INTRODUCTION TO BINOMIAL HEAPS

Binomial Heaps are a type of priority queue data structure that supports a collection of operations efficiently. Invented by Jean Vuillemin in 1978, they are based on binomial trees and provide an efficient implementation of merging two heaps, which is particularly useful in applications like Dijkstra's and Prim's algorithms.

Structure of Binomial Heaps

A Binomial Heap is a collection of binomial trees that are defined recursively:

- A binomial tree B_k of order k has a root node and k children.
- Each child is the root of a binomial tree B_{k-1}, B_{k-2},....., B_0.

Binomial trees are structured such that the depth of the B_k tree is k. The heap is composed of a forest of these trees, where no two trees have the same order, allowing for efficient merging of heaps.

Key Properties

- Order: Each tree in the heap is a binomial tree, and no two binomial trees have the same order.
- Min-Heap Property: The key of each node is greater than or equal to the key of its parent.

Binomial Heap Operations

Insert Operation

To insert a new element into a Binomial Heap, create a new single-node binomial tree (a B_0 tree) and merge it with the existing heap.

1. Create a binomial tree B_0 with the new key.

2. Merge this tree with the existing heap.

Time Complexity: O(log n)

Example:

Insert keys 3, 18, and 39 into an empty binomial heap.

1. Insert 3: The heap now contains a single binomial tree B_0(3).
2. Insert 18: The heap now contains two binomial trees B_0(3) and B_0(18).
3. Insert 39: The heap merges B_0(18) and B_0(39) to form B_1.

The heap structure:

Extract Minimum

The extract-minimum operation removes the tree with the minimum root and restructures the heap.

1. Find the root with the minimum key.
2. Remove this root and its tree.
3. Add the children of this root as separate binomial trees.
4. Merge these trees back into the heap.

Time Complexity: O(log n)

Example:

Extract the minimum from the heap containing B_1(3).

1. Minimum key is 3.

2. Remove 3, leaving trees B_0(18) and B_0(39).

3. Merge these trees back into the heap.

Resulting structure:

```
  18
 /
39
```

Decrease Key

To decrease the key of a node, reduce its value and then check if the heap property is violated. If so, perform a series of cuts and swaps to restore the heap property.

1. Decrease the key of the node.

2. If the new key is less than its parent's key, swap them.

3. Continue this process up the tree until the heap property is restored.

Time Complexity: O(log n)

Example:

Decrease key from 39 to 1 in the following structure:

```
  18
 /
39
```

Change 39 to 1 and swap with 18.

1

/

18

Delete Operation

To delete a node, decrease its key to negative infinity, making it the minimum element, and then extract it.

1. Decrease the key of the node to negative infinity.

2. Extract the minimum.

Time Complexity: O(log n)

Example:

Delete node with key 18 in the structure:

1

/

18

Decrease key of 18 to negative infinity, then extract minimum.

Resulting structure:

1

Union Operation

Union of two binomial heaps involves merging the root lists of the two heaps and combining trees of the same order.

1. Merge the root lists of both heaps, maintaining the binomial tree order.

2. Combine trees of the same order to maintain the heap properties.

Time Complexity: O(log n)

Example:

Union of two heaps:

Heap 1: 3

Heap 2: 18

Merge:

3 <-> 18

Resulting in a single binomial tree after merging trees of the same order.

Applications of Binomial Heaps

Binomial Heaps are used in:

- Graph Algorithms: Efficient implementation of Dijkstra's and Prim's algorithms.
- Priority Queue Operations: Handling a dynamic set of priorities.

Conclusion

Binomial Heaps provide a powerful and efficient data structure for priority queue operations, especially where merging of heaps is frequent. Their structure, based on binomial trees, allows for efficient implementation of a wide range of operations, making them suitable for numerous applications in computer science. Understanding the mechanics of Binomial Heaps enables better performance in algorithms and data handling tasks.