Binary Indexed Tree(Fenwick Tree)

1-based Fenwick Tree

Magic Operator: LowestOneBitMask (i & -i)

* i' = i - lowestOneBitMask(i) : calculates the next lower-indexntill which the current index is responsible to hold the sum. Lower-index i' does not contain upper-index i while storing the partial sum.

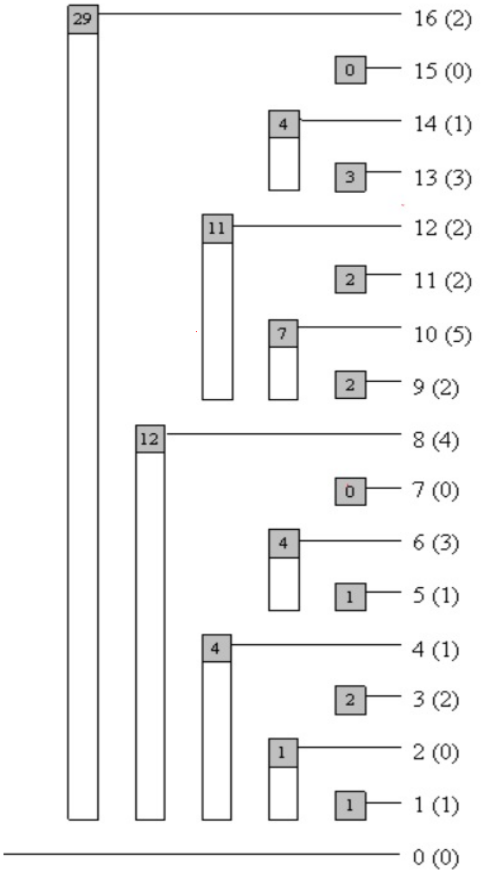
* i' = i + lowestOneBitMask(i) : calculates the next upper-index which is responsible to include the current element in its sum. Applying this recursively we can keep finding the next upper-index which is also responsible for 'i' which is like climbing the stairs.

* <u>Responsibilities of indexes in 1-based Fenwick tree</u>

1. odd-index is responsible only for itself. It means at odd index in fenwick tree we will find the original element itself taking 1-based transformation on input array.

2. all indexes of type 2^n, are responsible for all the indexes upto this point starting from 1. It means this index contains prefix sum of input-array upto this point.

3. all the even indexes are responsible for upto (lower-index + 1) that comes by dropping the lastSetBit of this index.



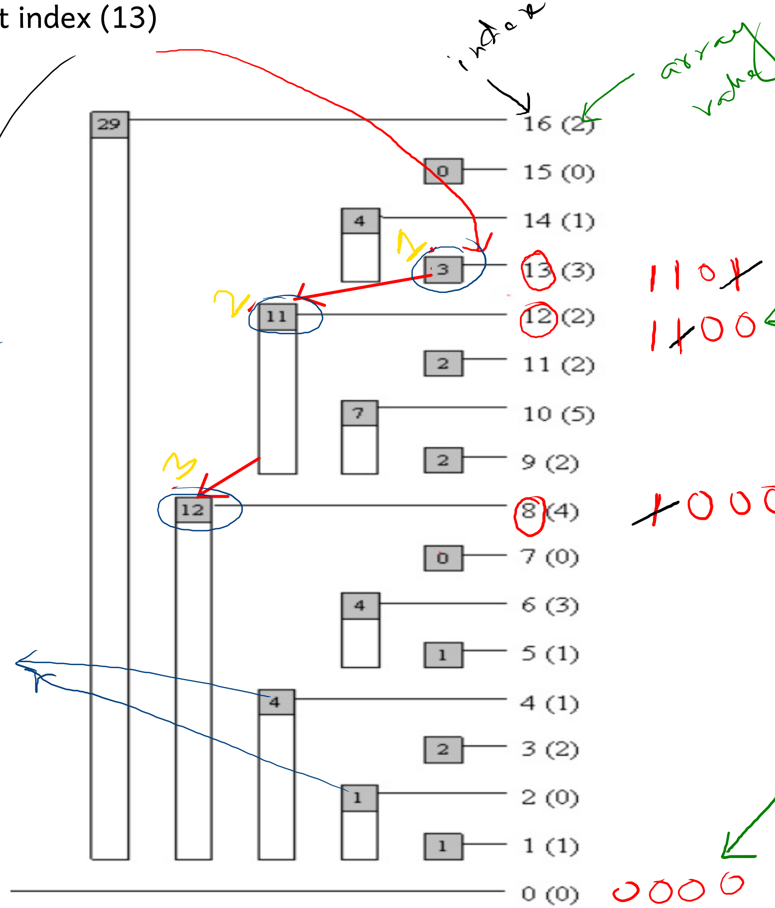
Tree Traversal :

Downward Traversal : By dropping lowest-one-bit position. Next index in traversal will always be even.

Upward Traversal : By adding 1 to lowest-one-bit position. Next index in traversal will always be even.

Note: Lower-odd-index is contained in immediate upper-even-index.

Downward traversal : Sum at index (13)



$Sum = \underline{3+11+12}$

$= 26$

$i' = i - (i \& -i)$

$110\cancel{1}$

$1\cancel{1}00$

$\cancel{1}000$

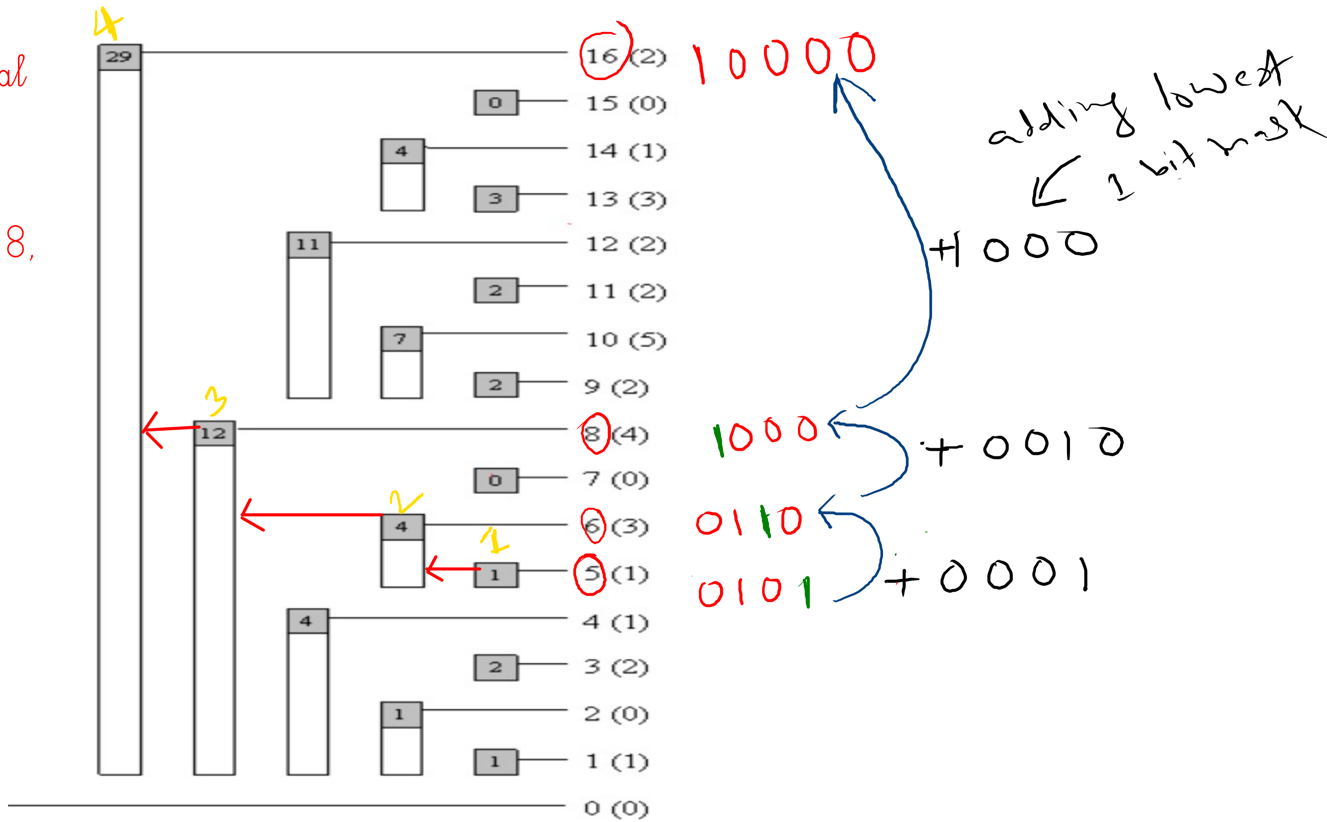
0000

Note : All the next traversal steps are even. Example: Traversal starting from index 13 includes indexes 12, 8, 0 i.e. all are evens.

Point Update at index 5

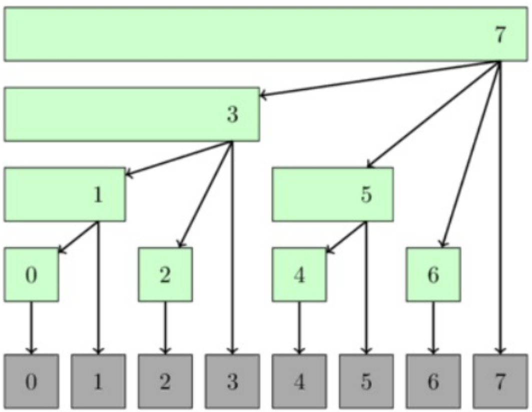
i' = i + lowestOneBitMask(i) : calculates the next upper-index which is responsible to include the current element in its sum. Applying this recursively we can keep finding the next upper-index which is also responsible for 'i' which is like climbing the stairs.

Note : All the next traversal steps are even. Example: Traversal starting from index 5 includes indexes 6, 8, 16 i.e. all are evens.



0-based fenwick tree

Magic Operator : i & (i+1) use to turn off the trailing 1's in a word, producing x if none (e.g. input: 10100111 output:10100000). This can be used to determine if an unsigned integer is of the form 2^n - 1.



Responsibilities of indexes in 0-based Fenwick tree

1. even-index is responsible only for itself. It means at even index in fenwick tree we will find the original element itself . This is because even number does not have any trailing 1's, so (i & (i+1)) will drop nothing.

2. all indexes of type (2^n -1), are responsible for all the indexes upto this point starting from 0. It means this index contains prefix sum of input-array upto this point. Dropping all the trailing 1's result in 0.

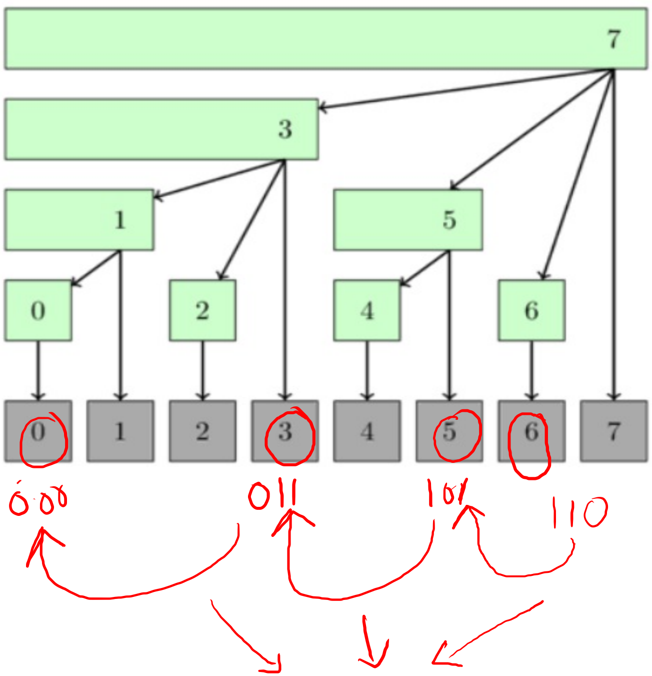
3. all the odd indexes are responsible for upto next lower-index that comes by dropping all the trailing 1's (i & (i+1)).

Downward traversal by dropping trailing 1's

Formula (DROP_TRAILING_1'S - 1) : $i' = (i\&(i+1)) - 1$.

All the next lower-index during traversal will be odd.

Example : Calculating sum at index 6



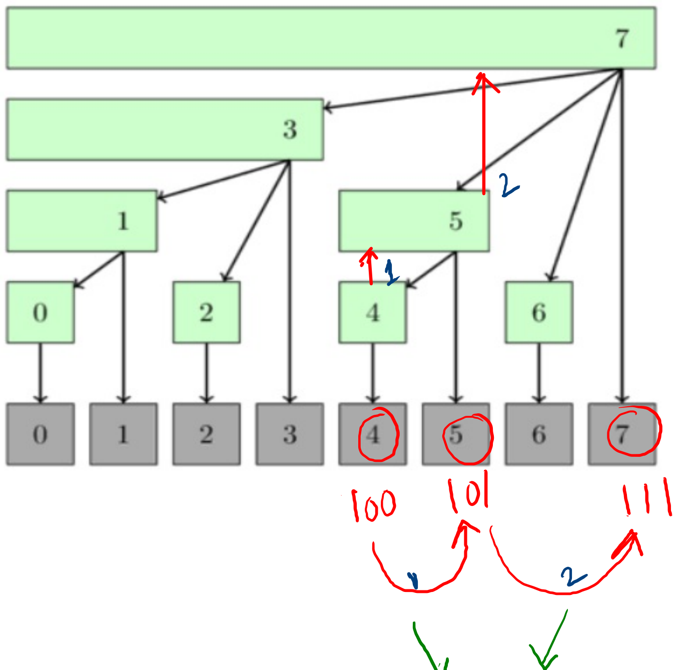
DROP_TRAILING_1'S - 1

Upward traversal by dropping trailing 1's

Formula (SET_RIGHTMOST_0_BIT): $x \mid (x + 1)$: use to turn on the rightmost 0-bit in a word, producing all 1's if none (e.g., input: 10100111 output:10101111)

All the next upper-index during traversal will be odd.

Example : pointUpdate at index 4



SET_RIGHTMOST_0_BIT