# CS60026 Parallel and Distributed Algorithms, Autumn 2018–2019

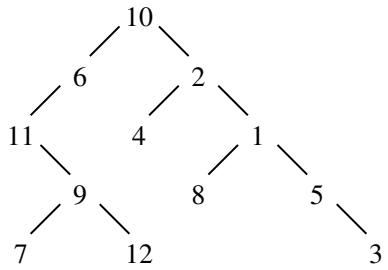
## Class Test

05–November–2018 | 06:15–07:15 pm | Maximum marks: 20

Roll No: _____ Name: _____

[ *Write your answers in the question paper itself. Be brief and precise. Answer all questions.*
*If you use any algorithm/result/formula covered in the class, just mention it, do not elaborate.* ]

The nodes of a binary tree $T$ are numbered $1,2,3,\ldots,n$. Assume that $n \geqslant 2$. The tree is presented to you by two arrays $L[\,]$ and $R[\,]$, where for each $v \in \{1,2,3,\ldots,n\}$, the array element $L[v]$ (resp. $R[v]$) denotes the left (resp. right) child of $v$. The following figure gives an example for $n = 12$.




(a) A binary tree

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $L$ | 8 | 4 | 0 | 0 | 0 | 11 | 0 | 0 | 7 | 6 | 0 | 0 |
| $R$ | 5 | 1 | 0 | 0 | 3 | 0 | 0 | 0 | 12 | 2 | 9 | 0 |

(b) Its child–pointer representation

Given $n$, $L[\,]$, and $R[\,]$ only, develop CREW/EREW PRAM algorithms for the following problems. In this question paper, you may assume that generalized prefix sums on a single (not multiple) linked lists can be computed in logarithmic time using linear work.

**1.** Prepare the parent array $P[\,]$ of size $n$ such that for each $v \in \{1,2,3,\ldots,n\}$, the element $P[v]$ stores the parent of $v$ in $T$ (or 0 is $v$ is the root of $T$). Your algorithm should take $O(1)$ time and do $O(n)$ work. **(5)**

*Solution* 1. For $v = 1,2,3,\ldots,n$, pardo:
Initialize $P[v] = 0$.

2. For $v = 1,2,3,\ldots,n$, pardo:
(a) If $L[v] \neq 0$, set $P[L[v]] = v$.
(b) If $R[v] \neq 0$, set $P[R[v]] = v$.

**2.** An Euler tour in $T$ replaces each undirected edge $(u,v)$ of $T$ by two directed edges $(u,v)$ and $(v,u)$, and specifies a successor of each directed edge on the tour. Propose an $O(1)$-time $O(n)$-work algorithm for generating an Euler tour on $T$ presented to you by $n$, $L[\,]$ and $R[\,]$ only. If needed, you may additionally use the parent array $P[\,]$ computed in Part 1. **(5)**

*Solution* Let us order the neighbors of a node $v$ in $T$ as follows: $L[v], R[v], P[v]$. Some of these neighbors may be non-existing. Using this ordering, a linked-list-based representation of $T$, suitable for computing the Eulerian tour on $T$, can be prepared in $O(1)$ time using $O(n)$ work. However, these linked lists need not be explicitly computed. We can keep the ordering in our head while designing the following algorithm. Let $(u,v)$ be a directed edge in $T$. We compute its successor of $S(u,v)$ in the Eulerian tour as follows.

**Case 1:** $v$ is a child of $u$ (left or right).

If $v$ has a left child $w$, set $S(u,v) = (v,w)$, else if $v$ has a right child $w$, set $S(u,v) = (v,w)$, else set $S(u,v) = (v,u)$.

**Case 2:** $u$ is the left child of $v$.

If $v$ has a right child $w$, set $S(u,v) = (v,w)$, else if $v$ has a parent $w$, set $S(u,v) = (v,w)$, else ($v$ is the root having only the left child), set $S(u,v) = (v,u)$.

**Case 3:** $u$ is the right child of $v$.

If $v$ has a parent $w$, set $S(u,v) = (v,w)$, else ($v$ is the root) if $v$ has a left child $w$, set $S(u,v) = (v,w)$, else ($v$ is the root having only the right child), set $S(u,v) = (v,u)$.

In each case, the successor of each edge can be computed in $O(1)$ time from the arrays $L[\,], R[\,], P[\,]$. Moreover, there are $2(n-1)$ directed edges in $T$, so the total work done is $\Theta(n)$.

**3.** Propose an $O(\log n)$-time $O(n)$-work algorithm to compute the height of $T$. **(5)**

*Solution* The root of $T$ is the unique node $r$ with $P[r] = 0$, and can be identified in $O(1)$ time using $O(n)$ work by making a parallel search in the parent array $P[\,]$. If $r$ has a right child, let $w$ be this child, else let $w$ be the left child of $r$. Convert the Euler tour to a linked list by setting the successor of $(w,r)$ to NULL. In this and the next parts, we assume that the Euler tour is converted to a linked list in this manner.

*Solution* [continued] In the class, we have seen how to compute the levels of the nodes in a tree. This consists of setting an array $U$ indexed by the directed edges of $T$ as $U[P[v],v] = 1$ and $U[v,P[v]] = -1$ for every non-root node $v$ of $T$, followed by computing the generalized prefix sum of $U[\,]$ in an array $V[\,]$ with respect to the Eulerian linked list. This can be done in $O(\log n)$ time using $O(n)$ work.

The height of $T$ is the maximum level of a node. The maximum of $V[\,]$ can be computed by the balanced-binary-tree method in $O(\log n)$ time using $O(n)$ work.

**4.** Propose an $O(\log n)$-time $O(n)$-work algorithm to compute the inorder numbering of each vertex of $T$. You should prepare an array $IN[1\ldots n]$ such that $IN[v]$ is the one-based inorder number of $v$. In the example on the first page, the inorder numbers $1,2,3,\ldots,12$ go to the vertices $11,7,9,12,6,10,4,2,8,1,5,3$, respectively. **(5)**

*Solution* Let $v$ be a node in $T$. The relevant edge $\varepsilon(v)$ of $v$ is defined as follows. If $v$ has a left child $w$, set $\varepsilon(v) = (w,v)$, else if $v$ has a parent $u$ (that is, $v$ is not the root), take $\varepsilon(v) = (u,v)$. If the root $r$ has only the right child, then $\varepsilon(r)$ is not defined. We will handle this case separately.

Prepare an array indexed by the directed edges $e$ of $T$ as follows. If $e = \varepsilon(v)$ for some vertex $v$, set $U[e] = 1$, else set $U[e] = 0$. Compute the generalized prefix sum of $U[\,]$ with respect to the Eulerian linked list, in an array $V[\,]$. If $\varepsilon(r)$ is defined, then $IN[v] = V[\varepsilon(v)]$ for all vertices $v$ of $T$. If $\varepsilon(r)$ is not defined, then $IN[v] = 1 + V[\varepsilon(v)]$ for all non-root vertices $v$ of $T$, and $IN[r] = 1$.

**Remark:** The special case of $\varepsilon(r)$ being not defined can be eliminated as follows. Add a new root $r'$ to $T$, and make the old root $r$ the left child of $r'$. Call the resulting augmented tree $T'$. The $L,R,P$ arrays and the Eulerian linked list for $T$ can be converted to those for $T'$ in constant time (sequentially, but tell how). Now, since $r$ has a parent, all cases ($r$ has a left child or not) can be treated uniformly.