← Notes

Binary Indexed Tree or Fenwick Tree

199 Fenwick-tree CodeMonk Binary-indexed-tree Data-structures range-query

Binary Indexed Tree also called Fenwick Tree provides a way to represent an array of numbers in an array, allowing prefix sums to be calculated efficiently. For example, an array is [2, 3, -1, 0, 6] the length 3 prefix [2, 3, -1] with sum 2 + 3 + -1 = 4). Calculating prefix sums efficiently is useful in various scenarios. Let's start with a simple problem.

We are given an array a[], and we want to be able to perform two types of operations on it.

1. Change the value stored at an index i. (This is called a **point update** operation)
2. Find the sum of a prefix of length k. (This is called a **range sum** query)

A straightforward implementation of the above would look like this.

```
int a[] = {2, 1, 4, 6, -1, 5, -32, 0, 1};
void update(int i, int v)   //assigns value v to a[i]
{
    a[i] = v;
}
int prefixsum(int k)    //calculate the sum of all a[i] such that 0 <= i < k
{
   int sum = 0;
   for(int i = 0; i < k; i++)
        sum += a[i];
   return sum;
}
```

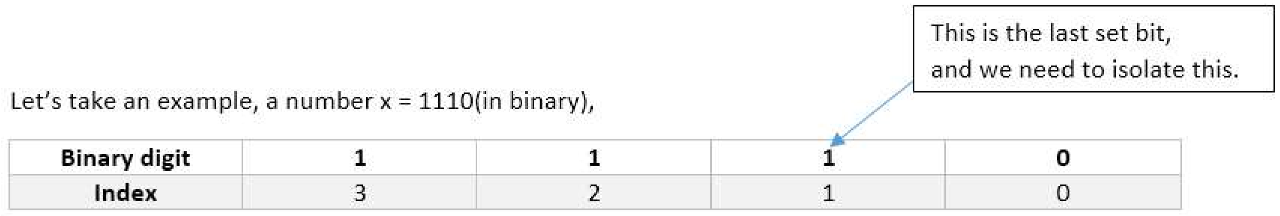
This is a perfect solution, but unfortunately the time required to calculate a prefix sum is proportional to the length of the array, so this will usually time out when large number of such intermingled operations are performed.

Can we do better than this? Off course. One efficient solution is to use segment tree that can perform both operation in O(logN) time.

Using binary Indexed tree also, we can perform both the tasks in O(logN) time. But then why learn another data structure when segment tree can do the work for us. It's because binary indexed trees require less space and are **very easy to implement** during programming contests (the **total code is not more than 8-10 lines**).

Before starting with binary indexed tree, we need to understand a particular bit manipulation trick. Here it goes.

Isolating the last set bit


This is the last set bit,
and we need to isolate this.

Let's take an example, a number x = 1110(in binary),

Binary digit	1	1	1	0
Index	3	2	1	0

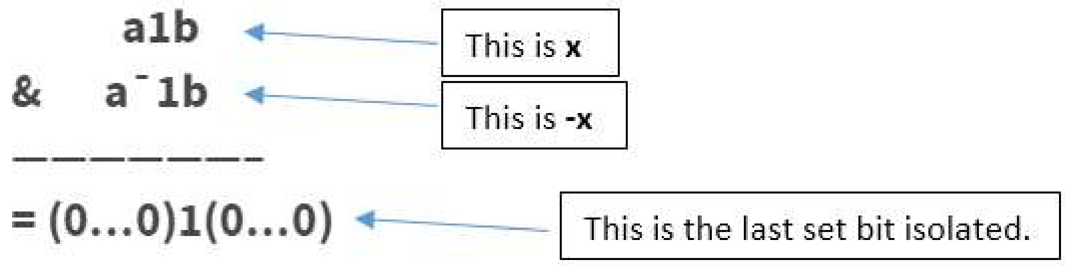
How to isolate?

x&(-x) gives the last set bit in a number x. How?

Let's say **x = a1b**(in binary) is the number whose last set bit we want to isolate.

Here **a** is some binary sequence of any length of 1's and 0's and **b** is some sequence of any length but of 0's only. Remember we said we want the LAST set bit, so for that tiny intermediate 1 bit sitting between **a** and **b** to be the last set bit, **b** should be a sequence of 0's only of length zero or more.

-x = 2's complement of x = **(a1b)'** + 1 = **a'0b'** + 1 = **a'0(0….0)'** + **1** = **a'0(1...1)** + **1** = **a'1(0…0)** = **a'1b**



Example: x = 10(in decimal) = 1010(in binary)

The last set bit is given by x&(-x) = (10)1(0) & (01)1(0) = 0010 = 2(in decimal)

But why do we need to isolate this weird last set bit in any number? Well we will be seeing that as you proceed further.

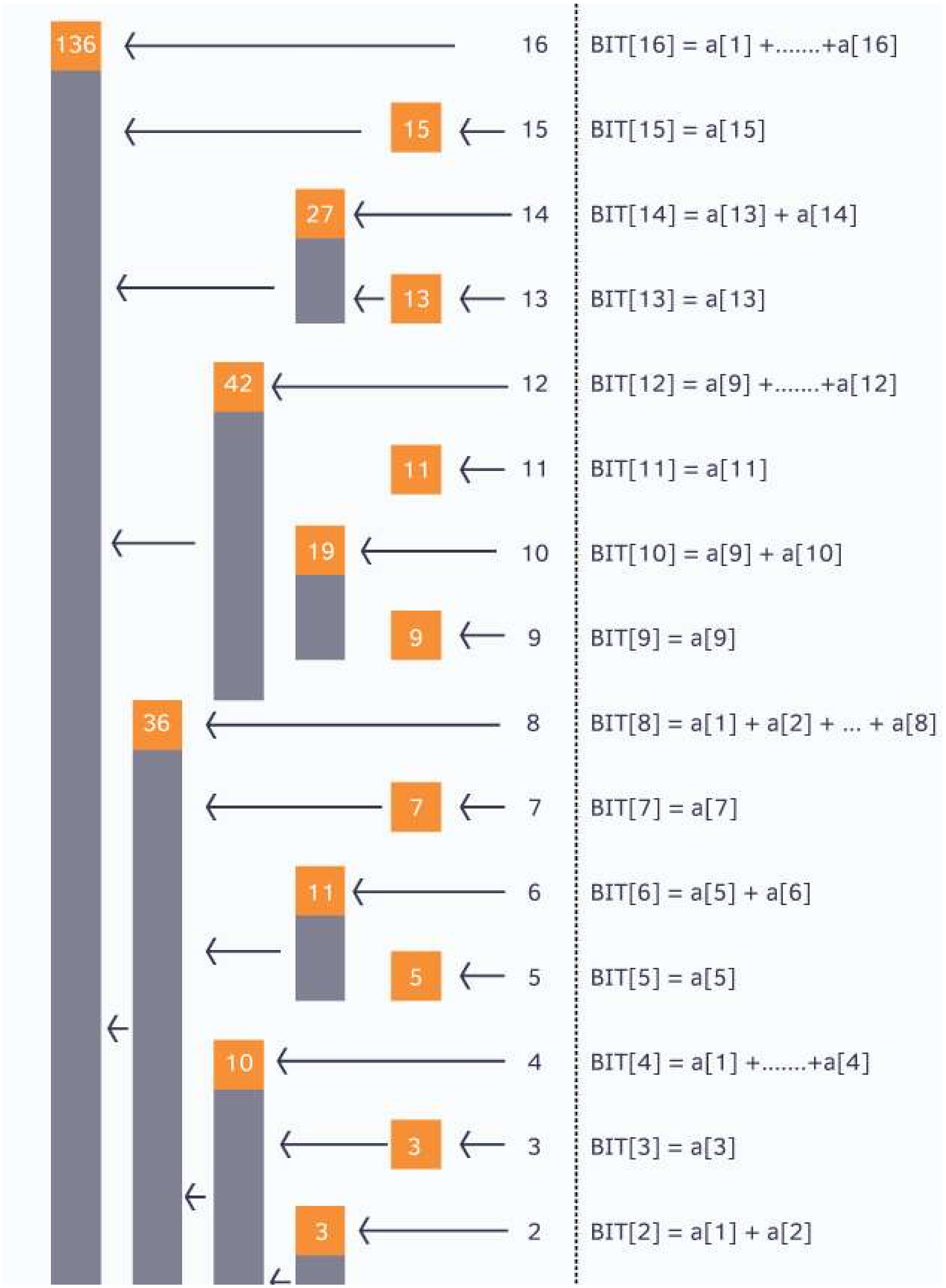
Now let's dive into Binary Indexed tree.

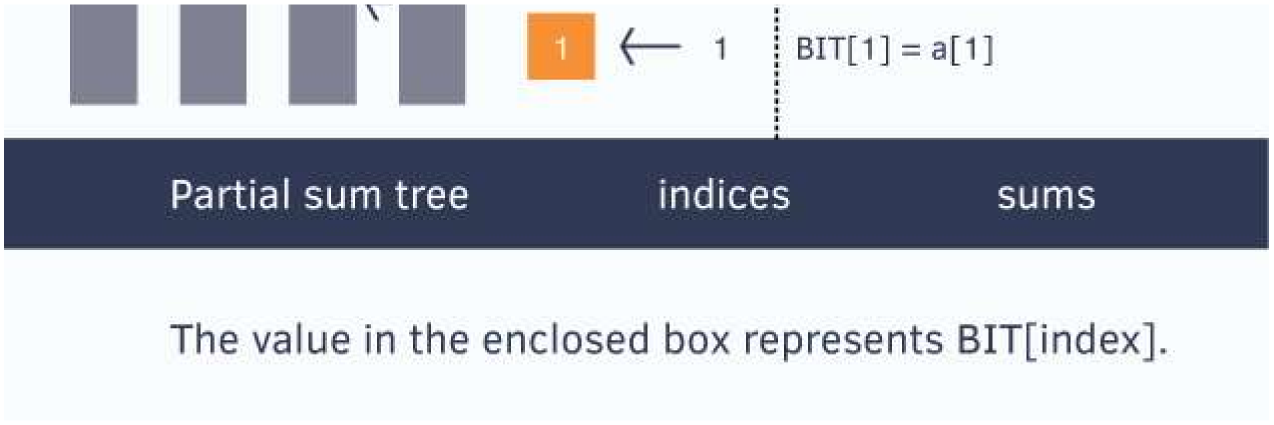
Basic Idea of Binary Indexed Tree:

We know the fact that each integer can be represented as sum of powers of two. Similarly, for a given array of size N, we can maintain an array BIT[] such that, at any index we can store sum of some numbers of the given array. This can also be called a partial sum tree.

Let's use an example to understand how BIT[] stores partial sums.

```
//for ease, we make sure our given array is 1-based indexed
int a[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16};
```





The above picture shows the binary indexed tree, each enclosed box of which denotes the value BIT[index] and each BIT[index] stores partial sum of some numbers.

Notice

```
                {            a[x],                  if x is odd
BIT[x] =                      a[1] + ... + a[x],    if x is power of 2
                }
```

> To generalize this every index **i** in the BIT[] array stores the cumulative sum from the index **i** to **i - (1<<r) + 1** (both inclusive), where **r** represents the last set bit in the index **i**

Sum of first 12 numbers in array a[] = **BIT[12] + BIT[8]** = (a[12] + … + a[9]) + (a[8] + … + a[1])

Similarly, **sum of first 6 elements** = **BIT[6] + BIT[4]** = (a[6] + a[5]) + (a[4] + … + a[1])

Sum of first 8 elements = **BIT[8]** = a[8] + … + a[1]

Let's see how to construct this tree and then we will come back to querying the tree for prefix sums. BIT[] is an array of size = 1 + the size of the given array a[] on which we need to perform operations. Initially all values in BIT[] are equal to 0. Then we call update() operation for each element of given array to construct the Binary Indexed Tree. The update() operation is discussed below.

```
void update(int x, int delta)       //add "delta" at index "x"
{
    for(; x <= n; x += x&-x)
          BIT[x] += delta;
}
```

Its okay if you are unable to understand how the above update() function works. Let's take an example and try to understand it.

Suppose we call **update(13, 2)**.

Here we see from the above figure that **indices 13, 14, 16 cover index 13** and thus we need to add 2 to them also.

Initially x is 13, we update BIT[13]

```
BIT[13] += 2;
```

Now isolate the last set bit of x = 13(1101) and add that to x , i.e. x += x&(-x)

Last bit is of x = 13(1101) is 1 which we add to x, then x = 13+1 = 14, we update BIT[14]

```
BIT[14] += 2;
```

Now 14 is 1110, isolate last bit and add to 14, x becomes 14+2 = 16(10000), we update BIT[16]

```
BIT[16] += 2;
```

In this way, when an update() operation is performed on index x we update all the indices of BIT[] which cover index x and maintain the BIT[].

If we look at the for loop in update() operation, we can see that the loop runs at most the number of bits in index x which is restricted to be less or equal to n (the size of the given array), so we can say that the **update operation takes at most O(log2(n)) time**.

How to **query** such structure for prefix sums? Let's look at the query operation.

```
int query(int x)      //returns the sum of first x elements in given array a[]
{
     int sum = 0;
     for(; x > 0; x -= x&-x)
         sum += BIT[x];
     return sum;
}
```

The above function query() returns the sum of first x elements in given array. Let's see how it works.

Suppose we call **query(14)**, initially **sum = 0**

x is 14(1110) we add BIT[14] to our sum variable, thus **sum = BIT[14]** = (a[14] + a[13])

now we isolate the last set bit from x = 14(1110) and subtract it from x last set bit in 14(1110) is 2(10), thus x = 14 – 2 = 12

we add BIT[12] to our sum variable, thus **sum = BIT[14] + BIT[12]** = (a[14] + a[13]) + (a[12] + … + a[9])

again we isolate last set bit from x = 12(1100) and subtract it from x

last set bit in 12(1100) is 4(100), thus x = 12 – 4 = 8

we add BIT[8] to our sum variable, thus

sum = BIT[14] + BIT[2] + BIT[8] = (a[14] + a[13]) + (a[12] + … + a[9]) + (a[8] + … + a[1])

once again we isolate last set bit from x = 8(1000) and subtract it from x

last set bit in 8(1000) is 8(1000), thus x = 8 – 8 = 0

since x = 0, the for loop breaks and we return the prefix sum.

Talking about complexity, again we can see that the loop iterates at most the number of bits in x which will be at most n(the size of the given array). Thus the **query operation takes O(log2(n)) time* *.

Here's the **full program** to solve efficiently, the problem that we discussed at the start of this article.

✓

```
int BIT[1000], a[1000], n;
void update(int x, int delta)
{
       for(; x <= n; x += x&-x)
         BIT[x] += delta;
}
int query(int x)
{
      int sum = 0;
      for(; x > 0; x -= x&-x)
         sum += BIT[x];
      return sum;
}

int main()
{
      scanf("%d", &n);
      int i;
      for(i = 1; i <= n; i++)
      {
            scanf(“%d”, &a[i]);
            update(i, a[i]);
      }
      printf(“sum of first 10 elements is %d\n”, query(10));
      printf(“sum of all elements in range [2, 7] is %d\n”, query(7) –
query(2-1));
      return 0;
}
```

When to use Binary Indexed Tree?

Before going for Binary Indexed tree to perform operations over range, one must confirm that the operation or the function is:

Associative. i.e f(f(a, b), c) = f(a, f(b, c)) this is true even for seg-tree

Has an inverse. eg:

1. addition has inverse subtraction (this example we have discussed)
2. Multiplication has inverse division
3. gcd() has no inverse, so we can't use BIT to calculate range gcd's
4. sum of matrices has inverse
5. product of matrices would have inverse if it is given that matrices are degenerate i.e. determinant of any matrix is not equal to 0

Space Complexity: **O(N)** for declaring another array of size N

Time Complexity: **O(logN)** for each operation(update and query as well)

Applications:

1. Binary Indexed trees are used to implement the arithmetic coding algorithm. Development of operations it supports were primarily motivated by use in that case.
2. Binary Indexed Tree can be used to count inversions in an array in O(N*logN) time.

Related problems:

1. Code Monk problems
2. Bubble Sort Graph
3. INVCNT
4. ORDERSET

5. DQUERY

Like 16 Tweet

COMMENTS (60) SORT BY: Relevance

Login/Signup to Comment

SANKET SAPKAL 5 years ago
it should be "i - r + 1" instead of "i - (1<<r) + 1"
0 votes

Chandan Mittal Author Edited 5 years ago
I am glad you raised the doubt, but it's "i - (1<<r) + 1" only.. if it would be "i - r + 1" the complexity won't turn out to be log2(n).
You may google for more information. :)
3 votes

preetham kowshik 4 years ago
I am little confused with "i - (1<<r) + 1" . I took i =14 -> 1110 ,its last set bit r = 0010= 2 and 1<<2 = 4 and 14 - 4 +1 = 11.So cumulative sum should be 11 to 14. But in the figure it shows bit[14] = a[13] + a[14]. Please let me know where I am going wrong . Thank You.
1 vote

uday kanth reddy 4 years ago
indexing starts from zero not 1. So it will be 1<<1 = 0010 = 2. And here r indicates the index of last set bit. not the value of the last set bit.
6 votes

Govind S a year ago
here you have defined r already as 1<<(index of last bit).r is not the index of last bit
0 votes

Apoorva Rathore 5 years ago
this is a real good article!!! thanks
9 votes

dinnu.keru Dinesh Kuruba 5 years ago
what is the difference between bit array and segment trees?
0 votes

Apoorva Rathore 5 years ago
well, first of all the space that a segment tree array takes is larger than BIT array...second, in Bit there are some special benchmarks (powers of 2)...
2 votes

Prem Kumar Tiwari 4 years ago
The BIT is specifically used for Dynamic - Cumulative stuffs, whereas the Segment tree is more general and can be used in all those situations where, the divide and conquer paradigm can be applied over queried interval.
All the jobs doable by BIT can essentially be done by segment tree but not vice-versa. :)
1 vote

?

uday kanth reddy 4 years ago
I wonder whether you have read the post. Everything is explained in great detail.
▲ 0 votes

Aditya Maheshwari 6 years ago
update should take difference of newvalue and oldvalue.
▲ 0 votes

Chandan Mittal Author 6 years ago
mentioned it as comment in update function. thanks for notifying. :)
▲ 2 votes

ABHISHEK RATHEE 5 years ago
Awesome explanation
▲ 2 votes

Aditya Ambikesh 3 years ago
It should be
sum = BIT[14] + BIT[12] + BIT[8] = (a[14] + a[13]) + (a[12] + … + a[9]) + (a[8] + … + a[1])
not sum = BIT[14] + BIT[2] + BIT[8] = (a[14] + a[13]) + (a[12] + … + a[9]) + (a[8] + … + a[1])
▲ 0 votes

Akash Kandpal 2 years ago
shi pakde hai xD
▲ 2 votes

Hitesh Garg 6 years ago
thanx for this beautiful xplanation .
▲ 1 vote

Chandan Mittal Author 6 years ago
I am glad you liked it. :)
▲ 0 votes

Arun Prasad 6 years ago
this is point update range query?
can you give code for range update range query as well ?
▲ 0 votes

Chandan Mittal Author 6 years ago
point sum, range update -->
https://github.com/calmhandtitan/algorepo/blob/master/DataStructures/BIT/BIT2.c
range sum, range update -->
https://github.com/calmhandtitan/algorepo/blob/master/DataStructures/BIT/BIT3.c
▲ 1 vote

Arun Prasad 6 years ago
what is the difference between PSRU and the above code in notes
update(i, a[i]);
update(i+1, -a[i]);
why have you done this there and not here?
▲ 0 votes

Chandan Mittal Author Edited 5 years ago
the basic binary indexed tree structure is discussed in this note, while we can cleverly use operations on this basic structure to perform many other operations

like point query with range update, range sum with range query.
Everywhere the binary indexed tree remains same, its just how we use it for our purpose.
▲ 0 votes

Arun Prasad 6 years ago
Thanks !
▲ 0 votes

Gural Nuriyev 5 years ago
Can you please provide some explanations to range sum, range update?
▲ 0 votes

Gural Nuriyev 5 years ago
Codemonk is Rock'n Rolling!
▲ 0 votes

Chandan Mittal Author 5 years ago
Indeed it was meant to be that way. :)
▲ 1 vote

TiyaChu Max 5 years ago
Maybe you could rename the variable 'val' to 'delta' or something, perhaps one would find it easier then to understand the whole idea just with your nice schematic and code.
▲ 1 vote

Chandan Mittal Author 5 years ago
Thanks for notifying. I have renamed the variable from "val" to "delta". :)
▲ 0 votes

Rajat Chauhan 5 years ago
what if given index is in form of binary string that too very large such that its decimal form cannot be assigned to long long int data type
▲ 1 vote

Dushyant Singh 5 years ago
Please update the example given in beginning. Update function should add value v to that index, NOT update it. Because in BIT we are adding value.
▲ 1 vote

Ritu Raj 4 years ago
Very nice article,thanks!
▲ 1 vote

Harsh Agarwal 4 years ago
Why inverse of the operation/function needs to exist for using BIT?
▲ 0 votes

Omkar Manjrekar 4 years ago
Query operation finds prefix sum for a K index. In order to have range sum you need two prefix sums for L and for R indices. And if you perform subtraction between the two only then you get range sum. That is why we need inversion.
▲ 1 vote

Hemakshi Sachdev 4 years ago
excellent article..... really well explained :)

▲ 1 vote

Omkar Manjrekar Edited 4 years ago
One more data structure to the arsenal! Thanks to you :D
▲ 1 vote

Robin Thakur 4 years ago
nice post.Thnx
▲ 1 vote

rushikesh fanse 4 years ago
great article man. i was struggling to understand the reason behind bit manipulation
▲ 1 vote

Sanjeeda Vempalli 3 years ago
Awesome explanation...thank u
▲ 1 vote

Akash Kandpal 2 years ago
Nice work. If possible could you provide a clean code for range sum range update BIT? Not the one which you provided below as that's not that clean :p and thanks for making some some issues cleared. Thanks
▲ 1 vote

Akash Dubey a year ago
That's a very great explanation better than a video
▲ 1 vote

satyam kumar a year ago
I tried to understand this concept from many articles, but nothing was making sense. But, when i read this, I understood completely. Thanks for putting effort for writing such explanation in lucid manner.
▲ 1 vote

Meet Shah a year ago
Great Article!!!
▲ 1 vote

Vignesh Mahalingam 6 years ago
Why there is no "code-monk" tag? Is this a part of code monk series or not?
▲ 0 votes

> Chandan Mittal Author 6 years ago
> tags added. thanks for notifying. :)
> ▲ 0 votes

Vignesh Mahalingam 6 years ago
Can you also give links for example problems?
▲ 0 votes

Himanshu Bhoria 6 years ago
how do you come up with values of BIT[x] where x is not a power of 2 and also not an odd number?
▲ 0 votes

> Chandan Mittal Author 6 years ago

values of such indexes can not be generalized, else I would have included the formula. :)
0 votes

Himanshu Bhoria 6 years ago
It can be generalized :P
http://www.algorithmist.com/index.php/Fenwick_tree
0 votes

Chandan Mittal Author 6 years ago
ohh that's true. I forgot it while writing the note. Thanks for notifying. :)
0 votes

Harshal Garg 5 years ago
kindly update it.
0 votes

Chandan Mittal Author 5 years ago
Hey harshal, it's already been added. Please check again.
0 votes

Mritunjay Dubey 6 years ago
can you give some sample problems ?
0 votes

Sujit yadav 6 years ago
“%d” is not equal to "%d" take that.... :p ur code isnt compiling :D
0 votes

Arjun Datta 6 years ago
Please provide some sample problems for practice.
0 votes

Arpit Srivastava 6 years ago
Very well explained Note
0 votes

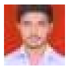
Chandan Mittal Author 6 years ago
thanks :)
0 votes

Ishpreet Singh 6 years ago
Thanks for this explanation.....Its quite Useful..
0 votes

Chandan Mittal Author 6 years ago
I am glad you find it useful :)
0 votes

Hitesh Kumar Regar 5 years ago
#python code
http://haikent.blogspot.in/2016/08/blog-post.html
0 votes

Surya Kahar a year ago
Thanks
0 votes



Mukul Kumar 9 months ago
Best editorial ever
▲ 0 votes

AUTHOR

Chandan Mittal
Software Development Eng...
Mumbai
1 note

TRENDING NOTES

Python Diaries Chapter 3 Map | Filter | For-else | List Comprehension
written by Divyanshu Bansal

Bokeh | Interactive Visualization Library | Use Graph with Django Template
written by Prateek Kumar

Bokeh | Interactive Visualization Library | Graph Plotting
written by Prateek Kumar

Python Diaries chapter 2
written by Divyanshu Bansal

Python Diaries chapter 1
written by Divyanshu Bansal

more ...

Competitive Programming

Start a Programming Club

Practice Machine Learning

?