Semester VI (B.Tech)

Jaypee University of Engineering and Technology, Guna

Advanced Programming Lab-3 18B17CI673

Lab Exercise 6: Binomial Tree and Heap

Title: Binomial Tree and Heap

Mode: Self Learning

Outcomes:

1. Understand the concept of binomial trees and heaps.
2. Implement binomial trees and heaps using arrays or linked lists.
3. Analyze the time and space complexity of various operations on binomial trees and heaps.
4. Apply binomial trees and heaps to solve real-world problems..

Methodology:

A binomial tree is a tree structure that satisfies certain properties, such as having a root node with a certain number of children, and each child being the root of a smaller binomial tree. A binomial heap is a collection of binomial trees that satisfy certain properties, such as having the minimum element at the root node, and each child having a smaller key value than its parent.

In this experiment, we will study the properties and operations of binomial trees and heaps. We will learn how to implement binomial trees and heaps using arrays or linked lists. We will also analyze the time and space complexity of various operations on binomial trees and heaps, such as insertion, deletion, and searching.

We will then apply binomial trees and heaps to solve real-world problems, such as priority queue management, graph algorithms, and data compression.

Steps:

1. Start by understanding the basics of binomial trees and heaps. Learn about the properties of binomial trees and heaps, and the different types of operations that can be performed on them.
2. Implement binomial trees and heaps using arrays or linked lists. Write code to create a new binomial tree or heap, and to perform operations such as insertion, deletion, and searching.
3. Analyze the time and space complexity of the binomial tree and heap operations. Use big-O notation to express the time and space complexity of each operation.

4. Apply binomial trees and heaps to solve real-world problems. For example, you could use a binomial heap to implement a priority queue for scheduling tasks in a computer system.
5. Experiment with different variations of binomial trees and heaps. For example, you could try implementing Fibonacci heaps, which are a type of binomial heap that has better amortized time complexity for certain operations.
6. Test the performance of your binomial tree and heap implementations on large datasets. Use profiling tools to measure the execution time and memory usage of your code.
7. Write a report summarizing your findings and conclusions. Include a discussion of the strengths and weaknesses of binomial trees and heaps, and recommendations for future improvements or applications.

Experiments:

1. Make a Binomial Heap by Inserting a sequence of elements taken from the user.
2. Merge two binomial Heaps
3. Delete the smallest element in the Binomial Heap