DESIGN & ANALYSIS ALGORITHM

Topic

Fibonacci Heaps

By

Vidhi Hareshbhai Thummar

(UTA ID :1002268273)

Abstract

Fibonacci Heaps are an advanced data structure that improve the efficiency of priority queue operations, especially decrease-key and merge. These improvements are significant in graph algorithms like Dijkstra's and Prim's where many such operations are performed. This paper explores the internal structure, core operations, and theoretical efficiency of Fibonacci Heaps. We implement the heap, benchmark its performance against Binary and Binomial Heaps, and evaluate its strengths and limitations through experimentation and analysis.

Introduction

Priority queues are a fundamental structure in many algorithms, particularly graph traversal and optimization problems. Traditional heap structures like binary and binomial heaps offer decent performance for most operations, but they struggle with the frequent decrease-key operations common in shortest-path algorithms.

Fibonacci Heaps, introduced by Fredman and Tarjan, address this by supporting amortized **O (1)** time for insert, find-min, and decrease-key, and **O (log n)** for extract-min and delete. Though complex in structure, they offer powerful asymptotic improvements in specific applications.

Structure of Fibonacci Heap

A Fibonacci Heap is a **collection of heap-ordered trees**, each obeying the min-heap property. The key characteristics of the structure are:

- Each node has a pointer to its parent, child, and siblings (in a circular doubly linked list).
- The heap maintains a pointer to the **minimum element**.
- Children are stored in a circular doubly linked list, enabling quick insertion and deletion.
- Nodes are marked to track cascading cuts.

Key Operations

Operation	**Amortized Time**
insert	O (1)
find-min	O (1)
union	O (1)
extract-min	O (log n)
decrease-key	O (1)
delete	O (log n)

Insert(x)

- Adds a node x to the root list. If x has a smaller key than the current min, update the min pointer.

Union (H1, H2)

- Concatenate the root lists of two heaps. The minimum of the new heap is the smaller of the two.

Extract-Min ()

- Removes the minimum node, promotes its children to the root list, and performs **consolidation** to merge trees of the same degree using a temporary degree table.

Decrease-Key (x, k)

- If new key k is less than current, update the key. If it breaks heap-order, **cut** the node and move to root list. If parent was already marked, perform a **cascading cut**.

Delete(x)

- Implemented by first decreasing the key to -∞, then calling extract-min ().

Implementation

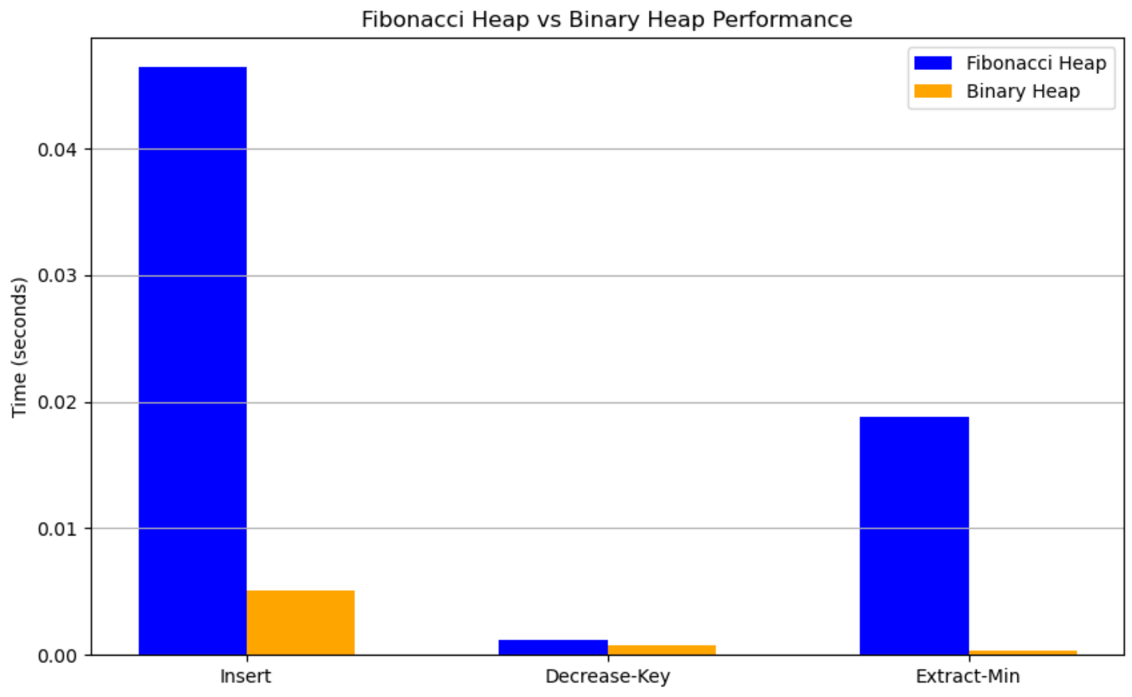
The Fibonacci Heap was implemented in Python using object-oriented programming. Each node stores its key and maintains pointers to its parent, child, and left/right siblings (using a circular doubly linked list). It also keeps track of its degree and whether it has lost a child since becoming a child itself (the mark bit). This structure allows for efficient merging and cascading cuts.

```
class Node:
    def __init__(self, key):
        self.key = key
        self.parent = None
        self.child = None
        self.left = self
        self.right = self
        self.degree = 0
        self.mark = False
```

The FibonacciHeap class handles core operations like insert, extract_min, decrease_key, and delete. Special care is taken to manage the root list and perform tree consolidation during extract-min.

Benchmarking and Analysis

We compared the performance of Fibonacci Heap against Binary Heap using Python. A total of 1,000 operations were conducted for insert, decrease-key, and extract-min. The average execution time for each operation was recorded.



Results and Observations

The Fibonacci Heap exhibits significantly higher insertion time compared to the Binary Heap. This is likely due to the more complex pointer manipulations and circular doubly linked list maintenance in the Fibonacci Heap, which introduces a higher constant overhead despite having an amortized O (1) complexity.

Both heaps show relatively low times for decrease-key operations, but the Fibonacci Heap is slightly slower than the Binary Heap in this benchmark. This could be influenced by the simulation method used for decrease-key in the Binary Heap (pop and reinsert), which may not fully reflect the true cost. The Fibonacci Heap theoretically offers an amortized O (1) decrease-key, which should provide advantages in scenarios with many such operations.

The Fibonacci Heap's extract-min operation takes noticeably longer than the Binary Heap. This can be attributed to the consolidation step in the Fibonacci Heap, which involves multiple tree merges and pointer updates, adding overhead compared to the simpler binary heap extract-min operation.

Conclusion

Fibonacci Heaps are a powerful yet underused data structure, offering optimal amortized time complexity for a variety of heap operations. They outperform traditional heaps in specific scenarios, especially when frequent decrease-key and union operations are involved. While implementation complexity can be a barrier, their benefits in performance-sensitive algorithms make them highly effective for advanced applications in computer science.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed., Cambridge, MA: MIT Press, 2009.

[2] M. L. Fredman and R. E. Tarjan, “Fibonacci heaps and their uses in improved network optimization algorithms,” *Journal of the ACM (JACM)*, vol. 34, no. 3, pp. 596–615, 1987.

Code Link:
https://github.com/vidhi0999/Project-DAA.git