Distributed Data Structures

- Labeling Schemes
- Nearest Common Ancestor

Philip Bille

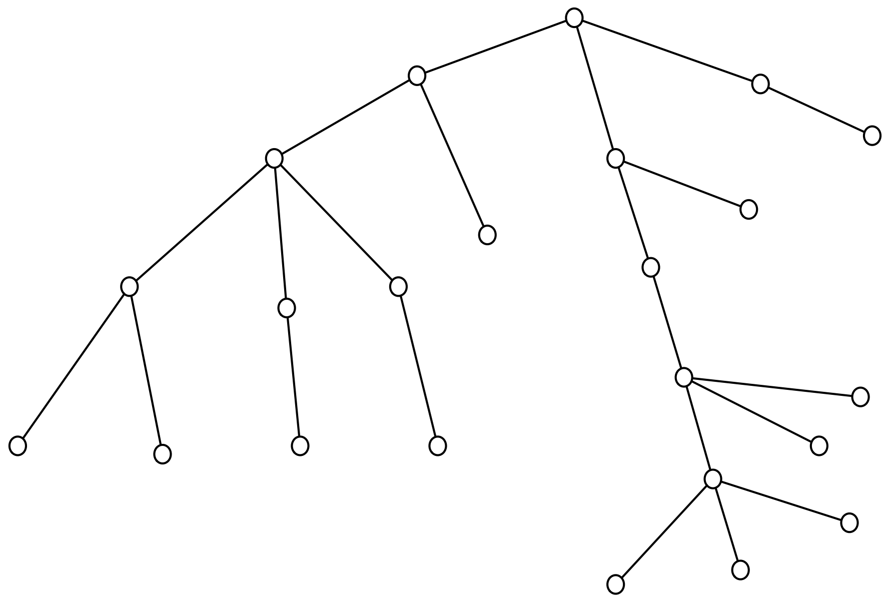
Distributed Data Structures

- Labeling Schemes
- Nearest Common Ancestor

Labeling Schemes

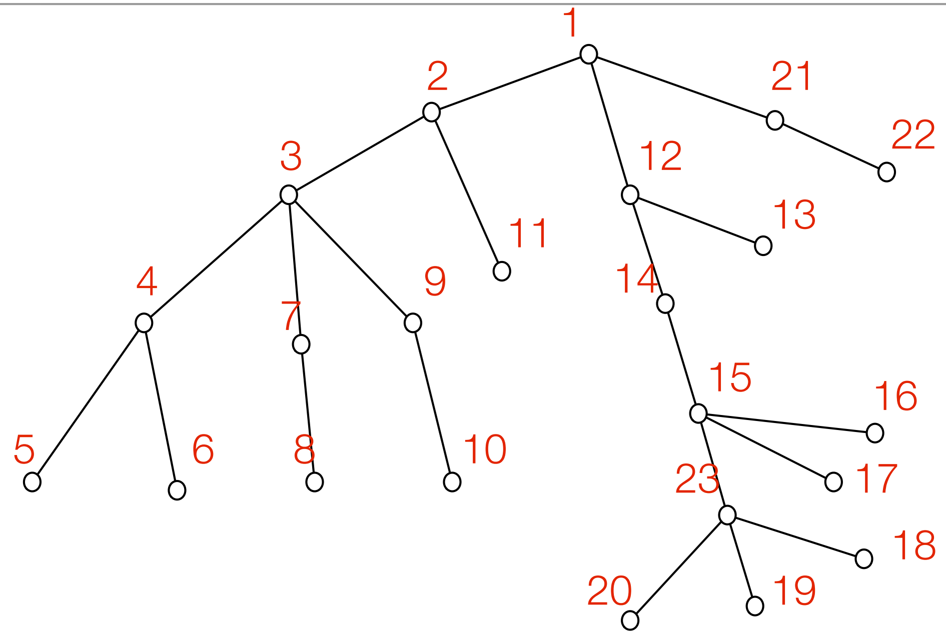
- Labeling scheme.
  - Input. Graph G and query q(.,.) on pairs of nodes.
  - Preprocess. Assign a label to each node v.
  - Query. Given only label(v) and label(w) compute q(v,w).
- Goals.
  - Minimize maximum length of labels.
  - Fast queries.

Labeling Schemes



- Parent labeling scheme.
  - Rooted tree with n nodes.
  - Parent queries. Is v a parent of w?
- How can we solve this?

Labeling Schemes



- Parent labeling scheme.
  - Assign unique ID to each node.
  - label(v) = ID(v) · ID(parent(v))
  - v is parent w iff ID(v) = ID(parent(w)).
- Analysis.
  - $2\lceil \log n \rceil$ bit labels.

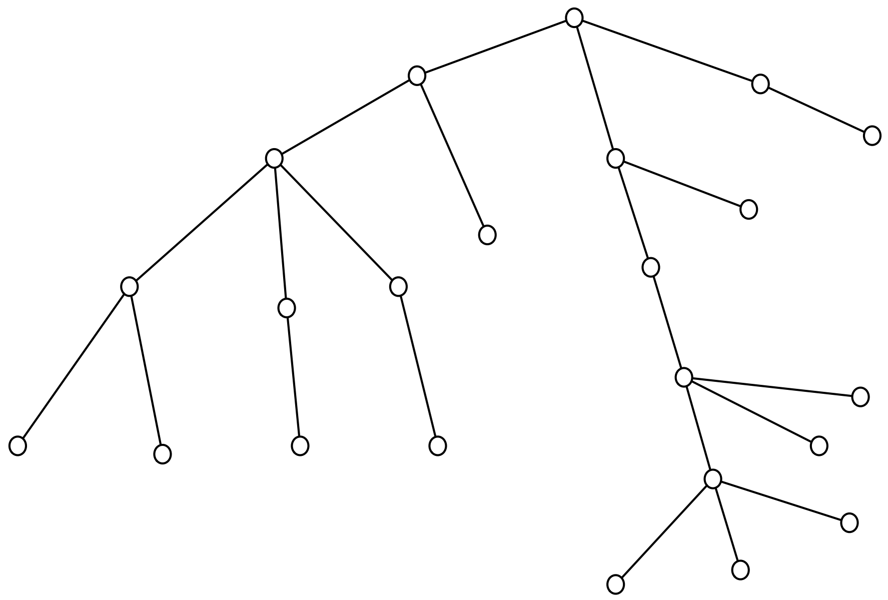
Labeling Schemes

- Applications.
  - Compact distributed data structures.
    - Network routing, graph representation, search engines, etc.
  - Graph theory.
    - Universal graphs, compression.
  - I/O complexity.
    - Minimal memory access.

Distributed Data Structures

- Labeling Schemes
- Nearest Common Ancestor

Nearest Common Ancestors



- Nearest common ancestors.
  - The ancestors of v is the set of nodes from v to the root.
  - The common ancestors of v and w are the ancestor of both v and w.
  - The nearest common ancestor of v and w, nca(v, w), is the common ancestor of greatest depth.
- Nearest common ancestor problem. Preprocess a rooted tree T to support
  - nca(v,w): return the nearest common ancestor of v and w.

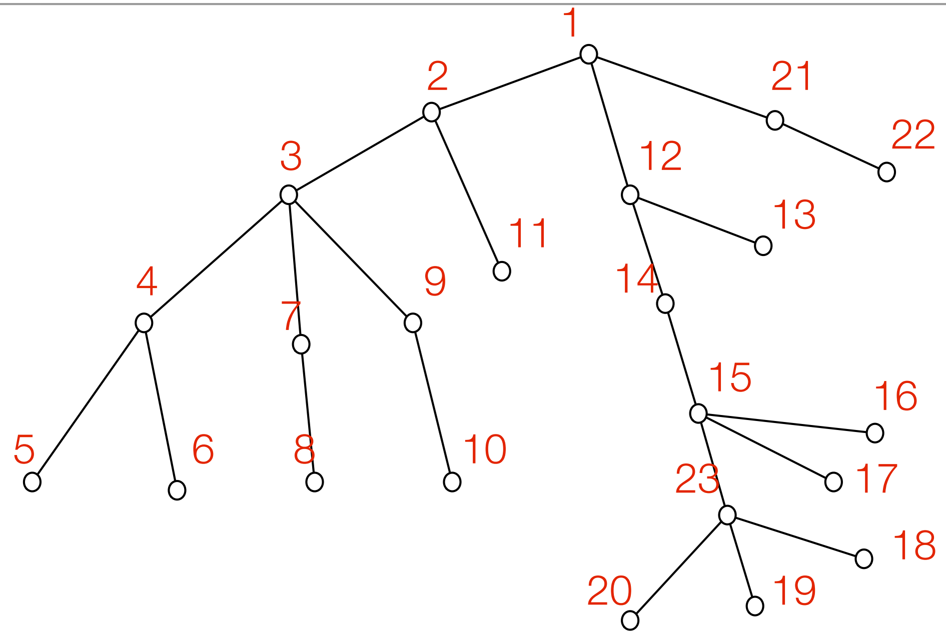
Nearest Common Ancestors

- Applications.
  - Weighted matching
  - Minimum spanning trees
  - Dominator trees
  - Approximate string matching
  - Dynamic planarity testing
  - Network routing
  - ....

Nearest Common Ancestors

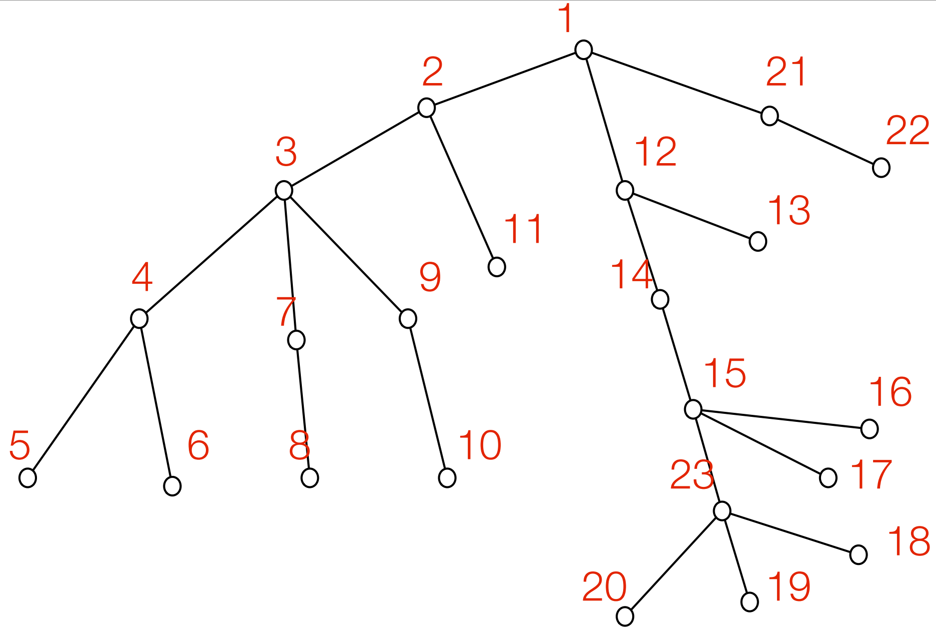
- Goal.
    - Labeling scheme for nearest common ancestor queries with O(log n) bits labels.
    - Query must output label(nca(v,w)).
- Solution in 3 steps.
    - ID encoding.
    - Heavy path decomposition.
    - Alphabetic codes.

Nearest Common Ancestors



- ID encoding.
    - Assign unique ID to each node.
- How can we use these for an nca labeling scheme?

Nearest Common Ancestors

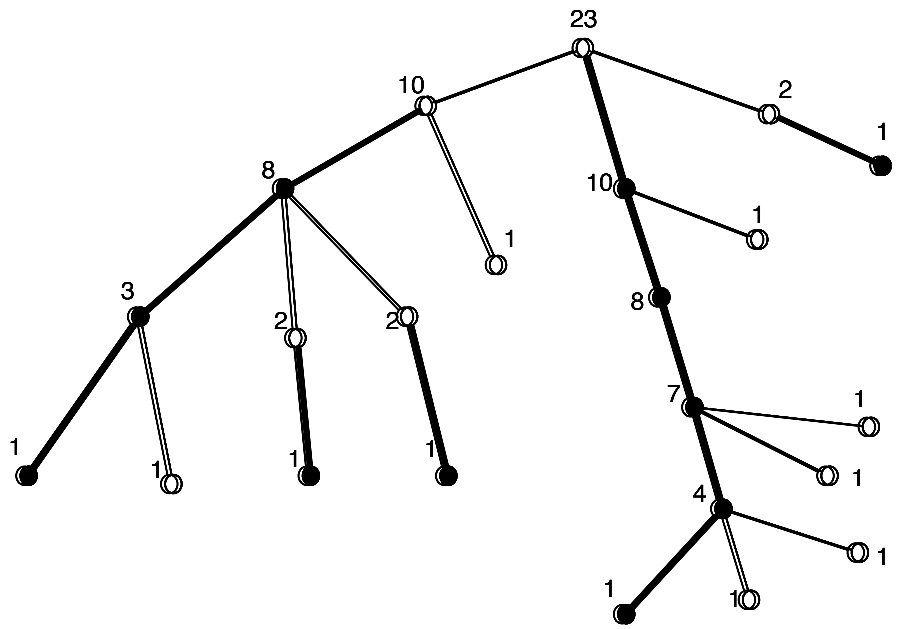


- ID encoding.
  - Assign unique ID to each node.
  - label(v) = ID(v_1) · ID(v_2)· · ·ID(v_k), where v_1, ..., v_k is the path from the root to v = v_k.
- Queries.
  - Compute the longest prefix of IDs.
- Analysis.
  - $h\lceil \log n \rceil$ = O(n log n) bit labels.

Nearest Common Ancestors

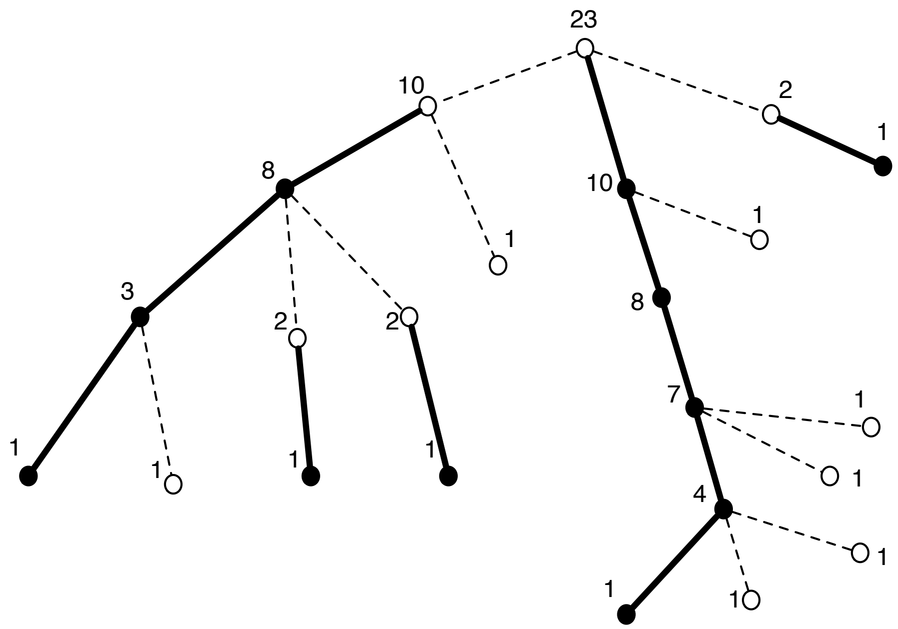
Labeling scheme	label length	query time
ID encoding	O(n log n)	

Nearest Common Ancestors



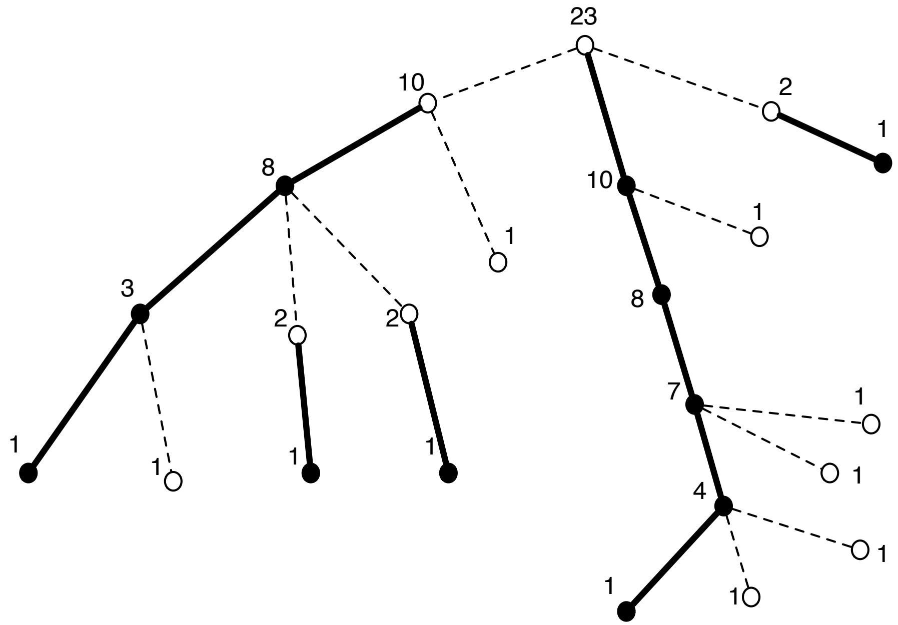
- Size. The size of a node v is number of descendants of v.
- Heavy and light nodes.
  - Root is light.
  - For each internal node v, pick child w of maximum size and classify it as heavy. The other children are light.
- Heavy and light edges. Edge to a heavy child is heavy and edge to a light child is light.
- Heavy path decomposition. Removing light edges partitions tree into heavy paths.

Nearest Common Ancestors



- Light depth.
  - depth(v) = #edges on the path from v to the root.
  - lightdepth(v) = #light edges on the path from v to the root.
- What bounds can we get for depth and lightdepth?
- Lemma. For any node v, lightdepth(v) = O(log n).

Nearest Common Ancestors

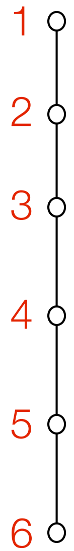


- Idea.
  - Find a good nca labeling scheme on a path.
  - Apply on each heavy path.

Nearest Common Ancestors

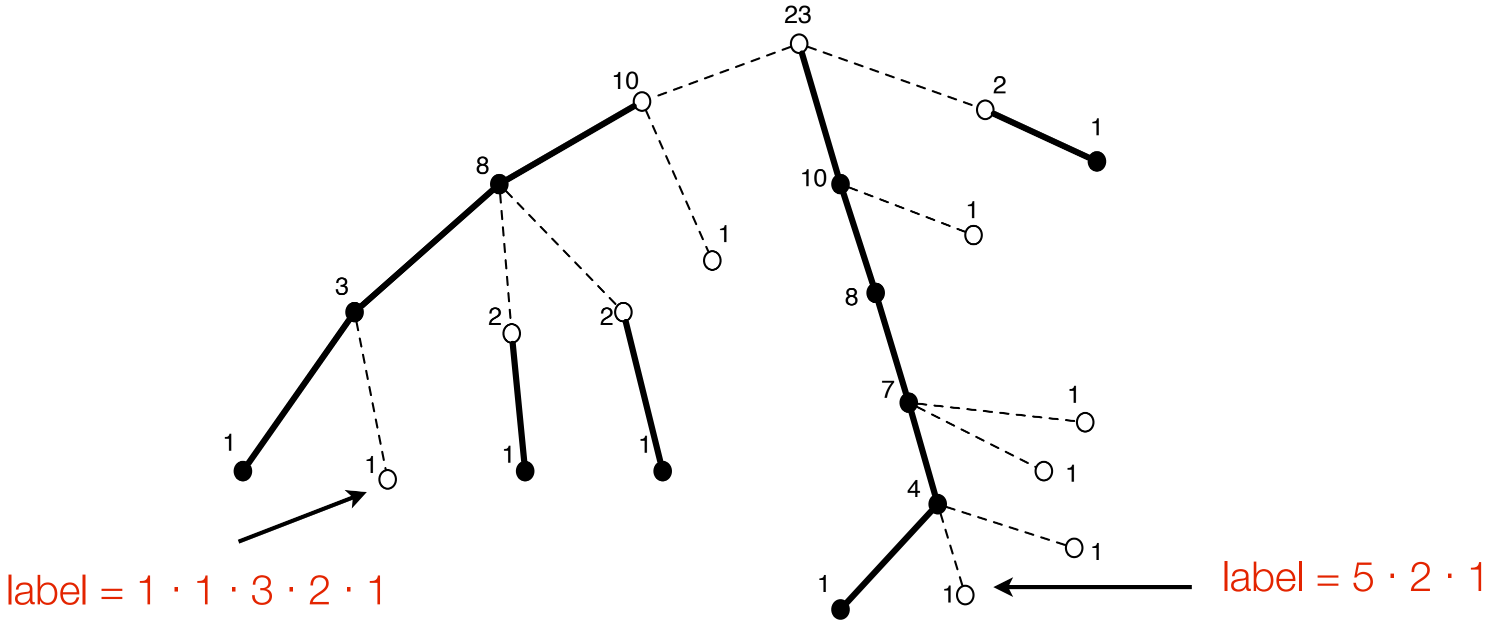
- Nearest common ancestors on a path.
  - How can we make an nca labeling scheme for a path?

Nearest Common Ancestors



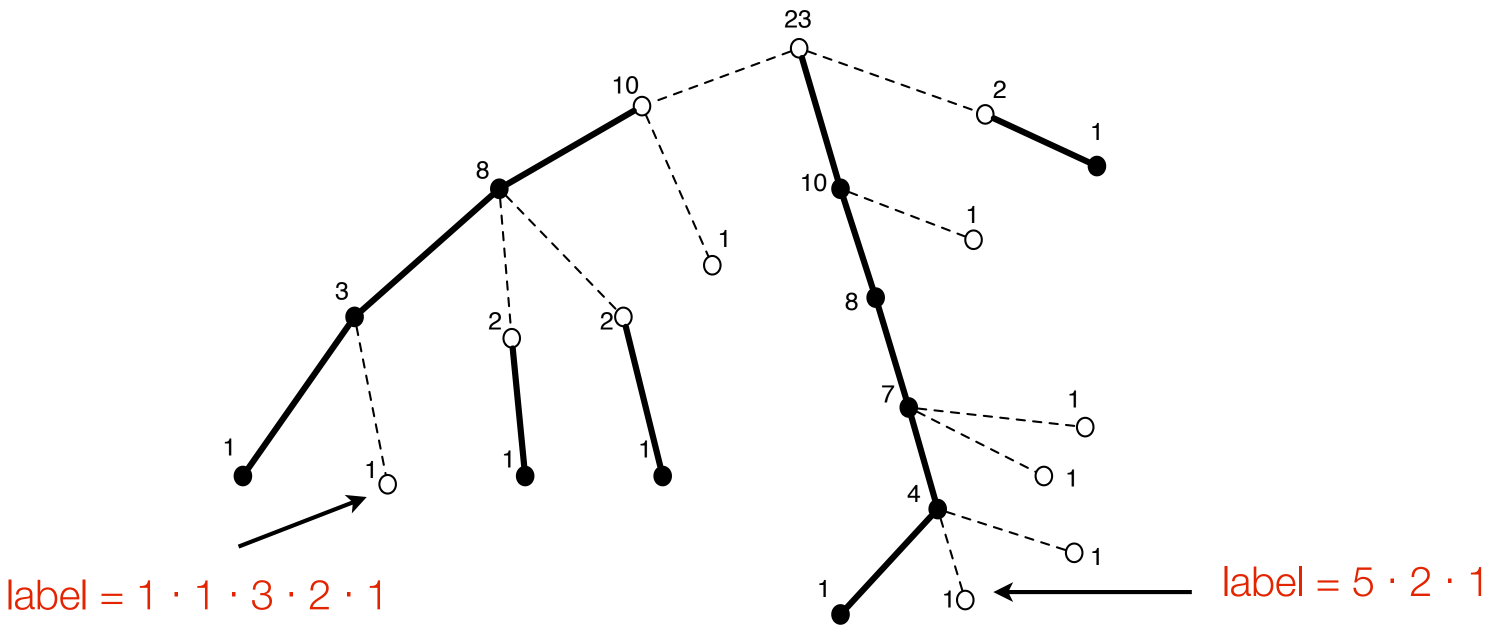
- Nearest common ancestors on a path.
    - Assign increasing IDs from root to leaf.
    - label(nca(v,w)) = min(ID(v), ID(w)).
- Analysis.
    - $\lceil \log n \rceil$ bit labels.

Nearest Common Ancestors



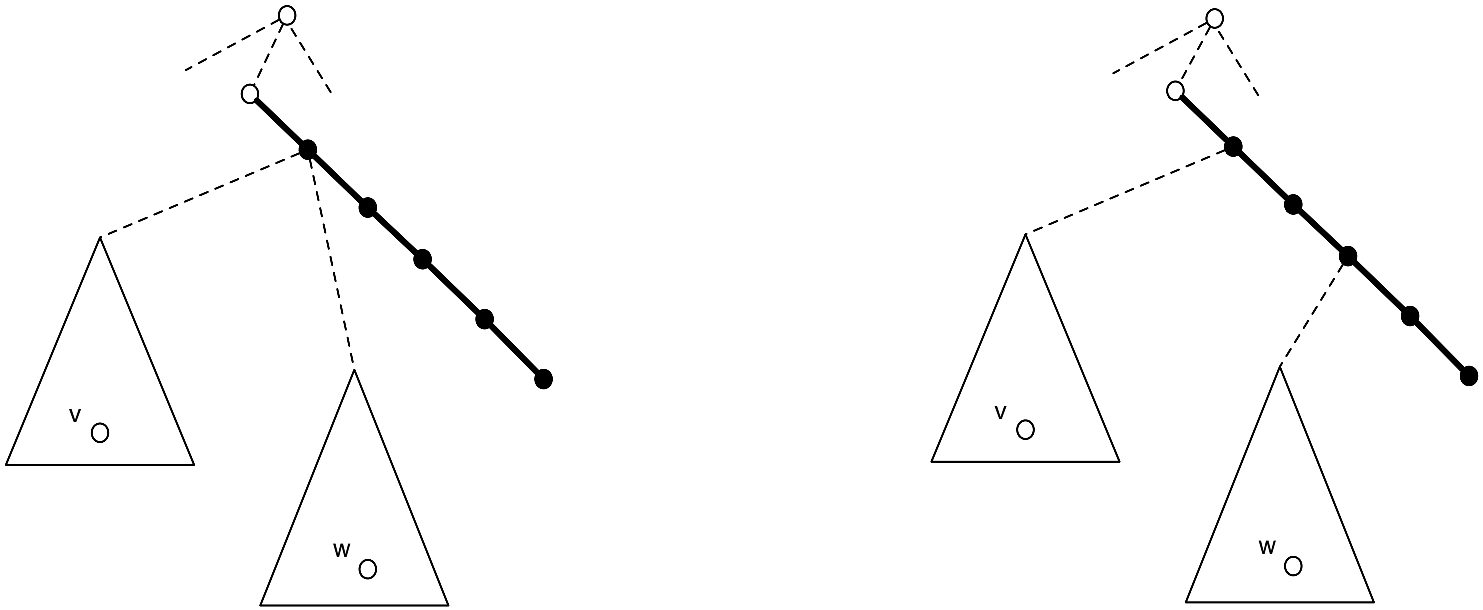
- Label construction.
  - For each heavy path $h_1 \cdots h_k$ from root to v store
    - HeavyID = deepest node on HP.
    - lightID = light child exit node in left-to-right order.
  - label(v) = heavyID(h_1) · lightID(h_1) · heavyID(h_2) · · · lightID(h_{k-1}) · heavyID(h_k)
- Analysis.
  - $2\lceil \log n \rceil$ bits per heavy path ⇒ $O(\log^2 n)$ bit label.

Nearest Common Ancestors



- Queries.
  - Compute longest common prefix L of IDs.
  - L contains either an even or odd number of IDs.

Nearest Common Ancestors

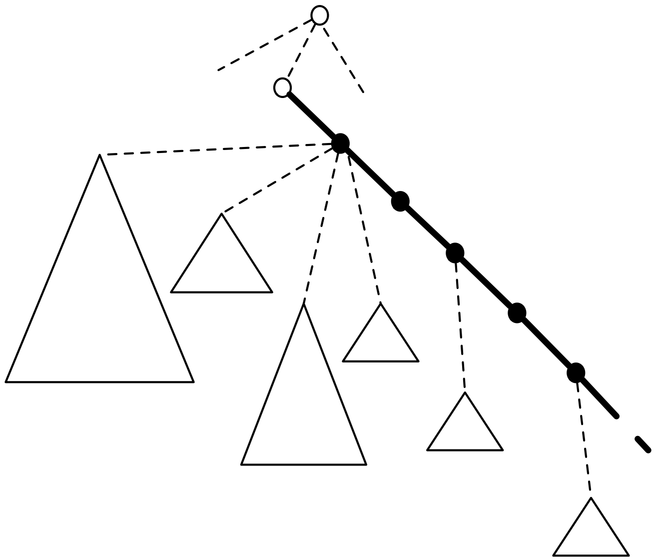


- Case 1. L contains odd number of IDs.
    - ⇒ last ID in L is heavyID
    - ⇒ v and w exit from same heavy path
    - ⇒ label(nca(v,w)) = L
- Case 2. L contains even number of IDs.
    - ⇒ last ID in L is lightID
    - ⇒ v and w enter same heavy path but leave at different exit points.
    - ⇒ label(nca(v,w)) = L · min(next ID in label(v), next ID in label(w))

Nearest Common Ancestors

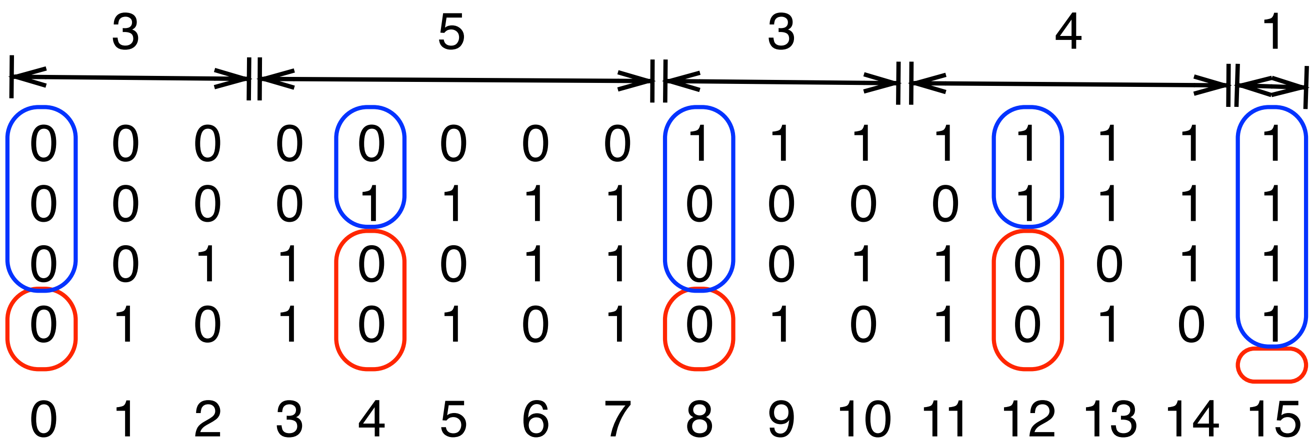
Labeling scheme	label length	query time
ID encoding	O(n log n)	
heavy path decomposition	$O(\log^2 n)$	

Nearest Common Ancestors



- Idea. Use variable length codes for IDs.
  - Small subtree ⇒ long IDs, large subtree ⇒ short IDs
  - LightID: need scheme to assign unique codes to distinct light children.
  - HeavyID: need scheme to assign unique codes to distinct nodes on heavy path that preserve order.

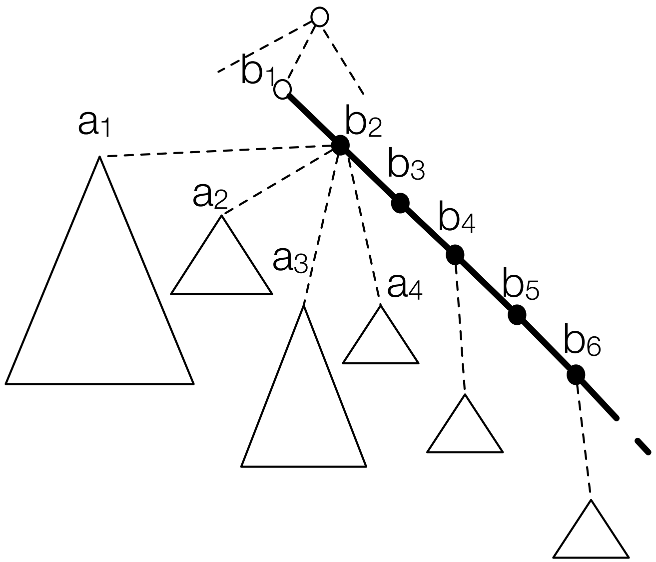
Nearest Common Ancestors



$Y = 3,5,3,4,1$ $s = 16$

- Alphabetic codes. Variable length code that preserves order.
  - Let $Y = y_1, y_2, \ldots, y_k$ be sequence of positive integers with $s = y_1 + y_2 + \cdots + y_k$.
  - Consider binary representation of $\{0, \ldots, s-1\}$.
  - Partition into intervals of sizes $y_1, y_2, \ldots, y_k$.
  - In interval i pick number z_i with $\lfloor \log y_i \rfloor$ least significant bits all 0.
  - Code for y_i is z_i with $\lfloor \log y_i \rfloor$ removed.
- Small $y_i \Rightarrow$ long code, large $y_i \Rightarrow$ short code.
- Preserves order by lexicographic order.

Nearest Common Ancestors



- Alphabetic codes and IDs.
  - Encode lightIDs and heavyIDs with alphabetic codes.
  - ⇒ O(log n) bits labels and O(1) query time.

Nearest Common Ancestors

Labeling scheme	label length	query time
ID encoding	O(n log n)	
heavy path decomposition	O(log^{2}n)	
alphabetic coding	O(log n)	O(1)