Fenwick Tree

...

Objetivos:

Arreglo A de talla n: A[1] ... A[n]

- suma(l,r) = A[l] + ... + A[r]
- update(i,delta): A[i] += delta
- update_range(l,r,delta): A[i] += delta (para cada l <= i <= r)

Complejidad

Operacion	Tiempo	Espacio
Construccion	O(n)	O(n) (un arreglo de mismo tamaño)
suma,update,update_rango	O(log n)	O(1)

Idea

Mantener un arreglo T que guarde sumas parciales:

T[i] = T[g(i) + 1] + ... + T[i]

```
int prefijo(i){
  int res=0;
  while(i>0){
    res+=T[i];
    i=g(i);
  }
  return res;
}
```

```
void update(int i, int delta){
  Para todo j con: g(j) <= i <= j:
    T[j] += delta
}
```

Ejemplo:

Si:

i	8	6	3	2
g(i)	6	3	2	0

A	1	2	3	4	5	6	7	8	9
T									

Como elegir g?

- g(i) = i-1 -> T es A

1	2	3	4
A[1]	A[2]	A[3]	A[4]

- g(i) = 0 -> T es la suma de prefijos

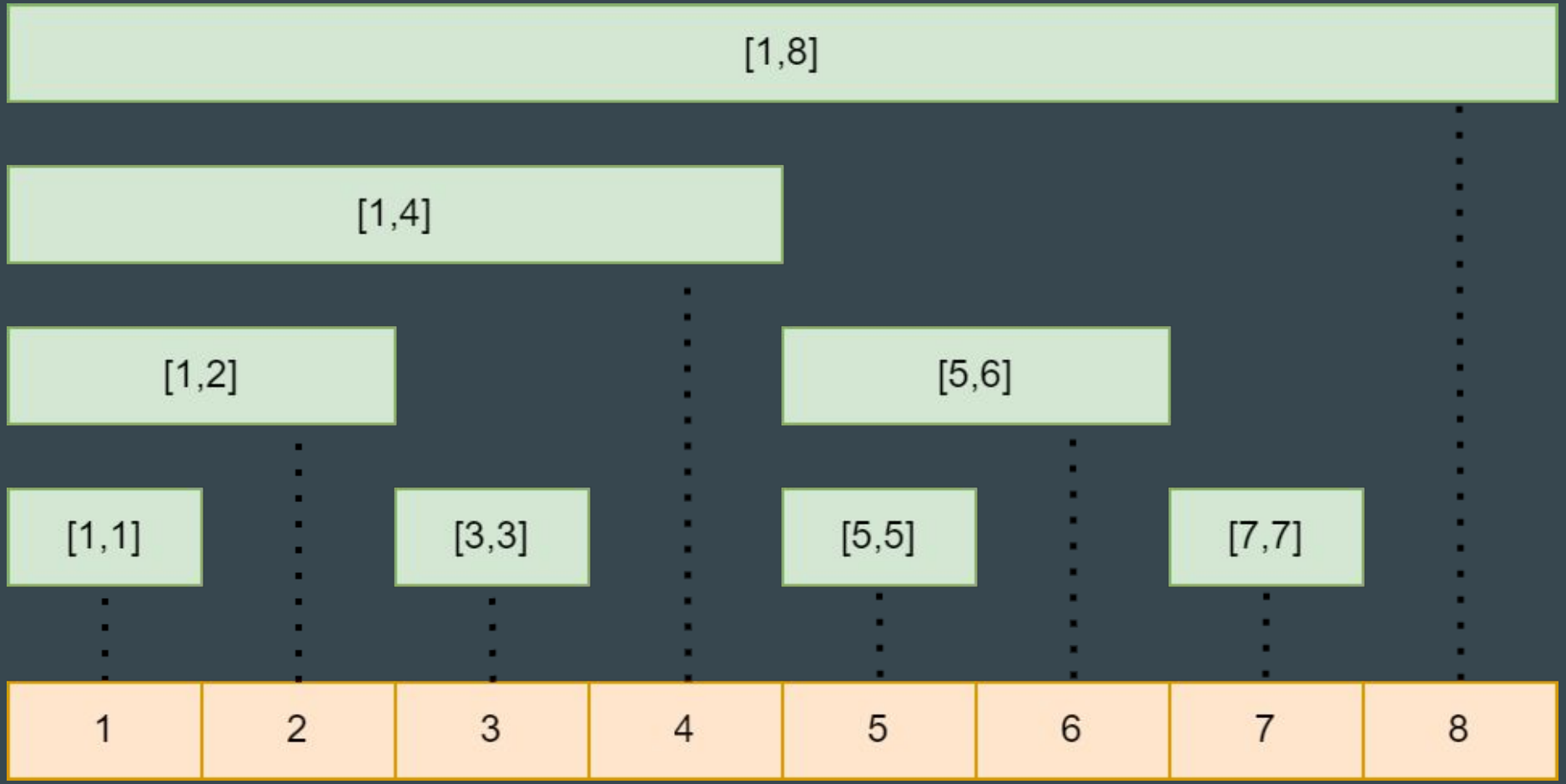
1	2	3	4
A[1]			
A[1]+A[2]			
A[1]+A[2]+A[3]			
A[1]+A[2]+A[3]+A[4]			

g

Poner el bit menos significativo en 0

- g(7) = g(111) = 110 = 6
- g(4) = g(100) = 000 = 0

T[i] tiene la suma de un segmento de largo el bit menos significativo de i.


[1,8]
[1,4]
[1,2]
[5,6]
[1,1]
[3,3]
[5,5]
[7,7]
1
2
3
4
5
6
7
8

Calcular g

Bit mas significativo: i & (-i)

i: a10...0 entonces -i es a'(10...0)' + 1 = a'01...1 + 1 = a'10000

g(i) = i - (i & (-i))

Suma(l,r)

```
int prefijo(int i, vector<int> &T){
  int res=0;
  while(i>0){
    res+=T[i];
    i-= i & (-i);
  }
  return res;
}
int sum(int l, int r,vector<int> &T){
  return prefijo(r,T)-prefijo(l-1,T);
```

Update?

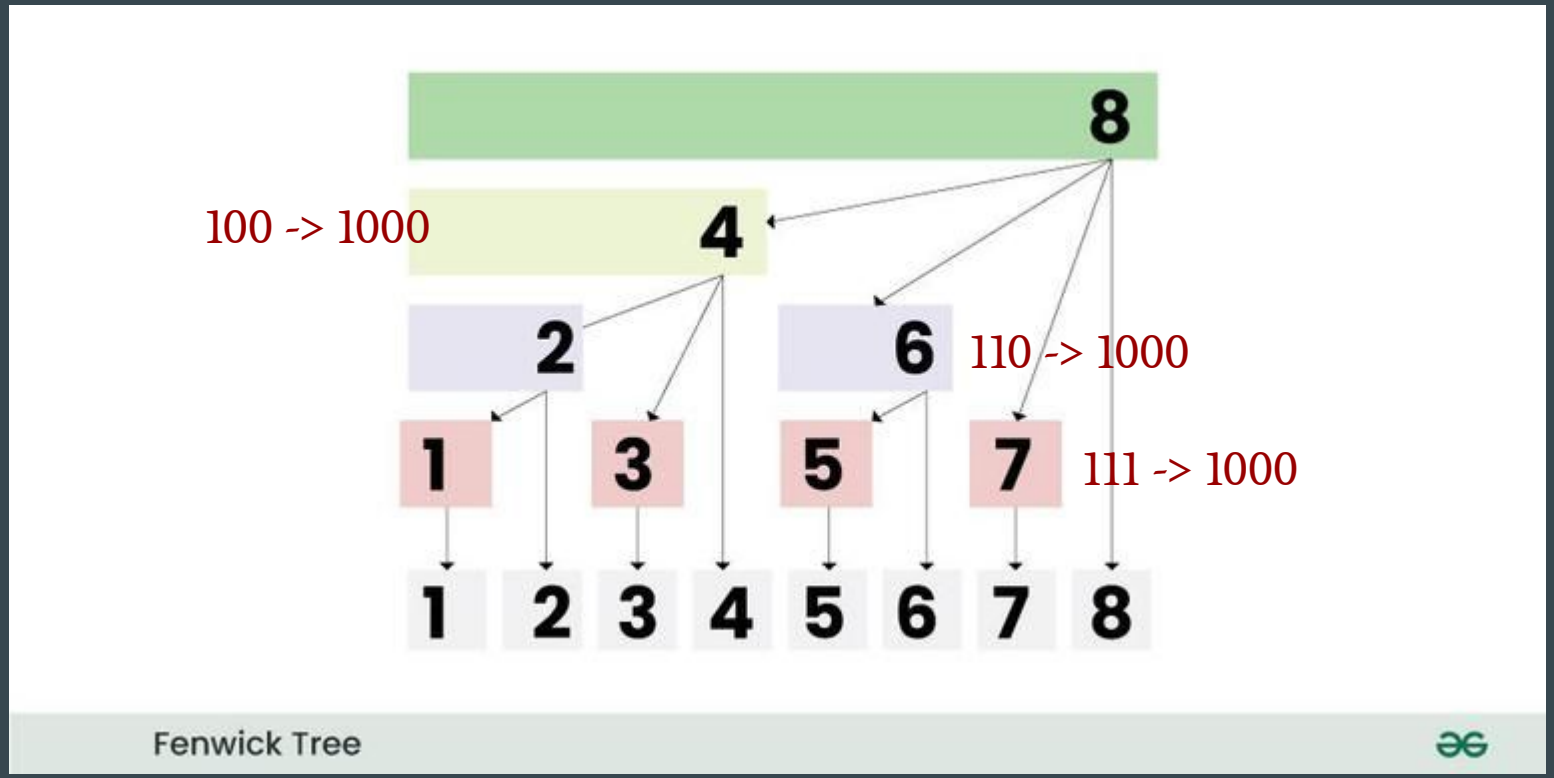
i: a 1 0...0

: b 0 0...0

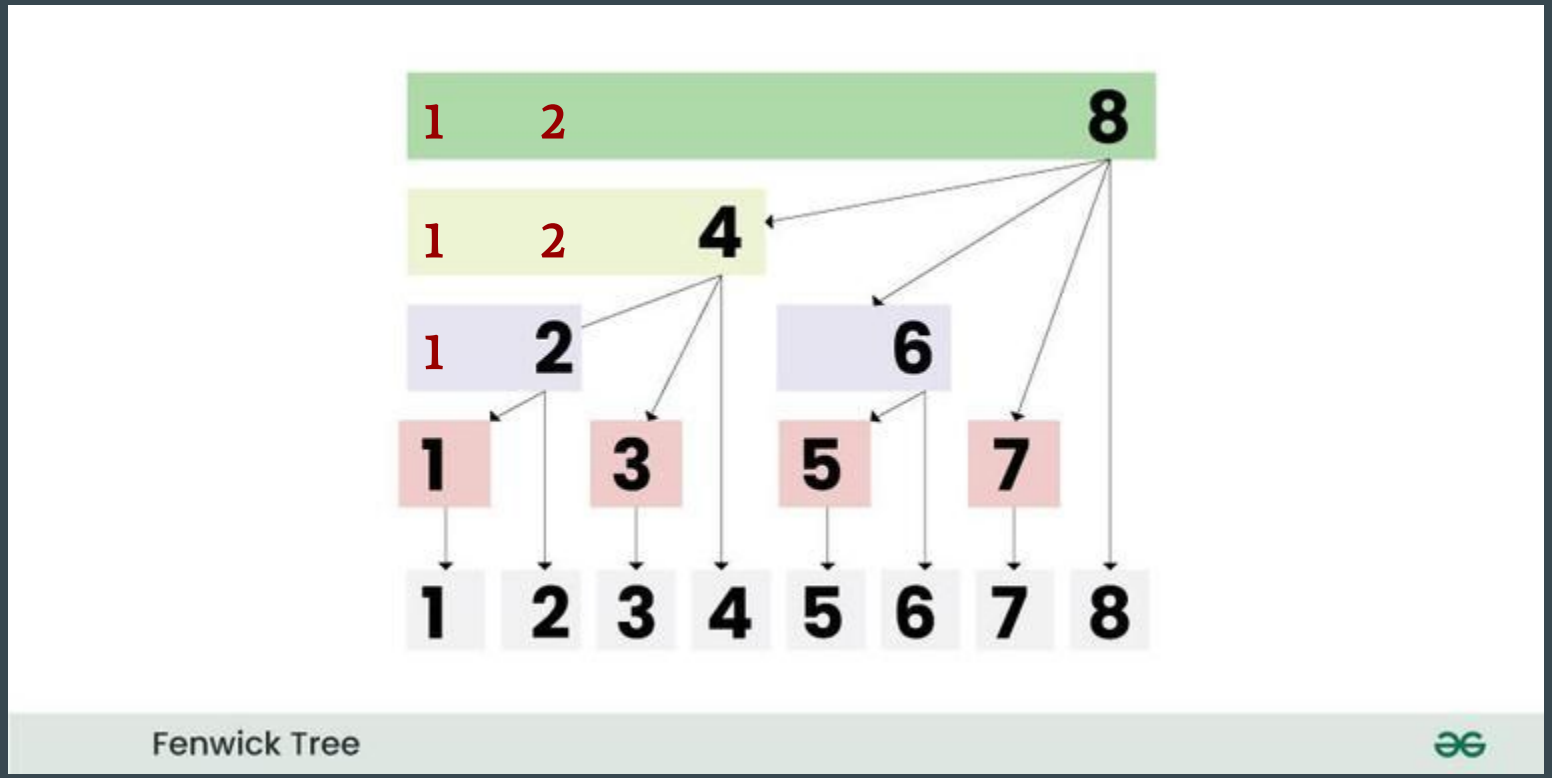
h(i) = i + (i & (-i))

```
void update(int i, int delta,vector<int> &T){
  int n=T.size()-1;
  while(i <= n){
    T[i]+=delta;
    i+= i&(-i);
  }
}
```

Vista de ‘arbol’



Construccion en O(n)



```
void init(vector<int> const &A, vector<int> &T){
  T.assign(A.size(),0);
  int n=A.size()-1;
  for(int i=1;i<=n;i++){
    T[i]+=A[i];
    int nxt=i+(i & (-i));
    if(nxt<=n) T[nxt]+=T[i];
  }
```

Update en rangos y querys en indices

A	1	…	l	…	r	r+1	…	n
Objetivo	0	…	+x		+x	0	…	0
+x en l	0	…	+x	0	0	0	…	0
pref(1,i)	0	…	+x	+x	+x	+x	…	+x
-x en r+1	0	…	+x	0	0	-x	…	0
pref(1,i)	0	…	+x	+x	+x	0	…	0

Updates y queries en rangos

	0 < i < L	[L … R]	R < i
	0	x * (i - (L-1))	x * (R - L + 1)
	i * 0 - 0	i * x - x * (L-1)	i * 0 - x * (L - 1 - R)
Pos	0	x	0
Neg	0	x * (L-1)	x * (L - 1 - R)

Queries:

```
int prefijo_rango(int i){
  return prefijo(i,pos)*i - prefijo(i,neg) + pref[i];
}
```

Updates:

```
void update_rango(int l, int r, int delta){
  update(l,delta,pos);
  update(r+1,-delta,pos);
  update(l,delta*(l-1),neg);
  update(r+1,-delta*r,neg);
}
```

Fuentes:

- C-P algorithms: https://cp-algorithms.com/data_structures/fenwick.html
- Geeks for Geeks: https://www.geeksforgeeks.org/competitive-programming/fenwick-tree-for-competitive-programming/