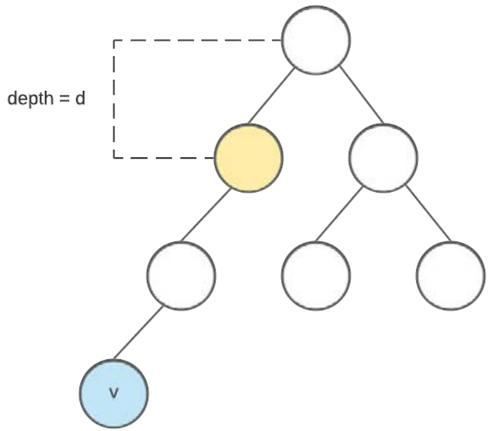
An Analysis of the Level Ancestor Problem

Carolyne Holmes & Matthew Mabrey

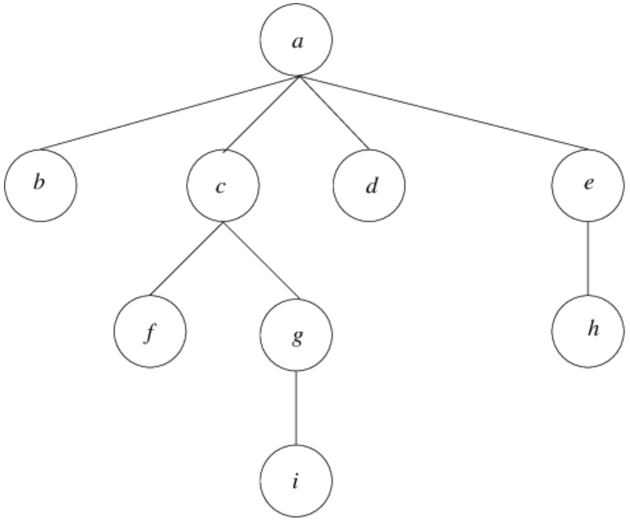
The Level Ancestor Problem

- In a tree T with n nodes, answer a query of the form LA(v, d)
  - Query aims to find the ancestor of the node *v* that is at a depth of *d* from the root
  - This can be done is O(n) time by tracing the path from the target node to the root until reaching depth *d*, but it is impractical for larger trees
- Preprocessing for the algorithms aims to store the tree in data structures so that queries can be completed more efficiently



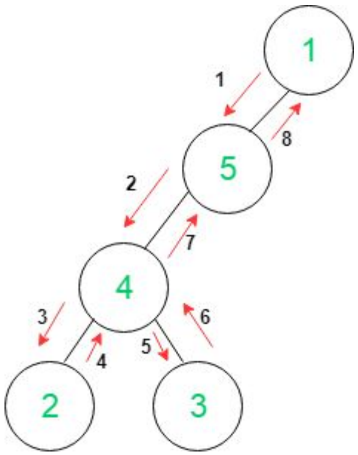
Applications

- Level ancestor implementations can be used to aid in many other algorithms
  - Least Common Ancestor
    - Used in string processing, computational biology, and complex distributed systems, among many other applications
  - Space-Efficient Ordinal Trees
    - XML document representation for XPath queries
  - Range-Aggregate Queries on trees

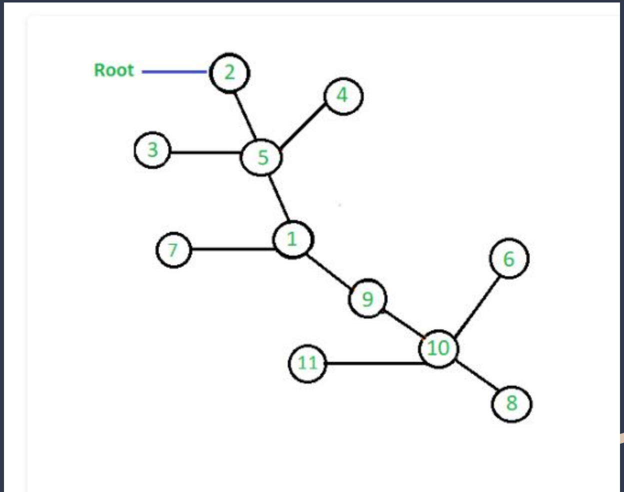


Our Project

- We aimed to code (in C) the algorithms outlined in different papers and compare the experimental time for tree preprocessing and querying
- We examined eight algorithms for static binary trees
- Random Tree Generation:
  - Trees were represented in Euler format that follows a Depth-First Search preorder
    - Binary Representation:
      - 1 represents moving down in the tree
      - 0 represents moving upwards in the tree
      - Example: 11101000
        - Euler Representation= 154243451
  - Generated via split-tree method
    - Creates trees where both the average node depth and tree depth are logarithmically bound
    - Allows for easy construction of Euler representation and skewed trees

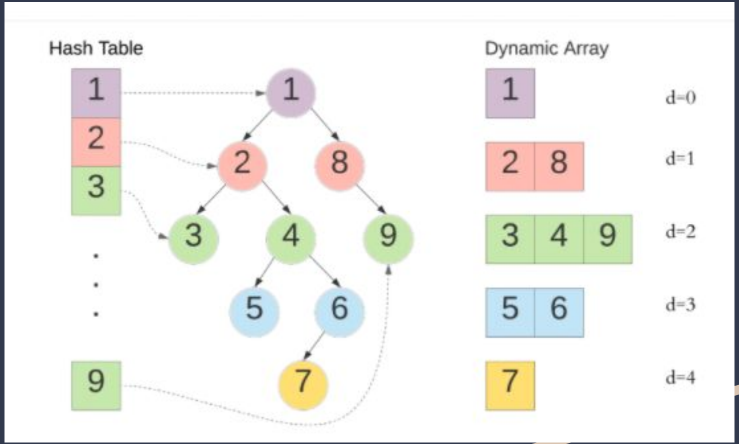


The Algorithms



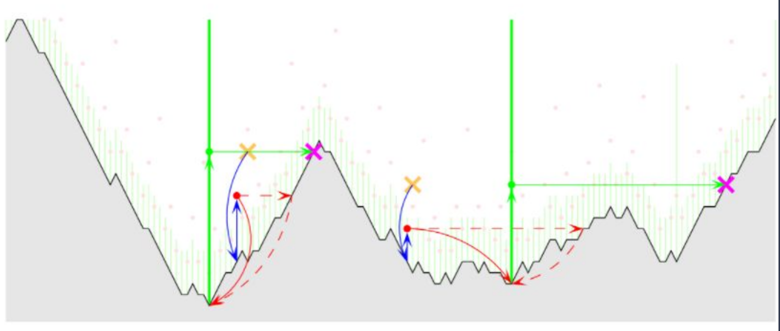
- Table Algorithm:
  - Preprocessing = $O(n^2)$, Query = O(1)
  - Uses a 2D-array to store the list of all ancestors of each node derived by a traversal in DFS order
- Jump-Pointer Algorithm:
  - Preprocessing = O(nlogn), Query = O(logn)
  - Uses pointers from each node to its ancestors, allowing for the distance to the ancestor at depth *d* to be traversed in jumps of at least half the remaining distance to the ancestor
- Ladder Algorithm:
  - Preprocessing = O(n), Query = O(logn)
  - Perform a longest path decomposition on the tree and extends the paths to reach the root so that a node at height *h* can be queried for ancestors of at least height 2*h*
- Jump Ladder:
  - Preprocessing = O(nlogn), Query = O(1)
  - Combines the Jump-Pointer and Ladder Algorithms
  - First jumps up the tree, then uses the ladder to move up towards the ancestor node

The Algorithms



- Macro-Micro Algorithm:
  - Preprocessing = O(n), Query = O(1)
  - Uses the other four algorithms
  - Nodes are divided into macro and micro subtrees based on their weight
  - Macro nodes have use of the jump-pointer method
  - Micro trees use table algorithm
- Menghani-Matani (Google-Facebook) Algorithm:
  - Preprocessing = O(n), Query = O(logn)
  - Perform a pre-order traversal of the tree where each node is given a integer tag starting at 1 for the root
  - Each node, in increasing tag value, is stored in a specific dynamic array depending on its depth
  - Find the largest node label less than the label of node *v* in the array for depth *d* and return the pointer for that label

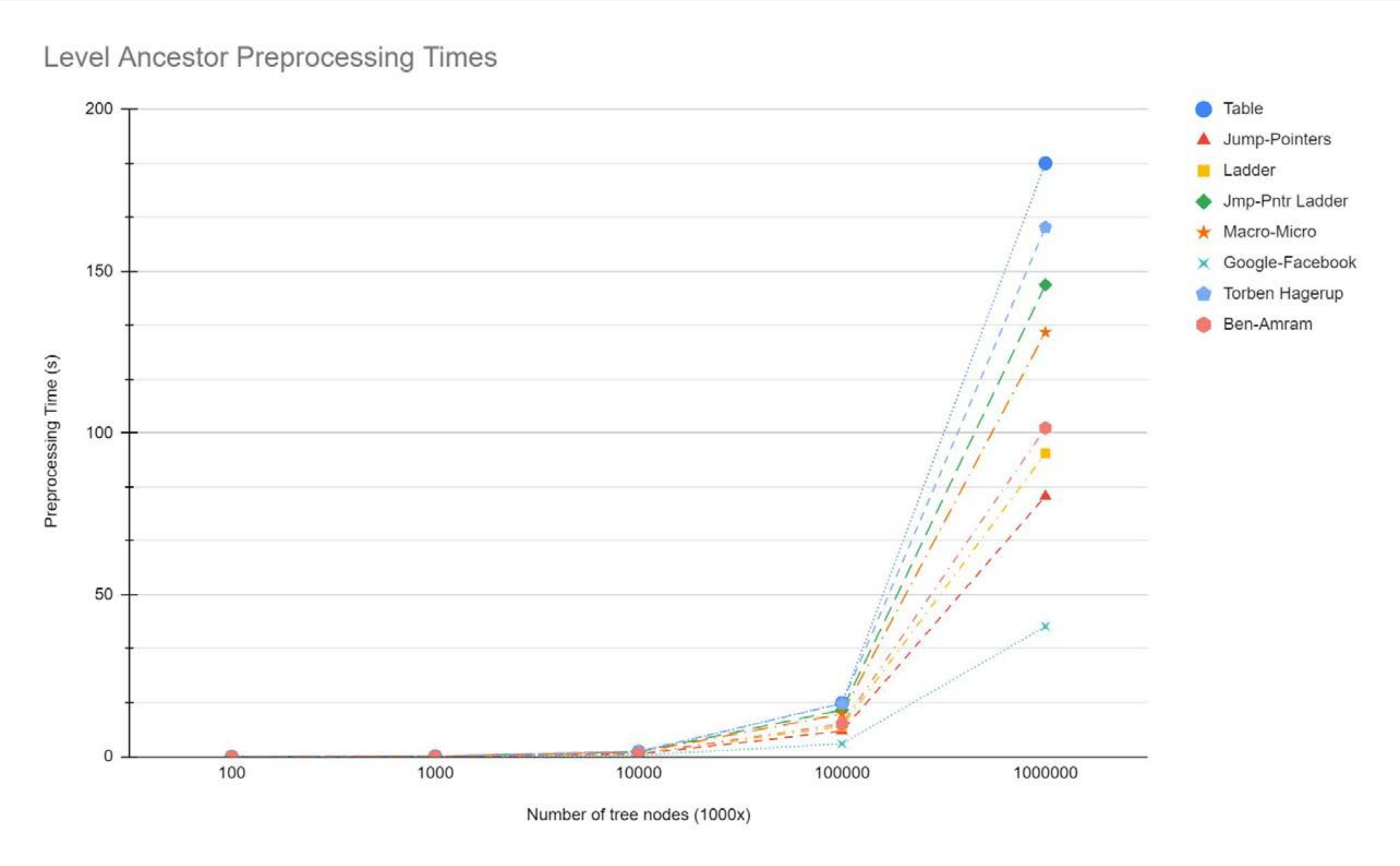
The Algorithms

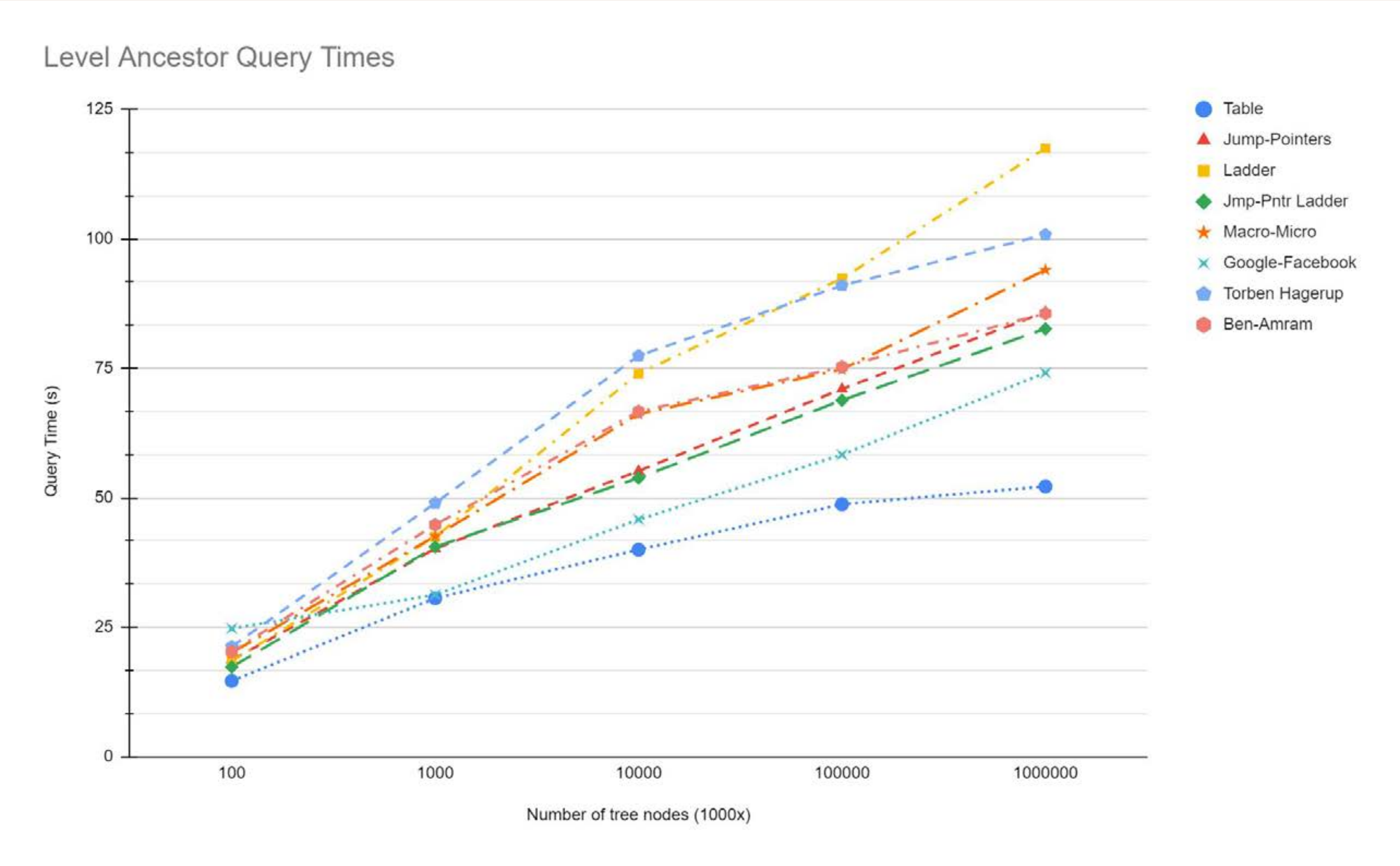


- Torben Hagerup:
  - Preprocessing = O(n), Query = O(1)
  - Uses the find-smaller method to create a 2D structure of peaks and valleys
  - Finds the deepest valley nearest to the node *v* and then uses the ladder method until reaching the height of the given depth *d*
  - Then the node to the right of this height in the 2D structure is selected
- Ben-Amran:
  - Preprocessing = O(nlogn), Query = O(n)
  - Uses the find-smaller methods which takes an array *A*, an index *x*, and an integer *y*
  - Finds an index $u > x$ such that $A_u \leq y$

Results

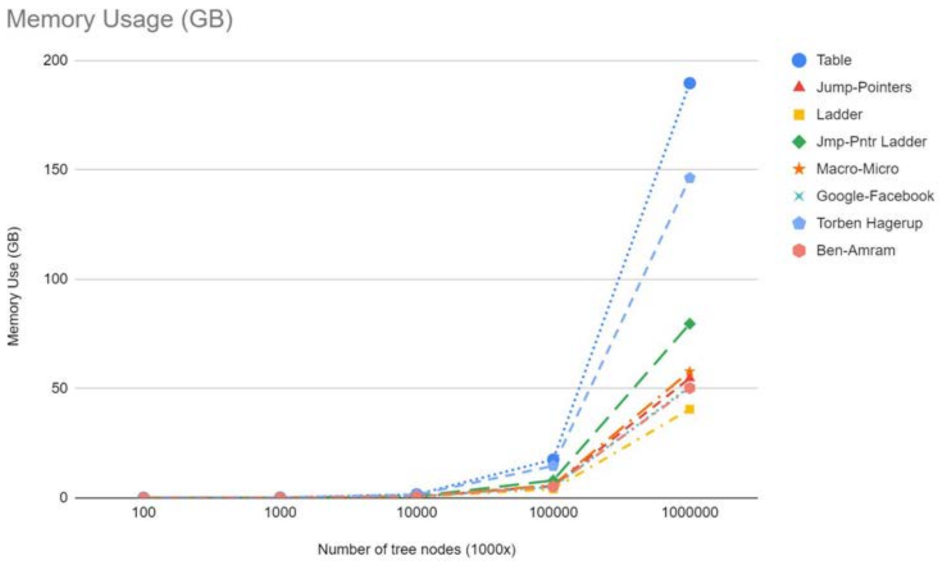
- Multiple trials were run with varying numbers of nodes from 100,000 to 1 billion and 100 million queries per tree
- Tests were run on the TCNJ cluster as submitted job scripts to an AMD EPYC2 processor with 512 GB of memory
- Preprocessing:
  - Fastest - Google-Facebook Algorithm
  - Slowest - Table Algorithm
- Querying:
  - Fastest - Table Algorithm
  - Slowest - Torben Hagerup & Ladder Algorithms


Level Ancestor Preprocessing Times
Preprocessing Time (s)
200
150
100
50
0
100
1000
10000
100000
1000000
Number of tree nodes (1000x)
Table
Jump-Pointers
Ladder
Jmp-Pntr Ladder
Macro-Micro
Google-Facebook
Torben Hagerup
Ben-Amram


Level Ancestor Query Times
Query Time (s)
125
100
75
50
25
0
100
1000
10000
100000
1000000
Number of tree nodes (1000x)
Table
Jump-Pointers
Ladder
Jmp-Pntr Ladder
Macro-Micro
Google-Facebook
Torben Hagerup
Ben-Amram

Results

- ➢ In addition to time taken, we also examined memory usage for each algorithm
  - ○ As expected, the Table Algorithm uses up the most memory space



Future Research

- Examine ways to improve upon the existing algorithms
  - Could incorporate more assembly code that speeds up processing time
- Incorporate dynamic algorithms that allow for adding leaves to trees
  - Alstrup & Holm Algorithm
    - Implements an improved version of the macro-micro algorithm
  - Dietz Algorithm
  - Adjust static algorithms to allow for dynamic trees

References

Stephen Alstrup and Jacob Holm. Improved algorithms for finding level ancestors in dynamic trees. In Automata, Languages and Programming, 27th International Colloquium, ICALP 2000, number 1853 in LNCS, pages 73–84. Springer Verlag, 2000.

Amir M. Ben-Amram. The euler path to static level-ancestors. CoRR, abs/0909.1030, 2009.

Michael A. Bender and Mart́ ın Farach-Colton. The level ancestor problem simplified. Theor. Comput. Sci., 321(1):5–12, June 2004.

Paul Dietz. Finding level-ancestors in dynamic trees. In Frank Dehne, Jrg-Rdiger Sack, and Nicola Santoro, editors, Algorithms and Data Structures, volume 519 of Lecture Notes in Computer Science, pages 32–40. Springer Berlin / Heidelberg, 1991. 10.1007/BFb0028247.

Omer Berkman and Uzi Vishkin. Finding level-ancestors in trees. Journal of Computer and System Sciences, 48(2):214 – 230, 1994.

Gaurav Menghani and Dhruv Matani. A simple solution to the level ancestor problem. 2019. Retrieved from https://arxiv.org/abs/1903.01387

Torben Hagerup. Still simpler static level ancestors. CoRR, abs/2005.11188, 2020.