ELE2742 / ENG1467 - PUC-Rio, 2019.1

Lecture 8: Application of Scan

DATE

Lecturer: Kanat Tangwongsan *Scribe: Kriangsak Thuiprakhon*

1 Paren-Matching

Recall Paren-Matching from DS, how do we make it Parallel? Here is what we can do. let's take an example string of parenthesis:

$$(())()$$

we can assign a score to each character: say 1 to an open and -1 to a close parenthesis. we will now have an array:

$$\{1, 1, -1, -1, -1, 1\}$$

From here on, apply PlusScan to the collection, then the result array,A, will be:

$$A = \{\{0, 1, 2, 1, 0, 1\}, 0\}$$

Now we can do **map**($\lambda x : x \geq 0, A$) resulting in:

$$A' = \{T, T, T, T, T, T\}$$

Finally, we can do **reduce**($\wedge, A'$) which will give the final Boolean **T**, meaning the string has the perfect parenthesis matching.

2 Left Copy

Let's say there is a group of friends at the restaurant trying to order some food but not all of them actually know what to order. So, some of them will take what ever the guy on the left does. That is:

$$O = \{a, None, b, None, None, d\}$$

To solve this problem, one can apple scan on the array O.

$$x * y = \begin{cases} x & \text{if } y \text{ is } None \\ y & \text{otherwise} \end{cases}$$

We can apply

$$scan(*, None, O)$$

3 Fibonacci sequence

Definition 3.1. the Fibonacci numbers, commonly denoted F_n, form a sequence, called the Fibonacci sequence, such that each number is the sum of the two preceding ones, starting from 0 and 1. That is,

$$F_0 = 0$$

$$F_1 = 1$$

and

$$F_n = F_{n-1} + F_{n-2}$$

for n > 1.

Now, let's look at this sequence in a linear algebra point of view. That is,

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_{n-1} \\ F_{n-2} \end{pmatrix} = \begin{pmatrix} F_{n-1} + F_{n-2} \\ F_{n-1} \end{pmatrix} = \begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix}$$

Notice that F_n can be rewritten as a some linear combination of F_{n-1} and F_{n-1}. That is,

$$F_n = aF_{n-1} + bF_{n-2}$$

Now, if we want the n-th number of Fibonacci sequence, we can do:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \cdots \underbrace{\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}}_{A} \begin{pmatrix} F_{n-1} \\ F_{n-2} \end{pmatrix}$$

now what we want looks somewhat like:

$$\{A^0, A^1, A^2, ..., \}$$

we can then apply scan as follows:

$$scan(\text{matrix multiply}, I, [A, A, A, ...A])$$

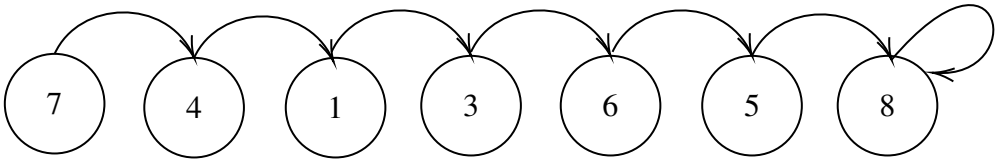
4 List Ranking

Let's say we have a linkedList and we want to give them some ranks (find the index of an element in Java Array). If this were to be a Java ArrayList, we could definitely call INDEXOF that return the index of element with linear time complexity. How about trying to solve it in parallel? how would one go about parallelizing this problem. In making it parallel, one can adopt SCAN function employing the technique of contraction. However, by contracting the list naively, there might be the case when a node get contracted twice with its (two) adjacent nodes. To prevent this from happening, we will go back to our old good friend, coin-flipping technique. that is to say, we will

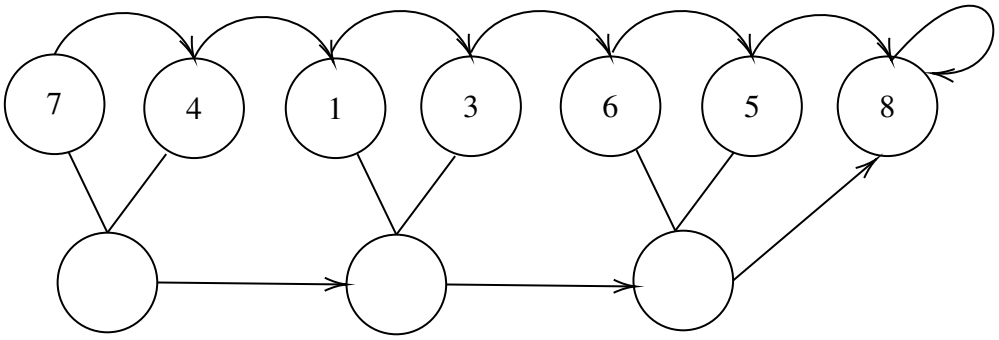
flip a coin for every node in the given LinkedList and will do contraction of two nodes if and only if the flips of the two nodes turn out to be exactly $H \to T$, this will be discussed in more detail when we touch upon TREECONTRACTION in the next lecture. Once we have tossed a coin for every node in our list, we can then proceed with node contractions. In general, we can represent a LinkedList in an array as follows:

2	3	7	6	1	8	5	4	8
0	1	2	3	4	5	6	7	8

Now, we can translate the array representation to the actual LinkedList:



Then contract the list:



5 Quick Sort

Quick sort is naturally very parallel as $Filter$ is very parallel

6 Merge Sort

Is Merge Sort parallel? Say if we halve an array

$$\{\underbrace{\ldots}_{n/2}\ \underbrace{\ldots}_{n/2}\}$$

then, the span of our mergeSort will be:

$$S(n) = S(n/2) + S_{merge}$$

The question is, what is the cost of merging two sotred arrays?

Claim 6.1. *given 2 sorted sequence A and B, there is a function, $k - th(k, A, B)$, that returns $l_A, l_B, l_A + l_B = k$ such that $A[: l_A]$ and $B[: l_B]$ are the smallest k elements of A+B done in $O(|A| + |B|)$ work, $O(log^2(|A| + |B|))$span*

This can be proven using Dual Binary Search merge(A,B)

---

if len(A) =0: return B
if len(B) =0: return A
m=(len(A)+len(B))/2
la,lb = kth(m, A,B)
A',B' = merge(A[:la],B[:lb]) || merge(A[la:],B[lb]:)
return A' +B'

Work and Span

$$W(n) = W(n/2) + W(n - n/2) + O(logn)$$

which solves to $O(n)$ where n = |A| + |B|

$$S(n) = S(n/2) + O(logn)$$

which solves to $O(log^2 n)$

Now that, $S_{merge} = O(log^2 n)$, we can go back to merge sort span analysis and complete it:

$$S_{mergeSort}(n) = S(n/2) + O(log^2 n)$$

which solves to $O(log^3 n)$