Binary Indexed Tree (Fenwick Tree)

Prof. Rhadamés Carmona

Last update: 21/11/2019

Content

- Motivation
- Operations
- Implementation
- Exercises
- Task
- References

Motivation

- Suppose we have n values, and we want to: [1] update one of the values, [2] get the sum of the first k<=n elements
- **First solution**: if we place the elements in an array of n positions, update is simply A[i]=x, O(1), and get the sum is O(n)
- **Second solution**: if we store the accumulated in another array B[i]=A[0]+...+A[i], get the sum is O(1), but update it is O(n)

Motivation

- **With Fenwick trees**, both operations are O(log)
- Fenwick trees are efficient for manipulating frequencies and frequency ranges
- Proposed by Peter Fenwick in 1994
- It is based in "associate" the sum of n terms in at most log(n) groups of sums

Operations

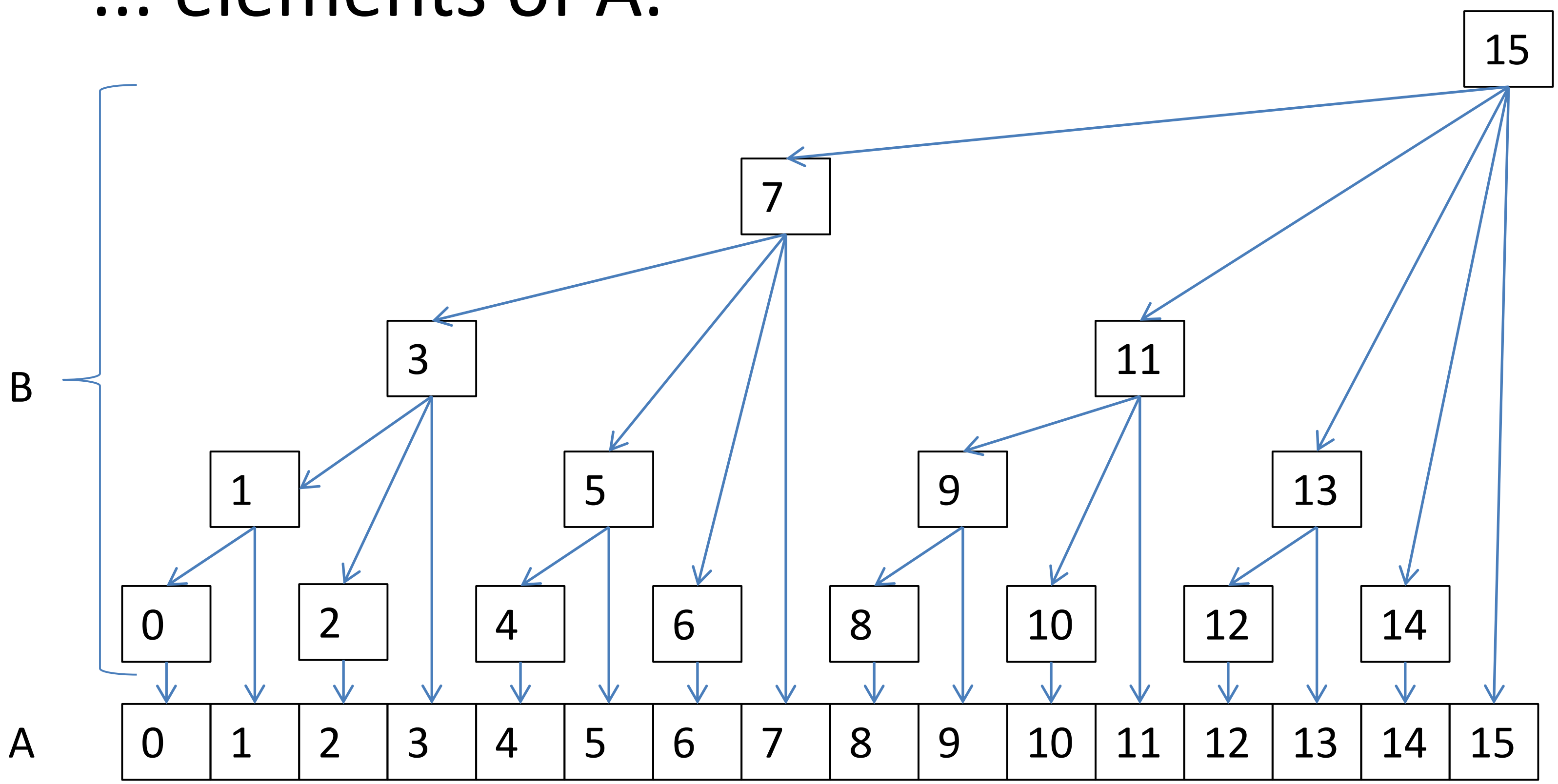
- getSum(i): gets the sum up to position i
- Increment(i,x): increments the frequence of the i-th element with the value x
- getFreq(i): reduced to getSum(i)-getSum(i-1)
  We may store in other array the individual frequencies (and not sums) to be O(1)
- getRange(i,j): If we want the sum of values in a range of elements i..j, it is reduced to getSum(j)-getSum(i-1)

Implementation

- Input: a frequencies array A
- Attribute: array B of n elements, B[0..n-1]. We will use 0-based index. There are other 1-based index implementations
- B[i] =
  - If i in binary is a sequence of 1s, it contains the sum of the first i elements: A[0].. A[i]
  - Otherwise, it contains the sum of the elements A[g(i)]+...+A[i], where g(i) is the number i turning off the continuous sequence of 1s (if any) starting at the least significant bit; g(i) = i & (i+1)

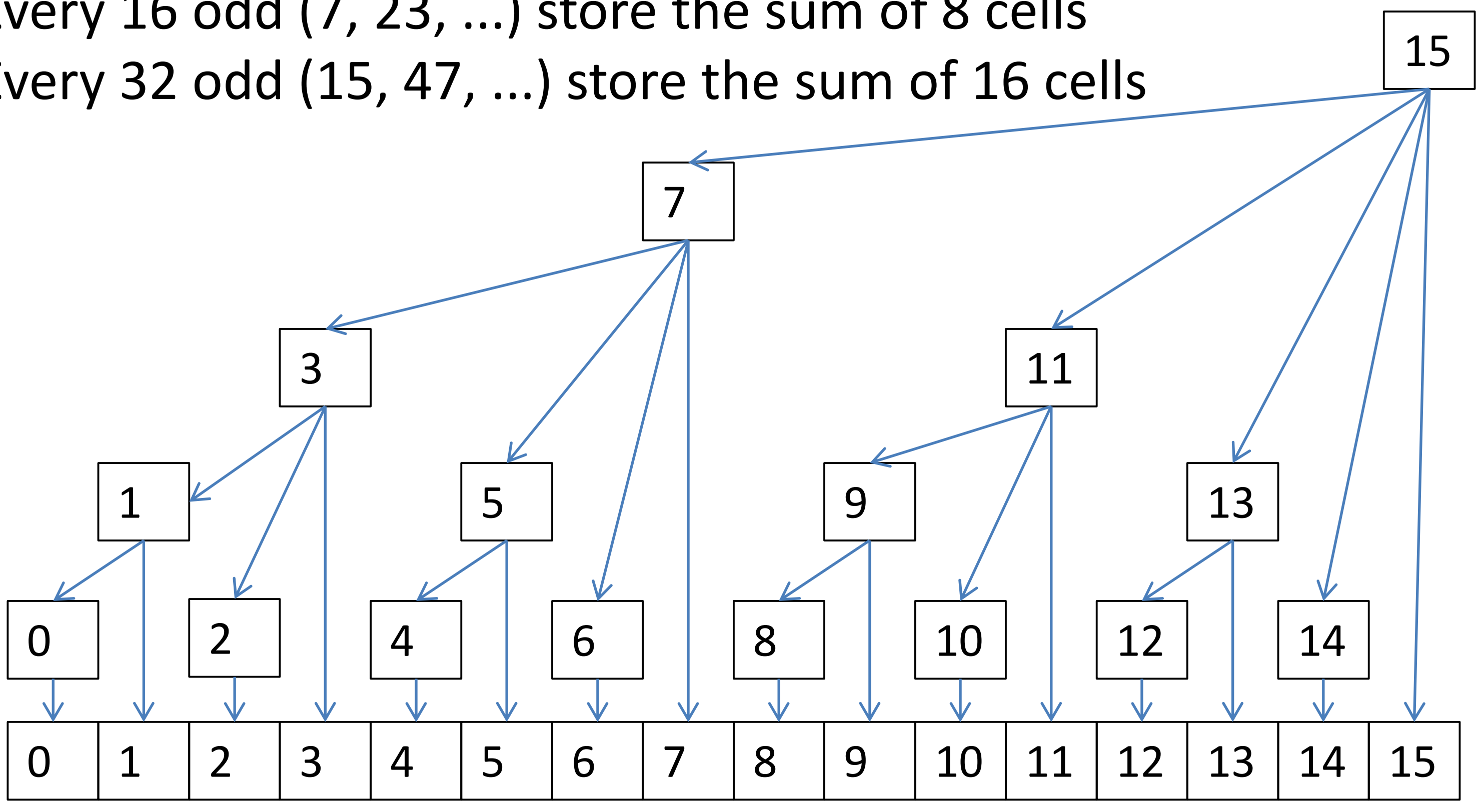
Implementation

- Each element of B can store the sum of 1, 2, 3, ... elements of A.



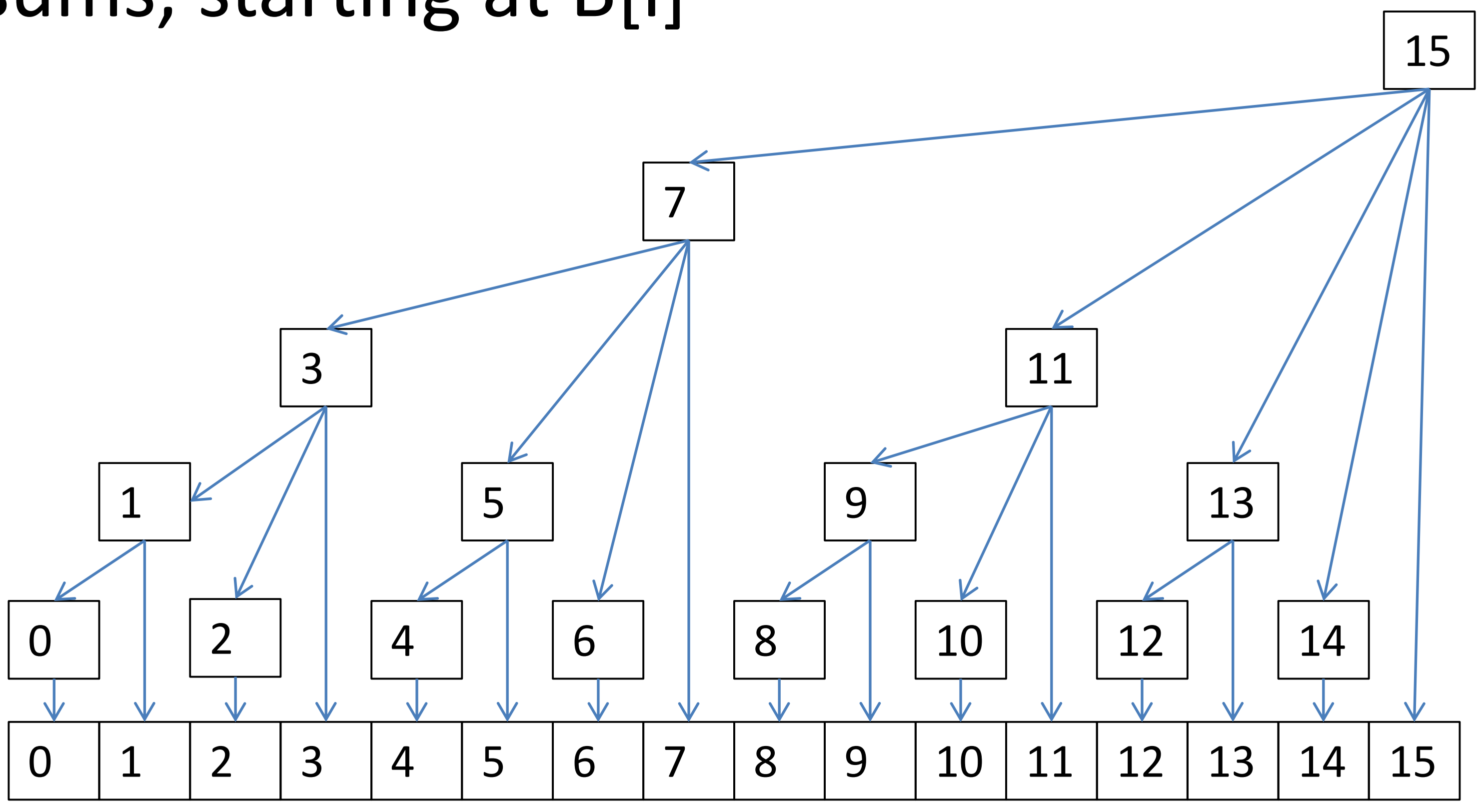
Implementation

- Pairs store only 1 cell value
- Every 4 odd (1, 5, ...) store the sum of 2 cells
- Every 8 odd (3, 11, ...) store the sum of 4 cells
- Every 16 odd (7, 23, ...) store the sum of 8 cells
- Every 32 odd (15, 47, ...) store the sum of 16 cells



Implementation: getSum(i)

- Given a number i, we look for the required sums, starting at B[i]



getSum(12) = B[12] +B[11]+B[7]

Implementation: getSum(i)

- **Example**: taking the 14th 1110_2
- We see that it is even, so we know that B[i] stores an element.
- We jump to B[13] = B[1101_2], where the 1's string is one element (stores the sum of 2 elements).
- We jumped to B[11] = B[1011_2]. The follow line is two 1's (stores the sum of 4 elements).
- B[7]=B(111_2), which stores the sum of 8 elements.
- When subtracting 8, we get -1 and the cycle ends.
- Notes that prev(i) = g(i)-1 = i & (i+1) -1.

Implementation: getSum(i)

- **Worst case**: numbers 2^k-2, because we have to sum element which positions with a sequence of zero 1's, one 1, two 1's, there 1's, …
- **Best case**: numbers 2^k-1, because all the bits are "on", and we only have to access one element of B.

```
int getSum(int i) {
        int res = 0;
         while (i >= 0) {
                res += B[i];
                i = prev(i);
        }
        return res;
}

int prev(int i) {
        return g(i)-1;
}

Int g(int i) {
        return i & (i+1);
}
```

Implementation: increment(i,x)

- This function must update all items in B that include in their sum to A[i]
- We start by updating B[i]+=x; then the next element of B that includes A[i] is

  i=i|(i+1)

- The process continues until i exceeds the size of the array

Implementation: increment(i,x)

- If i=6 and n=16, update 6,7,15
- If i=0 and n=16, update 0,1,3,7,15
- i=0 is the worst case ➔ O(log)

```
void increment(int i, int x)
{
        while ( i < n )
        {
                B[i] += x;
                i |= i+1;
        }
}
```

Tasks

- TULIPNUM - Tulip And Numbers

https://www.spoj.com/problems/TULIPNUM/

- FENTREE - Fenwick Trees

https://www.spoj.com/problems/FENTREE/

- PLNDTREE - Palindrome in a Tree

https://www.spoj.com/problems/PLNDTREE/

- MATSUM - Matrix Summation

https://www.spoj.com/problems/MATSUM/

- DCEPC206 - Its a Murder!

https://www.spoj.com/problems/DCEPC206/

References

- https://cp-algorithms.com/data_structures/fenwick.html