Quake Heaps: A Simple Alternative to Fibonacci Heaps

Abhishek Singh Rawat (T23191)
Guide: Varunkumar Jayapaul

26th January 2024

Overview

Introduction

Heap is a useful data structure when repeatedly removal needs to be done on the basis of priority.

There are various data structures that could perform such operations, such as Fibonacci Heaps, Relaxed Heaps, V-Heaps, Quake Heaps, and many more. But the described data structure is easiest to understand among all existing methods.

Quake heaps use a very simple idea to ensure balance: first, be lazy during updates and rebuild when structures get bad.

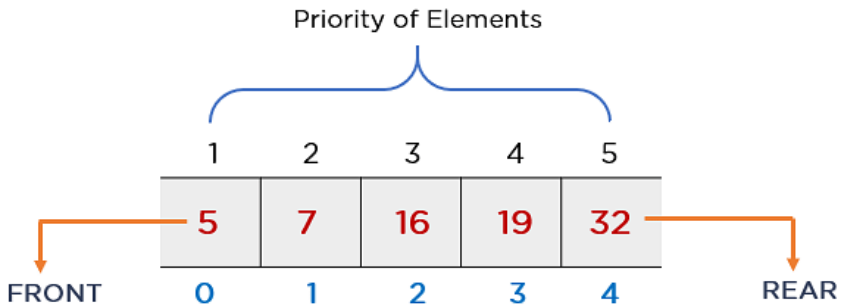

Figure: Priority Queue

Uses of Quake Heaps

Quake Heaps, a data structure with theoretical performance similar to Fibonacci heaps, efficiently supports `insert` operations in $O(1)$, `decrease-key` operations in $O(1)$ amortized time, and `delete-min` operations in $O(\log n)$ amortized time.

This makes Quake Heaps advantageous in scenarios where rapid execution of these operations is crucial, and its simplicity enhances practical applicability and suitability for educational purposes.

Structure of Quake Heaps

Internal Node:

`Height, Parent, Left Child, Right Child, and Leaf Reference.`

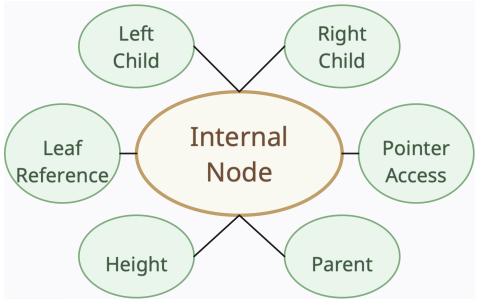

Figure: Internal Node Illustration

Leaf Node: `Key, Value, and Highest Reference.`

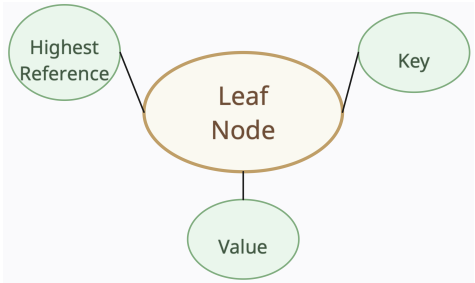

Figure: Leaf Node Illustration

Insert Operation

- Create a leaf node and update the key with the given value.
- Create an internal node and update the height with 0, the leaf reference with leaf, and the highest reference with the internal node.
- Store the internal node in the forest and perform update min.
- Update pointer access in the internal node with the stored address of the list.

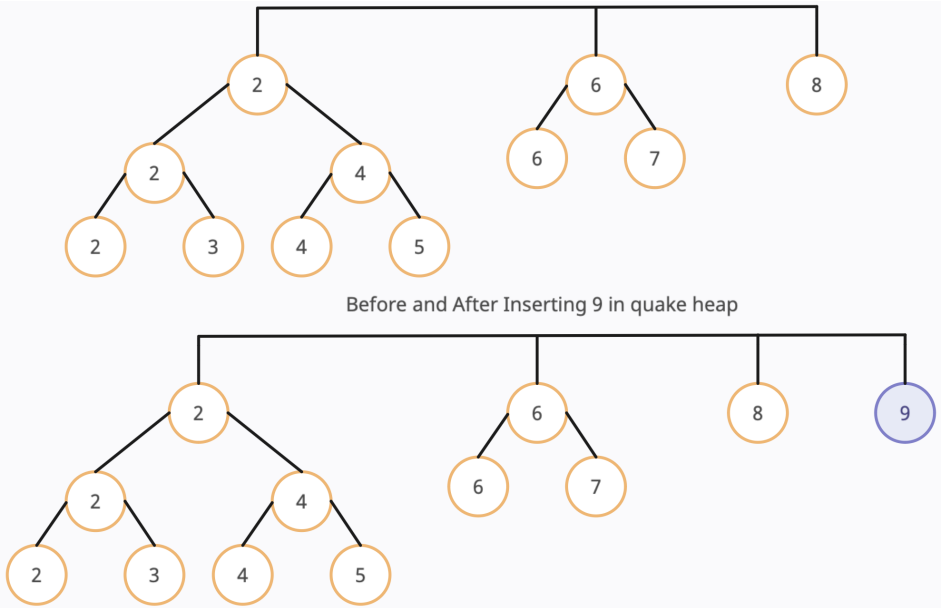

Figure: Insert Operation Illustration

Link Operation

- Create an internal node and update its left and right with node1 and node2 with the parent as null.
- Update the leaf reference after checking with the smallest leaf node.
- Update the parent of both nodes with the new node, and update the height.
- Insert the node in the forest and update the pointer access.
- Perform the update min operation.

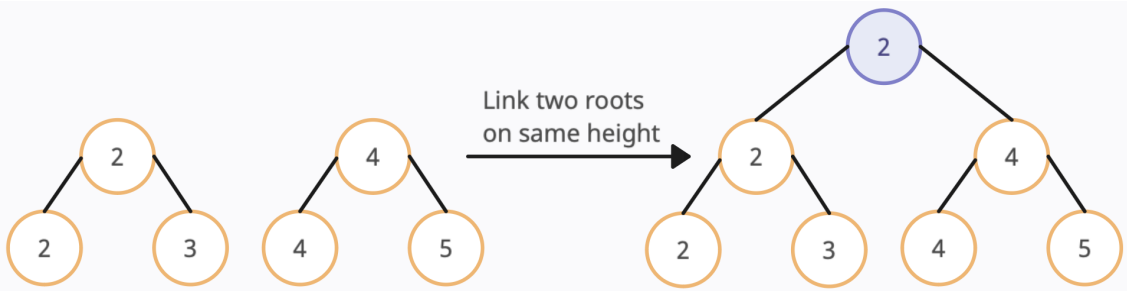

Figure: Link Operation Illustration

Cut Operation

- Get the leaf reference for the node.
- Update the child parent with a null that does not reference the leaf node, and update the forest.
- Update the node with the child that has the same leaf reference.

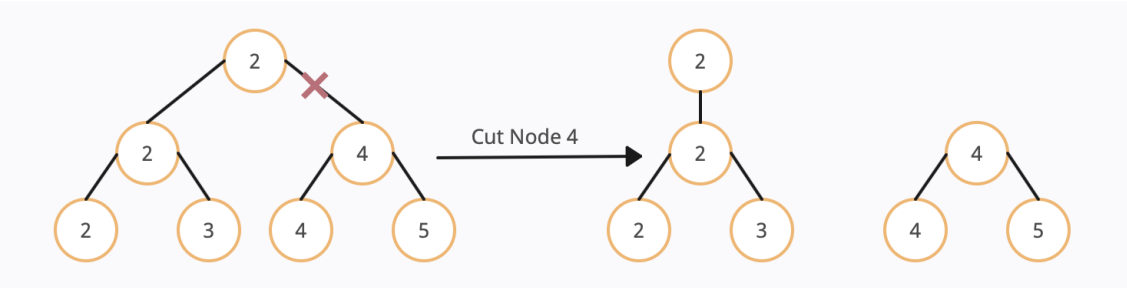

Figure: Cut Operation Illustration

Decrease-Key Operation

- Get the highest reference of the leaf node.
- If the parent is null, then update the key with the new key value.
- Otherwise, update the child of the parent pointer to null and update the forest.
- Perform the update min operation.

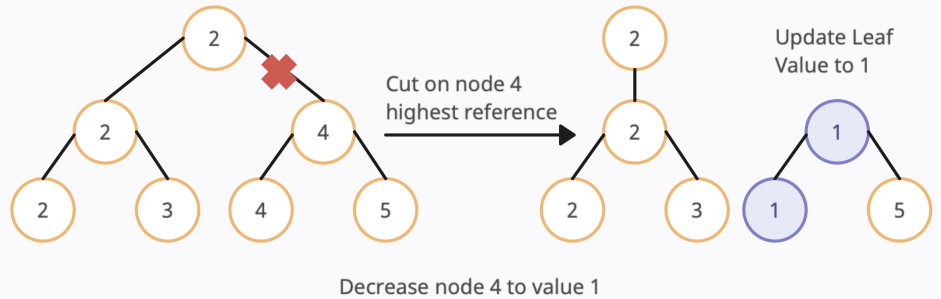

Figure: Decrease-Key Operation Illustration

Delete-Min Operation

- Perform `Cut Operation` with the help of the minimum pointer.
- Perform `Link Operation` of nodes with the same height.
- Check the off-balance condition if it exists height$_i$ < height$_{i-1}$ × α.
- If an off-balance condition exists, perform a cut on all the roots from height i to maximum.
- Perform the update min operation..

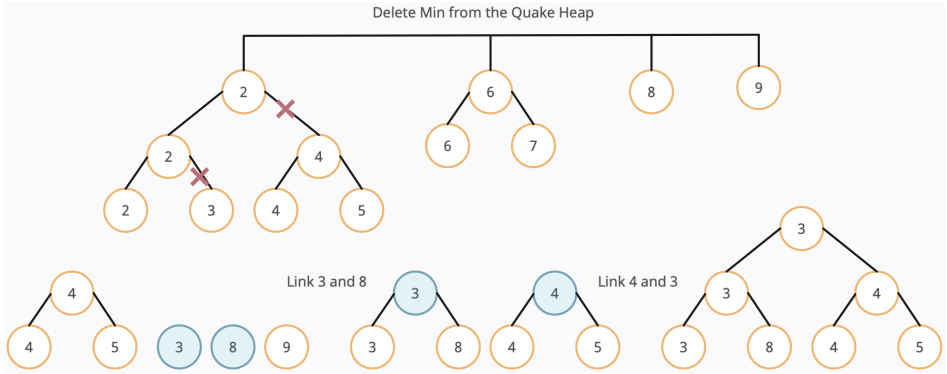

Figure: Delete-Min Operation Illustration

Results

After calculating asymptotic runtime analysis obtained from our exploration of quake heaps. The results below shared shows the comparison of amortized time complexity of various types of heaps.

Operation	**Regular**	**Binomial**	**Fibonacci**	**Quake**
Find Min	O(1)	O(log n)	O(1)	O(1)
Delete Min	O(log n)	O(log n)	O(log n)	O(log n)
Insert	O(1)	O(1)	O(1)	O(1)
Decrease-Key	O(log n)	O(log n)	O(1)	O(1)
Merge	O(n)	O(log n)	O(1)	O(1)

Recommendation

Building the foundation for the current work, here are some extensions that we can improve in the current working of Quake Heaps. We can look up to make the Delete Min operation with an unamortized complexity of O(log n).

Conclusion

Quake Heaps provide a simple alternative to Fibonacci Heaps with competitive theoretical performance. Their efficiency in handling key operations makes them suitable for various applications.