Level Ancestor

- Level Ancestor Problem
- Trade-offs

---

Level Ancestor

- Level ancestor problem. Preprocess rooted tree $T$ with $n$ nodes to support
  - $\text{LA}(v,k)$: return the $k$th ancestor of node $v$.  

![Diagram](image)

$\text{LA}(v,5) = u$

---

Applications.
- Basic primitive for navigating trees (any hierarchical data).
- Illustration of wealth of techniques for trees.
  - Path decompositions.
  - Tree decomposition.
  - Tree encoding and tabulation.
Level Ancestor

- Level Ancestor Problem
- Trade-offs

Goal. Linear space and constant time.
Solution in 7 steps (!).
- No data structure. Very slow, little space.
- Direct shortcuts. Very fast, lot of space.
- ...
- Ladder decomposition + jump pointers + top-bottom decomposition. Very fast, little space.

Solution 1: No Data Structure

Data structure. Store tree T (using pointers).
LA(v,k): Walk up.
Time. O(n)
Space. O(n)
Solution 2: Direct Shortcuts

- Data structure. Store each root-to-leaf in array.
- LA(v,k): Jump up.
- Time. O(1)
- Space. O(n^2)

Solution 3: Jump Pointers

- Data structure. For each node v, store pointers to ancestors at distance 1,2,4, ...
- LA(v,k): Jump to most distant ancestor no further away than k. Repeat.
- Time. O(log n)
- Space. O(n log n)

Solution 4: Long Path Decomposition

- Long path decomposition.
  - Find root-to-leaf path p of maximum length.
  - Recursively apply to subtrees hanging of p.
- Lemma. Any root-to-leaf path passes through at most O(n^{1/2}) long paths.
- Longest paths partition T \implies total length of all longest paths is < n

Solution 4: Long Path Decomposition

- Data structure. Store each long path in array.
- LA(v,k): Jump to kth ancestor or root of long path. Repeat.
- Time. O(n^{1/2})
- Space. O(n)
Solution 5: Ladder Decomposition

- Ladder decomposition.
  - Compute long path decomposition.
  - Double each long path.
- Lemma. Any root-to-leaf path passes through at most $O(\log n)$ ladders.
- Total length of ladders is $< 2n$.

Solution 6: Ladder Decomposition + Jump Pointers

- Data structure. Ladder decomposition + Jump pointers.
- $LA(v,k)$:
  - Jump to most distant ancestor not further away than $k$ using jump pointer.
  - Jump to $k$th ancestor using ladder.
- Time. $O(1)$
- Space. $O(n) + O(n \log n) = O(n \log n)$

Solution 7: Top-Bottom Decomposition

- Jump nodes. Maximal deep nodes with $\geq 1/4 \log n$ descendants.
- Top tree. Jump nodes + ancestors.
- Bottom trees. Below top tree.
- Size of each bottom tree $< 1/4 \log n$.
- Number of jump nodes is at most $O(n/\log n)$. 
Solution 7: Top-Bottom Decomposition

- Data structure for top.
  - Ladder decomposition + Jump pointers for jump nodes.
  - For each internal node pointer to some jump node below.
- LA(v,k) in top:
  - Follow pointer to jump node below v.
  - Jump pointer + ladder solution.
- Time. $O(1)$
- Space. $O(n) + (n/\log n \cdot \log n) = O(n)$

Solution 7: Top-Bottom Decomposition

- Tree encoding. Encode each bottom tree $B$ using balanced parentheses representation.
  - $< 2 \cdot 1/4 \log n = 1/2 \log n$ bits.
- Integer encoding. Encode inputs $v$ and $k$ to LA
  - $< 2 \cdot \log(1/4\log n) < 2 \log \log n$ bits.
- LA encoding. Concatenate into $\text{code}(B, v, k)$
  - $\Rightarrow |\text{code}(B, v, k)| < 1/2 \log n + 2 \log \log n$ bits.

Solution 7: Top-Bottom Decomposition

- Data structure for bottom.
  - Build table $A$ s.t. $A[\text{code}(B, v, k)] = \text{LA}(v, k)$ in bottom tree $B$.
- LA(v,k) in bottom: Lookup in $A$.
- Time. $O(1)$
- Space. $2^{\text{code}} < 2^{1/2 \log n + 2 \log \log n} = n^{1/2 \log 2} n = o(n)$.
- Combine bottom and top data structures $\Rightarrow O(n)$ space and $O(1)$ query time.

Solution 7: Top-Bottom Decomposition

- Theorem. We can solve the level ancestor problem in linear space and constant query time.
Level Ancestor

- Level Ancestor Problem
- Trade-offs