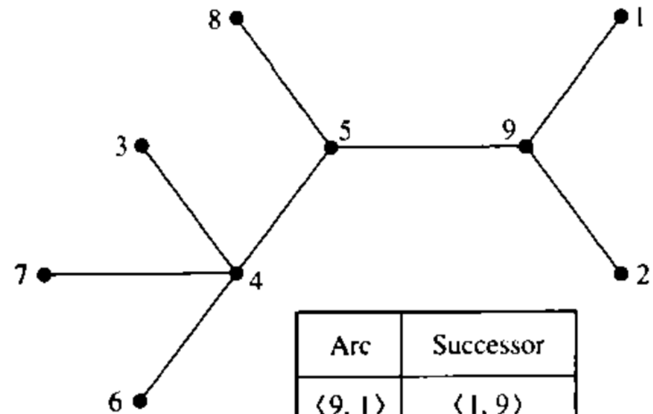
PRAM 3
Tree algorithms

CS121 Parallel Computing

Spring 2018

Euler tours

- An Euler tour of a graph is a cycle that goes through every edge of the graph.
  - It may go through a vertex multiple times.
- A connected, directed graph has an Euler tour if and only if the indegree and outdegree of every vertex are equal.
- Suppose we take an undirected graph, and for edge (u,v), create two directed edges (u,v) and (v,u).
  - Then every vertex has equal indegree and outdegree, and so has an Euler tour.
- Consider a tree where each edge has been doubled.
  - To find an Euler tour of the tree, first order the edges adjacent to each node arbitrarily.
  - Say the neighbors of a node v are ordered $u_0, \dots, u_{d-1}$. Then set the successor of edge (u_i, v) on the tour to $\left(v, u_{(i+1) \bmod d}\right)$.
- The Euler tour of a tree can be computed in O(1) parallel time.



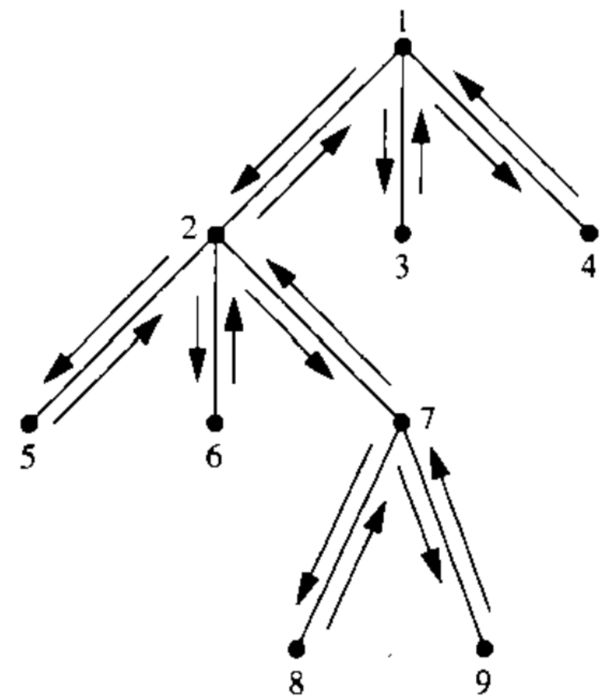
Arc	Successor
⟨9, 1⟩	⟨1, 9⟩
⟨9, 2⟩	⟨2, 9⟩
⟨4, 3⟩	⟨3, 4⟩
⟨5, 4⟩	⟨4, 3⟩
⟨3, 4⟩	⟨4, 7⟩
⟨7, 4⟩	⟨4, 6⟩
⟨6, 4⟩	⟨4, 5⟩
⟨8, 5⟩	⟨5, 4⟩
⟨4, 5⟩	⟨5, 9⟩
⟨9, 5⟩	⟨5, 8⟩
⟨4, 6⟩	⟨6, 4⟩
⟨4, 7⟩	⟨7, 4⟩
⟨5, 8⟩	⟨8, 5⟩
⟨5, 9⟩	⟨9, 2⟩
⟨2, 9⟩	⟨9, 1⟩
⟨1, 9⟩	⟨9, 5⟩

⟨9, 1⟩ → ⟨1, 9⟩ → ⟨9, 5⟩ → ⟨5, 8⟩ → ⟨8, 5⟩ → ⟨5, 4⟩ →
⟨4, 3⟩ → ⟨3, 4⟩ → ⟨4, 7⟩ → ⟨7, 4⟩ → ⟨4, 6⟩ → ⟨6, 4⟩ →
⟨4, 5⟩ → ⟨5, 9⟩ → ⟨9, 2⟩ → ⟨2, 9⟩ → ⟨9, 1⟩

Source: Introduction to Parallel Algorithms, Jaja

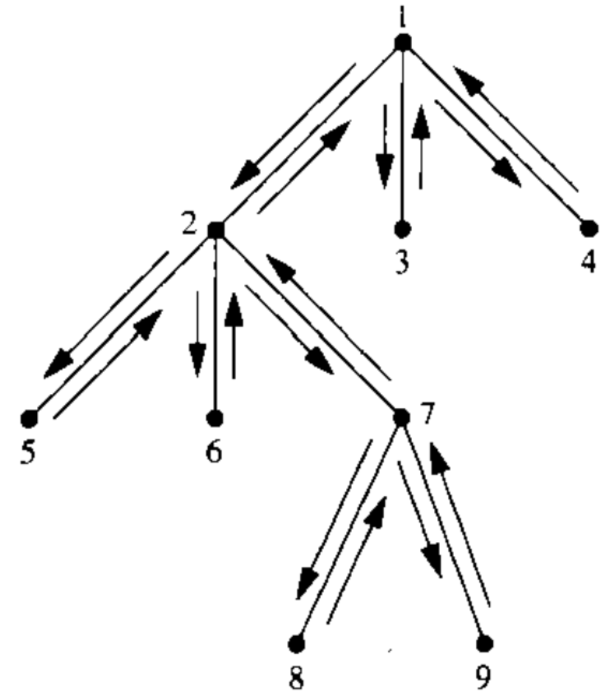
Parallel tree operations

- Many operations on trees can be done in parallel using Euler tours and prefix sum.
- These operations in turn are used in other parallel graph algorithms.
- We first root a tree in parallel.
  - I.e. set an arbitrary node r as the tree's root. Then each node v needs to compute p(v), its parent in the rooted tree.
  - To do this, assign a weight of 1 to each edge in an Euler tour of the tree.
  - Then compute the parallel prefix sum of the edges.
  - For each edge (u,v), set u=p(v) whenever the prefix sum of (u,v) is less than the prefix sum of (v,u).
  - Thus, we can root a tree with n nodes in O(log n) time and O(n) work.



Node depths

- For each node, compute its depth in a rooted tree.
  - For each node v, let p(v) be its parent.
  - Set the weight of edge (p(v), v) to 1, an the weight of edge (v, p(v)) to -1.
  - Compute a parallel prefix sum of the Euler tour starting at the root.
  - The depth of node v is the prefix sum of edge (p(v), v).
- For a tree with n nodes, this takes O(log n) time using O(n) work.

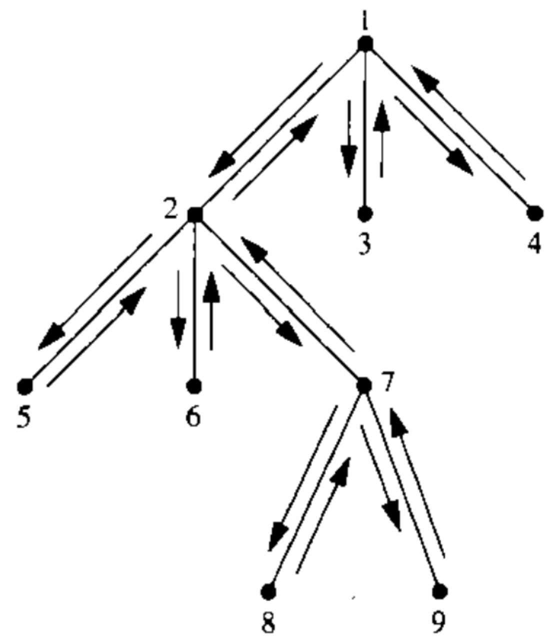


Postorder numbering

- Traverse a rooted tree in postorder, starting from the root r.
  - Start an Euler tour from r. For each node v, we want to visit v's children in the order of the tour, then visit v itself.
  - Ex

v	1	2	3	4	5	6	7	8	9
n(v)	9	6	7	8	1	2	5	3	4

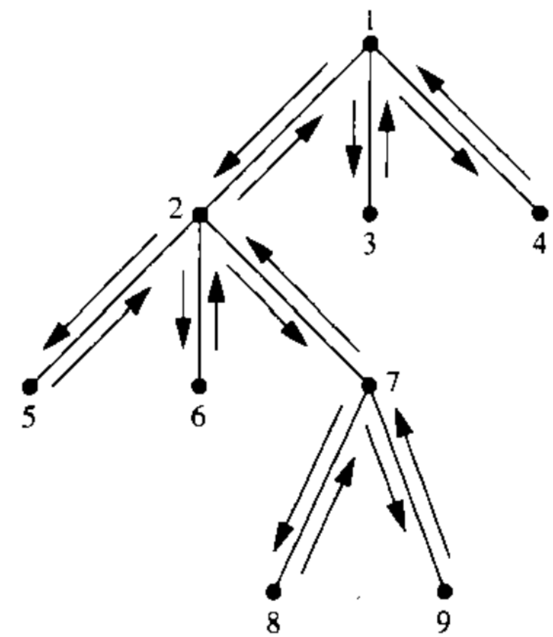
  - For each node v, set the weight of edge (v, p(v)) to 1, and the weight of (p(v), v) weight 0.
  - Compute a parallel prefix sum of the Euler tour.
  - For each $v \neq r$, set n(v) to the prefix sum of edge (v, p(v)). Set n(r)=n.
- In a tree with n nodes, we can compute the postorder numbering in O(log n) time and O(n) work.



Euler Path	Weight	Prefix Sums
⟨1, 2⟩	0	0
⟨2, 5⟩	0	0
⟨5, 2⟩	1	1
⟨2, 6⟩	0	1
⟨6, 2⟩	1	2
⟨2, 7⟩	0	2
⟨7, 8⟩	0	2
⟨8, 7⟩	1	3
⟨7, 9⟩	0	3
⟨9, 7⟩	1	4
⟨7, 2⟩	1	5
⟨2, 1⟩	1	6
⟨1, 3⟩	0	6
⟨3, 1⟩	1	7
⟨1, 4⟩	0	7
⟨4, 1⟩	1	8

Number of descendants

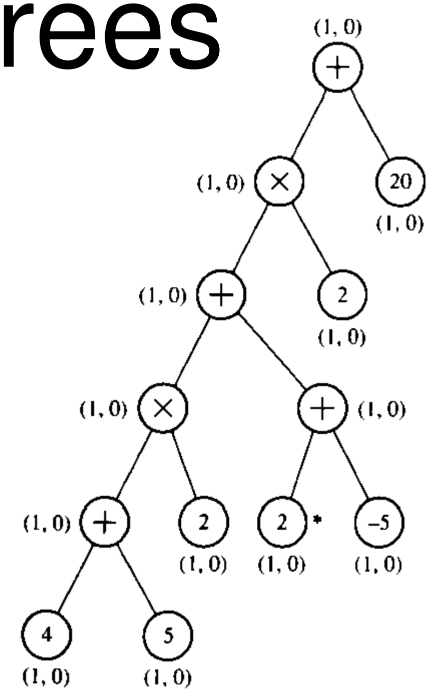
- For each node v in a rooted tree, compute the number of nodes in the subtree rooted at v.
- To do this, we compute prefix sums as in the postorder numbering.
- Then the number of descendants of a node v equals the prefix sum of (v, p(v)) minus the prefix sum of (p(v), v).
- In a tree with n nodes, we can compute the number of descendants in O(log n) time and O(n) work.

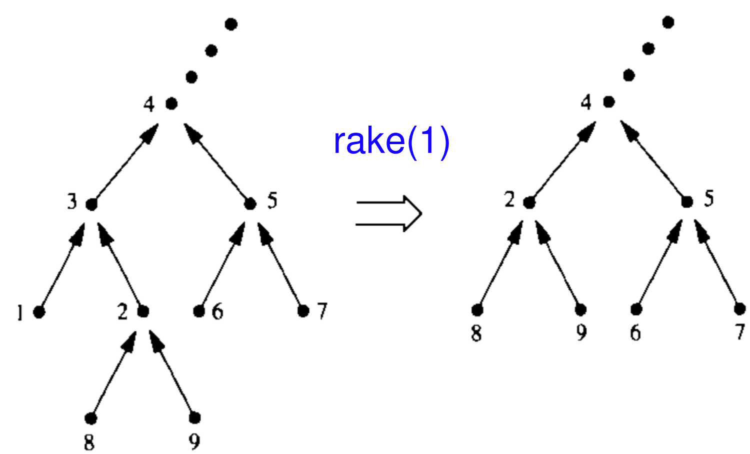


Euler Path	Weight	Prefix Sums
⟨1, 2⟩	0	0
⟨2, 5⟩	0	0
⟨5, 2⟩	1	1
⟨2, 6⟩	0	1
⟨6, 2⟩	1	2
⟨2, 7⟩	0	2
⟨7, 8⟩	0	2
⟨8, 7⟩	1	3
⟨7, 9⟩	0	3
⟨9, 7⟩	1	4
⟨7, 2⟩	1	5
⟨2, 1⟩	1	6
⟨1, 3⟩	0	6
⟨3, 1⟩	1	7
⟨1, 4⟩	0	7
⟨4, 1⟩	1	8

Evaluating expression trees

- An expression tree is a binary tree with values at the leaves and operators (+ or $\times$) in the interior nodes.
- We want to quickly evaluate an expression tree in parallel.
- The main tool is the rake operation.
  - Given a node u with sibling v, parent p and grandparent p', rake(u) removes u and p, and connects v with p'.
- We repeatedly rake an expression tree in parallel to contract it to 3 nodes with the same value as the original tree.



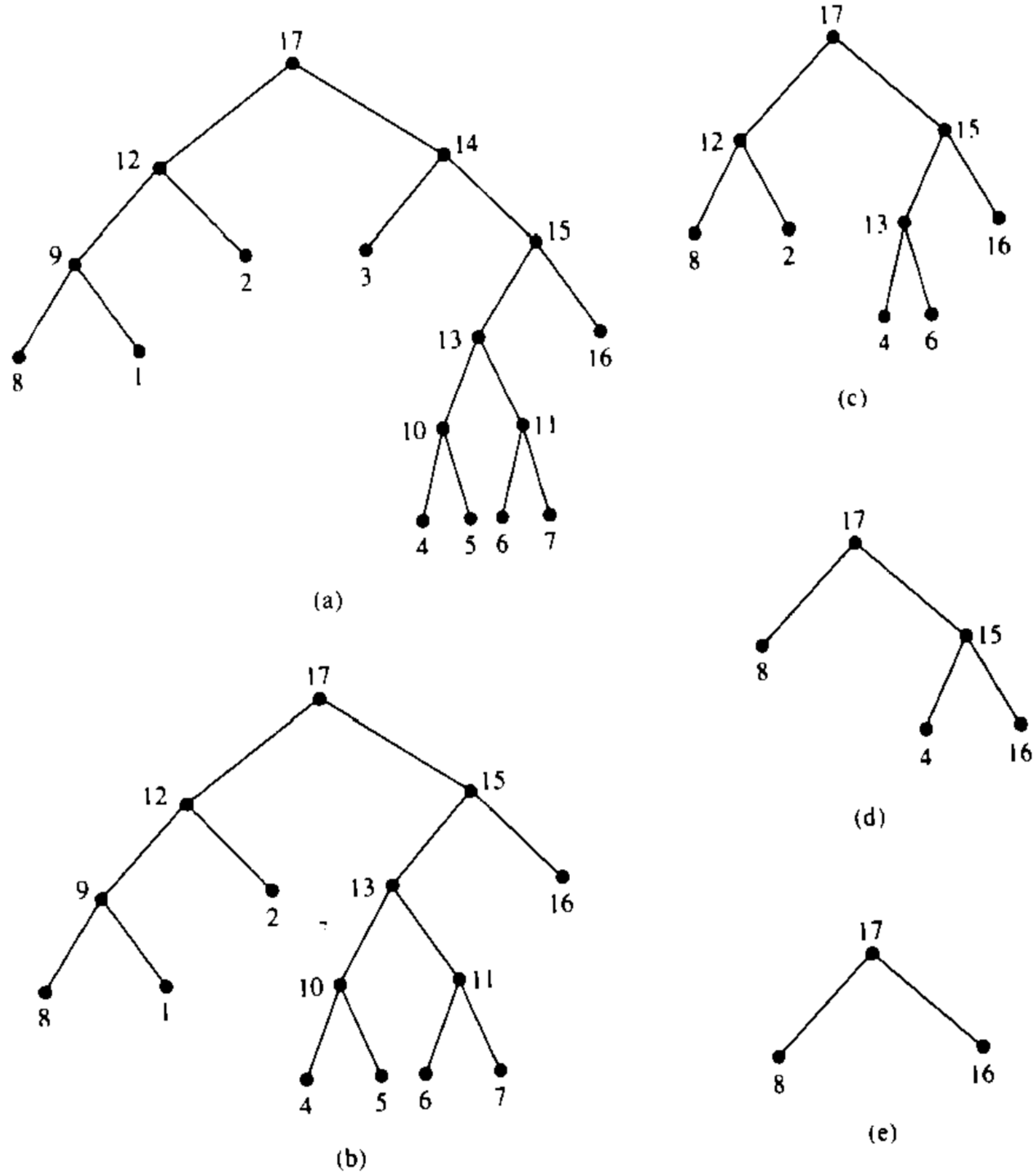


Raking in parallel

- When raking multiple nodes in parallel, we have to avoid concurrent changes to the same node.
- Given an expression tree, label the leaves from left to right (except for the first and last leaf), and call the set A.
  - Let A_{odd} and A_{even} be the subset of A with odd and even labels, resp.
- Repeat for $\lceil \log(n+1) \rceil$ rounds, where $n = |A|$.
  - Concurrently rake all the leaves in A_{odd} that are left children.
  - Concurrently rake the rest of the leaves in A_{odd}.
  - Set $A = A_{even}$.

Example

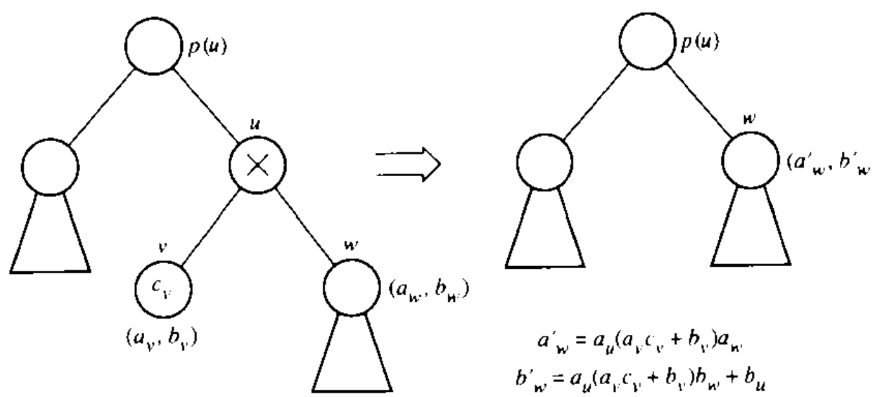
- First rake node 3.
- Then rake nodes 1, 5, 7.
- Then rake leaves 2, 6.
- Then rake leaf 4.



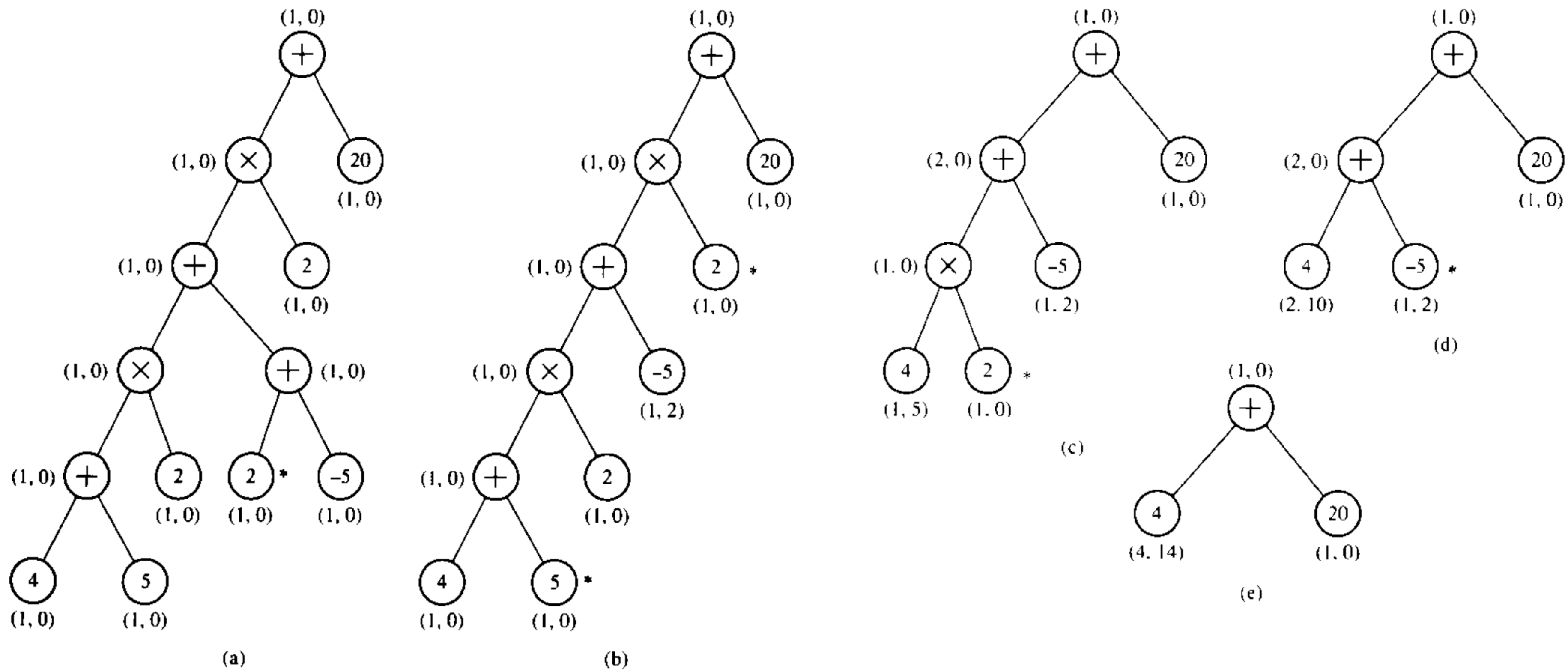
- Nodes raked concurrently don’t have the same parent. So the raking process works correctly.
- Each round reduces the number of leaves by a factor of 2.
- After $\lceil \log(n+1) \rceil$ rounds, there will be two leaves.

Parallel expression evaluation

- When raking an expression tree, we combine the raked node's value with its sibling's value.
- However, when we rake many nodes in parallel, some nodes may not have their values computed yet.
  - Imagine a chain graph with leaves hanging off the side. Nodes in the middle of the chain do not have values yet when they take part in a rake.
- Given a node v, we assign a label (a_v, b_v) to v.
  - If v is a leaf, v also has a numerical label x, and the numerical value of v is $a_v x + b_v$.
- When nodes are raked, we also update their labels.
  - Ex In the example below, suppose w has a still undermined value X. Then after raking v, w's value is $a_u(a_v c_v + b_v)(a_w X + b_w) + b_u = [a_u(a_v c_v + b_v)a_w]X + [a_u(a_v c_v + b_v)b_w + b_u]$.
    - Thus, the new label of w is $(a_u(a_v c_v + b_v)a_w, a_u(a_v c_v + b_v)b_w + b_u)$.
  - Leaves always have a numerical label, so their value can be immediately evaluated.
- Since raking in parallel reduces the tree to 3 nodes in $O(\log n)$ time, the tree's value can be evaluated in $O(\log n)$ time.



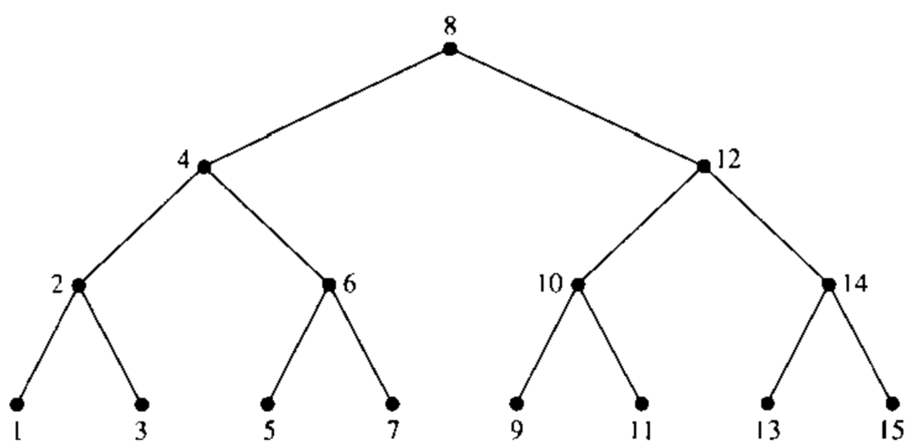
Example



Nodes marked by * are raked in parallel in each round.

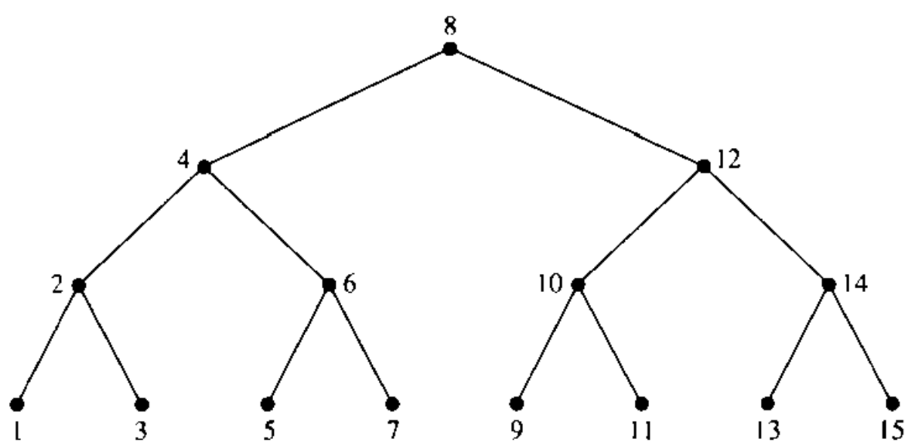
Lowest common ancestors

- Given two nodes u, v in a tree, LCA(u,v) is the lowest node in the tree with u and v as descendants.
  - Ex LCA(1,5) = 4, LCA(3,10) = 8.
- Given a tree, we want to preprocess it so that LCA queries can be answered in O(1) time.



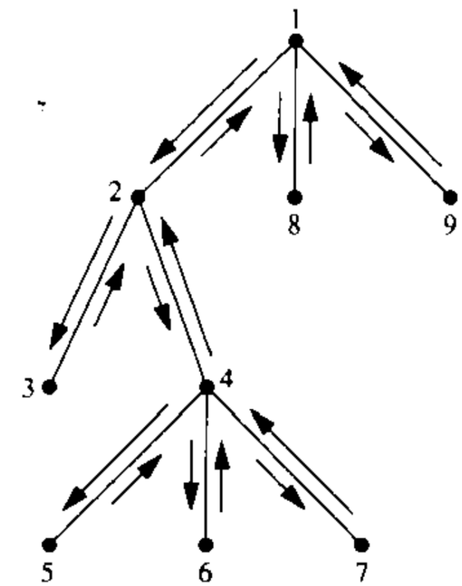
Simple cases

- For a path graph, the LCA of two nodes is whichever node is closer to the root, which can be computed in O(1) time.
- For a complete binary tree, suppose all nodes are labeled by their inorder number.
  - Given u and v, write their labels in binary. Suppose $u = (z_1 z_2 \ldots)_2$.
  - Let i be the most significant bit on which u and v differ.
  - Then $LCA(u, v) = (z_1 z_2 \ldots z_{i-1} 10 \ldots 0)_2$.
  - Ex 9 = $(1001)_2$ and 13=$(1101)_2$. The MSB on which they differ is 2. So LCA(9,13) = $(1100)_2$ = 12.
  - Thus, the LCA can be computed in O(1) time.



LCA and Euler tours

- Consider a general tree with n nodes.
- First compute an Euler tour of the tree, in O(1) parallel time.
  - Label nodes by their order of appearance in the tour.
- Next compute the depths of all nodes, in O(log n) parallel time.
- The Euler tour can be described by a size 2n-1 array A listing the order of nodes visited.
- Create a corresponding array B giving the levels (i.e. depths) of the nodes in A.

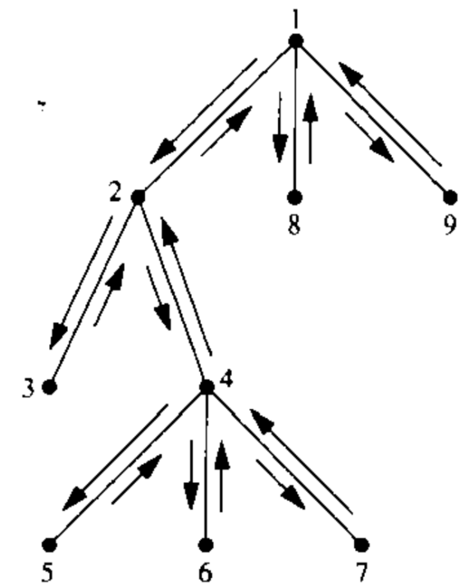


$A = (1, 2, 3, 2, 4, 5, 4, 6, 4, 7, 4, 2, 1, 8, 1, 9, 1)$

$B = (0, 1, 2, 1, 2, 3, 2, 3, 2, 3, 2, 1, 0, 1, 0, 1, 0)$

LCA and Euler tours

- Given a node v, define the following
  - level(v) is the level of v.
  - l(v), r(v) are the indices of the first, resp. last occurrence of v in A.
    - Ex l(4) = 4, r(4) = 10.
- Lem Given a node v, we have the following.
  - $l(v) = i$ iff $level(A[i-1]) = level(v) - 1$.
  - $r(v) = i$ iff $level(A[i+1]) = level(v) - 1$.
- Ex $l(4) = 4$, and $level(A[3]) = level(2) = 1 = level(4) - 1$.
- Ex $r(4) = 10$, and $level(A[11]) = level(2) = 1 = level(4) - 1$.
- Assume A and B are given. Then we can compute $l(v)$ and $r(v)$ for all nodes v in parallel O(1) time.
  - Use one processor for each node in A and check the conditions in the lemma.
  - If either condition holds for a node v, record l(v) or r(v).

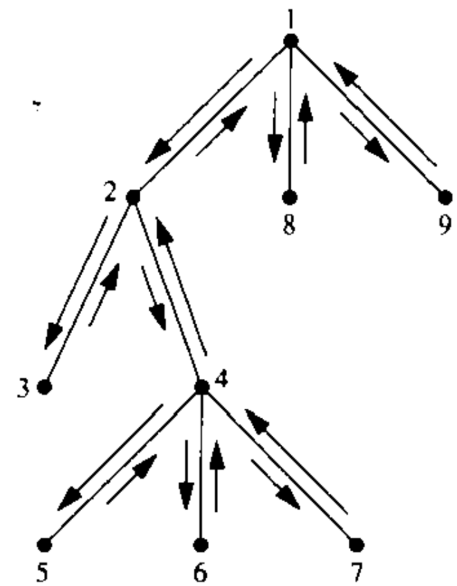


$A = (1, 2, 3, 2, 4, 5, 4, 6, 4, 7, 4, 2, 1, 8, 1, 9, 1)$

$B = (0, 1, 2, 1, 2, 3, 2, 3, 2, 3, 2, 1, 0, 1, 0, 1, 0)$

LCA and Euler tours

- Lem Given nodes u and v in the tree, the following properties hold
  - u is an ancestor of v if and only if $l(u) < l(v) < r(u)$.
  - If $r(u) < l(v)$, then LCA(u,v) is the node with the minimum level in the interval $[r(u), l(v)]$ of A.
- Ex 2 is an ancestor of 4, and $l(2) = 1 < l(4) = 3 < r(2) = 11$.
- Ex $r(4) = 10 < l(8) = 13$, and LCA(4,8) is the node with min level in $A[10{:}13]$, namely node 1 with level 0.
- Using the lemma, we can find LCAs in O(1) time, if we can find the node with min level in second property of lemma in O(1) time.
- In particular, given an array X and any interval $[i, j]$, we want to find the min value in $X[i{:}j]$ in O(1) time.

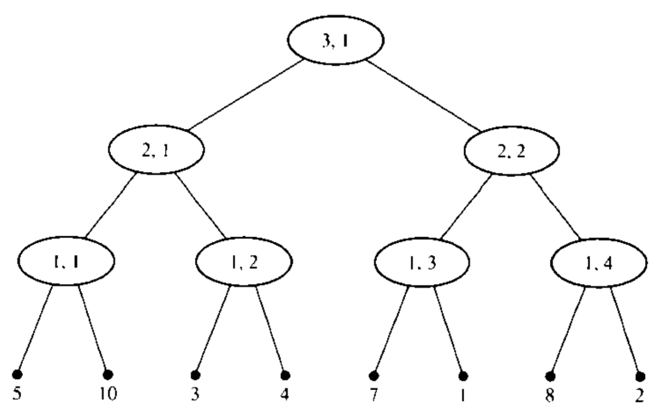


$A = (1, 2, 3, 2, 4, 5, 4, 6, 4, 7, 4, 2, 1, 8, 1, 9, 1)$

$B = (0, 1, 2, 1, 2, 3, 2, 3, 2, 3, 2, 1, 0, 1, 0, 1, 0)$

Range minima problem

- Given an array X of size n, preprocess X so that given any $1 \le i, j \le n$, we can find $Rmin[i,j] = \min(X_i, \dots, X_j)$ in O(1) time.
  - The preprocessing will take $O(\log n)$ parallel time and $O(n \log n)$ work.
- Assume for simplicity n is a power of 2.
- Form a complete binary tree with the values of X as the leaves.
  - Label each node in the tree by its height and index within a layer.
- For each node (h,j), create two arrays P(h,j) and S(h,j).
  - Let the values of the leaf nodes in the subtree rooted at (h,j) be $X_p, X_{p+1}, \dots, X_q$.
  - P(h,j) is the array of prefix minima of $(X_p, X_{p+1}, \dots, X_q)$, i.e. $P(h,j)[k] = \min(X_p, \dots, X_{p+k-1})$.
  - S(h,j) is the array of suffix minima of $(X_q, X_{q-1}, \dots, X_p)$, i.e. $S(h,j)[k] = \min(X_q, X_{q-1}, \dots, X_{q-k+1})$.



Ex Array $X = [5,10,3,4,7,1,8,2]$.

$P(2,1) = (5,5,3,3)$, $S(2,1) = (4,3,3,3)$

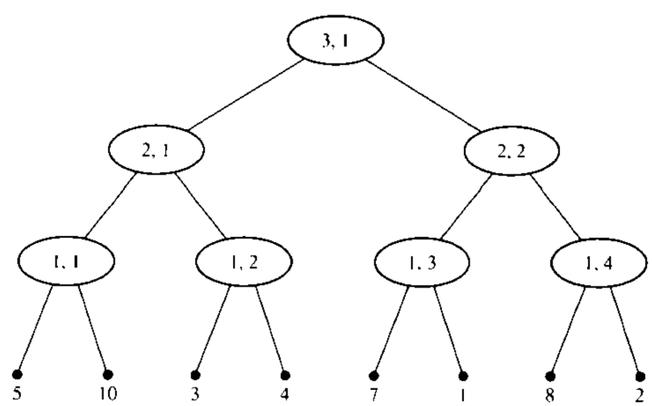
$P(2,2) = (7,1,1,1)$, $S(2,2) = (2,2,1,1)$

$P(3,1) = (5,5,3,3,3,1,1,1)$

$S(3,1) = (2,2,1,1,1,1,1,1)$

Computing range minima

- Given the P and S arrays, we can compute range minima in O(1) time.
- Given an interval [i,j], let $v = LCA(i,j)$.
  - Since X is represented by a complete binary tree, LCA can be computed in O(1) time.
  - Let u and w be the left, resp. right child of v.
  - Let X_i be the p'th node counting from the right in the subtree rooted at u.
  - Let X_j be the q'th node counting from the left in the subtree rooted at w.
  - Then $Rmin[i,j] = \min(S(u)[p], P(w)[q])$.
- Ex To find $Rmin[2,5]$, we have $LCA(2,5)$ is the root (3,1).
  - Then u = (2,1), w = (2,2). Also, $S(u) = (4,3,3,3)$ and $P(w) = (7,1,1,1)$.
  - Leaf 2 is the second child from the right in u's subtree, and leaf 5 is the second child from the left in w's subtree.
  - So $Rmin[2,5] = \min(10,3,4,7) = \min(S(u)[2], P(w)[2]) = \min(3,1) = 1$.

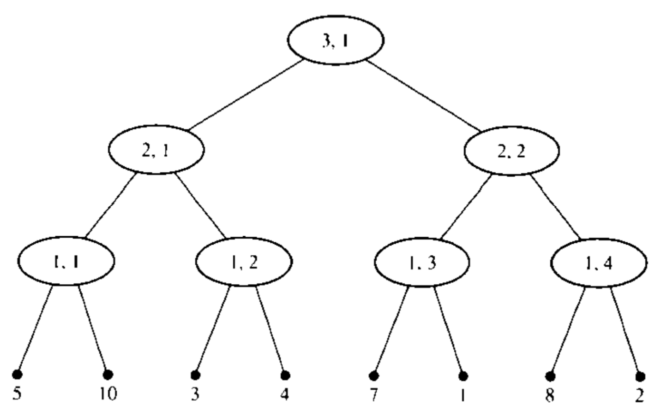


Ex Array $X = [5,10,3,4,7,1,8,2]$.

$P(2,1) = (5,5,3,3)$, $S(2,1) = (4,3,3,3)$
$P(2,2) = (7,1,1,1)$, $S(2,2) = (2,2,1,1)$
$P(3,1) = (5,5,3,3,3,1,1,1)$
$S(3,1) = (2,2,1,1,1,1,1,1)$

Computing P and S arrays

- We compute the P and S arrays from the bottom up.
- For a leaf with value X_i, the P and S arrays for it are just (X_i).
- Given a node v, let u and w be its left and right children, and suppose we've computed the P and S arrays for u and v.
  - Let x_u be the last value in $P(u)$, and x_w be the last value in $S(w)$.
  - Let $P'(w)$ be the elementwise min of $P(w)$ and x_u.
  - Let $S'(u)$ be the elementwise min of $S(u)$ and x_w.
- Then $P(v) = P(u) \circ P'(w)$, i.e. $P(u)$ followed by $P'(w)$. Also, $S(v) = S(w) \circ S'(u)$.

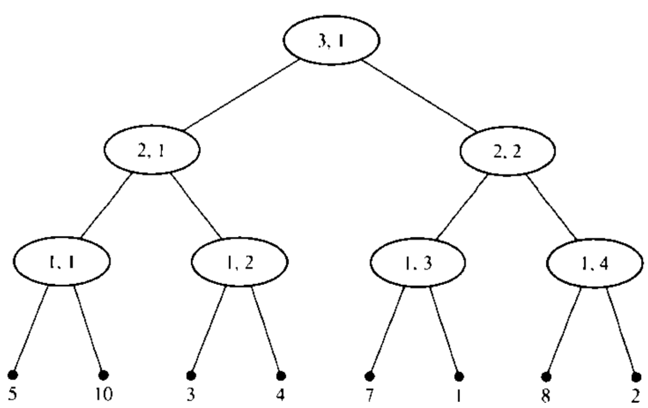


Ex Array $X = [5,10,3,4,7,1,8,2]$.

$P(2,1) = (5,5,3,3)$, $S(2,1) = (4,3,3,3)$
$P(2,2) = (7,1,1,1)$, $S(2,2) = (2,2,1,1)$
$P(3,1) = (5,5,3,3,3,1,1,1)$
$S(3,1) = (2,2,1,1,1,1,1,1)$

Computing P and S arrays

- Ex Suppose we computed the P and S arrays for nodes $u = (2,1)$ and $w = (2,2)$, and want to compute it for node $v = (3,1)$.
  - $x_u = 3$ and $x_w = 1$, so $P'(w) = (3,1,1,1)$ and $S'(u) = (1,1,1,1)$.
  - Then $P(3,1)$ and $S(3,1)$ are as shown below.
- Thm Given a tree with n leaves, all the P and S arrays in the tree can be computed in O(log n) parallel time and O(n log n) work.
- Proof Given two P (resp. S) arrays of size k, computing the parent P (resp. S) array takes O(1) parallel time and O(k) work.
  - On each layer of the tree, the total sizes of all P and S arrays is O(n).
  - So can compute each layer in O(1) parallel time and O(n) work.
  - There are O(log n) layers. So the theorem follows.
- The work can be reduced to O(n) using accelerated cascading.



Ex Array $X = [5,10,3,4,7,1,8,2]$.

$$P(2,1) = (5,5,3,3), \qquad S(2,1) = (4,3,3,3)$$
$$P(2,2) = (7,1,1,1), \qquad S(2,2) = (2,2,1,1)$$
$$P(3,1) = (5,5,3,3,3,1,1,1)$$
$$S(3,1) = (2,2,1,1,1,1,1,1)$$