Binary Indexed Tree (Fenwick Tree)

Prof. Rhadamés Carmona

Última revisión: 13/05/2019

Agenda

- Motivación
- Operaciones
- Implementación
- Ejercicios
- Tarea
- Referencias

Motivación

- Supongamos que tenemos n valores, y queremos: [1] actualizar uno de los valores, [2] obtener la suma de los primeros k<n elementos.
- Primera solución: si colocamos los elementos en un arreglo A de n posiciones, actualizar es A[i] = x, O(1), y obtener la suma es O(n).
- Segunda solución: si almacenamos el acumulado en otro arreglo B[i]=A[0]+…=A[i], obtener la suma es O(1), pero actualizar es O(n).

Motivación

- Con árboles de Fenwick, ambas operaciones son O(log).
- Lo árboles de Fenwick son eficientes para manipular frecuencias y rangos de frecuencias.
- Propuesta por Peter Fenwick en 1994.
- Se basa “asociar” la suma de n términos en a lo sumo log(n) grupos de sumandos.

Operaciones

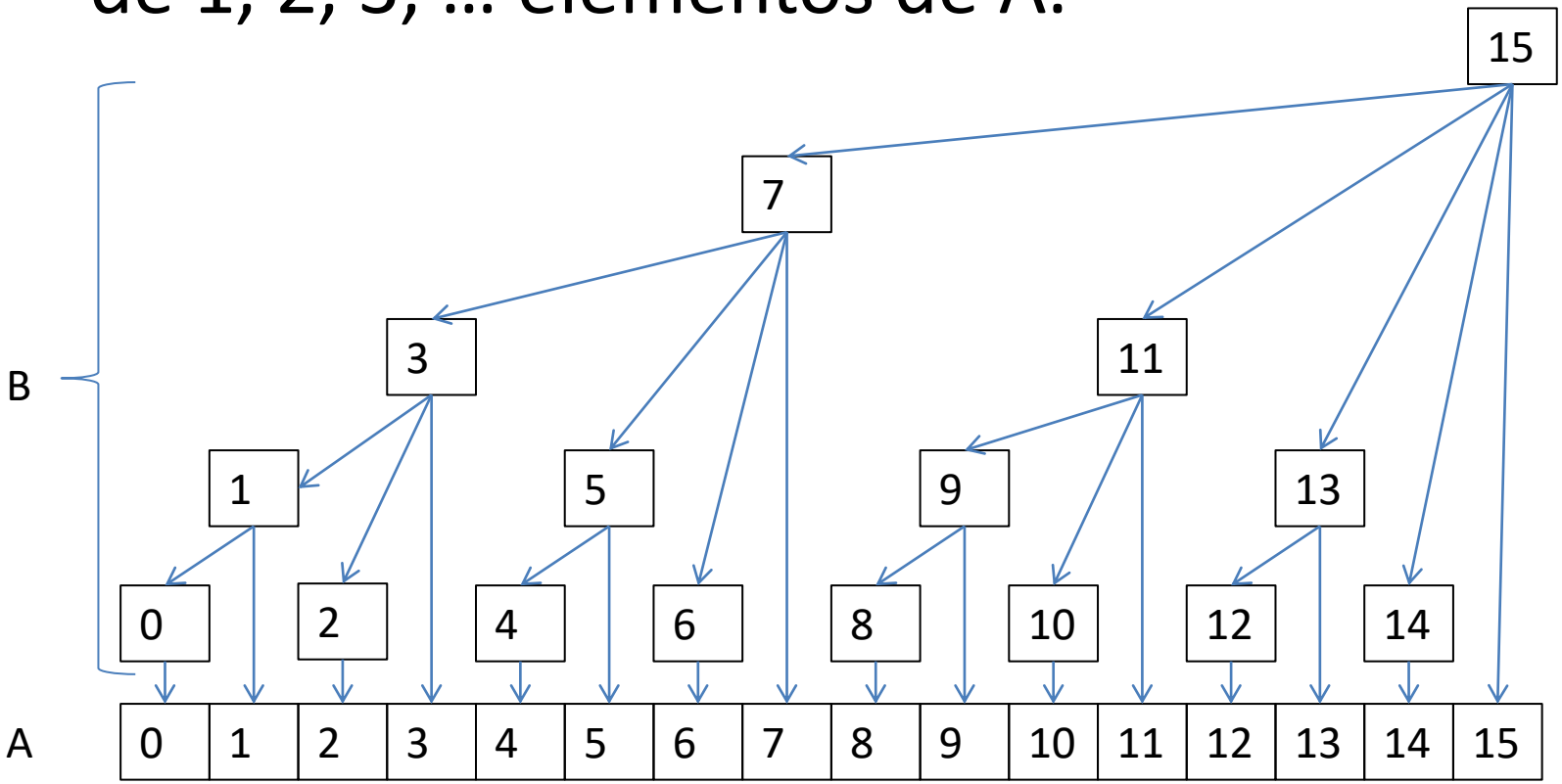
- getSum(i): obtiene la suma hasta la posición i
- Increment(i,x): incrementa el elemento i-ésimo con el valor x
- getFreq(i): se reduce a getSum(i)-getSum(i-1). Se podría almacenar adic. un arreglo de frecuencias (y no de sumas) para que sea O(1).
- getRange(i,j): Si queremos la suma de valores en un rango de elementos i..j, esto se reduce a getSum(j)-getSum(i-1)

Implementación

- Entrada: un arreglo de frecuencia A.
- Atributo: arreglo B de n elementos, B[0..n-1]. Utilizaremos 0-based index. Hay otras implementaciones 1-based index.
- B[i] =
  - Si i en binario es una secuencia de 1s, contiene la suma de los primeros i elementos: A[0]..A[i].
  - Sino, contiene la suma de los elementos A[g(i)]+...+A[i], donde g(i) es el número i apagando la secuencia continua de 1s (si existe) que empieza en el bit menos significativo; g(i) = i & (i+1)

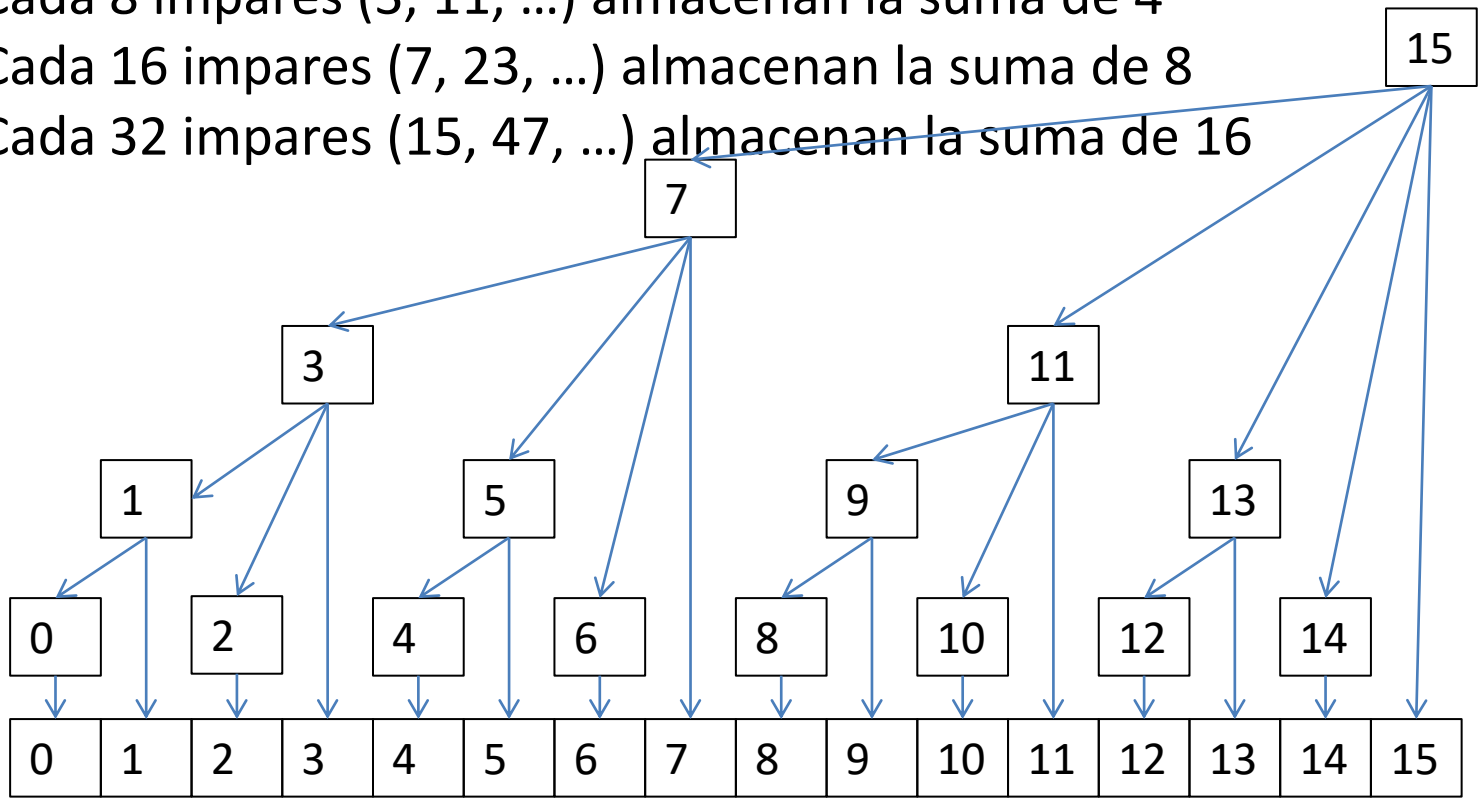
Implementación

- Cada elemento de B puede almacenar la suma de 1, 2, 3, … elementos de A.



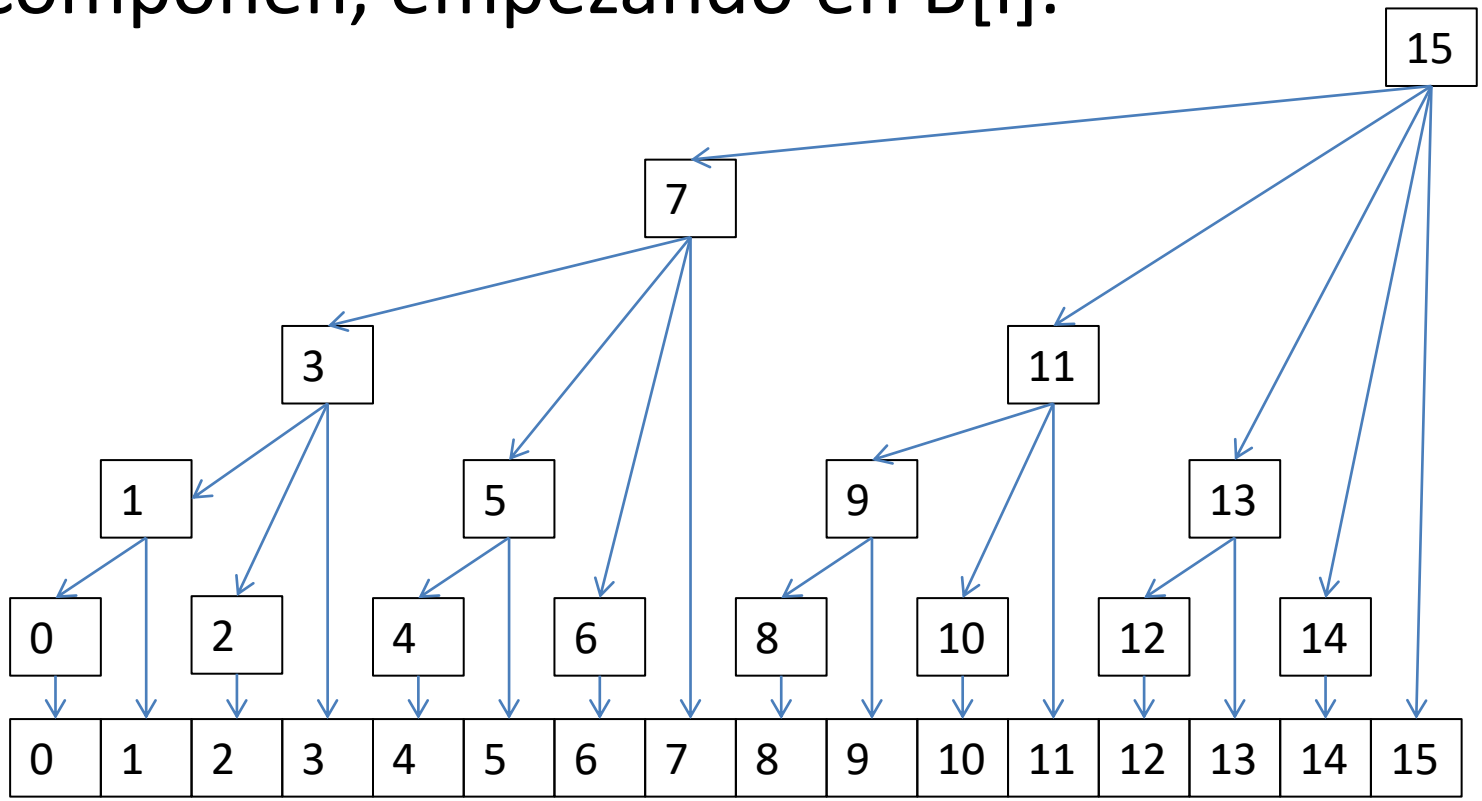
Implementación

- Los pares almacenan solo 1
- Cada 4 impares (1, 5, …) almacenan la suma de 2
- Cada 8 impares (3, 11, …) almacenan la suma de 4
- Cada 16 impares (7, 23, …) almacenan la suma de 8
- Cada 32 impares (15, 47, …) almacenan la suma de 16



Implementación: getSum(i)

- Dado un número i, buscamos las sumas que lo componen, empezando en B[i].



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

Implementación: getSum(i)

- **Ejemplo**: tomando el 14 = 1110_2
- Vemos que es par, así que sabemos que B[i] almacena un elemento.
- Saltamos a B[13]=B[1101_2], donde la seguidilla de 1's es de un elemento (almacena la suma de 2 elementos).
- Saltamos a B[11]=B[1011_2]. La seguidilla es de dos 1's (almacena la suma de 4 elementos).
- B[7]=B(111_2), que almacena la suma de 8 elementos.
- Al restar 8, obtenemos -1 y termina el ciclo.
- Note que previo(i) = g(i)-1 = i & (i+1) -1.

Implementación: getSum(i)

- Peor caso: números de la forma 2^k-2, pues requieren sumar posiciones con seguidillas de cero 1's, un 1, dos 1's, tres 1's, …
- Mejor caso: número de la forma 2^k-1, pues tienen todos los bits prendidos, y solo requieren de un acceso a B.

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

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

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

Implementación: increment(i,x)

- Esta función debe actualiza todas las posiciones en B que incluyan en su sumatoria a A[i].
- Empezamos por actualizar B[i] += x; luego el próximo elemento de B que incluye a A[i] es i=i|(i+1). El proceso continua hasta que i sobrepase el tamaño del arreglo.

Implementación: increment(i,x)

- Para i=6 y n=16, actualizamos 6,7,15.
- Para i=0 y n=16, actualizamos 0,1,3,7,15
- i=0 es el peor caso ➔ O(log)

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

Ejercicios

- Maratón suramericano 2017: F - Fundraising

http://matcomgrader.com/problem/9346/fundraising/

- Spoj: Tulip and Nubers

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

- UVA: potenciómetros

https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=24&page=show_problem&problem=3238

Tarea (no evaluada)

- Implemente una clase fenwick tree de enteros.
- Implemente al menos 2 problemas de spoj relacionados con fenwick trees. Intente pasar todos los tests.

Referencias

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