CSE341T/CSE549T **11/12/2014**

Last of Treaps and Max Flow

Lecture 21

Analyzing the Cost of Treap Operations

Recall last time that we saw how to implement UNION. We also saw that SPLITand JOINcan be implemented in $O(log|T|)$ time, where T is the input argument. Today, we will discuss the cost of implementing UNIONand INTERSECTION.

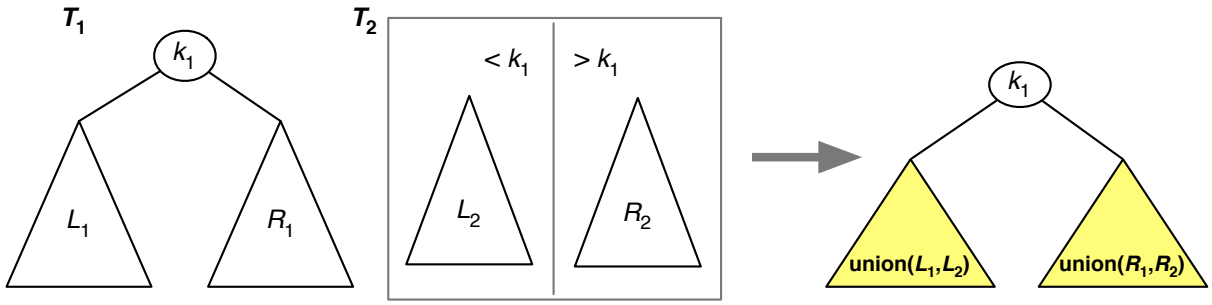
UNION(T_1, T_2)

1 **if** T_1 is empty,
2 **then return** T_2
3 $(L_1, R_1, (k_1, v_1)) \leftarrow$ EXPOSE(T_1)
4 $(L_2, v_2, R_2) \leftarrow$ SPLIT(T_2, k_1)
5 $L \leftarrow$ **spawn** UNION(L_1, L_2)
6 $R \leftarrow$ UNION(R_1, R_2)
7 **sync**
8 **return** JOIN$(L, (k_1, v_1), R)$

Let's recall the basic structure of UNION(T_1, T_2).

- For T_1 with key k_1 and children L_1 and R_1 at the root, use k_1 to split T_2 into L_2 and R_2.
- Recursively find $L_u =$ UNION(L_1, L_2) and $R_u =$ UNION(R_1, R_2).
- Now JOIN(L_u, k_1, R_u).

Pictorially, the process looks like this:



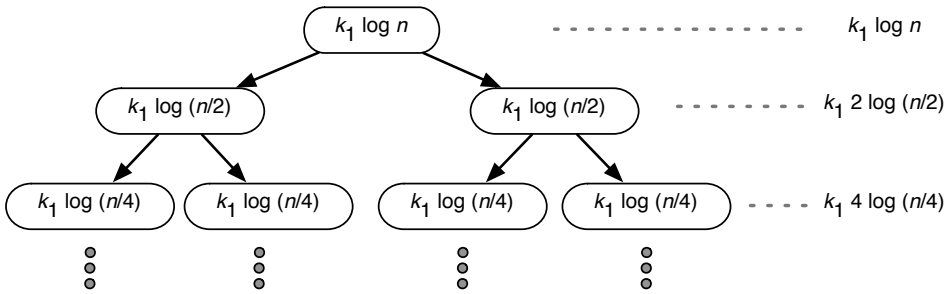
Note that each call to UNION makes one call to SPLIT and one to JOIN each which take $O(\log n)$ work, where n is the size of T_2. We assume that T_1 is the smaller tree with size m. For staters, we'll make the following assumptions to ease the analysis:

1. let's assume that T_1 it is perfectly balanced, and

2. each time a key from T_1 splits T_2, it splits it exactly in half.

The work recurrence is then $W(m, n) = 2W(m/2, n/2) + \Theta(\log n)$. In addition, if $m = 1$, then $W(1, n) = O(\lg(1 + n))$.

This recurrence deserves more explanation: When $m > 1$, expose gives us a perfect split, resulting in a key k_1 and two subtrees of size $n/2$ each, and by our assumption (which well soon eliminate), k_1 splits T_2 perfectly in half, so the subtrees that produces have size $m/2$. When $m = 1$, we know that expose give us two empty subtrees L_1 and R_1 , which means that both UNION(L_1, L_2) and UNION(R_1, R_2) will return immediately with values L_2 and R_2, respectively. Joining these together with T_1 costs at most $O(\log(1 + n))$ time.



Is this tree root or leaf dominated, or evenly sized? And how many levels will it have? It is not hard to see that this tree is dominated at the leaves. In fact, what's happening is that when we get to the bottom level, each leaf in T_1 has to split a subtree of T_2 of size n/m. This takes $O(\log(1+\frac{n}{m}))$ work. Since there are $O(m)$ such leaves, the total work at the leaves is $O(m \log(1 + \frac{n}{m}))$. Furthermore, the work going up the tree decreases geometrically.

To bound this more formally, since T_1 has m keys and it is split exactly in half, it will have $\log_2 m$ levels, so if we start counting from level 0 at the root, we have that the i-th level has 2^i nodes. The bottom level will have m nodes. Size of T_2 at each leaf is $n/2^{\lg m}$. The whole bottom level will cost

$$k_1 \cdot m \cdot \log\left(1 + n/2^{\log m}\right) = k_1 \cdot m \cdot \log\left(1 + \frac{n}{m}\right).$$

Since the tree is dominated at the leaves, the total work will be $O(m \log(1 + \frac{n}{m}))$, as desired. Hence, if the trees satisfy our assumptions, we have that UNION runs in $O(m \log(1 + \frac{n}{m}))$.

Of course, in reality, our keys in T_1 won't split subtrees of T_2 in half every time. But it turns out this only helps. We won't go through the rigorous analysis, but let us think informally. Instead of splitting exactly in half, T_2 splits unevenly. So now, let us say that the subtrees that reach the leaves

have sizes $n_1, n_2, \ldots, n_k$, where $\sum_{i=1}^{k} n_i \leq n$ since the subtrees are a partition of the original tree T_2. Therefore, the total cost of splitting these subtrees is

$$\sum_{i=1}^{k} k_1 \log(n_i) \quad \leq \quad \sum_{i=1}^{k} k_1 \log(n/k),$$

where we use the inequality of arithmetic and geometric means (or, the AM-GM inequality).[1] This shows that the total work is $O(m \log(1 + \frac{n}{m}))$.

Now, what is the work at the internal nodes? At each level of the tree, subtree sizes of T_2 sum up to n and you can apply Jensen's inequality again to get the same result to bound the work at each level. Thus, the tree is leaf domincated, even if T_2 is not split evenly.

What about T_1? In actuality, T_1 doesn't have to be perfectly balanced as we assumed. For treaps, we know that the depth of T_1 tree is at most $c \lg m$ for some constant c. Therefore, the depth of the above recursion tree will also be $c \lg m$. Again, at every level, the total sum of the nodes from T_1 is also at most n. Therefore, the recursion tree is still leaf dominated, and we can use the same analysis as above.

What abou the span? Well, at each level of recurrence, the span is $\theta(\lg n)$ with $c \lg m$ levels, so the span is $O(\lg^2 n)$. Actually, the span can be improved to $O(\log n)$ by changing the the algorithm slightly (akin to how one gets the $O(\lg n)$ span for the MERGE routine in mergesort), but it's more complicated, so we won't get into it.

Note that both the work and span are high-probability bounds. The cost of $f(n)$ in the recurrence above is dominated by the cost of split, which is $c \lg n$ *with high probability*. Thus, in order to get the bound on work and span, we need to argue that at each level of recurrence, the bound indicated by $f(n)$ holds with high probability. To get the probability of getting the final bounds on work and span, we take a union bound across all levels of recurrence.

In summary, this means that UNION can be implemented in $O(m \log(n/m))$ work and span $O(\log n)$. The same holds for the other similar operations (e.g. INTERSECTION).

Max Flow

In this lecture, we discussed serial algorithm for computing max flow, which can be found in the notes for lecture 22.

[1] Technically, we are using the fact that the logarithm function is concave and applying a variant of the so-called "Jensen's inequality."