15-492: Parallel Algorithms | **Lecturer:** Guy Blelloch
Topic: Trees | **Date:** September 20th
Scribe: Matthew Danish

8.1 Leaffix and Rootfix

Leaffix takes its name from a play on the word "prefix," as in the **PrefixSum** operation covered previously. **Leaffix** scans a tree from bottom to top and writes the sum of the children into every parent, up to the root. An example of **Leaffix** is given in figure 8.1.1.

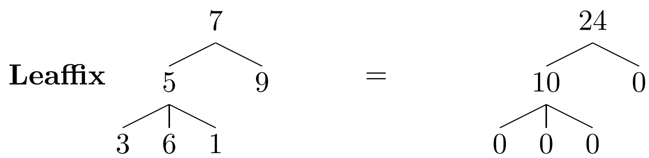

Figure 8.1.1: **Leaffix** example

Rootfix is a corresponding operation but operating in the opposite direction: from top to bottom. An example of **Rootfix** is given in figure 8.1.2.

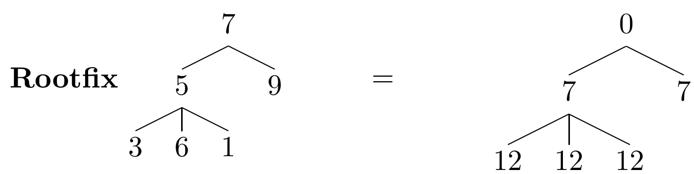

Figure 8.1.2: **Rootfix** example

Leaffix and **Rootfix** embody some basic concepts about trees: namely, size and depth. If you assign the value 1 to every node, then the **Leaffix** operation will find the size of every node, while the **Rootfix** operation will find the depth of every node.

We would like to be able to perform these operations in $\mathcal{O}(\log n)$ depth and $\mathcal{O}(n)$ work. There are two "degenerate" cases

Balanced binary tree The depth of the tree is $\log n$ so it should be clear that it is possible to perform the operations in $\mathcal{O}(\log n)$ depth.

Linked list A linked list is really a tree where every node has only one child. We know of a good $\mathcal{O}(\log n)$ depth algorithm for linked lists using symmetry-breaking.

The "interesting" case is a mixture of the two, potentially unbalanced or list-like trees. These are the subject of the following two sections.

8.2 Euler Tours

An Euler tour is a traversal of a tree such that every node is visited twice: once on the way in, and once on the way out. The entry to a node N is denoted ${}_\circ N$ and the exit is denoted $N_\bullet$. After entering a node, the tour proceeds to recursively visit every child in turn, left to right. An example is shown in figure 8.2.3.

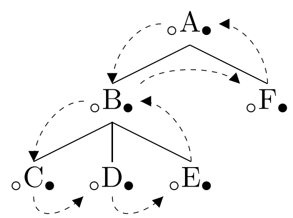

Figure 8.2.3: Euler tour example

An Euler tour can be stored in an array as shown by this example based on figure 8.2.3.

${}_\circ A$	${}_\circ B$	${}_\circ C$	$C_\bullet$	${}_\circ D$	$D_\bullet$	${}_\circ E$	$E_\bullet$	$B_\bullet$	${}_\circ F$	$F_\bullet$	$A_\bullet$

8.2.1 Leaffix with Euler tours

The algorithm for **Leaffix** operates as follows:

1. For every node N, write the value of N into ${}_\circ N$.
2. Run the **PrefixSum** operation on the array.
3. For every node N, the result is $N_\bullet - {}_\circ N - N$.

The following tables demonstrates the algorithm on the example from figure 8.1.1. First, write the values into the entry nodes:

${}_\circ A$	${}_\circ B$	${}_\circ C$	$C_\bullet$	${}_\circ D$	$D_\bullet$	${}_\circ E$	$E_\bullet$	$B_\bullet$	${}_\circ F$	$F_\bullet$	$A_\bullet$
7	5	3		6		1			9		

Then, run **PrefixSum**:

${}_\circ A$	${}_\circ B$	${}_\circ C$	$C_\bullet$	${}_\circ D$	$D_\bullet$	${}_\circ E$	$E_\bullet$	$B_\bullet$	${}_\circ F$	$F_\bullet$	$A_\bullet$
0	7	12	15	15	21	21	22	22	22	31	31

from this table, you can see that the result for A is 24, the result for B is 10, and the rest are 0.

8.2.2 Rootfix with Euler tours

The algorithm for **Rootfix** using an Euler tour array is

1. For every node N, write the value of N into ${}_\circ N$ and the negation of that value into $N_\bullet$.
2. Run the **PrefixSum** operation.
3. For every node N, read the result from ${}_\circ N$.

Again, with our example, we start off with this table:

${}_\circ A$	${}_\circ B$	${}_\circ C$	$C_\bullet$	${}_\circ D$	$D_\bullet$	${}_\circ E$	$E_\bullet$	$B_\bullet$	${}_\circ F$	$F_\bullet$	$A_\bullet$
7	5	3	-3	6	-6	1	-1	-5	9	-9	-7

Then, run **PrefixSum**:

${}_\circ A$	${}_\circ B$	${}_\circ C$	$C_\bullet$	${}_\circ D$	$D_\bullet$	${}_\circ E$	$E_\bullet$	$B_\bullet$	${}_\circ F$	$F_\bullet$	$A_\bullet$
0	7	12	15	12	18	12	13	12	7	16	7

and you can compare the results to figure 8.1.2 to see that they are the same.

8.2.3 Representation in Nesl

A common NESL representation is a triple of lists:

- The set of nodes in *preorder*. Ex: $[A, B, C, D, E, F]$.
- The positions of ${}_\circ N$ for every N. Ex: $[0, 1, 2, 4, 6, 9]$.
- The positions of $N_\bullet$ for every N. Ex: $[3, 5, 7, 8, 10, 11]$.

It should be easy to see how to construct the Euler tour array, of size $2n$, from this representation quickly; where n is the size of the set of nodes.

8.2.4 Summary on Euler tours

Euler tours give us a simple way to use our **PrefixSum** operation on arbitrary trees. The algorithms given have been exclusively concerned with *sums*. In fact, this technique can be used on any set of values and binary operation which form a *group*: the binary operation must be associative, every value must have an inverse, and there exists an identity value.

8.3 Rake/Compress

The *group* requirement can be quite onerous: consider the example of multiplication when the set of values includes 0. Other commonly desired operations are max and min which do not have inverses. We would like an algorithm which works on all trees, and all *monoids*, in expected $\mathcal{O}(n)$ work and $\mathcal{O}(\log n)$ depth.

Rake/Compress is a tree-contraction based algorithm which involves the interleaving of two contraction operations. We will assume, for the moment, that all nodes have less than three children.

Rake The rake is an operation which contracts the children of a node into the node, using the binary operation to combine them. It applies when the node has more than one child.

Compress A node is considered a *chain* node if it has only one child. Compress shortens a chain by using the random-mate technique drawn from symmetry-breaking algorithms on linked lists.

In the "degenerate" cases listed above, the rake is the best tool for dealing with balanced binary trees, and compress is the best for linked lists. However, arbitrary trees must be attacked with a combination of these operations. In the next class, we will prove the following theorem:

Theorem 8.3.1 *After $\mathcal{O}(\log n)$ expected rake and compress steps, a tree is reduced to a single node.*

and in the process, the following claim:

Claim 8.3.2 *Rake combines all the children into the parent. Compress makes a new link with the combined values of two nodes. The correct values for the eliminated nodes can be recovered even after interleaving these steps.*