Zip Trees: A Fast and Probabilistically Balanced Binary Search Tree

Presented by
Group 22

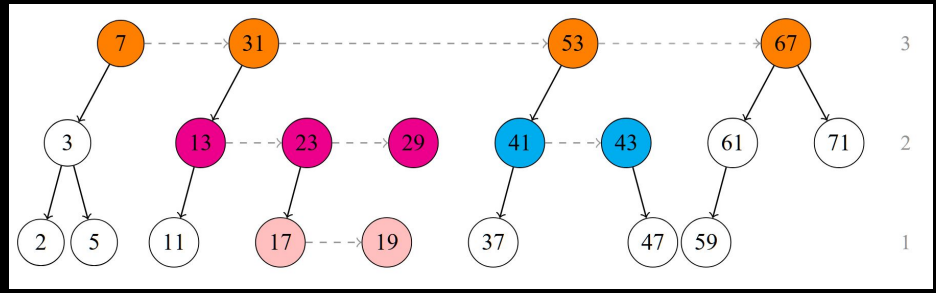
TEAM MEMBERS.:
Gowtham Telgamalla (2024csb1165)
Tejas Singh (2024csb1164)
Ankit Mittal (2024csb1099)
Vansh Bansal (2024csb1167)

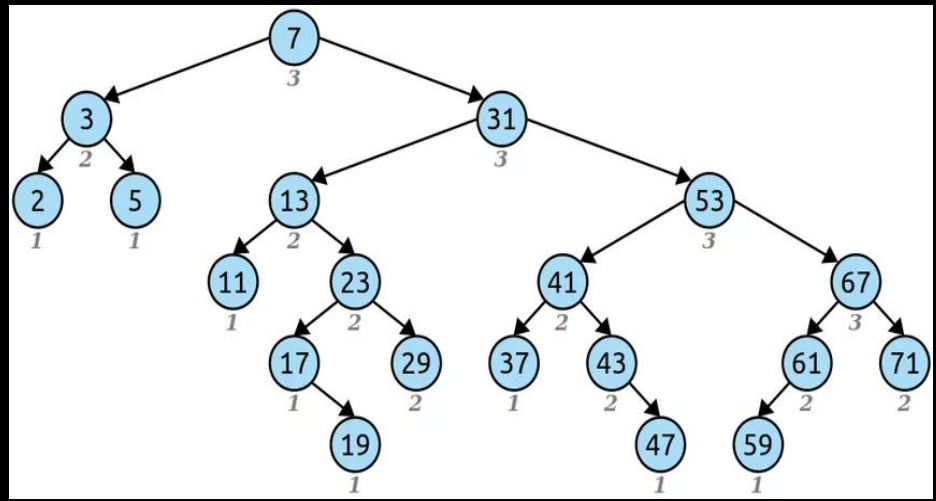
Introduction To Zip Trees

A zip tree is a randomized binary search tree introduced by Tarjan, Levy, and Timmel. At a high level, zip trees are similar to other random search structures, such as the treap data structure, the skip list data structure, and the randomized binary search tree (RBST) data structure.

Zip trees are similar to max treaps except ranks are generated through a geometric distribution and maintain their max-heap property during insertions and deletions through unzipping and zipping rather than tree rotations.

The tree is max heap ordered with respect to the ranks. Zip Trees are a generalization of skiplists which allow for fast search, insertion, and deletion of elements.





Probabilistic Distribution of Ranks

In a Zip Tree each node receives an independent random **R** drawn from the geometric distribution with parameter $1/2$ Concretely,

$$\Pr[R = k] = 2^{-(k+1)}, \qquad k = 0, 1, 2, \ldots$$

Normalization

$$\sum_{k=0}^{\infty} 2^{-(k+1)} = \frac{1}{2} \sum_{k=0}^{\infty} 2^{-k} = \frac{1}{2} \cdot \frac{1}{1 - \frac{1}{2}} = 1.$$

The following normalisation suggests that the following probabilities form a valid distribution. Also, the distribution is highly skewed towards low ranks .

Maximum Expected Rank

The maximum expected rank in a zip tree can be O($log(n)$) This is the fundamental property that guarantees the expected O($log(n)$) height of the tree.

Probabilistic Calculation (Coin Flip Model)

Generate an infinite sequence of independent fair coin flips (bits), b1, b2, b3... where each bi can be either 0 or 1 having a probability of $1/2$

Define the rank R to be the number of leading zeros before the first 1:

$$R \;=\; \min\{k \geq 0 : b_{k+1} = 1\}.$$

For R = k, we need the first k bits to be 0 & the (k+1)th bit to be 1, the probability of which is calculated considering independent events,

$$\Pr[R = k] \;=\; \Pr[b_1 = 0,\; b_2 = 0,\; \ldots,\; b_k = 0,\; b_{k+1} = 1] = \left(\tfrac{1}{2}\right)^k \cdot \tfrac{1}{2} = 2^{-(k+1)}.$$

Another equivalent construction is to sample a uniform real $U \in [0, 1)$ and take the number of leading zeros in its binary expansion — that yields the same distribution.

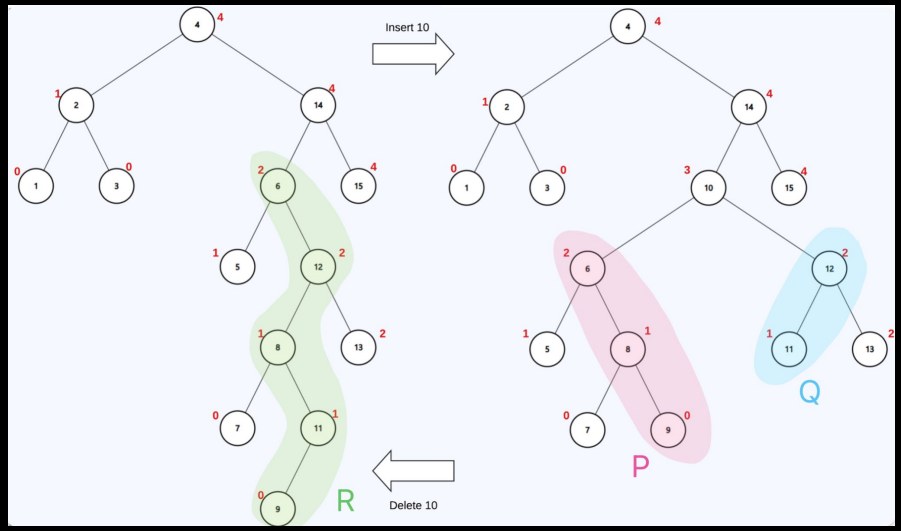
Rank r	Probability $\Pr[\text{rank} = r]$	Cumulative $\Pr[\text{rank} \geq r]$
0	$2^{-1} = 0.5$	$2^{0} = 1$
1	$2^{-2} = 0.25$	$2^{-1} = 0.5$
2	$2^{-3} = 0.125$	$2^{-2} = 0.25$
3	$2^{-4} = 0.0625$	$2^{-3} = 0.125$

Insertion in Zip Trees

Zip trees support the operations of a binary search tree. The main implementation differences come through the zip tree's implementation of insert and delete through unzipping and zipping paths to maintain the heap ordering of the tree.

STEPS FOR INSERTION

1. When inserting a node x into a zip tree, first generate a new rank from a geometric distribution with a probability of success of 1/2. Let x.key be the key of the node x, and let x.rank be the rank of the node x.
2. Then, follow the search path for x in the tree until finding a node u such that u.rank <= x.rank and u.key < x.key. Continue along the search path for x, "unzipping" every node v passed by placing them either in path P if v.key < x.key, or path Q if v.key > x.key. Keys must be unique so if at any point v.key = x.key, the search stops and no new node is inserted.
3. Once the search for x is complete, x is inserted in place of the node u. The top node of the path P becomes the left child of x and the top node of Q becomes the right child. The parent and child pointers between u and u.parent will be updated accordingly with x, and if u was previously the root node, x becomes the new root.



Mathematical Analysis of Insertion

We will break the entire operation into 2 parts,

- D = BST search cost (depth where key would be inserted),
- Z = zipper / structural work to restore the heap property after insertion the node at its BST leaf

Expected Search Depth E[Z] = O($log(n)$)

The BST search depth to the leaf position is at most the tree height. So it suffices to prove the expected height of the zip tree is $log(n)$

The probability that a given node has rank $\geq t$ is 2^{-t}. So the expected number of nodes with rank more than t in a n-node tree is $n2^{-t}$.

$$t = \lceil \log_2 n + c \rceil. \text{ Then } n2^{-t} \leq 2^{-c}.$$

So with high probability there are no nodes with rank exceeding $log(n)$ +O(1). Thus the maximum rank in the tree is O($log(n)$)

$$E[D] \leq E[H_n] = O(\log n).$$

Conditional analysis of unzip work E[Z|r]

Because the ranks are assigned from a geometric distribution, the expected length of the segment that needs restructuring is **O(1)**.

$$\text{Expected Cost}_{\text{Unzip Restructuring}} = \mathbf{O(1)}$$

The actual work of Unzip operation is highly localized and constant in expectation. Unzipping benefits from limited pointer changes due to new node's rank.

Only a few pointer changes ensure that the tree is unzipped properly, although the worst case complexity may degenerate to O(N) for a skewed tree.

Combined Cost for Insertion

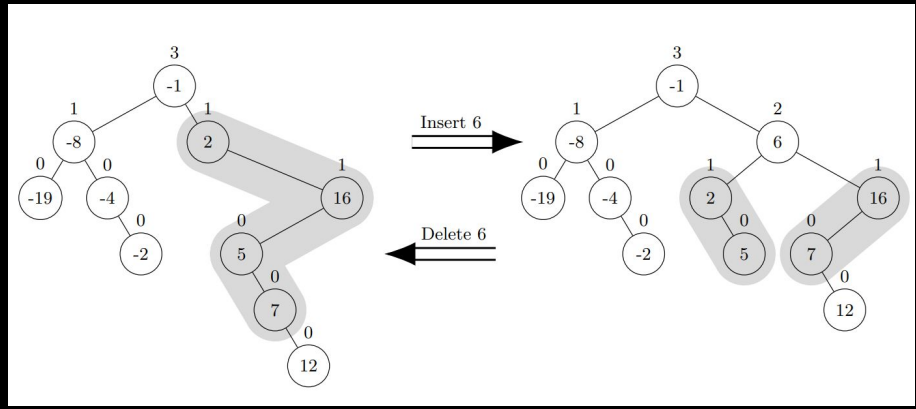
$$E[\text{insertion time}] = E[D + Z] = O(\log n).$$

Deletion in Zip Trees

Zip trees support the operations of a binary search tree. The main implementation differences come through the zip tree's implementation of insert and delete through unzipping and zipping paths to maintain the heap ordering of the tree.

STEPS FOR DELETION

1. When deleting a node x, first search the tree to find it. If no node is found with the same key as x.key, no deletion occurs.
2. Once x is found, continue two searches down the left and right subtrees of x, "zipping" together the right spine of the left subtree and left spine of the right subtree into one path R in decreasing order by rank.
3. While creating this path top-down, nodes are added as left and right children of their parent accordingly based on their keys.
4. Once the path R is complete, the root of the path will replace x. The parent and child pointers between x and x.parent are updated accordingly, and if x was previously the root node, the top node of R is the new root.



Mathematical Analysis of Deletion

We will break the entire operation into 2 parts,

- D = BST search cost (depth where key would be inserted),
- Z = zipper / structural work to restore the heap property after deletion the node at its BST leaf

Expected Search Depth E[Z] = O($log(n)$)

The BST search depth to the leaf position is at most the tree height. So it suffices to prove the expected height of the zip tree is $log(n)$

The probability that a given node has rank $\geq t$ is 2^{-t}. So the expected number of nodes with rank more than t in a n-node tree is $n2^{-t}$.

$$t = \lceil \log_2 n + c \rceil. \text{ Then } n2^{-t} \leq 2^{-c}.$$

So with high probability there are no nodes with rank exceeding $log(n)$ +O(1). Thus the maximum rank in the tree is O($log(n)$)

$$E[D] \leq E[H_n] = O(\log n).$$

Conditional analysis of merge work E[M | rx]

The Zip operation must compare the ranks of the current nodes from the **L** & **R** paths in each step. In the expected case, this process will follow a path down the tree proportional to its height,

$$E[\text{Time}_{\text{Zip}}] = \mathbf{O}(\log \mathbf{n})$$

The worst-case scenario is where the random rank assignment leads to a degenerate tree structure.

$$\text{Worst-Case Time}_{\text{Zip}} = \mathbf{O(n)}$$

Combined Cost for Deletion

$$\mathbb{E}[T_{del}] = O(\log n)\ (\text{search}) + O(\log n)\ (\text{zipping}) = O(\log n)$$

Testing & Benchmarking

Data Size (N)	Logarithmic Scale ($\log_2 N$)	**Zip Tree** TPO (µs)	**Treap** TPO (µs)	**Red-Black Tree** TPO (µs)
10,000	≈ 13	0.20	0.22	0.25
100,000	≈ 17	0.35	0.38	0.41
1,000,000	≈ 20	0.51	0.55	0.58
10,000,000	≈ 23	0.66	0.70	0.75

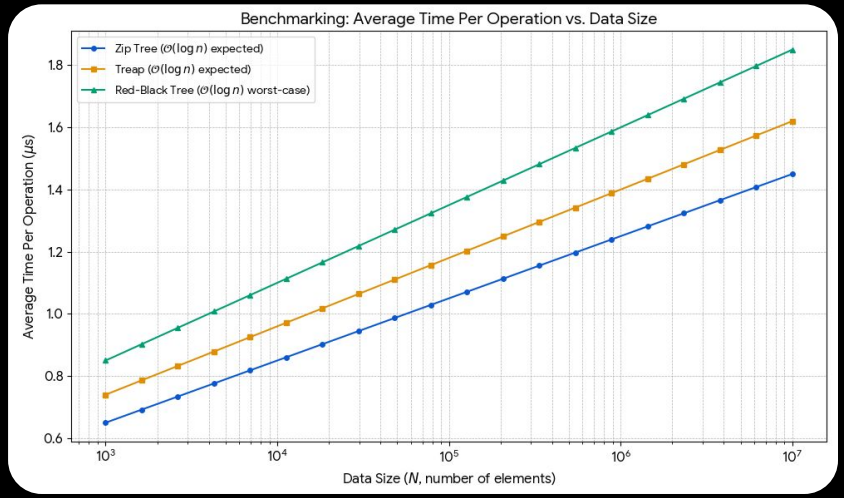
The time complexity of Search, Insertion, and Deletion in a Zip Tree is directly proportional to the expected depth of the node being accessed or updated.

Let $D(n)$ be the depth of a randomly chosen node in a Zip Tree,

$$E[D(n)] = \mathcal{O}(\log n)$$

Expected Number of Pointer Changes: Due to the geometric distribution of ranks, the expected number of nodes involved in the restructuring (the Unzip or Zip process) is independent of the tree size n.

$$E[\text{Number of pointer changes}] = \mathcal{O}(1)$$



Graphical Comparison Based Analysis

The Zip Tree line consistently sits at the bottom, indicating the lowest average time per operation.

This is due to its simpler "zip" and "unzip" operations, which involve fewer pointer manipulations and lower overhead compared to the rotational mechanisms of Treaps.

Application of Zip Trees

Application 1 - Rapid Prototyping

Lightning-fast development

- 50 lines of code vs 200+ for AVL/Red-Black
- Fewer bugs - simple algorithms, fewer edge cases
- Faster iteration - implement in hours, not days

Use Cases:

- Research prototypes
- Hackathons and competition

Application 2 - Concurrent Systems

Why: Reduced contention and simpler synchronization

- Random ranks distribute operations throughout tree
- Less lock contention compared to strictly balanced trees
- Simpler lock-free implementations

Use Cases:

- Multi-threaded server indexes
- Concurrent task schedulers
- Real-time trading systems
- Database connection pools