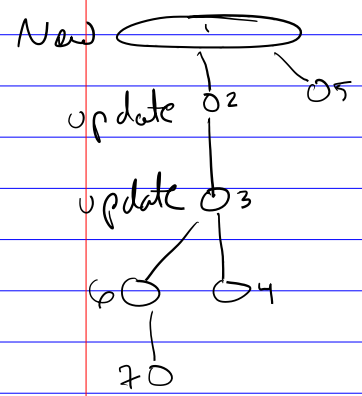
Goals

1. sequential semantics (functional)
2. Cost model & provable bound
3. Work efficiency
4. Low span
5. As simple as sequential
6. Sequential as a "byproduct".

Open Question: Does there exist a breadth first search that achieves 3, 4.

Problem with pure array

1. persistence (immutability) requires copying.
   see "Purely functional arrays" (POPL '17)
   (Haskell is an example of a pure language with an impure cost semantics.)
2. Solution: new type system keep a history of changes



Version tree

Goal: everytime we update, it's cheap.

Here they're all cheap except updates 5 & 6.

	Leaf		Interior Node		$n = \|A\|$
	W	S	W	S	
subA	$O(1)$	—	$O(\log n)$	$O(\log n)$	
updateA	$O(1)$		$O(n)$	$O(\log n)$	
injectAV	$O(\|V\|)$	$O(1)$	$O(\|V\|+n)$	$O(\log n)$	

(l = label)

$\delta : l \longrightarrow \{+, -\}$

leaf, interior node

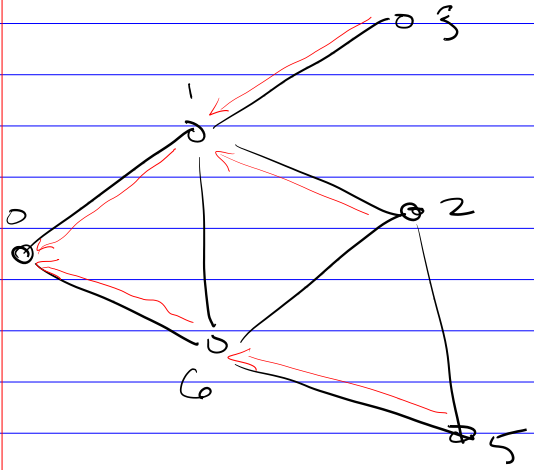
$\delta\ l \longrightarrow \delta', v, w, d$

a = pure array
l = label (either "leaf" or "interior")

$A : (a, l) \qquad \delta[l \to +] \quad \text{new } l'$

$\delta,\ \text{update}(A, v) \longrightarrow \delta[l \to -, l' \to +], (\text{update}(a, v.i), l'), !, !$

Breadth First Search (BFS)



Often the datastructure used for BFS is a queue. Not appropriate.

Want to search a given depth in parallel

$W = O(m+n)$

$S = O(d \cdot \log n)$

where $m = |\text{Edges}|$

$n = |\text{vertices}|$

$d = \text{diameter}$

Algorithm

Repeat over levels F, P

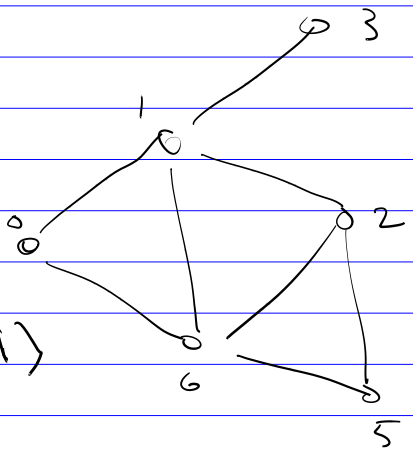
1) Neighbors of F
2) Filter those not yet visited
3) remove duplicate

How to represent a graph in a way that facilitates parallelism.

Don't want adjacency lists, per se.

$\langle \langle 1,4 \rangle, \langle 0,2,3 \rangle, \langle 1,6,5 \rangle, \ldots \rangle$

vertex — neighbors



$F = \langle 1,4 \rangle$

After visiting $\langle 1,4 \rangle$,

$P = \langle 0,0,-1,-1,0,-1 \rangle$