Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms

MICHAEL L. FREDMAN

University of California, San Diego, La Jolla, California

AND

ROBERT ENDRE TARJAN

AT&T Bell Laboratories, Murray Hill, New Jersey

Abstract. In this paper we develop a new data structure for implementing heaps (priority queues). Our structure, *Fibonacci heaps* (abbreviated *F-heaps*), extends the binomial queues proposed by Vuillemin and studied further by Brown. F-heaps support arbitrary deletion from an n-item heap in $O(\log n)$ amortized time and all other standard heap operations in $O(1)$ amortized time. Using F-heaps we are able to obtain improved running times for several network optimization algorithms. In particular, we obtain the following worst-case bounds, where n is the number of vertices and m the number of edges in the problem graph:

(1) $O(n \log n + m)$ for the single-source shortest path problem with nonnegative edge lengths, improved from $O(m \log_{(m/n+2)} n)$;
(2) $O(n^2 \log n + nm)$ for the all-pairs shortest path problem, improved from $O(nm \log_{(m/n+2)} n)$;
(3) $O(n^2 \log n + nm)$ for the assignment problem (weighted bipartite matching), improved from $O(nm \log_{(m/n+2)} n)$;
(4) $O(m\beta(m, n))$ for the minimum spanning tree problem, improved from $O(m \log \log_{(m/n+2)} n)$, where $\beta(m, n) = \min \{i \mid \log^{(i)} n \le m/n\}$. Note that $\beta(m, n) \le \log^* n$ if $m \ge n$.

Of these results, the improved bound for minimum spanning trees is the most striking, although all the results give asymptotic improvements for graphs of appropriate densities.

Categories and Subject Descriptors: E.1 **[Data]**: Data Structures—*trees*; *graphs*; F.2.2 **[Analysis of Algorithms and Problem Complexity]**: Nonnumerical Algorithms and Problems—*computations on discrete structures*; *sorting and searching*; G.2.2 **[Discrete Mathematics]**: Graph Theory—*graph algorithms*; *network problems*; *trees*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Heap, matching, minimum spanning tree, priority queue, shortest path

A preliminary version of this paper appeared in the *Proceedings of the 25th IEEE Symposium on the Foundations of Computer Science* (Singer Island, Fla., Oct. 24–26). IEEE, New York, 1984, pp. 338–346, © IEEE. Any portion of this paper that appeared in the preliminary version is reprinted with permission.

This research was supported in part by the National Science Foundation under grant MCS 82-04031.

Authors' addresses: M. L. Fredman, Electrical Engineering and Computer Science Department, University of California, San Diego, La Jolla, CA 92093; R. E. Tarjan, AT&T Bell Laboratories, Murray Hill, NJ 07974.

1. *Introduction*

A *heap* is an abstract data structure consisting of a set of *items*, each with a real-valued *key*, subject to the following operations:

make heap:	Return a new, empty heap.
insert (i, h):	Insert a new item i with predefined key into heap h.
find min (h):	Return an item of minimum key in heap h. This operation does not change h.
delete min (h):	Delete an item of minimum key from heap h and return it.

In addition, the following operations on heaps are often useful:

meld (h_1, h_2):	Return the heap formed by taking the union of the item-disjoint heaps h_1 and h_2. This operation destroys h_1 and h_2.
decrease key (Δ, i, h):	Decrease the key of item i in heap h by subtracting the nonnegative real number Δ. This operation assumes that the position of i in h is known.
delete (i, h):	Delete arbitrary item i from heap h. This operation assumes that the position of i in h is known.

Note that other authors have used different terminology for heaps. Knuth [15] called heaps "priority queues." Aho et al. [1] used this term for heaps not subject to melding and called heaps subject to melding "mergeable heaps."

In our discussion of heaps, we assume that a given item is in only one heap at a time and that a pointer to its heap position is maintained. It is important to remember that heaps do *not* support efficient searching for an item.

Remark. The heap parameter h in *decrease key* and *delete* is actually redundant, since item i determines h. However, our implementations of these operations require direct access to h, which must be provided by an auxiliary data structure if h is not a parameter to the operation. If melding is allowed, a data structure for disjoint set union [26] must be used for this purpose; otherwise, a pointer from each item to the heap containing it suffices.

Vuillemin [27] invented a class of heaps, called *binomial queues*, that support all the heap operations in $O(\log n)$ worst-case time. Here n is the number of items in the heap or heaps involved in the operation. Brown [2] studied alternative representations of binomial heaps and developed both theoretical and experimental running-time bounds. His results suggest that binomial queues are a good choice in practice if meldable heaps are needed, although several other heap implementations have the same $O(\log n)$ worst-case time bound. For further discussion of heaps, see [1], [2], [15], and [24].

In this paper we develop an extension of binomial queues called *Fibonacci heaps*, abbreviated *F-heaps*. F-heaps support *delete min* and *delete* in $O(\log n)$ amortized time, and all the other heap operations, in particular *decrease key*, in $O(1)$ amortized time. For situations in which the number of deletions is small compared to the total number of operations, F-heaps are asymptotically faster than binomial queues.

Heaps have a variety of applications in network optimization, and in many such applications, the number of deletions is relatively small. Thus we are able to use F-heaps to obtain asymptotically faster algorithms for several well-known network optimization problems. Our original purpose in developing F-heaps was to speed up Dijkstra's algorithm for the single-source shortest path problem with non-negative length edges [5]. Our implementation of Dijkstra's algorithm runs in $O(n\log n + m)$ time, improved from Johnson's $O(m\log_{(m/n+2)}n)$ bound [13, 24].

Various other network algorithms use Dijkstra's algorithm as a subroutine, and for each of these, we obtain a corresponding improvement. Thus we can solve both the all-pairs shortest path problem with possibly negative length edges and the assignment problem (bipartite weighted matching) in $O(n^2\log n + nm)$ time, improved from $O(nm\log_{(m/n+2)} n)$ [24].

We also obtain a faster method for computing minimum spanning trees. Our bound is $O(m\beta)(m, n))$, improved from $O(m\log\log_{(m/n+2)}n)$ [4, 24], where

$$\beta(m, n) = \min\{i \mid \log^{(i)}n \le m/n\}.$$

All our bounds for network optimization are asymptotic improvements for graphs of intermediate density ($n \ll m \ll n^2$). Our bound for minimum spanning trees, which is perhaps our most striking result, is an asymptotic improvement for sparse graphs as well.

The remainder of the paper consists of five sections. In Section 2 we develop and analyze F-heaps. In Section 3 we discuss variants of F-heaps and some additional heap operations. In Section 4 we use F-heaps to implement Dijkstra's algorithm. In Section 5 we discuss the minimum spanning tree problem. In Section 6 we mention several more recent results and remaining open problems.

2. *Fibonacci Heaps*

To implement heaps we use heap-ordered trees. A *heap-ordered tree* is a rooted tree containing a set of items, one item in each node, with the items arranged in *heap order*: If x is any node, then the key of the item in x is no less than the key of the item in its parent $p(x)$, provided x has a parent. Thus the tree root contains an item of minimum key. The fundamental operation on heap-ordered trees is *linking*, which combines two item-disjoint trees into one. Given two trees with roots x and y, we link them by comparing the keys of the items in x and y. If the item in x has the smaller key, we make y a child of x; otherwise, we make x a child of y. (See Figure 1.)

A *Fibonacci heap* (*F-heap*) is a collection of item-disjoint heap-ordered trees. We impose no explicit constraints on the number or structure of the trees; the only constraints are implicit in the way the trees are manipulated. We call the number of children of a node x its *rank* $r(x)$. There is no constant upper bound on the rank of a node, although we shall see that the rank of a node with n descendants is $O(\log n)$. Each node is either *marked* or *unmarked*; we shall discuss the use of marking later.

In order to make the correctness of our claimed time bounds obvious, we assume the following representation of F-heaps: Each node contains a pointer to its parent (or to a special node *null* if it has no parent) and a pointer to one of its children. The children of each node are doubly linked in a circular list. Each node also contains its rank and a bit indicating whether it is marked. The roots of all the trees in the heap are doubly linked in a circular list. We access the heap by a pointer to a root containing an item of minimum key; we call this root the *minimum node* of the heap. A minimum node of null denotes an empty heap. (See Figure 2.) This representation requires space in each node for one item, four pointers, an integer, and a bit. The double linking of the lists of roots and children makes deletion from such a list possible in $O(1)$ time. The circular linking makes concatenation of lists possible in $O(1)$ time.