Quake Heaps: A Simple Alternative to Fibonacci Heaps

Exploratory Work Technical Project Report

to be submitted by

Abhishek Singh Rawat

T23191

for the partial fulfillment of the degree

of

MASTERS OF TECHNOLOGY IN COMPUTER SCIENCE ENGINEERING

SCHOOL OF COMPUTER AND ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY MANDI

KAMAND-175075, INDIA

JANUARY, 2024

Contents

Chapter 1

Introduction

1.1 Motivation

Efficient management of data is very important in every computer application. Heap data structure is important in applications where priority-based scheduling plays a pivotal role. Take the example of an operating system as software that has to manage all the running processes on the basis of priority. It has to take care that the user must get a good experience while using it, which will happen if user processes are taken as priorities. All the operations that are performed by the user should be executed in an ordered manner and also ensure that the system's performance is optimized. Consideration of priority Fibonacci Heaps is popular among theoretically optimized operations like insertion, deletion, and priority update. It does consist of complex operations, which usually cause difficulty in understanding in the educational arena. Students usually find it difficult to comprehend its workings. This work [1] is done to find an alternative that will strike a balance between performance and educational understanding.

1.2 State-of-the-Art

The count of those already invented is various, several with the same features as the Fibonacci heaps [2] proposed by Fredman and Tarjan's that set the standard for insertion and decrease key operations in amortize time of $O(1)$ and delete min operations in amortize time of $O(\log n)$. Still, understanding its workings is complex, which makes it complicated for learners. There are also various alternatives, which consist of V-Heaps [3], Relaxed Heaps [4], Rank Pairing Heaps [5], Violation Heaps [6], and Thin-Fat Heaps [7]. Each of them has unique features and benefits. Some of them can lower amortized time, while others could ensure balancing the structure.

1.3 Relevance of Quake Heaps

Among all heaps, Quake Heaps stands out as a distinct methodological choice. They provide the same insertion and deletion operations in amortized time of $O(1)$ and delete min operations in amortized time of $O(\log n)$. While replicating the same theoretical work as Fibonacci Heaps, it also has ease of understanding for educational purposes. The theoretic understanding is easy, which makes it an appealing option for teaching purposes. Quake Heaps solves the problem of providing the same theoretical value while easing the implementation. Their simplicity is enhanced by not using cascading cuts, which are found in Fibonacci heaps.

Transition

As we progress in the chapters, we will see in-depth construction, functions, and theoretical aspects to understand this unique heap structure and its uses.

Chapter 2

Problem Definition, Model, Approach

2.1 Problem Definition

Heaps are important structures in computer science, a major application where retrieval of data is important on the basis of priority. Priority queues are used in places where such features are required. However, those do not provide insertion and decrease key operations in $O(1)$. While Fibonacci heaps are able to perform all operations with a given theoretical runtime complexity, their complex structure makes them difficult to understand for learning and creates difficulty for teaching purposes. Evolving a heap that provides the same theoretical aspect with a simpler approach while proving the inherent features of the heaps.

2.2 Model

Here we have used a tournament tree approach that is simple and efficient. The inherent feature of tournament trees came in use to order the element based on value. As tournament trees work on the basis of this data structure, providing a clear understanding will help in understanding the data structure. A tournament tree with a simple structure allows for the formation of order elements based on rank, which will be required as the base of quake heaps.

2.3 Approach

The approach taken here is to make strategic use of tournament trees by having a lazy update. The operations of lazy update and insertion are used in such a manner that they can provide the same theoretical performance as Fibonacci heaps. This will also take care of the balance between simplicity and performance by performing updates only when necessary. Which will lead to the ability to perform operations in constant time.

2.4 Structure of Quake Heaps

Quake Heaps take an innovative approach to heap architectures, offering a simpler yet more effective alternative to sophisticated systems. The structure consists of two types of nodes: internal nodes and leaf nodes.

Internal Node

The internal node in Quake Heaps encapsulates the following attributes:

- **Height:** Denotes the height of the internal node.
- **Parent:** Points to the parent node.
- **Left Child:** References the left child node.
- **Right Child:** References the right child node.
- **Leaf Reference:** Points to the associated leaf node.

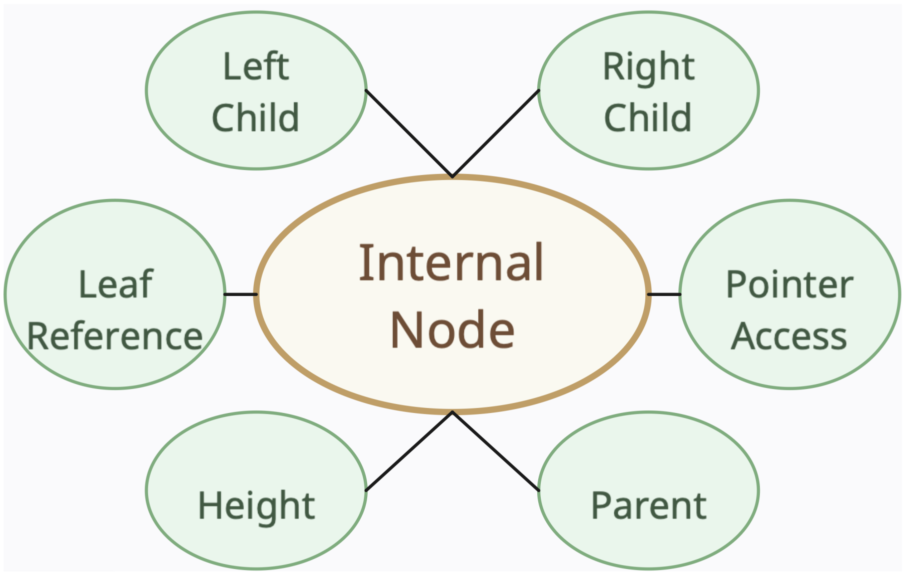

Fig. 2.1: Internal Node Illustration

Leaf Node

The leaf node in Quake Heaps consists of the following fields:

- **Key:** Represents the key associated with the node.
- **Value:** Stores the value corresponding to the key.
- **Highest Reference:** Points to the highest internal node referencing this leaf.

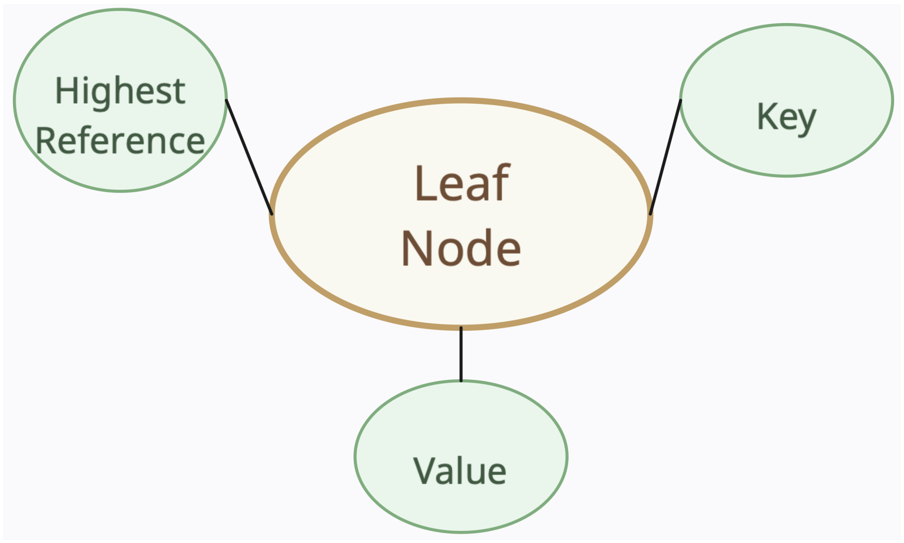

Fig. 2.2: Leaf Node Illustration

2.5 Quake Heaps Operations

Insert Operation

1. Create a leaf node and update the key with the given value.
2. Create an internal node with height 0, referencing the created leaf node.
3. Store the internal node in the forest and perform an Update-Min operation.
4. Update pointer access in the internal node with the stored address of the list.

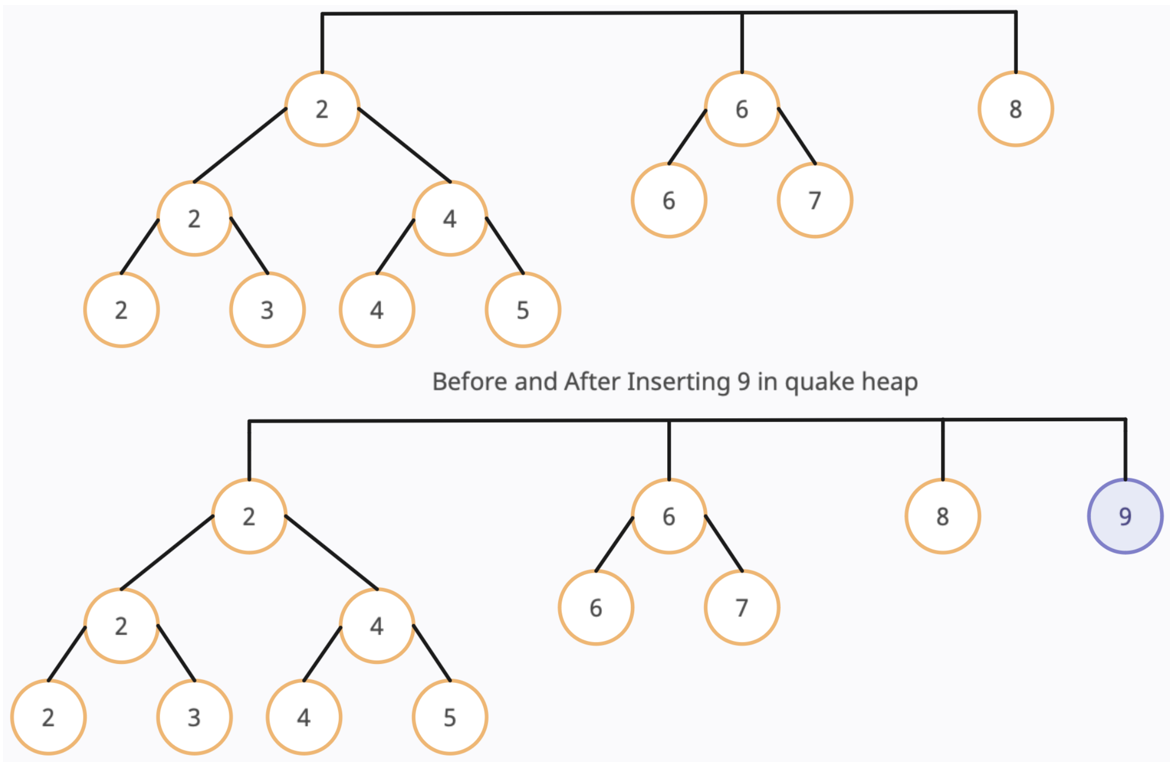

Fig. 2.3: Insert Operation Illustration

Link Operation

1. Create an internal node and update its left and right with node1 and node2 with the parent as null.
2. Update the leaf reference after checking with the smallest leaf node.

3. Update the parent of both nodes with the new node, and update the height.

4. Insert the node in the forest and update the pointer access.

5. Perform the update min operation.

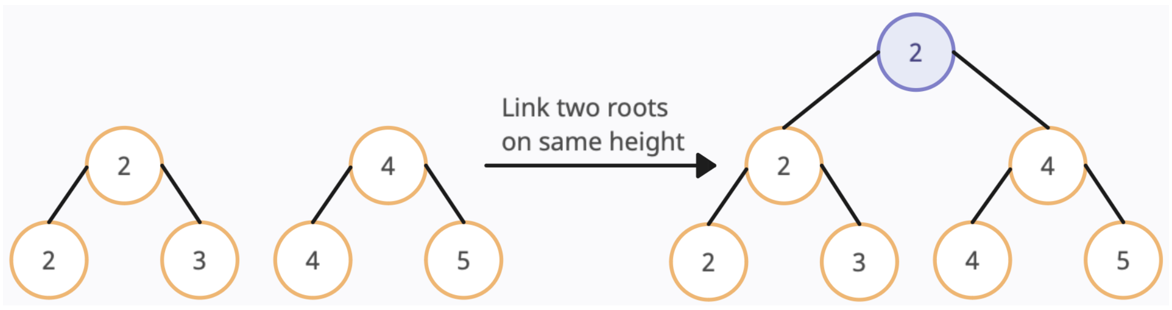

Fig. 2.4: Link Operation Illustration

Cut Operation

1. Get the leaf reference for the node.

2. Update the child parent with null who does not have same leaf reference as of the current node.

3. Update the node with the child that has the same leaf reference.

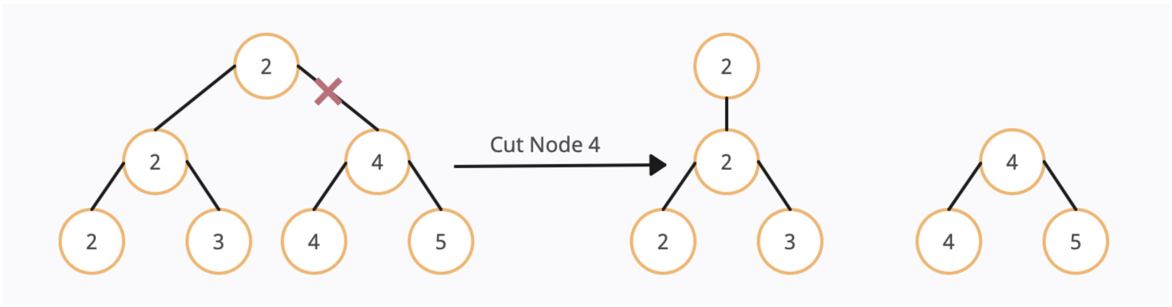

Fig. 2.5: Cut Operation Illustration

Decrease-Key Operation

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

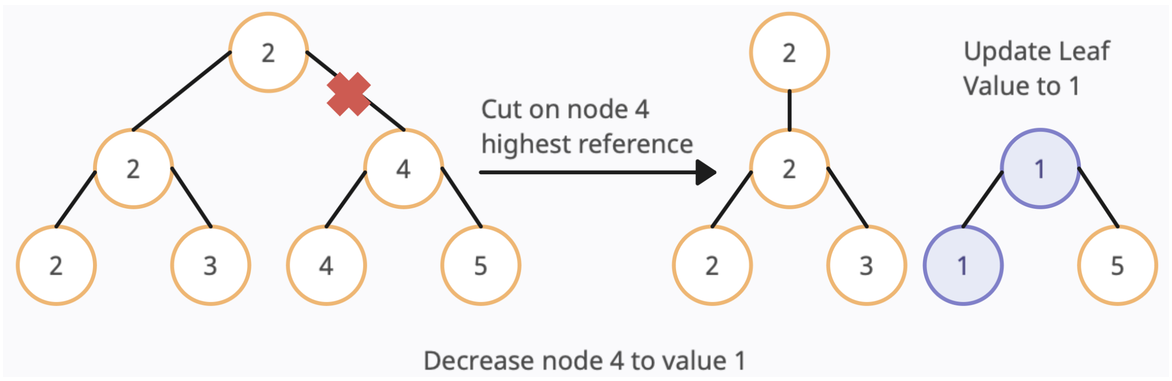

Fig. 2.6: Decrease-Key Operation Illustration

Delete-Min Operation

1. Perform the cut operation with the help of the minimum pointer.
2. Perform the link operation of nodes with the same height.
3. Check the off-balance condition if it exists ($\text{height}_i < \text{height}_{i-1} \times \alpha$).
4. If an off-balance condition exists, perform a cut on all the roots from height i to maximum.
5. Perform the update min operation.

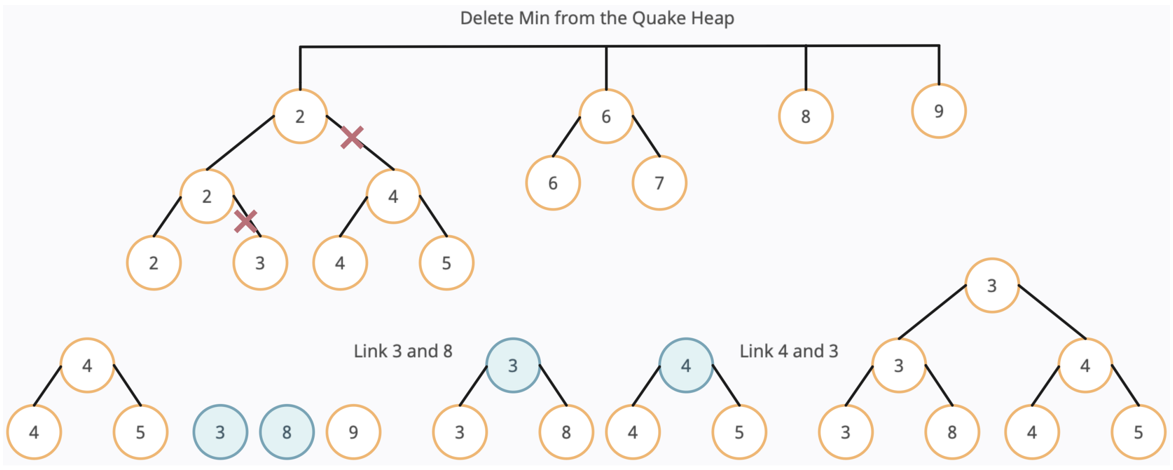

Fig. 2.7: Delete-Min Operation Illustration

The integration of Quake Heaps into the exploration of heap structures provides a promising alternative, addressing challenges posed by more complex solutions. This chapter establishes the foundation for the subsequent detailed examination of the mechanics and theoretical foundations of Quake Heaps.

Chapter 3

Future Work

3.1 Extensions

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), which can be done if we can include some constant-time operation of linking at the time of insert and decreasing, which will not affect their runtime but will improve the runtime operation of Delete Min.

3.2 Conclusion

Concluding with a comprehensive summary of the findings that put emphasis on the use of quake heaps as a simple and effective alternative heap data structure. We reflect the contribution done through this work as a base point for future endeavors and further improvement of Quake Heaps so that it could have better potential advancements.

Bibliography

[1] T. M. Chan, “Quake heaps: A simple alternative to fibonacci heaps,” *Springer Berlin Heidelberg*, pp. 27–32, 2013.

[2] M. Fredman and R. Tarjan, “Fibonacci heaps and their uses in improved network optimization algorithms,” *Journal of the ACM*, vol. 34, pp. 596–615, 1987.

[3] G. Peterson, “A balanced tree scheme for meldable heaps with updates,” Georgia Institute of Technology, Technical Report GIT-ICS-87-23, 1987.

[4] J. Driscoll, H. Gabow, R. Shrairman, and R. Tarjan, “Relaxed heaps: An alternative to fibonacci heaps with applications to parallel computation,” *Communications of the ACM*, vol. 31, pp. 1343–1354, 1988.

[5] B. Haeupler, S. Sen, and R. E. Tarjan, “Rank-pairing heaps,” *SIAM Journal on Computing*, vol. 40, pp. 1463–1485, 2011.

[6] A. Elmasry, “The violation heap: A relaxed fibonacci-like heap,” *Discrete Mathematics, Algorithms and Applications*, vol. 2, pp. 493–504, 2010.

[7] H. Kaplan and R. Tarjan, “Thin heaps, thick heaps,” *ACM Transactions on Algorithms*, vol. 4, no. 1, p. 3, 2008.

Report

by Abhishek SINGH RAWAT

Submission date: 29-Jan-2024 01:29PM (UTC+0530)
Submission ID: 2281049688
File name: Exploratory_Report.pdf (1.43M)
Word count: 1849
Character count: 9791

Quake Heaps: A Simple Alternative to Fibonacci Heaps

Exploratory Work Technical Project Report

to be submitted by

Abhishek Singh Rawat

T23191

for the partial fulfillment of the degree

of

**MASTERS OF TECHNOLOGY IN
COMPUTER SCIENCE ENGINEERING**

SCHOOL OF COMPUTER AND ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY MANDI

KAMAND-175075, INDIA

JANUARY, 2024

Contents

Chapter 1

Introduction

1.1 Motivation

Efficient management of data is very important in every computer application. Heap data structure is important in applications where priority-based scheduling plays a pivotal role. Take the example of an operating system as software that has to manage all the running processes on the basis of priority. It has to take care that the user must get a good experience while using it, which will happen if user processes are taken as priorities. All the operations that are performed by the user should be executed in an ordered manner and also ensure that the system's performance is optimized. Consideration of priority Fibonacci Heaps is popular among theoretically optimized operations like insertion, deletion, and priority update. It does consist of complex operations, which usually cause difficulty in understanding in the educational arena. Students usually find it difficult to comprehend its workings. This work [1] is done to find an alternative that will strike a balance between performance and educational understanding.

1.2 State-of-the-Art

The count of those already invented is various, several with the same features as the Fibonacci heaps [2] proposed by Fredman and Tarjan's that set the standard for insertion and decrease key operations in amortize time of $O(1)$ and delete min operations in amortize time of $O(\log n)$. Still, understanding its workings is complex, which makes it complicated for learners. There are also various alternatives, which consist of V-Heaps [3], Relaxed Heaps [4], Rank Pairing Heaps [5], Violation Heaps [6], and Thin-Fat Heaps [7]. Each of them has unique features and benefits. Some of them can lower amortized time, while others could ensure balancing the structure.

1.3 Relevance of Quake Heaps

Among all heaps, Quake Heaps stands out as a distinct methodological choice. They provide the same insertion and deletion operations in amortized time of $O(1)$ and delete min operations in amortized time of $O(\log n)$. While replicating the same theoretical work as Fibonacci Heaps, it also has ease of understanding for educational purposes. The theoretic understanding is easy, which makes it an appealing option for teaching purposes. Quake Heaps solves the problem of providing the same theoretical value while easing the implementation. Their simplicity is enhanced by not using cascading cuts, which are found in Fibonacci heaps.

Transition

As we progress in the chapters, we will see in-depth construction, functions, and theoretical aspects to understand this unique heap structure and its uses.

Chapter 2

Problem Definition, Model, Approach

2.1 Problem Definition

Heaps are important structures in computer science, a major application where retrieval of data is important on the basis of priority. Priority queues are used in places where such features are required. However, those do not provide insertion and decrease key operations in $O(1)$. While Fibonacci heaps are able to perform all operations with a given theoretical runtime complexity, their complex structure makes them difficult to understand for learning and creates difficulty for teaching purposes. Evolving a heap that provides the same theoretical aspect with a simpler approach while proving the inherent features of the heaps.

2.2 Model

Here we have used a tournament tree approach that is simple and efficient. The inherent feature of tournament trees came in use to order the element based on value. As tournament trees work on the basis of this data structure, providing a clear understanding will help in understanding the data structure. A tournament tree with a simple structure allows for the formation of order elements based on rank, which will be required as the base of quake heaps.

2.3 Approach

The approach taken here is to make strategic use of tournament trees by having a lazy update. The operations of lazy update and insertion are used in such a manner that they can provide the same theoretical performance as Fibonacci heaps. This will also take care of the balance between simplicity and performance by performing updates only when necessary. Which will lead to the ability to perform operations in constant time.

2.4 Structure of Quake Heaps

Quake Heaps take an innovative approach to heap architectures, offering a simpler yet more effective alternative to sophisticated systems. The structure consists of two types of nodes: internal nodes and leaf nodes.

Internal Node

The internal node in Quake Heaps encapsulates the following attributes:

- **Height:** Denotes the height of the internal node.
- **Parent:** Points to the parent node.
- **Left Child:** References the left child node.
- **Right Child:** References the right child node.
- **Leaf Reference:** Points to the associated leaf node.

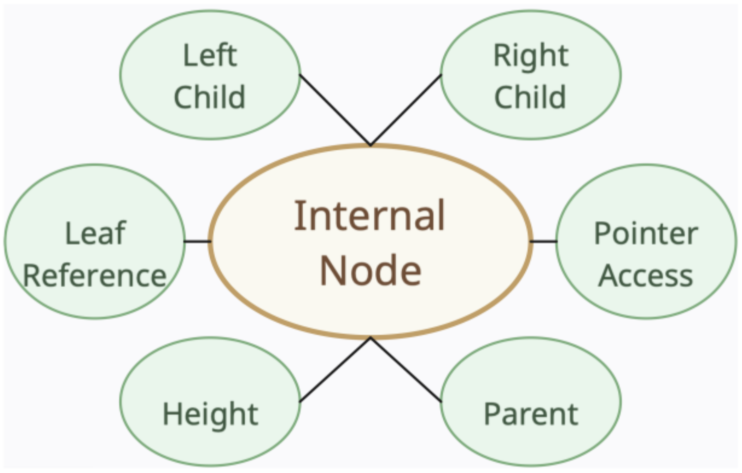

Fig. 2.1: Internal Node Illustration

Leaf Node

The leaf node in Quake Heaps consists of the following fields:

- **Key:** Represents the key associated with the node.
- **Value:** Stores the value corresponding to the key.
- **Highest Reference:** Points to the highest internal node referencing this leaf.

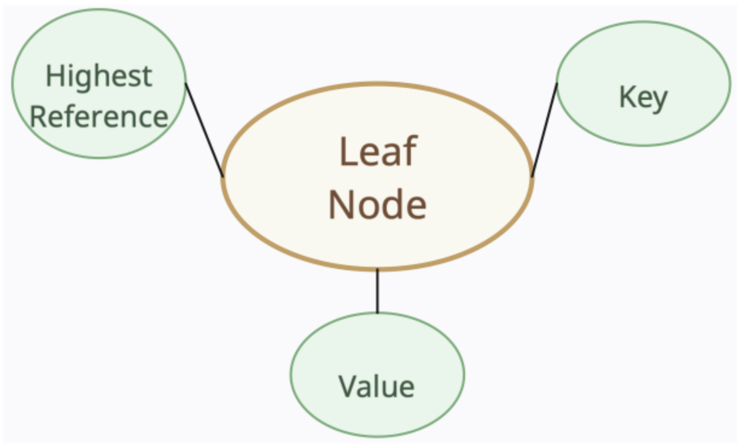

Fig. 2.2: Leaf Node Illustration

2.5 Quake Heaps Operations

Insert Operation

1. Create a leaf node and update the key with the given value.
2. Create an internal node with height 0, referencing the created leaf node.
3. Store the internal node in the forest and perform an Update-Min operation.
4. Update pointer access in the internal node with the stored address of the list.

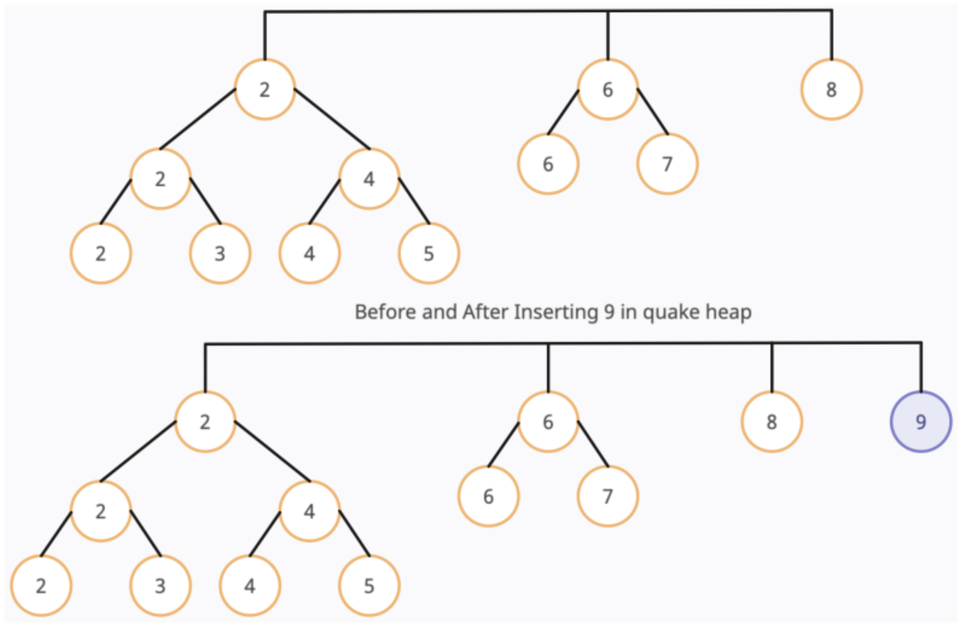

Fig. 2.3: Insert Operation Illustration

Link Operation

1. Create an internal node and update its left and right with node1 and node2 with the parent as null.
2. Update the leaf reference after checking with the smallest leaf node.

3. Update the parent of both nodes with the new node, and update the height.

4. Insert the node in the forest and update the pointer access.

5. Perform the update min operation.

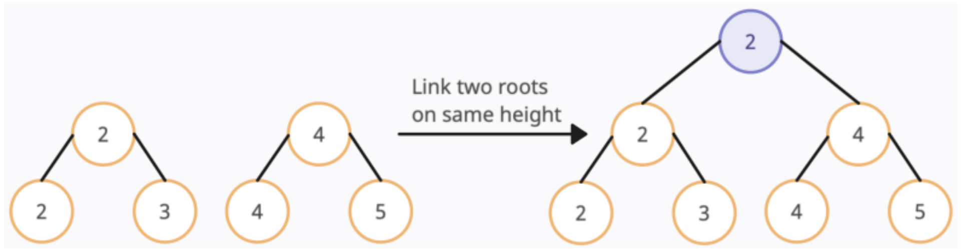

Fig. 2.4: Link Operation Illustration

Cut Operation

1. Get the leaf reference for the node.

2. Update the child parent with null, removing the reference to the leaf node.

3. Update the node with the child that has the same leaf reference.

4. Perform it until the root refers to null.

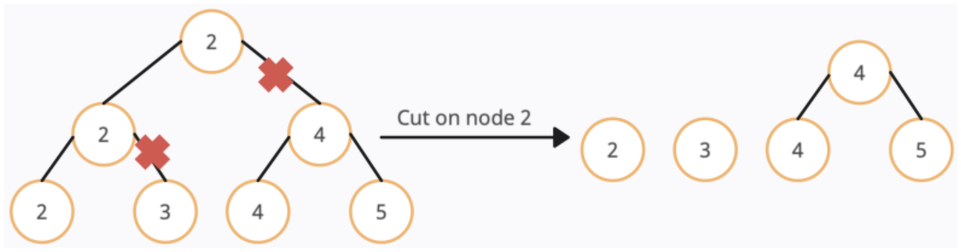

Fig. 2.5: Cut Operation Illustration

Decrease-Key Operation

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

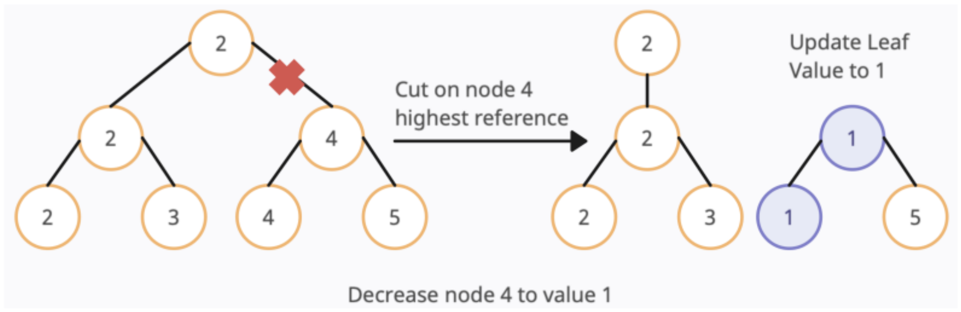

Fig. 2.6: Decrease-Key Operation Illustration

Delete-Min Operation

1. Perform the cut operation with the help of the minimum pointer.
2. Perform the link operation of nodes with the same height.
3. Check the off-balance condition if it exists ($\text{height}_i < \text{height}_{i-1} \times \alpha$).
4. If an off-balance condition exists, perform a cut on all the roots from height i to maximum.
5. Perform the update min operation.

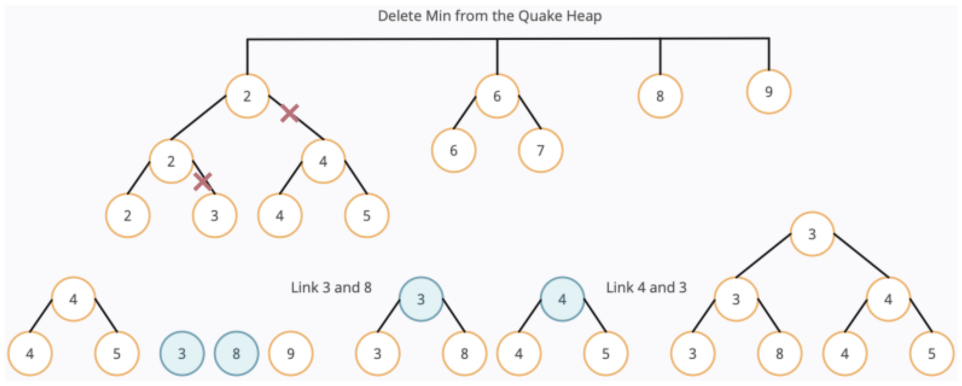

Fig. 2.7: Delete-Min Operation Illustration

The integration of Quake Heaps into the exploration of heap structures provides a promising alternative, addressing challenges posed by more complex solutions. This chapter establishes the foundation for the subsequent detailed examination of the mechanics and theoretical foundations of Quake Heaps.

Chapter 3

Results

3.1 Overview

Here, we provide the results obtained from our exploration of quake heaps. It shows the amortized run-time complexity for all the basic operations that we perform in heaps. The results shared in Table 3.1 show the comparison of amortized time complexity of various types of heaps.

Table 3.1: Amortized Comparison of Various 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)

3.2 Run Time Analysis

Below is the analysis of the key operations in the Quake Heaps. The analysis utilizes the potential method, where potential is defined as $N + T + \frac{1}{2\alpha-1} \cdot B$, with N being the number of nodes, T the number of trees, and B the number of degree 1 nodes.

Insert Operation

- Actual cost: $O(1)$
- Changes: N and T increase by 1
- Amortized cost: $O(1)$

Decrease-Key Operation

- Actual cost: $O(1)$
- Changes: T and B increase by 1
- Amortized cost: $O(1)$

Delete-Min Operation

- Analysis for removing root path till link trees:
  - Actual cost is bounded by $T(0) + O(\log n)$.
  - Change in T: $O(\log n) - T(0)$.
  - Change in B: Nonpositive.
  - Amortized cost: $O(\log n)$.
- Analysis for alpha condition:
  - Actual cost is bounded by $\sum_{j>i} n_j$.
  - Change in N: At most $-\sum_{j>i} n_j$.
  - Change in T: At most $+n_i$.
  - Change in B: Non-positive.
  - Amortized cost: Non-positive.

Therefore, the overall amortized cost for `delete-min()` is $O(\log n)$.

Chapter 4

Future Work

4.1 Extensions

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), which can be done if we can include some constant-time operation of linking at the time of insert and decreasing, which will not affect their runtime but will improve the runtime operation of Delete Min.

4.2 Conclusion

Concluding with a comprehensive summary of the findings that put emphasis on the use of quake heaps as a simple and effective alternative heap data structure. We reflect the contribution done through this work as a base point for future endeavors and further improvement of Quake Heaps so that it could have better potential advancements.

Bibliography

[1] T. M. Chan, "Quake heaps: A simple alternative to fibonacci heaps," *Springer Berlin Heidelberg*, pp. 27–32, 2013.

[2] M. Fredman and R. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *Journal of the ACM*, vol. 34, pp. 596–615, 1987.

[3] G. Peterson, "A balanced tree scheme for meldable heaps with updates," Georgia Institute of Technology, Technical Report GIT-ICS-87-23, 1987.

[4] J. Driscoll, H. Gabow, R. Shrairman, and R. Tarjan, "Relaxed heaps: An alternative to fibonacci heaps with applications to parallel computation," *Communications of the ACM*, vol. 31, pp. 1343–1354, 1988.

[5] B. Haeupler, S. Sen, and R. E. Tarjan, "Rank-pairing heaps," *SIAM Journal on Computing*, vol. 40, pp. 1463–1485, 2011.

[6] A. Elmasry, "The violation heap: A relaxed fibonacci-like heap," *Discrete Mathematics, Algorithms and Applications*, vol. 2, pp. 493–504, 2010.

[7] H. Kaplan and R. Tarjan, "Thin heaps, thick heaps," *ACM Transactions on Algorithms*, vol. 4, no. 1, p. 3, 2008.

Report

ORIGINALITY REPORT

18%	16%	14%	7%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	tmc.web.engr.illinois.edu Internet Source	4%
2	repository.ubn.ru.nl Internet Source	3%
3	www.pmg.lcs.mit.edu Internet Source	2%
4	Submitted to Indian Institute of Technology Student Paper	2%
5	uwspace.uwaterloo.ca Internet Source	1%
6	livrepository.liverpool.ac.uk Internet Source	1%
7	László Kozma, Thatchaphol Saranurak. "Smooth Heaps and a Dual View of Self-Adjusting Data Structures", SIAM Journal on Computing, 2019 Publication	1%
8	Lecture Notes in Computer Science, 2004. Publication	1%

9	hjemmesider.diku.dk Internet Source	1 %
10	ipfs.io Internet Source	1 %
11	silo.pub Internet Source	<1 %
12	www.cs.indiana.edu Internet Source	<1 %
13	www.qucosa.de Internet Source	<1 %
14	"Space-Efficient Data Structures, Streams, and Algorithms", Springer Science and Business Media LLC, 2013 Publication	<1 %

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		