Nearest Common Ancestors

Philip Bille
Outline

• Distributed data structures
  • Parent labeling scheme
• Nearest common ancestor problem
• Nearest common ancestor labeling scheme
  • A first attempt
  • Heavy path decomposition
• Alphabetic Codes
Distributed Data Structures
Distributed Data Structures

- A *labeling scheme* for a graph G supporting query $q(v, w)$ for any pair of nodes $v$ and $w$:
  - Preprocessing: To each node $v$, assign *label* (just a bitstring).
  - Query: Given label($v$) and label($w$) compute query on $v$ and $w$. No other info!

- Goal:
  - Primary: Minimize the maximum size of labels.
  - Secondary: Fast queries, fast preprocessing, total space.
Parent Labeling Schemes for Trees
Let $T$ be a rooted tree with $n$ nodes.

Assign label to each node to support

- $lparent(label(v), label(w))$: return true iff $v$ is parent of $w$.

Goal: Minimize the maximum length of a label.

What can we store in labels to answer $lparent$?
• Solution: Assign unique ID to each node.

• \( \text{label}(v) = \text{ID}(v) \cdot \text{ID}(\text{parent}(v)) \)

• \( \text{parent(label}(v), \text{label}(w)) \): true iff \( \text{ID}(v) = \text{ID}(\text{parent}(w)) \)

• Use \( \lceil \log n \rceil \) bits to store ID => label length is 2 \( \lceil \log n \rceil \)
Summary

• Theorem: There is a parent labeling scheme for trees with maximum label length of $2 \lceil \log n \rceil$ bits.

• Other properties:
  • O(1) time queries
  • O(n) preprocessing and space
Applications

• Why study labeling schemes?
• Ultra compact distributed data structures:
  • Network routing, graph representation, XML search engines, ...
• Few memory accesses:
  • Limited memory access process query => Little I/O overhead.
• Graph theoretical connection:
  • Universal graphs.
Nearest Common Ancestor Problem
• $T$ is a rooted tree with $n$ nodes.
• The *ancestors* of node $v$ is the set of nodes from $v$ to the root (both inclusive)
• The *common ancestors* of nodes $v$ and $w$ are the ancestor of both $v$ and $w$.
• The *nearest common ancestor* of nodes $v$ and $w$ ($nca(v, w)$) is the common ancestor of greatest depth.
Nearest Common Ancestor Problem

- The nearest common ancestor problem: Preprocess a rooted tree $T$ with $n$ nodes to support
  - $\text{nca}(v,w)$: return the nearest common ancestor of $v$ and $w$.
  - A.k.a. the lowest common ancestor problem or the least common ancestor problem (lca)
  - How about most common ancestor (mca) as compromise between l and n?
Applications

- Key primitive in algorithms for: Weighted matching, minimum spanning trees, dominator trees, approximate string matching, dynamic planarity testing, network routing, computational geometry, computational biology, ....

- Main point: Trees are everywhere in science and nca is a very basic primitive for trees.

- Nice illustration of data structure design techniques.
We study algorithms for nca in labeling scheme context:

Assign label to each node supporting \textit{label-nca}

- \text{Inca}(\text{label}(v),\text{label}(w)) \colon \text{return label(nca}(v, w))

Goal:

- Primary: Maximum label length of $O(\log n)$ bits.
- Secondary:
  - Inca in $O(1)$ time.
  - $O(n)$ preprocessing and space
Overview

• Solution in three steps:
  • A first attempt
  • Heavy path decomposition
  • Alphabetic Codes
A First Attempt
A First Attempt

• Simplifying assumption: Ignore secondary goals (time to compute Inca, preprocessing, total space).

• Focus on maximum label length instead.
  • What information suffices to support Inca?
• Suppose we add unique ID to each node (as in parent labeling scheme)
• Which IDs could we store in a label to answer Inca queries?
Labels and Queries

- \( \text{label}(v) = \text{ID}(v_1) \cdot \text{ID}(v_2) \cdot \cdots \cdot \text{ID}(v_k) \) (\( r = v_1, \ldots, v_k = v \) is path of nodes from root to \( v \))

- \( \text{Inca}(\text{label}(v), \text{label}(w)) \): longest common prefix of IDs.

- Use \( \lceil \log n \rceil \) bits to store ID

- \( \Rightarrow \) label length is at most \( h \lceil \log n \rceil \), where \( h \) is height of tree.
• Theorem: There is a nearest common ancestor labeling scheme for trees with height $h$ with maximum label length of $h \lceil \log n \rceil$ bits.

• Nasty problem: $h$ might be $\Omega(n) \Rightarrow \Omega(n \log n)$ bit labels.

• How can we get better bounds?
Overview

• Improve to $O(\log n)$ bit labels in two steps:
  • Heavy-path decomposition.
  • Alphabetic codes.
Heavy Path Decomposition
Heavy Path Decomposition

• Technique to:
  • Balance trees (sort of).
  • Reduce problems on tree to problems on *paths* with logarithmic overhead.
Subtree Size

- For each node $v$ compute:
  - $\text{size}(v) = \#\text{of descendants (including } v\text{ itself)}$
Heavy and Light Nodes

- Classify nodes as heavy or light:
  - 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

- Classify edges as heavy or light:
  - Edge to heavy child is heavy and edge to light child is light.
  - If we remove light edges we partition T into heavy paths.
**Light Depth**

- \( \text{depth}(v) = \# \text{edges on path from } v \text{ to root} \)
- \( \text{lightdepth}(v) = \# \text{light edges on path from } v \text{ to root} \)
- \( \text{depth}(v) = n-1 \) for a worst case tree.
- What about \( \text{lightdepth}(v) \)?
Light Depth

- Each light edge on path from root decreases size by at least half.
- => For any node v, lightdepth(v) = O(log n).
- => Number of heavy paths on a root to leaf path is O(log n).
Heavy Path Decomposition and NCA Labeling Schemes

• How can we use heavy-path decomposition to improve our labeling scheme?

• Idea:
  • Find a good solution for paths.
  • Apply to the $O(\log n)$ heavy paths on root to leaf path.
NCA for Paths

- What is nca(v,w) on a path?
- How can we label nodes for Inca queries on a path?
NCA for Paths

- Assign increasing numbers to nodes from root to leaf.
- \( \text{lnca}(\text{label}(v), \text{label}(w)) = \min(\text{label}(v), \text{label}(w)) \)
Heavy Path Decomposition and NCA Labeling Schemes

• How can we use path idea to get to $O(\log^2 n)$ bit labels?

• (E.g. $O(\log n)$ bits per heavy path on the path from root to node)
Light and Heavy IDs

- For each heavy path $h_1 \cdots h_k$ from root to $v$ store:
  - HeavyID: The final node on heavy path (where we exit to light descendant or stop if final heavy path)
  - LightID: The light child we exit to among the other children. E.g. number in a left-to-right ordering.

- $\text{label}(v): \text{heavyID}(h_1) \cdot \text{lightID}(h_1) \cdot \text{heavyID}(h_2) \cdots \cdot \text{lightID}(h_{k-1}) \cdot \text{heavyID}(h_k)$
Label Length

• 2 \( \lceil \log n \rceil \) bits per heavy path (heavyID and lightID)

• \( \Rightarrow \) maximum label length is \( 2 \lceil \log n \rceil \cdot O(\log n) = O(\log^2 n) \)
Computing Queries

- $\text{Inca}(\text{label}(v), \text{label}(w))$: Almost as before
  - Compute longest common prefix $L$ of IDs.
  - 2 cases to consider: $L$ contains an even or odd number of IDs.
Case 1: L contains odd number of IDs

- Final ID in L is a heavy ID.
- v and w exit from same heavy path node to different light children.
- Inca(label(v), label(w)) = L (the longest common prefix of IDs)
Case 2: L contains even number of IDs

- Final ID in L is a lightID.
- v and w enter same heavy path but leave at different exit points on path.
- \( \text{Inca}(\text{label}(v), \text{label}(w)) = L \cdot \min(\text{next ID in label}(v), \text{next ID in label}(w)) \)
Summary

• Theorem: There is a nearest common ancestor labeling scheme for trees with maximum label length of $O(\log^2 n)$ bits.

• How do we get down to $O(\log n)$ bits?
Shaving a Log
What do we need of heavyIDs and LightIDs?

- Do we need binary ⌊log n⌋ bit numbers for heavyIDs and lightIDs?
  - LightID: Any method that assigns unique codes to distinct light siblings.
  - HeavyID: Any method that assigns unique codes to distinct nodes on heavy path and allows us to determine ordering from top to bottom.
What do we need of heavyIDs and LightIDs?

- How do we minimize the length of labels?

- Intuition:
  - Use variable length codes for heavyIDs and lightIDs to average out lengths.
  - Small subtree => long IDs and large subtree => short IDs.
What do we need of heavyIDs and LightIDs?

• Solution: Alphabetic codes
  • Variable length codes (bitstrings) preserving order (lexicographic order).
Alphabetic Codes
Lexicographic Ordering

• Let a and b be bitstrings.
• $a <_{\text{lex}} b$ if a is lexicographically smaller than b:
  • a is prefix of b or
  • the first bit where a and b differ is 0 in a and 1 in b.
• Example: 000 $<_{\text{lex}} 01$ $<_{\text{lex}} 100$ $<_{\text{lex}} 11$ $<_{\text{lex}} 1111$
Alphabetic Sequences

• Let \( Y = y_1, y_2, \ldots, y_k \) be a sequence of integers

• An *alphabetic sequence* for \( Y \) is a sequence \( B = b_1, b_2, \ldots, b_k \) such that
  • \( b_1 <_{\text{lex}} b_2 <_{\text{lex}} \cdots <_{\text{lex}} b_k \)

• Hence, \( B \) is an order-preserving coding (using \(<_{\text{lex}}\) of \( Y \).
Lemma:

- Let $Y = y_1, y_2, ..., y_k$ be a sequence of integers with $y_1 + y_2 + \cdots + y_k = s$
- There exists an alphabetic sequence $B = b_1, b_2, ..., b_k$ for $Y$ such that
  - For all $i$, $|b_i| \leq \lceil \log s \rceil - \lfloor \log y_i \rfloor = \log s - \log y_i + O(1)$

Example: $Y = 3, 5, 3, 4, 1$.

- $s = 3 + 5 + 3 + 4 + 1 = 16$, and $\log s = 4$
- $B = b_1, b_2, b_3, b_4, b_5 = 000, 01, 100, 11, 1111$ ($000_{\text{lex}} <_{\text{lex}} 01 <_{\text{lex}} 100 <_{\text{lex}} 11 <_{\text{lex}} 1111$)
  - $|b_1| = 4 - \lfloor \log 3 \rfloor = 4 - 1 = 3$
  - $|b_2| = 4 - \lfloor \log 5 \rfloor = 4 - 2 = 2$
  - $|b_3| = 4 - \lfloor \log 3 \rfloor = 4 - 1 = 3$
  - $|b_4| = 4 - \lfloor \log 4 \rfloor = 4 - 2 = 2$
  - $|b_5| = 4 - \lfloor \log 1 \rfloor = 4 - 0 = 4$
Construction

\[
\begin{array}{cccccccccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
\end{array}
\]

\[Y = 3, 5, 3, 4, 1 \quad s = 16\]

- Consider binary representation of integers \{0,..., s-1\}. 
Construction

\[ Y = 3, 5, 3, 4, 1 \quad \text{s} = 16 \]

• Consider binary representation of integers \{0, ..., s-1\}.

• Partition into consecutive intervals of sizes \( y_1, y_2, ..., y_k \)
Consider binary representation of integers \( \{0, \ldots, s-1\} \).

Partition into consecutive 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 (why does \( z_i \) exist?).
Construction

- Consider binary representation of integers \( \{0, \ldots, s-1\} \).
- Partition into consecutive intervals of sizes \( y_1, y_2, \ldots, y_k \).
- In interval \( i \) pick number \( z_i \) with \( \lceil \log y_i \rceil \) least significant bits all 0 (why does \( z_i \) exist?).
- \( b_i \) is \( z_i \) with \( \lfloor \log y_i \rfloor \) least significant bits removed.

\[
Y = 3, 5, 3, 4, 1 \quad s = 16
\]
Summary

- Lemma:
  - Let $Y = y_1, y_2, ..., y_k$ be a sequence of integers with $y_1 + y_2 + \cdots + y_k = s$
  - There exists an alphabetic sequence $B = b_1, b_2, ..., b_k$ for $Y$ such that
    - For all $i$, $|b_i| \leq \lceil \log s \rceil - \lfloor \log y_i \rfloor = \log s - \log y_i + O(1)$
  - $B$ is an alphabetic code for $Y$. 
An O(log n) Labeling Scheme
Overview

• Alphabetic encoding of lightIDs and heavyIDs.
• Handling variable length encoded lightIDS and heavyIDs in labels.
Alphabetic Encoding of LightIDs and HeavyIDs
Light Sizes

- Define $lsize(v) = \sum_{c \text{ is light child}} size(c)$
- For heavy path $h$, define $lsize(h) = \sum_{v \text{ is on } h} lsize(v)$
LightID Encoding

- LightIDs for children $c_1, c_2, ..., c_k$ encoding:
  - Alphabetic code $B = b_1, ..., b_k$ for size($c_1$), ..., size($c_k$).
  - We have $\text{size}(c_1) + \cdots + \text{size}(c_k) = \text{lsize}(v)$
  - $\implies |b_i| \leq \log (\text{lsize}(v)) - \log (\text{size}(c_i)) + O(1)$
LightID Encoding

• What is the total length $l$ of lightIDs in label($v$)?
  
  • $l \leq \log(lsize(v_2)) - \log(size(v_3)) + O(1) + \log(lsize(v_4)) - \log(size(v_5)) + O(1) + \cdots$
  
  • We have $size(v_3) > lsize(v_4)$
  
  • $\Rightarrow l \leq \log(lsize(v_2)) + O(1) - \log(size(v_5)) + O(1) + \cdots$
  
  • Telescoping sum of $O(\log n)$ terms $\Rightarrow l = O(\log n)$
HeavyID Encoding

- HeavyIDs for nodes $z_1, z_2, ..., z_k$ on heavy path $h$ encoding:
  - Alphabetic code $B = b_1, ..., b_k$ for $\text{lsize}(z_1), ..., \text{lsize}(z_k)$.
  - We have $\text{lsize}(z_1) + \cdots + \text{lsize}(z_k) = \text{lsize}(h)$
  - $|b_i| \leq \log \text{lsize}(h) - \log \text{lsize}(z_i) + O(1)$
What is the total length $l$ of heavyIDs in label($v$)?

- $l \leq \log(\text{lsize}(h_1)) - \log(\text{lsize}(v_2)) + O(1) + \log(\text{lsize}(h_2)) - \log(\text{lsize}(v_4)) + O(1) + \cdots$

- We have $\text{lsize}(v_2) > \text{lsize}(h_2)$

- $\Rightarrow l \leq \log(\text{lsize}(h_1)) + O(1) - \log(\text{lsize}(v_4)) + O(1) + \cdots$

- Telescoping sum of $O(\log n)$ terms $\Rightarrow l = O(\log n)$
Summary

• The total length of lightIDs in label(v) is $O(\log n)$
• The total length of heavyIDs in label(v) is $O(\log n)$
• The total length of IDs in labels is $O(\log n)$
• How do we distinguish between start and end of IDs in label(v)?
Handling Variable Length LightIDs and HeavyIDs
Variable Length Encodings

- Add additional *indicator label* containing 1 at the end (or start) of each ID in label.
- => Unique decoding of IDs in label.
- Doubles length of label.
- => maximum length label remains $O(\log n)$. 
The Labeling Scheme

• Theorem: There is a nearest common ancestor labeling scheme for trees with maximum label length of $O(\log n)$ bits.

• Also:
  • We can compute lnca in $O(1)$ time.
  • We can compute all labels in $O(n)$ time.
  • Total space is $O(n)$ ($n \cdot \log n$ bits).
Summary

- Distributed data structures
  - Parent labeling scheme
- Nearest common ancestor problem
- Nearest common ancestor labeling scheme
  - A first attempt
  - Heavy path decomposition
- Alphabetic Codes
References


• Scribe notes from MIT