← Notes

Binary Indexed Tree

18 Tree Algorithm

A **BIT (Binary Indexed Tree) or Fenwick Tree** is a data structure providing efficient methods for calculation and manipulation of the prefix sums of a table of values. Fenwick tree calculate prefix sums and modify the table in **O(log n)** time, where n is the size of the table. It is very useful for solving queries of a particular type, Let us suppose we have an array of elements and we need to do two types of queries frequently.

1)Change the value of any element in the list.

2)Get the sum of all the elements in any range within the given list.

The naive solution has time complexity of O(1) for query of type 1 and O(n) for query of type 2. suppose we make M queries.The worst case (when all queries are of type 2) has time complexity O(n*M). Binary Indexed Trees are easy to code and have worst time complexity O(Mlogn). i.e each query of the type 2 takes at most O(log n).

Let us consider an initial array of size N(can be any valid size within Integer Range).We can easily construct another array of similar size N with values which is used to hold the cumulative values e.g.

index	0	1	2	3	4	5	6	7	8
Int arr[]	0	6	1	3	-1	2	9	1	2
Int cumulative[]	0	6	7	10	9	11	20	21	23

The underlying working principle of the Binary Indexed Tree is achieved by the help of responsibility array. It's an array of size>=N size where each index holds some specific values (called the responsibility sum).Let us declare 'int res[N]'

Basic Idea

We know that each integer can be represented by sum of powers of 2 e.g.

9=8+1 ----> 1001(Binary Notation)

37=32+4+1 --> 00100101(Binary Notation)

15=8+4+2+1 -> 1111(Binary Notation)

We can see that the maximum number of 1 in the binary notation of 15 is ceil(log215)=4 ,similarly for 37 and 9.

for 37 ceil(log237)=6

for 9 ceil(log29)=4

So, number of digits of any number in 2's power form would be log2N ,this is what exactly the property being made use in BIT.

Similarly, instead of storing the cumulative frequencies for the entire array we can store the sum of some sub frequencies (Not the entire values from 0 to that index) in particular indexes, the purpose of which will be a bit clear in a short while.

->suppose idx be some index(not value) of the array of size N (therefore,0<=idx<=N)

->suppose r be the position of the last occurrence of 1 in the binary notation of idx from left to right.(See the example below) therefore, 1<=r<=log2N

The responsibility range is the range of **(idx-2^r+1) to idx**(inclusive).And the responsibility array contains the sum of all the index in the frequency array corresponding to the range. for example let idx=12(1100),hence r=2 and thus res[12], the range of indexes of the frequency array covered by it is ,

[12-(2^2)+1 to 12] i.e [9-12] hence ,

res[12]=(frequency[9]+frequency[10]+frequency[11]+frequency[12])

index	0	1	2	3	4	5	6	7	8
Int arr[]	0	6	1	3	-1	2	9	1	2
Int cumulative[]	0	6	7	10	9	11	20	21	23
Responsibility range	0	1	1..2	3	1..4	5	5..6	7	1..8
Int res[]	0	6	7	3	9	2	11	1	23

In order to read the cumulative value at any index e.g 13 we just need to remove the last one bit(i.e r) each time from the idx till the value is greater then 0

```
    13th index(1101) = res[13] + res[12] + res[8]

                          1101   +  1100   +  1000 (here we are removing the
```

We can also check from the above table e.g. cumulative value of 6th index (0110)=res[6]+res[4]=9+11=20 i.e

0110 =6

0100 =4

LOGIC :

we need to find a fast way of finding the r at each step. This can be easily done using Val=(num & -num) where num is our initial number. In binary notation num can be represented as 'a1b', where a represents binary digits before the last digit and b represents zeroes after the last digit.

```
    -num = (a1b)¯ + 1
     = a¯0b¯ + 1
     = a¯0(0...0)¯ + 1
     = a¯0(1...1) + 1
     = a¯1(0...0)
     = a¯1b.
```

Now, we can easily isolate the last digit, using bitwise operator AND (in C++, Java it is &) with num and -num:

```
         a1b
  &     a¯1b
--------------------
 = (0...0)1(0...0)
```

The number of iterations in this function is number of bits in idx, which is at most log N.Time complexity: O(log N).

READ LOGIC :

```
     int read(int idx)
     {
              int sum = 0;
              while (idx > 0){
                      sum += res[idx];
                      idx -= (idx & -idx);
              }
              return sum;
      }
```

So,in order to get the sum between say two points a,b, we do read(b)-read(a-1)

UPDATE LOGIC :

```
void update(int idx ,int val)
{
        while (idx <= N){
                res[idx] += val;
                idx += (idx & -idx);
        }
}
```

Remember you should never query or update on the 0th element. that gives an error because your elements are a1 , a2 , ...a[n] so be careful with that.

AUTHOR

Raj kumar verma
Problem-Setter at HackerE...
Gwalior
1 note

TRENDING NOTES

Matrix exponentiation
written by Mike Koltsov

Graph Theory - Part II
written by Pawel Kacprzak

Computational Geometry - I
written by Arjit Srivastava

Rendering Performance in Android - Overdraw
written by Vishnu Sosale

Sparse table
written by Mike Koltsov

more ...

Recruit

Developer Sourcing

Lateral Hiring

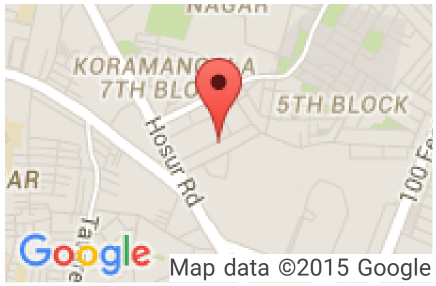
Campus Hiring

FAQs

Customers

Annual Report

Connect with us



IIIrd Floor, Salarpuria Business Center,
4th B Cross Road, 5th A Block,
Koramangala Industrial Layout,
Bangalore, Karnataka 560095,
India.

Reach us

contact@hackerearth.com
+91-80-4155-4695
+1-650-461-4192