

Tree Computations

Introduction

1. This lesson covers how to apply parallelism to algorithms based on trees
 - Basic principle is to split trees into lines and break into segments
 - Requires applying randomization

Tree Warm Up

1. Can store a tree as an array pool, just like we did with linked lists
 - Number the nodes of the tree
 - Store the parent of each node i at index i
2. Serial algorithm for finding the root of the tree:
 - Running time = $O(n)$

```
root(P[1:n])
{
  if n < 1 then return 0
  node <- (any node, 1..n)
  while P[node] == 0 do
    node <- P[node]
  return node
}
```

3. How do we make this parallel?
 - Idea: Explore from all nodes simultaneously. At each node, change parent to grandparent
 - Eventually, each node points to the root

```
hasGrandparent(k, P[1:n])
{
  return (k > 0) and (P[k] > 0) and (P[P[k]] > 0)
}
```

```
adopt(P[1:n], G[1:n])
{
  parfor i <- 1 to n do
    if hasGrandparent(i, P[:])
      G[i] <- P[P[i]]
    else
      G[i] <- P[i]
}
```

```
findRoots(P[1:n], R[1:n])
{
  let Pcur[1:n] <- P[:]
  let Pnext[1:n] <- (tempbuf)
  for l <- 1 to ceil(log(n)) do
    adopt(Pcur[:], Pnext[:])
    Pcur[:] <- Pnext[:]
  R[:] <- Pcur[:]
}
```

Parallel Root Finder

1. Which of these claims about findRoots are true?

- Uses pointer jumping (true)
- Is work-optimal
- Has polylogarithmic span (true)
- Works on a forest, not just one tree (true)

Work-Optimal List Scan/Prefix-Sum - Part 1

1. Wyllie's algorithm uses prefix-sums to compute list ranks, but is not work-optimal
 - $W(n) = O(n * \log(n))$
 - $D(n) = O(\log(n))$
2. Consider a trick where we shrink the list to size $m < n$
 - Then, run Wyllie $O(m \log(m))$ and restore full list and ranks
3. Assume step 2 dominates overall work. What choice of 'm' leads to work- optimality?
 - $O(\log(n))$
 - $O(n / \log(n))$ (true)
 - $O(n * \log(n))$
 - $O(\sqrt{n})$
 - $O(n)$
 - $O(n^2)$
4. 1 and 4 are not possible because $O(m * \log(m))$ would be asymptotically less than n
5. 3, 5, and 6 are not good choices because they give sub-optimal algorithms

Parallel Independent Sets - Part 1

1. An independent set is a subset I of vertices such that any vertex in the set does not also have its successor in the set
2. Consider the following linked list
 - $4 \rightarrow 2 \rightarrow 7 \rightarrow 1 \rightarrow 3 \rightarrow 5 \rightarrow 6 \rightarrow 8$
 - $N[i] = [3, 7, 5, 2, 6, 8, 1, 0]$
 - $I = \{3, 7, 8\}$ is an independent set
 - $I = \{3, 4, 6, 8\}$ is not an independent set because 8 is a successor to 6
3. Computing an independent set in serial is easy; just skip nodes
4. Computing an independent set in parallel is more difficult due to the problem of symmetry (all nodes look the same)
 - Need a scheme to break the symmetry
 - At every node, flip an unbiased coin ($\Pr[\text{heads}] = \Pr[\text{tails}] = 1/2$)
 - If the coin is heads, the node is a candidate to join the set
 - If a node sees its successor has a head, it changes to a tail

```

ParIndSet(N[1:n], I[:])
{
  let C[1:n], Chat[1:n] = space for coins
  parfor i <- 1 to n do
    C[i] <- flipcoin(H or T)
    Chat[i] <- C[i] // make a copy
  parfor i <- 1 to n do
    if(Chat[i] == H) and (N[i] > 0) and (Chat[N[i]] == H) then
      C[i] <- T
  I[:] <- gatherIf(1:n, C[1:n])
}

```

Parallel Independent Sets - Part 2

1. Consider the following linked list and initial coin flips
 - $4 \rightarrow 2 \rightarrow 7 \rightarrow 1 \rightarrow 3 \rightarrow 5 \rightarrow 6 \rightarrow 8$

- $N[i] = [3, 7, 5, 2, 6, 8, 1, 0]$
 - $C[i] = [H, T, T, H, T, H, H, H]$
2. After breaking symmetry
- $C[i] = [H, T, T, H, T, T, T, H]$
 - Independent set: $[1, 4, 8]$

Parallel Independent Sets - Part 3

1. What are the work and span of the parallel independent set algorithm?
 - $W(n) = O(n)$
 - Loops go from 1 to n , work per iteration is constant
 - $D(n) = O(\log(n))$
 - A parallel for-loop with constant work per iteration is $O(\log(n))$

Parallel Independent Sets - Part 4

1. What is the average number of vertices that end up in the independent set?
 - $1n / 8$
 - $3n / 4$
 - $1n / 4$
 - $7n / 8$
 - $1n / 2$
2. The answer is $n / 4$. Initially, it's $n / 2$, but after applying the correction in the parallel independent sets algorithm for when the next node is also heads, only one of the four options remain

Work-Optimal List Scan/Prefix-Sum - Part 2

1. To shrink the list for Wyllie's algorithm, we can use the parallel independent set algorithm
 - Similar to pointer-jumping, but only over the elements of the independent set
 - After we've calculated the independent set, we remove it from the list
 - This requires updating the next pointers and pushing each nodes' rank to its neighbor
 - We might need to repeat this process until we reach our desired size of $n / \log(n)$
 - Once we reach the desired size, we can do the list scan to get the ranks
 - To get the ranks for the other nodes, we need to run the process for shrinking the list in reverse
 - This isn't necessarily difficult, but requires lots of bookkeeping. Psuedocode is available in instructor notes

Work-Optimal List Scan/Prefix-Sum - Part 3

1. How many times do you need to run the independent set to shrink the list to $O(n / \log(n))$ in length?
 - $O(1)$
 - $O(\log(n))$
 - $O(\log(\log(n)))$
 - $O(\sqrt{\log(n)})$
2. The answer is $O(\log(\log(n)))$
 - After each iteration, the list is roughly $3/4$ the size of the previous list
 - After k calls, the list length is $(3/4)^k * n \leq n / \log(n)$
 - Solving this relation gives $O(\log(\log(n)))$
 - Work and span:
 - $W(n) = O(n * \log(\log(n)))$
 - $D(n) = O(\log(n) * \log(\log(n)))$
3. Additional considerations:
 - This is the average case, but there's some distribution around this case and we care what the distribution is

- Is the distribution skinny or fat? How much weight is in the tails?
- Also, how much bookkeeping is required to shrink and restore the list?

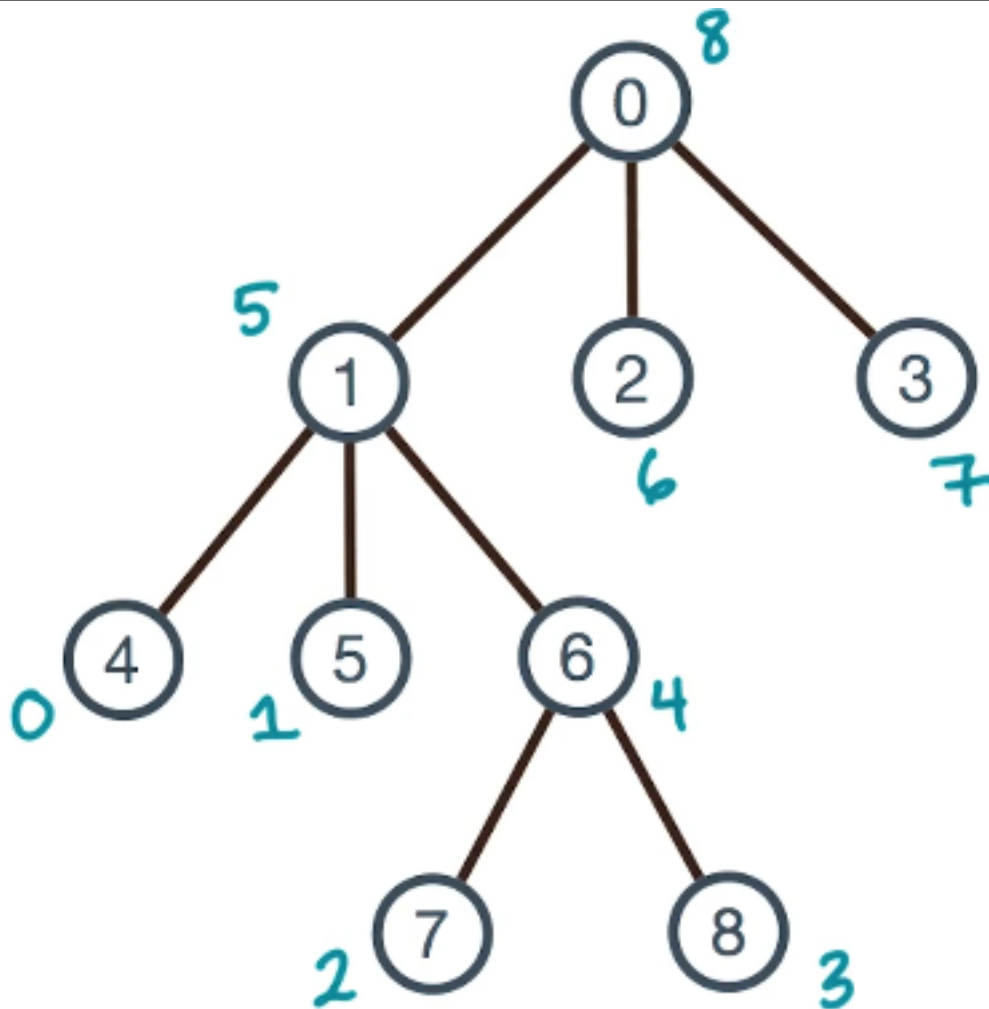
A Seemingly Sequential Tree Algorithm

1. Suppose you want to compute a postorder numbering of a tree

```
postorder(root, V[1:n], v0)
{
  v <- v0
  foreach C in children(root) do
    v = postorder(C, V[:], v) + 1
  V[root] = v
  return v
}
```

2. How do we do this in parallel?

- Looks inherently sequential, but looks similar to list ranking
- Need a way to convert a tree to a list

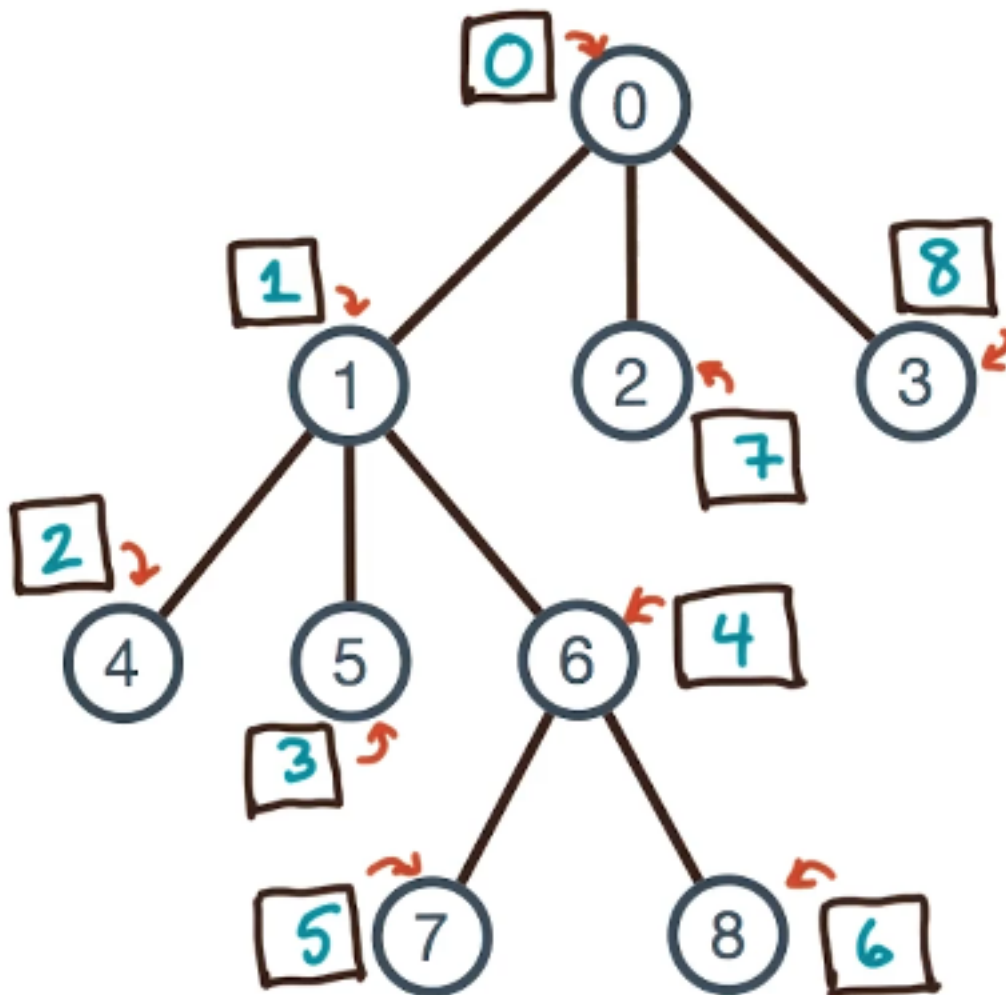


Postorder Notation

Another Tree Traversal

1. Consider the same tree from the previous slide. What would a preorder notation look like for this tree?

```
preorder(root, V[1:n], v0)
{
  v <- v0
  V[root] = v
  foreach C in children(root) do
    v = preorder(C, V[:], v) + 1
  return v
}
```



Preorder Notation

The Euler Tour Technique

1. Take every undirected edge of the tree and represent it as a pair of directed edges
 - At each node, the number of incoming and outgoing edges are the same
 - This fact makes the graph Eulerian
2. Euler circuit: A closed path that uses every edge once

3. If we use this notation and come up with a clever choice of initial values, prefix-summing over the tree will give the postorder values
 - Set the parent-to-child sinks to 0
 - Set the child-to-parent sinks to 1
4. Summary:
 - Convert the tree to a list (Euler circuit)
 - Apply the numbering to each node
 - Prefix sum over the list

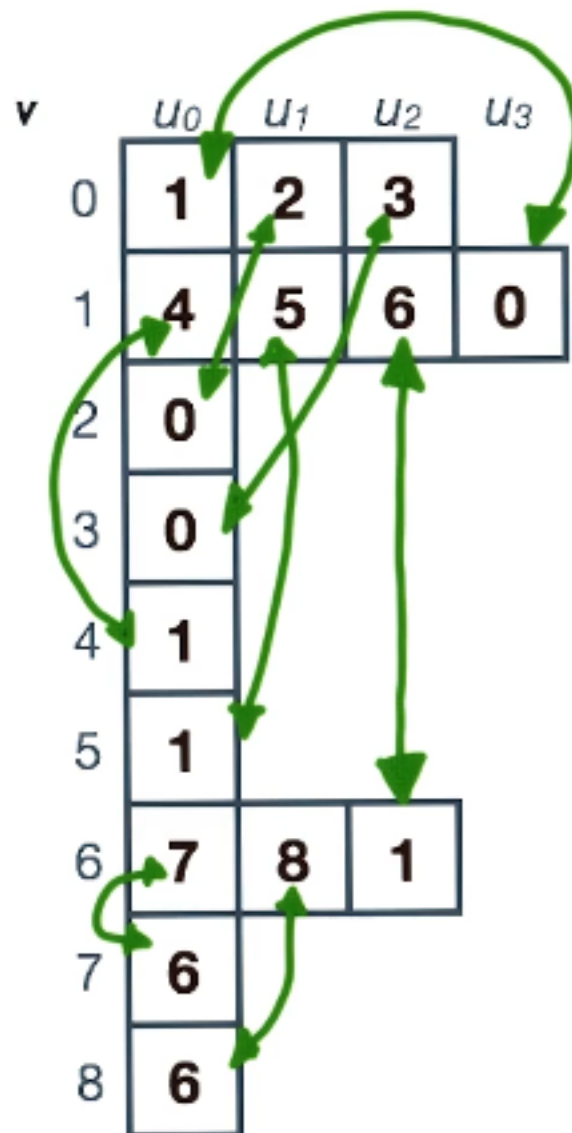


Euler Tour of a Tree

The Span of an Euler Tour

1. The work of the Euler tour is $W(n) = O(n)$
2. The span of the prefix sum is $O(\log(n))$
3. Should the overall span of the Euler tour algorithm depend on the overall depth of the tree, or is it just $O(\log(n))$?
 - $D(n) = O(\log(n))$
 - After we've converted the tree into a list, the span no longer depends on the shape of the tree

- Start with a version of the tree in which each undirected edge is represented by a pair of directed edges
 - For each node v , we define its adjacency list as the set of its outgoing neighbors
2. Successor function:
- $s(U_i, v) = (v, U_{(i+1) \% d_v})$
 - Given an edge that goes from U_i to v , the successor function returns the next neighbor in v 's adjacency list
 - Modulo makes the list circular
3. Is the cost of applying the successor function constant?
- Yes, but we need to add the cross-edges to the adjacency list
 - Cross-edge: What next node does the end of a list point to?



Cross Edges

Successor or Failure-or

1. What is $s(s(s(6,8)))$ for the following adjacency list?

- $s(6,8) = (8,6)$
- $s(8,6) = (6,1)$
- $s(6,1) = (1,0)$

v	<i>u</i> ₀	<i>u</i> ₁	<i>u</i> ₂	<i>u</i> ₃
0	1	2	3	
1	4	5	6	0
2	0			
3	0			
4	1			
5	1			
6	7	8	1	
7	6			
8	6			

Adjacency List

Conclusion

1. Two frameworks for performing parallel operations on trees
 - One is built on top of work-optimal lists
 - The other is on the rank-compress framework for evaluating expression trees
2. Linearizing a tree is an important concept for achieving load balance

- Load balancing is required for performing parallel algorithms at scale