Least Common Ancestor
COMP 2860

Bernhard Scholz, Michael Charleston

The University of Sydney
Australia
Given a rooted Tree $T(V, E, r)$

Find the least common ancestor of two nodes

Example: least common ancestors of $k$ and $i$ is $b$. 

![Tree Diagram]

- root (a)
- Node b
- Nodes d and e
- Nodes f, g, h, i
- Nodes j, k, l, m, n, o, p
The set of ancestors $a(u)$ of a node $u \in V$ is defined as

$$a(x) = \begin{cases} 
\{x\} \cup a(parent(x)), & x \neq \text{root} \\
\{x\}, & \text{otherwise}
\end{cases}$$

where $parent(x)$ is the parent of a node in the tree.

$$a(i) = \{i, e, b, a\}$$
Common Ancestors

- The set of common ancestors $ca(u, v)$ of nodes $u$ and $v$ is defined as
  $$ca(u, v) = a(u) \cap a(v)$$

- The least common ancestors $lca(u, v)$ is a common ancestors of $u$ and $v$ with maximal depth, i.e. order common ancestors $ca(u, v) = \{x_1, \ldots, x_k\}$ according to their level:
  $$l(x_1) < l(x_2) < l(x_3) \ldots < l(x_k)$$

  where $l(u)$ is the level of a node.

- $x_1$ is the root vertex $r$.

- $x_k$ is the least common ancestor.
Properties of Least Common Ancestor

- **Identity**
  \[ \forall u \in V : lca(u, u) = u \]

- **Commutativity**
  \[ \forall \{u, v\} \subseteq V \times V : lca(u, v) = lca(v, u) \]

- **Number of different lca pairs**
  \[
  \binom{n}{2} = \frac{n(n - 1)}{2}
  \]
Exercise

Implement an LCA algorithm to compute $lca(u, v)$ with following interface:

```java
interface Tree {
    // return number of nodes in the tree
    int getNumNodes();

    // return the parent of node
    TreeNode getParent(TreeNode x);

    // return root node of the tree
    TreeNode getRoot();
}
```
do {
    s1.pushItem(u);
    if (u != tree.getRoot())
        u = tree.getParent(u);
} while (u != tree.getRoot());
do {
    s2.pushItem(v);
    if (v != tree.getRoot())
        v = tree.getParent(v);
} while (v != tree.getRoot());
while (s1.topItem() == s2.topItem()){
    lca = s1.popItem(); s2.popItem();
}
return lca;
Worst-case of simple algorithm is $O(|V|)$.  
Example:

![Diagram of a tree with nodes labeled a, b, c, d, e, and an edge labeled $lca(e,e)$]
while (tree.getLevel(u) < tree.getLevel(v)) {
    u = tree.getParent(u);
}
while (tree.getLevel(v) < tree.getLevel(u)) {
    v = tree.getParent(v);
}
while (u != v) {
    u = tree.getParent(u);
    v = tree.getParent(v);
}
return u;
Worst-case of improved algorithm is still $O(|V|)$.

However, simpler to program – only level information is required.

Example:
Improvements

How can we improve runtime of LCAs? This is important because there are many applications in

- Bio-Informatics
- Compilers
- Graph Theory
- ...

Brute-Force Approach:

- compute $lca(u, v)$ in a pre-processing step
- requires $O(|V|^2)$ space
- lca query is done in $O(1)$ (i.e. table lookup)
- for huge trees not acceptable (e.g. phylogenetic trees)
Harel and Tarjan’s Algorithm

The algorithm has following properties:

- linear in time
- linear in space

Operations:

- pre-processing: takes $O(|V|)$
- $lca(u, v)$: takes $O(1)$ time

Disadvantage:

- Very difficult to program
- Perl implementation

http://search.cpan.org/~matkin/Algorithm-Tree-NCA-0.01/NCA.pm
Consider special case of a tree:

- tree is a binary balanced tree
- labelling is a symmetric order
```java
int labelCtr = 1;
void labelNode(TreeNode u) {
    if (!isLeaf(u)) {
        labelNode(u.left);
        u.label = labelCtr++;
        labelNode(u.right);
    } else {
        u.label = labelCtr++;
    }
}
```
Properties of Symmetric Order

The vertices of height $h$ are numbered $2^h$, $3 \cdot 2^h$, $5 \cdot 2^h$, \ldots from left to right.

Proof?
More Properties

- The height of a node $u$ is the largest integer $h$ such that $2^h$ divides the label of $u$.
- The descendants of a node $u$ are those vertices with the label in the range $[u - 2^{h(u)} + 1, u + 2^{h(u)} - 1]$.
- If $u$ is a node and $h$ is a height such that $h \geq h(u)$ then the ancestor of $u$ of height $h$ has the number $2^{h+1} \lfloor u/2^{h(u)} - 1 \rfloor$.
- If $u$ and $v$ are two unrelated vertices, the height of the least common ancestor is $\lfloor \log_2 u \oplus v \rfloor$, where $x \oplus y$ is the integer whose binary representation is the bitwise exclusive or of the binary representations of $x$ and $y$. 
Algorithm for LCAs of Binary Balanced Trees

Compute $lca(u, v)$:

1. If $u$ is a descendant of $v$ return $v$ (see range check)
2. If $v$ is a descendant of $u$ return $u$ (see range check)
3. Otherwise compute height of ancestors and then the ancestors

See C-Code (resource materials)
Generalisation

Algorithm:
- A tree is mapped to a binary balanced tree
- Lca is performed in the binary balanced tree
- Result of the lca operation is mapped back to the original tree

Reference:
Summary

- Concepts of least common ancestors
- A simple algorithm based on stacks
- An improved algorithm based on node levels
- Brief overview of Harel and Tarjan’s least common ancestor algorithm