Tutorial 10 Exercise Solutions

COMP526: Efficient Algorithms

09–10 December, 2024

Exercise 1. In our lectures on parallel algorithms, we saw a PRAM algorithm that solves string matching for searching for a pattern $P[0..m)$ in a text $T[0..n)$ with span $\Theta(m)$ and work $\Theta(n)$. The output of this algorithm, however, was different from the original setting of pattern matching we discussed earlier in the semester. In particular, the output of a parallel algorithm was an array $M[0..n)$ such that $M[i] = 1$ if T contains a match to P at index i and $M[i] = 0$ otherwise.

(a) Devise a PRAM algorithm that modifies the array M such that after applying your algorithm, $M[n-1]$ stores the total number of matches of P in T. The span of your procedure should be $O(\log n)$ and its work should be $O(n)$.
(Hint: try a divide and conquer approach.)

(b) Explain how your procedure from part (a) can be modified (or extended) to produce the index of the first instance of P in T (assuming there is a match). The span and work of the updated procedure should be (asymptotically) no worse than your first procedure.

For simplicity, you may assume that n, the length of the text, is a power of 2, say $n = 2^k$.

Solution. For the first part, first observe that the total number of occurrences of P in T is the sum of the entries in M. This is because match of P in T corresponds to exactly one 1-entry in M. Therefore, our goal for the first part is to update M such that $M[n-1]$ is the sum of the original values in M.

Following the suggestion to use the divide and conquer approach, a natural way of dividing the array M would be to split it in half by index. We can sum the values in $M[0..n/2)$ and $M[n/2..n)$ independently of each other, then add the two sums to get the total number of 1s in M. A recursive implementation of this approach would give the following procedure:

```
procedure SUM($M[\ell..r)$)▷ Sum the elements of $M$ from indices $\ell$ to $r-1$ and store the result at $M[r-1]$
    if $r = \ell + 1$ return
    $m \leftarrow (\ell + r)/2$                         ▷ The midpoint of the interval
    SUM($M[\ell, m)$)                 ▷ Sum the left half and store sum in $M[m-1]$
    SUM($M[m, r)]$)                   ▷ Sum the right half and store sum in $M[r-1]$
    $M[r-1] = M[m-1] + M[r-1]$            ▷ Store the sum of sums in $M[r-1]$
end procedure
```

Note that the depth of recursion for this solution is $\Theta(\log n)$. Moreover, the recursive calls can be performed in parallel, as they are independent of each other.

In order to analyze the work and span of a parallelized variant of the Sum procedure, it is instructive to write a non-recursive version of the same procedure that performs the same operations of Sum. To this end, consider the operations performed by Sum at depth $k-1$ (where $n = 2^k$). In this case the two recursive calls to Sum don't do anything, so only Line 6 has any effect. Specifically, after all calls at depth $k-1$ are completed, the effect is that

$$\begin{gathered} M[1] \leftarrow M[0]+M[1] \\ M[3] \leftarrow M[2]+M[3] \\ M[5] \leftarrow M[4]+M[5] \\ \vdots \\ M[n-1] \leftarrow M[n-2]+M[n-1] \end{gathered}$$

Similarly, at depth $k-2$, the values are updated as follows:

$$\begin{gathered} M[3] \leftarrow M[1]+M[3] \\ M[7] \leftarrow M[5]+M[7] \\ M[11] \leftarrow M[9]+M[11] \\ \vdots \\ M[n-1] \leftarrow M[n-3]+M[n-1] \end{gathered}$$

More generally, at depth $k-d$, each index i that is one less than a multiple of 2^d is updated to the sum $M[i]+M[i-2^{d-1}]$. After this operation, $M[i]$ stores the sum the original entries of $M[i-2^d+1..i]$.

Unrolling the recursive computations in this way, we obtain the following parallel procedure:

```
1: procedure ParallelSum($M[0..2^k)$, $n = 2^k$)
2:     for $d = 1, 2, \ldots, k$ do
3:         $w \leftarrow 2^d$                    ▷ the width of the subinterval being summed
4:         for $i = w-1, 2w-1, \ldots n-1$ in parallel do
5:             $M[i] \leftarrow M[i] + M[i - w/2]$
6:         end for
7:     end for
8: end procedure
```

To analyze the span of the procedure, observe that the inner loop (lines 4–6) has span $\Theta(1)$ because all operations are performed in parallel. The iterations of the outer loop (lines 2–7) are performed sequentially, but there are only $\log n$ iterations performed, each with span $O(1)$ Thus, the overall span is $\Theta(\log n)$. For the work, note that for $w = 2^d$, there are $n/2^d$ iterations of the inner loop, and iteration does $\Theta(1)$ work. Summing over the iterations of the outer loop, we find the number of iterations performed is

$$\frac{n}{2}+\frac{n}{4}+\cdots+1=\sum_{j=1}^{k}\frac{n}{2^j}<n\sum_{j=1}^{\infty}\frac{1}{2^j}=n.$$

Thus, the total work is $\Theta(n)$.

To modify the procedure to find first index where P matches T, note that we are searching for the first index i for which $M[i] > 0$. We can use the array M produced by running PARALLELSUM. Specifically, after running PARALLELSUM, for any odd positive integer c, $M[c2^d - 1]$ stores the number matches between indices $(c-1)2^d$ and $c2^d - 1$. To find the smallest index i with $M[i] > 0$, we can perform binary search, starting with $j = n - 1 = 2^k - 1$. An iterative version of binary search is implemented with the following pseudocode:

```
procedure FirstMatch($M[0..n]$, $n = 2^k$)
    $j \leftarrow n - 1$
    for $w = 2^{k-1}, 2^{k-2}, \ldots, 1$ do
        if $M[j - w] > 0$ then
            $j \leftarrow j - w$
        end if
    end for
end procedure
```

This procedure runs in $\Theta(k) = \Theta(\log n)$ sequential steps from a single processors. Thus, performing this after running PARALLELSUM has an overall span of $\Theta(\log n)$ and an additional $\Theta(\log n)$ work. □

Exercise 2. Consider the text $T =$ `abbabbaa$`. What is n here? (Exactly follow the convention from the lecture!) Construct/draw the

(a) standard (not compacted) trie of all suffixes of T,

(b) suffix tree of T (human version) with string labels on edges and leaves,

(c) suffix tree of T (computer version) as it is stored, i.e., offsets in nodes, starting index in leaves, first characters on edges.

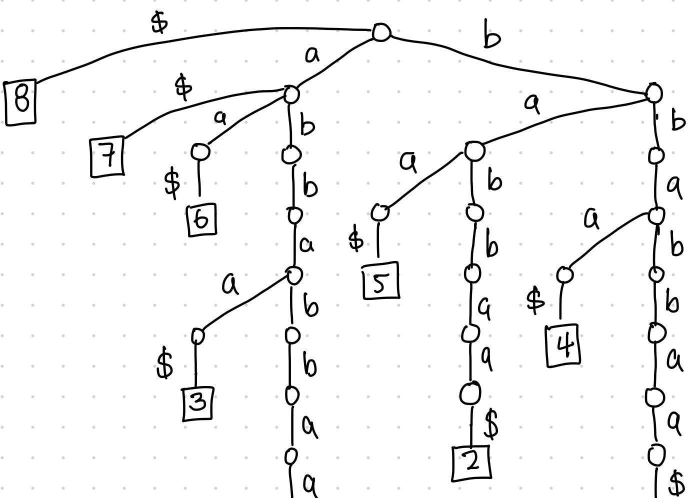
Solution. The value of n is 8. The trees are drawn on the following page. □

T = abbabbaa$

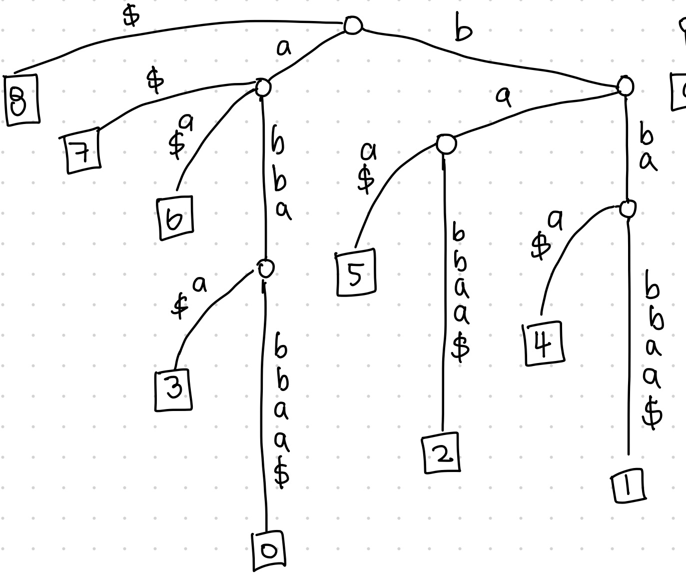
Prefixes :

0	1	2	3	4	5	6	7	8
a	b	b	a	b	b	a	a	$
	b	b	a	b	b	a	a	$
		b	a	b	b	a	a	$
			a	b	b	a	a	$
				b	b	a	a	$
					b	a	a	$
						a	a	$
							a	$
								$

Trie (with unlabeled leaves)



Human Readible Suffix Tree



Actual Suffix Tree