Student Name: A Z Hasnain Kabir
Student ID: 200042102

CSE 4303 Data Structures
Assignment on
Binary Indexed Tree

Introduction

A binary indexed tree is a data structure stored as an array that can efficiently calculate prefix sum and perform update operations. It is also called a Fenwick Tree. Given an array of inputs, the tree will store the sum of some elements in each of its nodes. There is another data structure called a segment tree which can also perform similar tasks. However, in contrast to a segment tree, a binary indexed tree will consume less space, and the implementation of a binary indexed tree is also easier.

Internal understanding of binary indexed tree

In a scenario where we store the sum of given inputs in a binary indexed tree, we can express each integer value as the sum of powers of 2. For example, 21 can be expressed as 16+4+1 where each term is a power of 2. Therefore, each node in a binary indexed tree will store sum of n elements as a power of 2.

Operations on a binary indexed tree

We can perform query operations and update operations in a standard binary indexed tree.

Advantages of a Binary indexed tree

- A binary indexed tree will always be more space efficient compared to a segment tree.
- A binary indexed tree is easier and faster in terms of implementation in code.

Sample Code

```
1  #include <iostream>
2
3  using namespace std;
4
5  int getSum(int tree[], int index)
6  {
7      int sum = 0;
8      index = index + 1;
9
10     while (index>0)
11     {
```

```
12
13          sum += tree[index];
14          index -= index & (-index);
15      }
16      return sum;
17  }
18
19  void update(int tree[], int n, int index, int val)
20  {
21      index = index + 1;
22      while (index <= n)
23      {
24      tree[index] += val;
25      index += index & (-index);
26      }
27  }
28
29  int *constructTree(int arr[], int n)
30  {
31      int *tree = new int[n+1];
32      for (int i=1; i<=n; i++)
33          tree[i] = 0;
34      for (int i=0; i<n; i++)
35          update(tree, n, i, arr[i]);
36      return tree;
37  }
```

Complexity Analysis

The construction of a binary indexed tree has a time complexity of O(nlog n). Similar to a segment tree, both query() and update() operation has a time complexity of O(log n).

Problems involving binary indexed tree

Range sum query

In this problem, we are required to calculate the sum of some elements of a given array within an input range. We also have to update numbers for given indices throughout our calculations.

Solution

We can use a binary indexed tree and sum of multiple ranges will be stored inside that tree so that we can find sum of our desired range within O(log n) time complexity.