

and the operations INSERT, DELETEMIN and MERGE in $O(\log n)$ time.

13.3 Randomized Meldable Heaps

Next we consider another implementation of heaps that uses randomization. The performance of this implementation is based on properties of random walks in binary trees. Take any binary tree T and augment it into a binary tree T' by adding one or two leaves to any node that has one or zero children, respectively. Note T' is *full*; all internal nodes of T' have 2 children, T' has $n + 1$ leaves and n internal nodes. A *random walk* in T' is a root-to-leaf path obtained as follows: Start at the root, and while the current node is not a leaf, toss a fair coin and go to the left child of the current node or the right child of the current node depending on whether the coin comes up heads or tails, respectively.

Let us consider the expected length of a random walk in T' . Observe that, if a leaf v of T' has depth $\text{depth}(v)$ then the probability that the walk terminates at v is exactly $p_v = 2^{-d}$. Thus, the expected length of the random walk is

$$\sum_{v \text{ is a leaf of } T'} 2^{-\text{depth}(v)} \text{depth}(v) = \sum_{v \text{ is a leaf of } T'} p_v \log(1/p_v) = H$$

where H is the entropy of the probability distribution that the random walk induces over the leaves of T' . We saw in Chapter 5 that a probability distribution of r items has entropy at most $\log_2 r$, so the value of H above is at most $\log(n + 1)$. That is, a random walk in a full binary tree with $n + 1$ leaves has expected length at most $\log(n + 1)$, and *this is true regardless of the shape of the tree*.

Next we see how to use this fact to obtain an efficient algorithm for merging two heaps. Let T_1 and T_2 be two heap-ordered binary trees with n_1 and n_2 nodes, respectively. If either T_1 or T_2 is empty, then we are done. Otherwise we first check which of the two trees has a smaller root, since this will become the root of the new heap. Suppose, wlog, that T_1 has the smaller root. Then we toss a fair coin and recursively merge T_2 with the left or right subtree of T_1 depending on whether the coin came up heads or tails, respectively.

What is the expected running time of this merge operation? We can view the merge operation as taking two random walks on the augmented full binary trees T'_1 and T'_2 , where we interleave the steps of the random walks depending on the values in the trees. The merge operation completes when one of the random walks reaches a leaf. Together, the two random walks have expected length at most $\log_2(n_1 + 1) + \log_2(n_2 + 1)$. Therefore, the expected cost of a merge operation on two heaps of size n_1 and n_2 is $O(\log n_1 + \log n_2)$. Again, this does not depend on the shapes of the trees representing the two heaps.

Now, once we have an efficient merge operation, all other operations are easy to implement. To insert a new element x into a heap we create a new heap containing only x and merge it with the old heap. To delete the minimum element in a heap we simply remove the root and merge the left and right subtrees. To delete an arbitrary element in a heap we delete it and perform two merges to merge the resulting three heaps. And so on.

Theorem 32. *Randomized meldable heaps support the operations MAKEQUEUE and FINDMIN in constant time and the operations INSERT, DELETE, DELETEMIN and MERGE in $O(\log n)$ expected time.*

With a little more work, Theorem 32 can be strengthened to show that all the operations on heaps can be done in $O(\log n)$ time with high probability. This is because each step of a random walk has probability at least $1/2$ of decreasing the size of the current subtree by a factor of at least $1/2$. An easy application of Chernoff's bounds then shows that the probability that a random walk requires more than $(2 + c) \log_2 n$ steps is $O(n^{-c})$.

13.4 Skew Heaps

This section is still to be written.

13.5 Discussion and References

The linked-list data structure for disjoint sets is due to Tarjan, though much more efficient data structures are possible [3]. Leftist heaps are described by Knuth [2]. Randomized meldable heaps are due to Gambin and Malinowsky [1].

Bibliography

- [1] Gambin and Malinowski. Randomized meldable priority queues. In *Theory and Practice of Informatics, Seminar on Current Trends in Theory and Practice of Informatics*, volume 25 of *LNCS*. 1998.
- [2] D. E. Knuth. *The Art of Computer Programming*, volume 3: Sorting and Searching. Addison-Wesley, Reading, MA, 2nd edition, 1996.
- [3] R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, 1975.