Theory of Trinomial Heaps

Tadao Takaoka

Department of Computer Science, University of Canterbury
Christchurch, New Zealand
tad@cosc.canterbury.ac.nz

Abstract. We design a new data structure, called a trinomial heap, which supports a decrease-key in $O(1)$ time, and an insert operation and delete-min operation in $O(\log n)$ time, both in the worst case, where n is the size of the heap. The merit of the trinomial heap is that it is conceptually simpler and easier to implement than the previously invented relaxed heap. The relaxed heap is based on binary linking, while the trinomial heap is based on ternary linking.

1 Introduction

The Fibonacci heap was invented by Fredman and Tarjan [3] in 1987. Since then, there have been two alternatives that can support n insert and delete-min operations, and m decrease-key operations in $O(m + n \log n)$ time. The relaxed heaps by Driscoll, et. al [2] have the same overall complexity with decrease-key with $O(1)$ worst case time, but are difficult to implement. The other alternative is the 2-3 heap invented by the author [5], which supports the same set of operations with the same time complexity. Although the 2-3 heap is simpler and slightly more efficient than the Fibonacci heap, the $O(1)$ time for decrease-key is in the amortized sense, meaning that the time from one decrease key to the next can not be smooth. Two representative application areas for these operations will be the single source shortest path problem and the minimum cost spanning tree problem. Direct use of these operations in Dijkstra's [1] and Prim's [4] algorithms with those data structures will solve these two problems in $O(m + n \log n)$ time, where n and m are the numbers of vertices and edges of the given graph. Logarithm here is with base 2, unless otherwise specified.

A Fibonacci heap is a generalization of a binomial queue invented by Vuillemin [6]. When the key value of a node v is decreased, the subtree rooted at v is removed and linked to another tree at the root level in the Fibonacci and 2-3 heap. This removal of a subtree may cause a chain effect to keep structural properties of those heaps, resulting in worst case time greater than $O(1)$, although it is $O(1)$ amortized time. As an alternative to the Fibonacci heap, Driscoll, et. al. proposed a data structure called a relaxed heap, whose shape is the same as that of a binomial queue. The requirement of heap order is relaxed in the relaxed heap; a certain number of nodes are allowed to have smaller key values than those of their parents. Those nodes are called bad children in [2] and inconsistent nodes in this paper. On the other hand, the 2-3 heaps is proposed as

D.-Z. Du et al. (Eds.): COCOON 2000, LNCS 1858, pp. 362–372, 2000.

another alternative to the Fibonacci heap. While the Fibonacci heap is based on binary linking, 2-3 heaps are based on ternary linking; we link three roots of three trees in increasing order according to the key values. We call this path of three nodes a trunk. We allow a trunk to shrink by one. If there is requirement of further shrink, we make adjustment by moving a few subtrees from nearby positions. This adjustment may propagate, taking time more than $O(1)$.

In this paper, we combine the ideas of the relaxed heap and 2-3 heap; we use ternary linking and allow a certain number of inconsistent nodes. Ternary linking gives us more flexibility to keep the number of inconsistent nodes under control. The new data structure is called a trinomial heap, and is simpler and easier to implement than the relaxed heap. In the relaxed heap which is based on binary linking, we must keep each bad child at the rightmost branch of its parent, causing difficult book-keeping. The trinomial heap is constructed by ternary linking of trees repeatedly, that is, repeating the process of making the product of a linear tree and a tree of lower dimension. This general description of r-ary trees is given in Section 2. The definition of trinomial heaps and their operations are given in Section 3. In Section 4, we implement decrease-key in $O(1)$ amortized time. In Section 5, we implement it with $O(1)$ worst case time. In Section 6, we give several practical considerations for implementation. Section 7 concludes the paper.

In this paper we analyze the number of key comarisons for computing time as the times for other operations are proportional to it. We mainly deal with decrease-key and delete-min in this paper since the main application areas are the single source shortest path problem and the minimum cost spanning tree problem and we can build up the heap of size n at the beginning. The nodes not adjacent with the source can be inserted with infinite key values at the beginning. As shown in Section 2, the time for building the heap can be $O(n)$. Thus we can concentrate our discussion on decrease-key and delete-min after the heap is built.

2 Polynomial Queues and Their Analysis

We borrow some materials from [5] in this section, as the trinomal heap and the 2-3 heap share the same basic structure. We define algebraic operations on rooted trees as the basis for priority queues. A tree consists of nodes and branches, each branch connecting two nodes. The root of tree T is denoted by $root(T)$. A linear tree of size r is a liner list of r nodes such that its first element is regarded as the root and a branch exists from a node to the next. The linear tree of size r is expressed by bold face $\mathbf{r}$. Thus a single node is denoted by $\mathbf{1}$, which is an identity in our tree algebra. The empty tree is denoted by $\mathbf{0}$, which serves as the zero element. A product of two trees S and T, $P = ST$, is defined in such a way that every node of S is replaced by T and every branch in S connecting two nodes u and v now connects the roots of the trees substituted for u and v in S. Note that $\mathbf{2} * \mathbf{2} \neq \mathbf{4}$, for example, and also that $ST \neq TS$ in general. The symbol " $*$ "