Fenwick Tree

Ajay Kumar

Date 13 Feb 2023

Introduction

- Just because a data structure has a word tree in it, doesn’t mean it’s hard to understand or implement.
- One such data structure is Binary Indexed Trees a.k.a Fenwick Trees. BIT is a data structure that you can understand a lot more easily as compared to other data structures that are capable of performing the same tasks which BIT does and also BIT is much much much easier to implement especially in short contests or situations where you have time constraints.
- Just FYI the full code of BIT can be written in less than 10 lines. That’s how easy to code it is.
-

Applications

- BIT is ds which in general is used to perform range queries over an array or let’s generalize and say a collection that can be indexed.
- In Competitive programming, you will encounter a lot of questions which require you to perform range queries as well as updates in a collection or an array the simplest one being

->For Q queries, in each query update ith index of an array with some value or provide a sum from index l to r in the array.

- Segment Trees is a popular data structure that can solve these kinds of problems and though segment trees are not that hard to understand they can be quite tedious to write and consume a lot of time to debug as well if you do a mistake in case.
- BIT is a very good alternative to segment trees for a lot of questions and when I say alternative I don't at all mean that BIT can replace Segment trees in all the situations.
- ***Segment Trees can solve a lot more complex problems related to range queries which BIT cannot do but for a lot of questions where BIT can be used, it's just bliss for all competitive programmers.***.
- Also as most of the people use recursive segment trees and binary indexed trees are based on bit manipulation, hence BIT have faster execution times as compared to recursive segment trees even though the overall complexities of both of them is O(log(n))

Binary Indexed Trees

/

Fenwick Trees

Made Easy

Code and Implementation

```
const int N = 1e5+10;
int bit[N];

void update(int i, int x){
    for(; i < N; i += (i&-i))
        bit[i] += x;
}

int sum(int i){
    int ans = 0;
    for(; i > 0; i -= (i&-i))
        ans += bit[i];
    return ans;
}
```