to the operation that caused the cut two extra credits and deposit them to the account of  $p$ . Not only does this give two extra credits to use later, it also marks  $p$  as having had one child cut already. When a second child is cut from  $p$ , cut  $p$  from its parent  $p'$  and **meld**  $p$  into the heap, paying for it with one of the extra credits that was deposited to the account of  $p$  when its first child was cut. The other credit is left in the account of  $p$  in order to maintain the invariant that each tree in the heap have a credit on deposit. If  $p$  was the second child cut from its parent  $p'$ , then  $p'$  is cut from its parent; again, this is already paid for by the operation that cut the first child of  $p'$ . These cuts can continue arbitrarily far up the tree; this is called *cascading cuts*. However, all these cascading cuts are already paid for. Thus **decrement** is  $O(1)$ , and **delete** will still be  $O(\log n)$  provided our precautions have guaranteed that the sizes of trees are still exponential in their rank.

**Theorem 9.1** *The size of a tree with root  $r$  in a Fibonacci heap is exponential in rank ( $r$ ).*

*Proof.* Fix a point in time. Let  $x$  be any vertex and let  $y_1, \dots, y_m$  be the children of  $x$  at that point, arranged in the order in which they were linked into  $x$ . We show that rank ( $y_i$ ) is at least  $i - 2$ . At the time that  $y_i$  was linked into  $x$ ,  $x$  had at least the  $i - 1$  children  $y_1, \dots, y_{i-1}$  (it may have had more that have since been cut). Since only trees of equal rank are linked,  $y_i$  also had at least  $i - 1$  children at that time. Since then, at most one child of  $y_i$  has been cut, or  $y_i$  itself would have been cut. Therefore the rank of  $y_i$  is at least  $i - 2$ .

We have shown that the  $i^{\text{th}}$  child of any vertex has rank at least  $i - 2$ . Let  $F_n$  be the smallest possible tree of rank  $n$  satisfying this property. The first few  $F_n$  are illustrated below.

Miller



Observe that  $F_0, F_1, F_2, F_3, F_4, F_5, \dots$ , are of size 1, 2, 3, 5, 8, 13, ..., respectively. This sequence of numbers is called the *Fibonacci sequence*, in which each number is obtained by adding the previous two. It therefore suffices to show that the  $n^{\text{th}}$  Fibonacci number  $f_n = |F_n|$  is exponential in  $n$ .


Specifically, we show that  $f_n \geq \varphi^n$ , where  $\varphi = \frac{1+\sqrt{5}}{2} \approx 1.618\dots$ , the positive root of the quadratic  $x^2 - x - 1$ . The proof proceeds by induction on  $n$ .

For the basis,  $f_0 = 1 \geq \varphi^0$  and  $f_1 = 2 \geq \varphi^1$ . Now assume that  $f_n \geq \varphi^n$  and  $f_{n+1} \geq \varphi^{n+1}$ . Then

$$\begin{aligned} f_{n+2} &= f_{n+1} + f_n \\ &\geq \varphi^{n+1} + \varphi^n \\ &= \varphi^n(\varphi + 1) \\ &= \varphi^n \cdot \varphi^2 \text{ since } \varphi^2 = \varphi + 1 \\ &= \varphi^{n+2}. \end{aligned}$$

□

The real number  $\varphi$  is often called the *golden ratio*. It was considered the most perfect proportion for a rectangle by the ancient Greeks because it makes the ratio of the length of the longer side to the length of the shorter side equal to the ratio of the sum of the lengths to the length of the longer side.



$$\varphi = \frac{a}{b} = \frac{a+b}{a}$$

(The picture is actually 81pt  $\times$  50pt, giving a ratio of 1.62. Apologies to the ancient Greeks.)

The golden ratio  $\varphi$  is more closely related to the Fibonacci sequence than is apparent from the proof of Theorem 9.1. Consider the linear system

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} f_n \\ f_{n+1} \end{bmatrix} = \begin{bmatrix} f_{n+1} \\ f_{n+2} \end{bmatrix} \quad (14)$$

which generates the Fibonacci sequence:

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^n \begin{bmatrix} f_0 \\ f_1 \end{bmatrix} = \begin{bmatrix} f_n \\ f_{n+1} \end{bmatrix}.$$

Let  $F$  denote the  $2 \times 2$  matrix in (14). The eigenvalues of  $F$  are  $\varphi$  and  $\varphi' = \frac{1-\sqrt{5}}{2}$ , the two roots of its characteristic polynomial

$$\det(xI - F) = x^2 - x - 1.$$

The eigenvectors associated with  $\varphi$  and  $\varphi'$  are

$$\begin{bmatrix} 1 \\ \varphi \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 \\ \varphi' \end{bmatrix},$$

respectively, of which the former is dominant. Successive applications of a matrix to a vector with a nonzero component in the direction of a dominant eigenvector, suitably scaled, will generate a sequence of vectors converging to that dominant eigenvector. Thus

$$\left( \varphi^{-1} \cdot \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \right)^n \begin{bmatrix} f_0 \\ f_1 \end{bmatrix} = \varphi^{-n} \cdot \begin{bmatrix} f_n \\ f_{n+1} \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ \varphi \end{bmatrix}$$

as  $n \rightarrow \infty$ ; in other words, the ratio of successive Fibonacci numbers tends to  $\varphi$ .

## 9.2 Fibonacci Heaps and Dijkstra's Algorithm

We can use Fibonacci heaps to implement Dijkstra's single-source shortest-path algorithm (Algorithm 5.1) in  $O(m + n \log n)$  time. We store the elements of  $V - X$  in a Fibonacci heap. The value of the element  $v$  is  $D(v)$ . The initialization uses the **makeheap** operation and takes linear time. We use the **decrement** operation to implement the statement

$$D(v) := \min(D(v), D(u) + \ell(u, v)).$$

This requires constant time for each edge, or  $O(m)$  time in all. We use the **deletemin** operation to remove a vertex from the set of unreached vertices. This takes  $O(\log n)$  time for each deletion, or  $O(n \log n)$  time in all.

Another application of Fibonacci heaps is in Prim's algorithm for minimum spanning trees. We leave this application as an exercise (Homework 4, Exercise 1).