Washington University in St. Louis

Engineering

# Dynamic Trees and Path Sets

Jon Turner
Computer Science & Engineering
Washington University

www.arl.wustl.edu/~jst

# Dynamic Trees

- Collection of trees, with each node having a cost; tree edges directed from child to parent; operations are:
  - *findroot*($v$): return the root of the tree containing node $v$
  - *findcost*($v$): return pair [$w$,$x$] where $x$ is the min cost of a node on tree path from $v$ to *findroot*($v$) and $w$ is last node on path with cost $x$
  - *addcost*($v$,$x$): add $x$ to cost of every node on the path from $v$ to *findroot*($v$)
  - *link*($v$,$w$): combine the trees containing vertices $v$ and $w$ by adding the edge [$v$,$w$]; $v$ must be a root
  - *cut*($v$): divide tree containing $v$ into two trees by deleting the edge between $v$ and $p(v)$
- For efficient implementation, each tree is decomposed into a set of linked paths

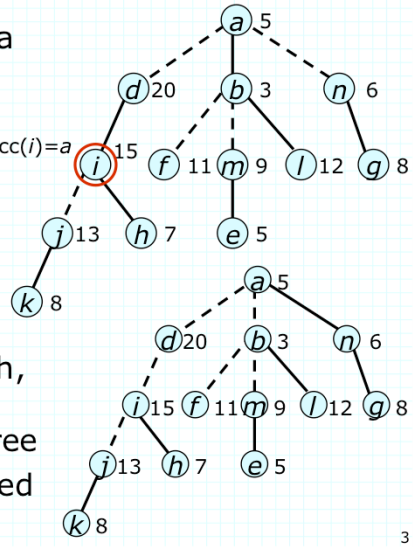
2

Washington University in St. Louis

Engineering

# Representing Trees As Paths

- A tree can be represented as a set of linked paths
  - » edges linking different paths are called *dashed* others are *solid*
  - » direction of paths is up the tree
- All nodes in a path are ancestors of path *head*
  - » so, each node has at most one solid edge to a child
- If *v* is *canonical* node of a path, *succ*(*v*) is parent of the tail of the path containing *v* in the tree
- A given tree can be represented in many different ways



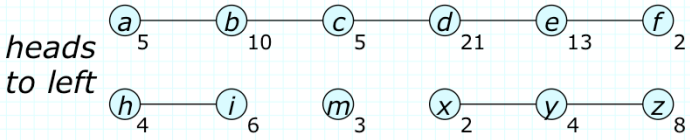
# Path Sets

- Collection of paths in which each node has a cost; the first node on a path is called the *head*, the last is called the *tail*; the operations on a path set are:
  - *findpath*(*v*): return the path (actually, the canonical node) containing node *v*
  - *findtail*(*p*): return the tail of *p*
  - *findpathcost*(*p*): return pair [*w*,*x*], where *x* is the min cost of any node on path *p* and *w* is the last node of cost *x*
  - *addpathcost*(*p*,*x*): add *x* to the cost of every node on *p*
  - *join*(*p*,*v*,*q*): return the new path formed by adding an edge from the tail of *p* to *v* and from the *v* to the head of *q*; either *p* or *q* may be empty, but *v* must be a single node path
  - *split*(*v*): return the pair [*p*,*q*] formed by dividing path containing *v* at *v*; either *p* or *q* (or both) may be empty
- Path ops can be implemented to run in $O(\log n)$ time

Washington University in St. Louis

Engineering

# Example of Path Operations



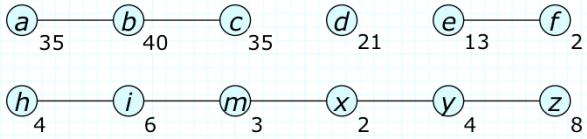

*heads to left*

*tails to right*

split(*d*)

join(findpath(*i*),*m*,findpath(*y*))

addpathcost(findpath(*b*),30)

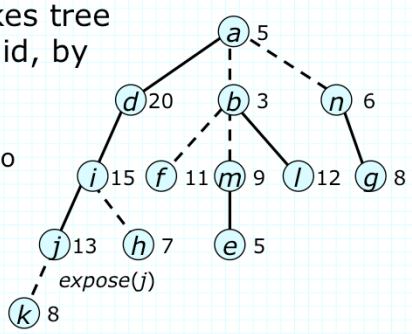



findcost(findpath(*b*)) = [*c*,35]

5

# *Exposing* Paths

- If tree path from a node *v* to *findroot*(*v*) is one of the pathset paths, we can use the *findpathcost* op on this path to implement the *findcost* op in the dynamic tree
- The *expose*(*v*) operation makes tree path from *v* to *findroot*(*v*) solid, by
  - » converting dashed edges along the path to solid
  - » converting solid edges incident to the path to dashed
  - » returns resulting path
- Using expose and *pathset* ops, we can implement all the dynamic tree ops
  - » *findcost*(*u*) in tree ⇒ *expose*(*u*) then *findpathcost*(*u*)

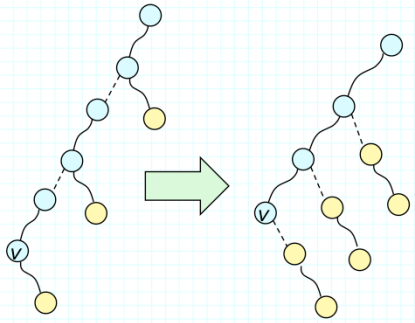


# Implementing Dynamic Trees

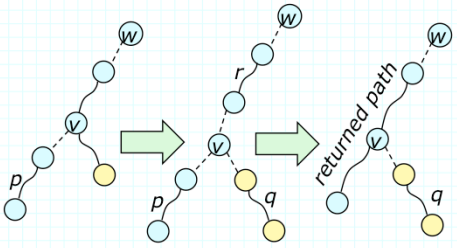
```
node function findroot(node v);
    return findtail(expose(v));
end;

[node,real] function findcost(node v);
    return findpathcost(expose(v));
end;

procedure addcost(node v, real x);
    addpathcost(expose(v),x);
end;
```

```
procedure link(node v,w);
    node u;
    u := join(null,expose(v),
                   expose(w))
    succ(u) := null;
end;

procedure cut(node v);
    path q,r;
    expose(v);
    [q,r] := split(v);
    succ(v) := null;
    succ(r) := null;
end;
```

```
path function expose(node v);
    path p; p := null;
    do v ≠ null ⇒
        [p,v] := splice(p,v);
    od;
    succ(p) := null;
    return p;
end;
```

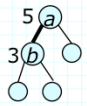



```
[path, node] function splice(path p, node v);
    path q,r; node w;
    w := succ(findpath(v));
    [q,r] := split(v);
    if q≠null ⇒ succ(q) := v fi;
    return [join(p,v,r),w];
end
```



# Analysis of Dynamic Trees

- Number of path ops required to implement a sequence of $m$ tree operations is $O$(# of splices)
  - » each splice takes $O(1)$ path operations
- Let *size* of a node $v$ be number of descendants it has
  - » call an edge from $v$ to its parent $w$ *heavy* if $size(v) > size(w)/2$ and *light* otherwise



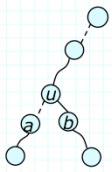

- *Lemma 5.1*. If $v$ is any node, there is at most one heavy edge from $v$ to a child of $v$ and $\leq \lfloor \lg n \rfloor$ light edges on the path from $v$ to *findroot*($v$)

  The proof follows directly from the definitions
- Note that each splice during an expose (except first) converts a dashed edge into a solid edge
  - » these edges are either heavy or light; by lemma, there are at most $\lfloor \lg n \rfloor$ light edges and the rest are heavy

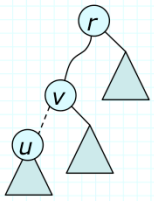
- If $x$=number of heavy solid edges created by splices during an entire sequence of $m$ operations,

  $$(\#\text{ of splices}) \le (\#\text{ of exposes})(\lfloor \lg n \rfloor + 1) + x$$

- Every heavy solid edge created during a splice increases the total number of heavy solid edges
- If $y$ is number of heavy solid edges destroyed by various operations, then $x - y \le n - 1$ since there can be at most $n-1$ edges at the end of the sequence
  - » thus, $x \le y + n - 1$
- A splice destroys a heavy solid edge only if it is the first splice of an expose or if it converts a light edge from dashed to solid
- So, each expose destroys at most $\lfloor \lg n \rfloor + 1$ heavy solid edges, so all exposes destroy $\le 2m(\lfloor \lg n \rfloor + 1)$
  - » still need to account for those destroyed by other operations



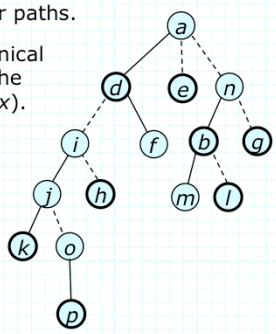
- A cut destroys $\leq \lfloor \lg n \rfloor + 1$ heavy solid edges outside of those destroyed by the expose; there are $\leq m/2$ cuts
- None of the other tree operations destroys any heavy solid edges other than those destroyed by exposes




- Thus, $y \leq (5/2)m(\lfloor \lg n \rfloor + 1)$, $x \leq (5/2)m(\lfloor \lg n \rfloor + 1) + n - 1$ and number of splices is $\leq (9/2)m(\lfloor \lg n \rfloor + 1) + n - 1$
- *Theorem 5.1*. A sequence of $m$ dynamic tree ops on $n$ vertices requires $O(m \log n)$ path set ops
- Path sets can be implemented with binary search trees
  - » to implement *addpathcost*, need *differential cost representation*
  - » with *balanced* trees, each op takes $O(\log n)$ time, leading to a running time of $O(m (\log n)^2)$ for dynamic trees data structure
  - » can improve to $O(m \log n)$ using self-adjusting search trees

# Exercises

1. The diagram below shows a dynamic tree represented as a collection of paths that are linked together. The nodes with a heavy outline are the canonical nodes for their paths.



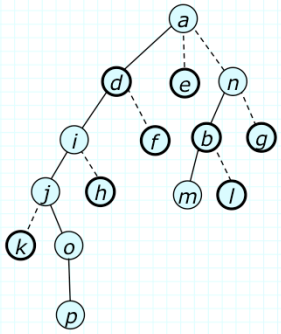

For each canonical node *x*, give the value of succ(*x*).

*succ(k)=d,*
*succ(d)=null,*
*succ(p)=j,*
*succ(e)=a,*
*succ(b)=a,*
*succ(g)=n,*
*succ(l)=b,*
*succ(h)=i*

Which edges on the path from *p* to the root of the tree are heavy?

*Edges ij, di and ad are heavy*

Show how the tree and the succ values change following an expose at node *p*. Assume that when two path are combined, the node that comes first alphabetically becomes the canonical element.




*succ(k)=j, succ(f)=d. Nothing else changes.*

12

2. In the tree from problem 1, which edges are heavy and solid before the expose?

   *ij, ad and bn are heavy and solid before.*

   Which are heavy and solid after the expose?

   *ij, ad, di and bn are heavy and solid after.*

   Suppose we did a second expose at node *g*. How does this change the set of heavy solid edges?

   *Edges ad and bn are no longer solid, so they drop out of the set of heavy solid edges. There are no other changes.*

   Suppose we did a cut at node *j* following the second expose. How does this change the set of heavy solid edges?

   *Edge ij is no longer an edge after the cut, so it drops out of the set. Also, edge di is no longer heavy after the cut, it drops out. This leaves us with no heavy solid edges.*

3. Consider a dynamic tree implemented using the path sets data structure. Suppose that the path from node *u* to the root of the tree includes 12 dashed edges, four of which are heavy. What does this tell you about the number of nodes in the tree?

   *It means that there are at least 256 nodes in the tree, since the path has 8 light edges and any path from a node to the root can have no more than lg n light edges.*

   How many heavy solid edges are created during an expose that starts at *u*?

   *The 4 dashed heavy edges along the path will become heavy solid edges during the expose, so the number of heavy solid edges that are created by the expose is four.*

   Give an upper bound on the number of heavy solid edges that are destroyed during the expose.

   *If u has a child that connects to it by a heavy solid edge, then this heavy solid edge is destroyed. For each of the light edges (x,p(x)) along the path, there may be a heavy solid edge from p(x) to another child. These edges are destroyed by the expose. This gives us a total of no more than 9 heavy solid edges that are destroyed by the expose.*