P THINK FAST S

Competitive Programming

From Problem 2 Solution in O(1)

Data Structures

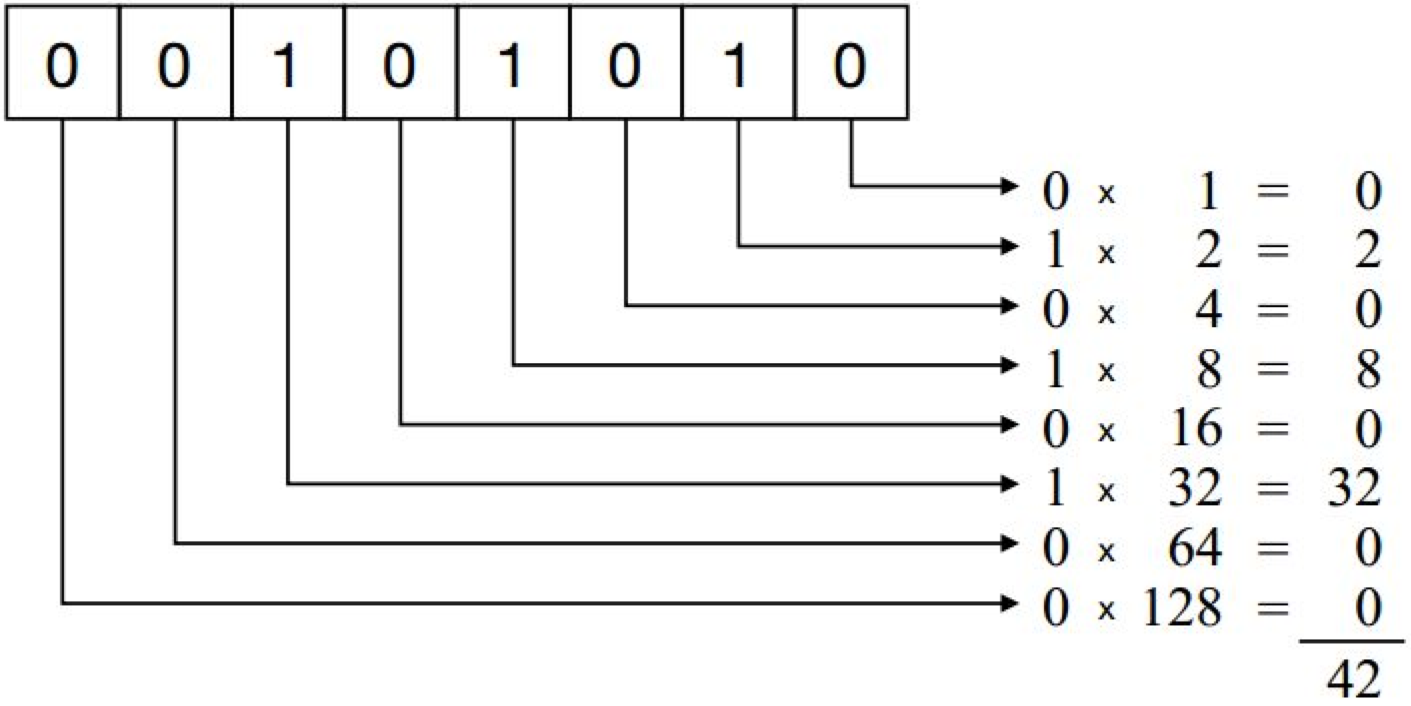
Binary Indexed Tree (Fenwick)

Mostafa Saad Ibrahim
PhD Student @ Simon Fraser University

Background

- BIT is based on binary properties
- Let’s revise some binary properties first

Binary Representation

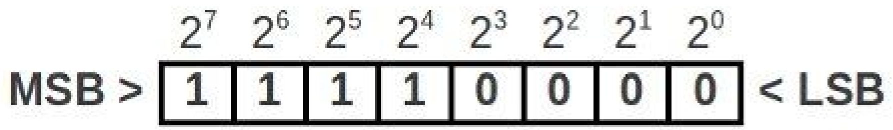


Src: http://www.c-jump.com/bcc/common/Talk2/Cxx/BitByteHexASCII/BitByteHexASCII.html

Removing bits from mask

- **mask** = think in integer **bits**
- Assume we have numbers X, Y
- If **for every position** in Y with 1 AND X has 1
  - X = 10010100
  - Y = 00010100
- X - Y removes all Y 1s from X
- Another longer/general way to do so:
  - X & ~Y
  - 10010100 &
  - 11101011 =
  - 10000000

Least/Most Significant Bit



Src: http://www.electronique-et-informatique.fr/Digit/images/MSB_LSB.gif
Src: http://tronixstuff.com/wp-content/uploads/2011/05/binnum.jpg

Least Significant ONE Bit

1	$\boxed{1}$	0	0	0	0	0	0	192
1	0	1	0	$\boxed{1}$	0	0	0	168
0	0	0	0	0	0	0	0	0
0	1	1	0	0	$\boxed{1}$	0	0	100

Let’s call it **the last bit**

Src: http://billconner.com/techie/binary-2.gif

One's Complement Representation

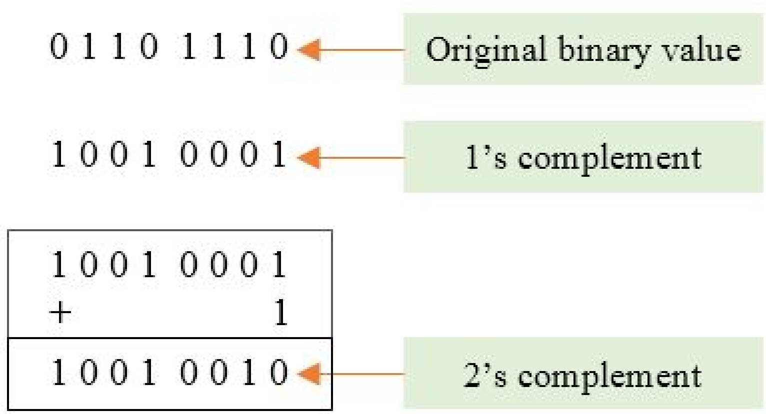
The 1's complement of a binary number is just the inverse of the digits. To form the 1's complement, change all 0's to 1's and all 1's to 0's.

For example, the 1's complement of 11001010 is
00110101

Src: http://district.bluegrass.kctcs.edu/kevin.dunn/files/Arithmetic_Operations_and_Circuits/paste_image3.png

Two's Complement Representation

- Start to flip AFTER the "last bit"
- **-number** = 2's complement of number
- One way to compute **manually**:
  - Get 1's complement...then add 1



Src: http://2.bp.blogspot.com/-7pIXtOAOaq0/VdA5fZ_-EmI/AAAAAAAACyY/GPgQwR-NOqA/s1600/twos%2Bcomplement%2Bof%2Bbinary%2Bvalue.PNG

Two's Complement Representation

Number in decimal	Number in two's complement binary
5	0000000000000101
4	0000000000000100
3	0000000000000011
2	0000000000000010
1	0000000000000001
0	0000000000000000
-1	1111111111111111
-2	1111111111111110
-3	1111111111111101
-4	1111111111111100
-5	1111111111111011

Src: http://patentimages.storage.googleapis.com/WO2002095573A1/imgf000007_0001.png

Removing Last Bit

- Get last bit using **index & -index**
  - +20 = 00010100
  - -20 = **11101**100
  - 20 & -20 = 00000100
- Remove last bit
  - Get it...subtract it
  - **index - (index & -index)**
  - 00010100 - 00000100 = 00010000
- We can remove last bit using other ways too

Integer as sums of powers of 2

Binary Expansion

- Any integer N can be written as a sum of powers of 2.
- Start with the largest $2^k \le N$, subtract of it, and repeat the process.
- $147 = 128 + 19$; $19 = 16 + 3$; $3 = 2 + 1$

  So

  $147 = 128 + 16 + 2 + 1$ = **010010011**

  with $k = 7, 4, 1, 0$

Src: http://images.slideplayer.com/5/1546412/slides/slide_22.jpg

Our problem

- Let's move to our problem
- Given an array of integer N
  - Assume index 0 always will be 0 (NOT in use)
- 2 query types:
  - Add value **val** to position **index**
  - **Sum** values from 1 to index
- Segment Tree can be used to such problem
  - O(N) preprocess, O(NlogN) queries, O(nlogn) memory
- BIT has a better memory order (shorter code)
  - O(n) memory + O(NlogN) queries

Problem Example

0	1	2	3	4	5	6	7	8	9	10	11

xx	3	2	**-1**	6	5	4	-3	3	7	2	3

- Accumulative Sum (1, 3): 3 + 2 - 1 = 4
- Accumulative Sum (1, 5): 3 + 2 - 1 + 6 + 5 = 15
- Add: index 3, value = 5

xx	3	2	**4**	6	5	4	-3	3	7	2	3

- Accumulative Sum (1, 3): 3 + 2 + 4 = 9
- Accumulative Sum (1, 5): 3 + 2 + 4 + 6 + 5 = 21

Motivation

- Integer = Sum of Powers of 2
- Accumulative Sum = Sum of **Sub sums**
- Recall: 147 = 128 + 16 + 2 + 1
- Think in accumulative sum (1 to 147)
  - Sum of last 1 number +
  - Sum of next 2 numbers +
  - Sum of next 16 numbers +
  - Sum of next 128 numbers
- Sum(1,147) =
  - Sum(147,147) + Sum(146,145) + Sum(144,129) + Sum(128,1)
  - 147 ⇒ positions {147, 146, 144, 128} with ranges {1, 2, 16, 128}

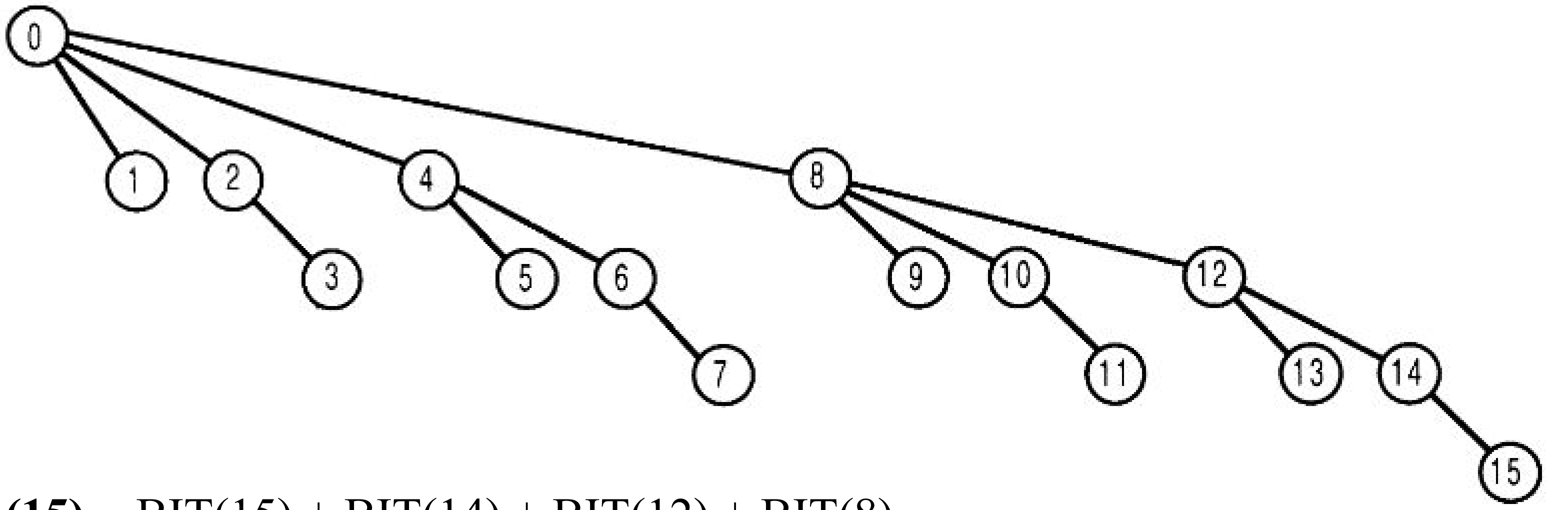
Motivation

- To get starting positions fast? Remove last bit
  - 147 = 010010011 [remove last 1 bit]
  - 146 = 010010010 [remove last 1 bit]
  - 144 = 010010000 [remove last 1 bit]
  - 128 = 010000000 [remove last 1 bit]
  - 0 = DONE
- How to interpret:
  - 147 responsible for range 147 to > 146
  - 146 responsible for range 146 to > 144
  - 144 responsible for range 144 to > 128
  - 146 responsible for range 128 to > 0

Binary Indexed Tree

- Create a new array of Length N, name it BIT
- BIT[position] = sum of its responsible range
- Then For each Query
  - **Sum(147)**= BIT(147) + BIT(146) + BIT(144) + BIT(128)
  - That is: 4 steps only to get the answer
  - **Sum(144)** = BIT(144) + BIT(128)
  - **Sum(15)** = BIT(15) + BIT(14) + BIT(12) + BIT(8)
  - Recall: 1111 = 1111, 1110, 1100, 1000, 0
  - **Sum(11)** = BIT(11) + BIT(10) + BIT(8)
  - Recall 1011: 1011, 1010, 1000, 0
  - **Sum(7)** = BIT(7) + BIT(6) + BIT(4) $\Rightarrow$ 111, 110, 100, 0

Binary Indexed Tree

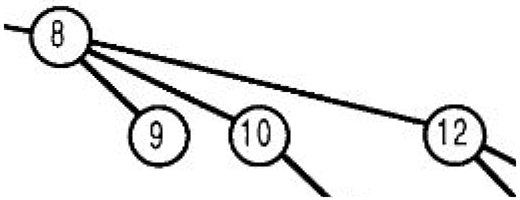


Sum(15) = BIT(15) + BIT(14) + BIT(12) + BIT(8)

E.g. node 12 has values: BIT[12] = val[12] + val[11]+val[9]
12 = 1100 ⇒ removing last 1 bit ⇒ 1000 = 8
Then parent of 12 ⇒ 8 (e.g. next closest position 12 is **not covering**)
Notice: we removed bit at position 2 ⇒ 12 covers 2^2 numbers = 12 - 8 = 4

Src: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.14.8917&rep=rep1&type=pdf

Binary Indexed Tree



Notice: 8 = 1000 => has 3 trailing zeros. Try to replace each 0 with 1

1001 = 9

1010 = 10

1100 = 12

# of trailing zeros = # children … child remove last bit => go to parent

Src: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.14.8917&rep=rep1&type=pdf

Get Interval Accumulation

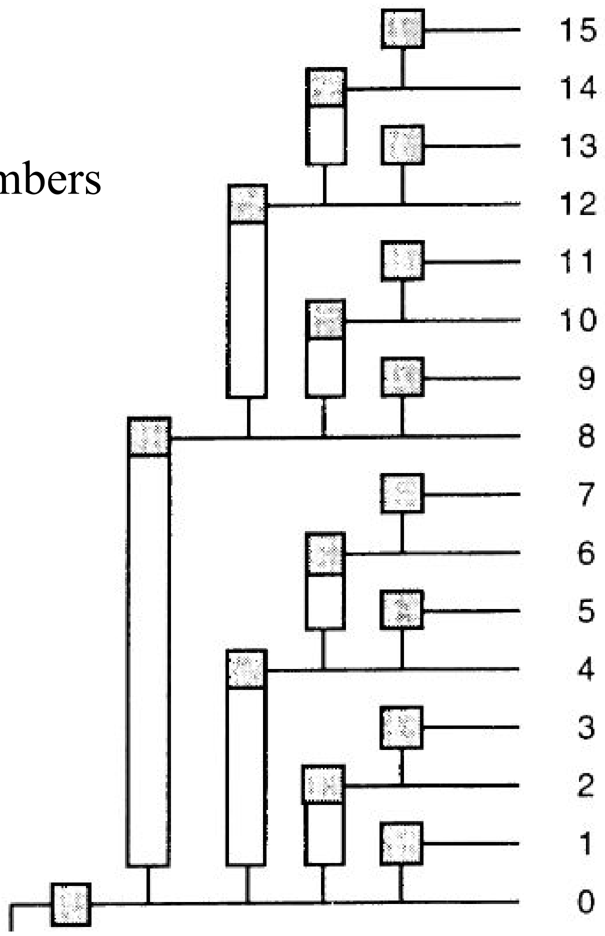
Sum(15) = BIT(15) + BIT(14) + BIT(12) + BIT(8)
= 1111 ⇒ 1110 ⇒ 1100 ⇒ 1000 ⇒ 0 = STOP

15 is **responsible for** 1 number, 14 for 2, 12 for 4, 8 for 8 numbers

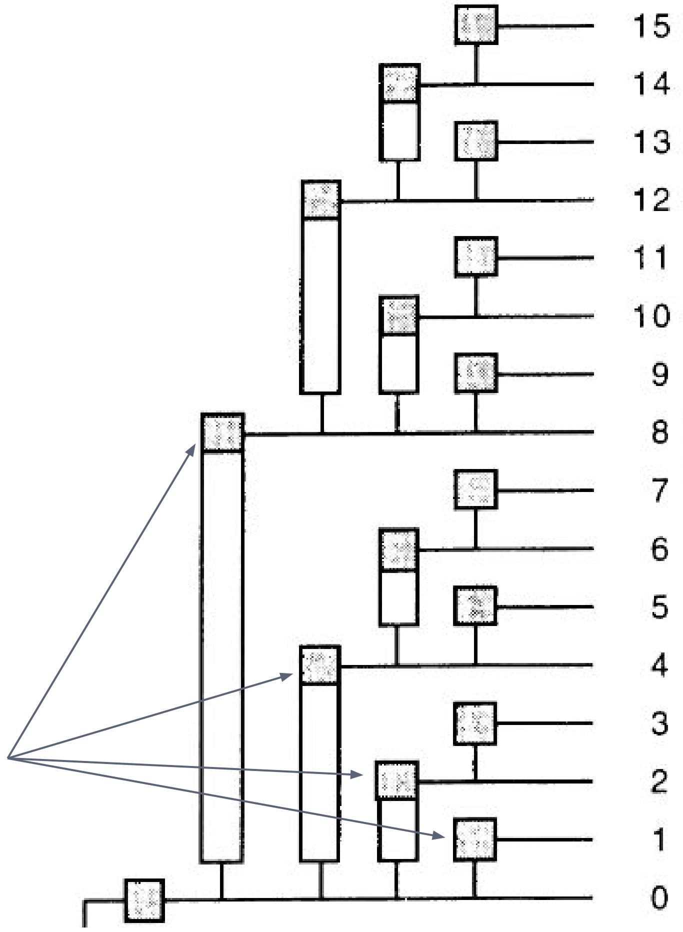
```
const int MAX_VAL = 30000;
int BITTree[MAX_VAL] = {0};

int getAccum(int idx){
    int sum = 0;

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



Updating position



Position 1: Covered by 4 intervals ⇒ 1, 2, 4, 8
Add -3 to 1 ⇒ add -3 to these 4 intervals

Given index, how to get **smallest** position **covering** it?
E.g. 1 ⇒ 2 6 ⇒ 8 10 ⇒ 12 13 ⇒ 14
Then 1 goes to 2...2 goes to 4..4 goes to 8 [recursive]

Updating position

- Recall given number **idx** it covers 2^r values
  - r is position of “last bit”
  - It covers numbers from **idx** to **idx – 2^r + 1**
- All following numbers cover 8 values
  - 0001000 $\Rightarrow$ r = 3 $\Rightarrow$ 2^3 = 8
  - 0101000
  - 1101000
  - 1111000
  - 1001000
- So our focus on “last bit”, NOT before that

Updating position

- 1000 covers 8 numbers
  - 1000 - 000 = 1000
  - 1000 - 001 = 0111
  - 1000 - 010 = 0110
  - 1000 - 011 = 0101
  - 1000 - 100 = 0100
  - 1000 - 101 = 0011
  - 1000 - 110 = 0010
  - 1000 - 111 = 0001
- Each of these 8 numbers covered by 1000
- But 1000 is NOT their smallest cover number

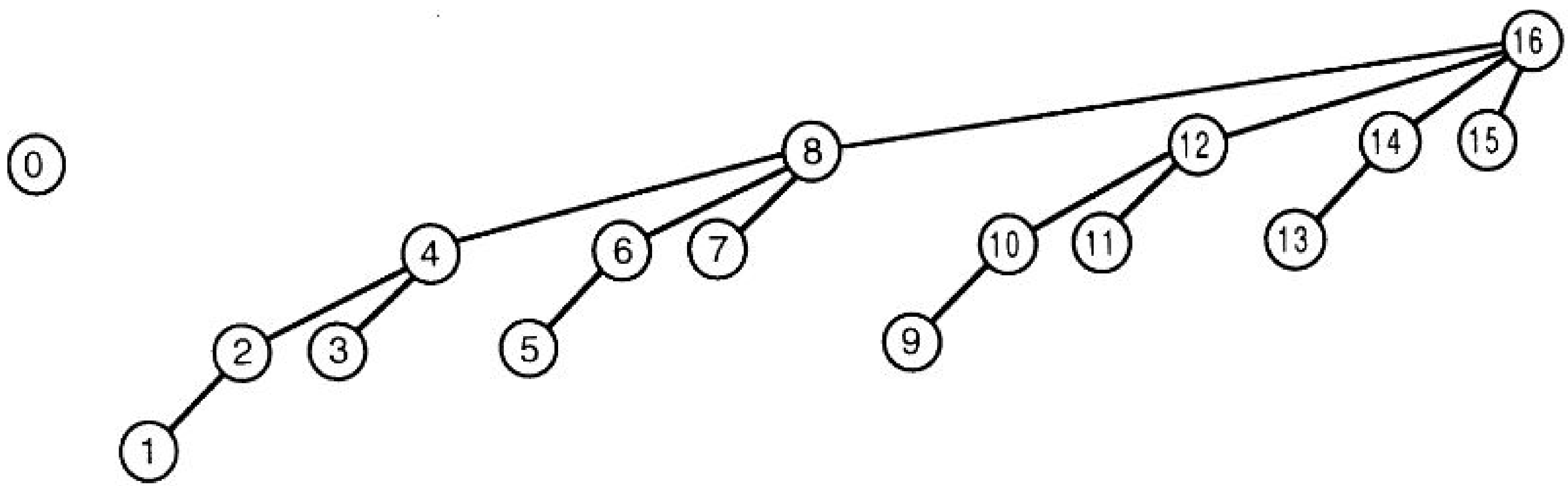
Updating position

- Let’s get who covers 4 = 0100
  - 4 has “last bit” at k = 2
  - When target number enumerate its 2^r, one contains 100
  - So we need to go at least 1 bit higher than k
  - E.g. re-set last bit k = 3 $\Rightarrow$ 1000 $\Rightarrow$ first one to cover 0100
- Let’s get who covers 5 = 0101
  - k = 0
  - We need target number to include our 1 at k = 0
  - The earlier one should exist in smallest coverer number
  - So again, shift k = 0 1 step to be in its enumeration
  - E.g. re-set last bit k = 1 $\Rightarrow$ 110. Note, 1000 also cover 5

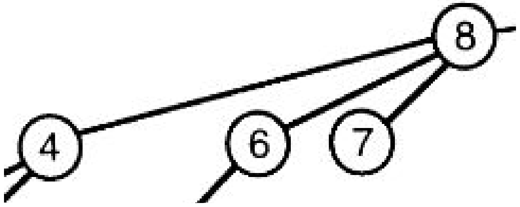
Updating position

- Let's get who covers 3 = 0011
  - "last bit" at k = 0
  - We need enumeration includes whole 11
  - So parent need to be a 1 before these 11
  - E.g. ⇒ 0**1**00
- So general rule
  - 100100001000 100100011100
  - 100100010000 100100100000
- How to get that number easily?
  - Just add 2^k ⇒ if one or more bits ⇒ shifted
  - E.g. 100100011100 + 000000000100 = 100100100000

Updating tree



Updating tree



Notice: 8 = 1000 => has 3 trailing zeros. Remove last bit, and add 1, 2, 3...trailing ones

0100 = 4

0110 = 6

0111 = 7

# of trailing zeros = # children

Updating tree

```
void add(int idx ,int val){
    while (idx < MAX_VAL) {
        BITTree[idx] += val;
        idx += (idx & -idx);
    }
}
```

Building initial tree from input: just iterate on input and add it to its position

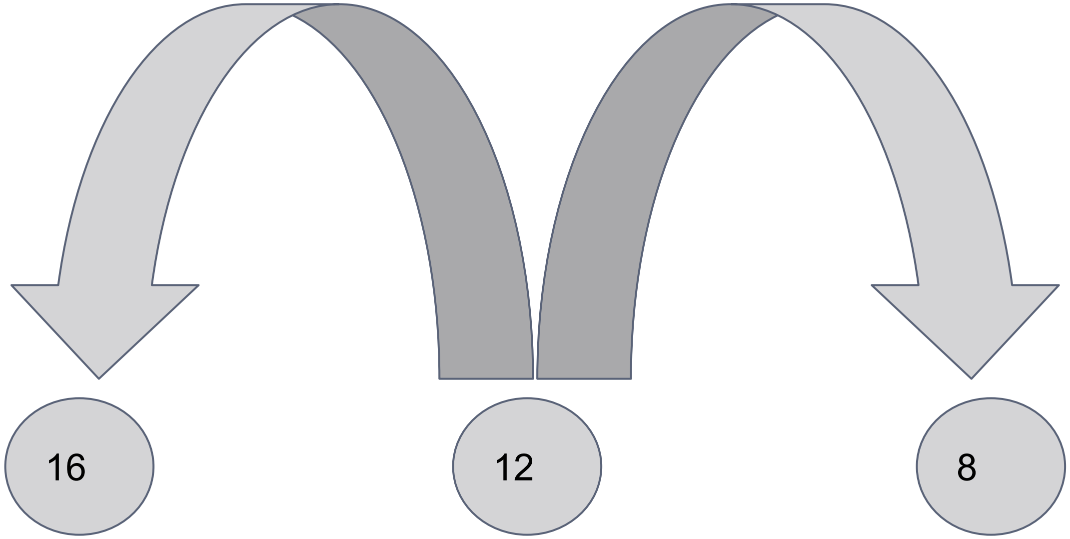
Index perspective

Smallest idx **cover** 12 is 16
16 responsible for: 16, 15,...1

idx += (idx & -idx)
12 += (12 & -12) ⇒ 16

12 is **responsible** down to 8+1
12 responsible for: 12, 11, 10, 9

idx -= (idx & -idx)
12 -= (12 & -12) ⇒ 8



Index with cumulative sum

- Assume we have array of values >= 0
- Accumulate it $\Rightarrow$ increasing sequence
- Find **first index** with accumulation >= value
- Given that it is increasing, using binary search is direct
- BIT maintain such accumulation by definition, if all values >= 0

Index with cumulative sum

```
int getValue(int idx) {
    return getAccum(idx) - getAccum(idx-1);
}

// Prerequisite : input array is positive
int getIdx(int accum) {
    int s = 1, e = MAX_VAL;

    while(s < e) {
        int midIdx = s + (e-s)/2, val = getAccum(midIdx);
        if(val >= accum)
            e = midIdx;
        else s = midIdx+1;
    }
    return s;       // s is the least x for which p(x) is true
}
```

2D BIT

- BIT can be extended to higher dimensions
  - In 2D: query add value to cell
  - Or Rectangle sum (0, 0) to (x, y)
- Define 2D array with MAX_X and MAX_Y
  - Think in each row (x indexed) as independent tree on y
  - X is responsible for set of trees
  - Y is responsible for a single tree
- Add val to bit2d[x][y]
- bit2d[x] is a 1D tree at position x
  - Update normally cross different bit2d[x][y]

2D BIT

```
void update(int x , int y , int val){
    while (x <= max_x){
        updatey(x , y , val);
        // this function should update array tree[x]
        x += (x & -x);
    }
}
```

The function **updatey** is the "same" as function **update**:

```
void updatey(int x , int y , int val){
    while (y <= max_y){
        tree[x][y] += val;
        y += (y & -y);
    }
}
```

References

- Paper
- TopCoder Article

تم بحمد الله

علمكم الله ما ينفعكم

ونفعكم بما تعلمتم

وزادكم علماً

Problems

- 2D Bit: http://codeforces.com/contest/341/problem/D
- SRM-310-D1-500