Unidad 3

Fundamentos de Programación Competitiva

Logro de sesión

Al finalizar la sesión, el estudiante comprenderá los conceptos de árboles de segmentos

Semana 9

Arbol de Segmentos

Contenido:

- Árbol de Segmentos - Construcción
- Árbol de Segmentos - Update
- Árbol de Segementos – Query

Motivación

- Problema: Tenemos un arreglo de enteros de N elementos. ($N \leq 100{,}000$)

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

- Se le pide **hallar la suma de todos los números entre las posiciones [*x*, *y*]**
  - Ejemplo:
  - Si $x = 1$; $y = 3$
  - Respuesta: $5 + 2 + 3 = 10$

Motivación

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

- **Propuesta 1**
- Recorrer todos los elementos desde *x* hasta *y* (inclusive) y acumular la suma.

```
int suma = 0;
for(int i = x; i <= y; i++)
  suma += v[i];
cout << suma;
```

- Este algoritmo es **O(*N*)**
- En el peor caso: $x = 0$; $y = N - 1$

Motivación

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

- Definiremos la operación *“Consulta” (Query)*.
  - *Consulta*(x, y) = Suma de elementos entre x e y *(inclusive)*.
  - Ej. *Consulta*(0, 7) = 39
  - Ej. *Consulta*(1, 3) = 10
- Si hubiera una única operación *consulta*, el algoritmo de la propuesta 1 funcionaría bien.
  - Leer los N elementos del arreglo: $O(N)$
  - En total (lectura + 1 *consulta*): $O(N + N) = O(N)$
- **¿Y si tuviéramos que responder *Q consultas*?**

Motivación

- Responder cada *consulta* toma O(N).
- Se le dan Q *consultas*.
- <u>**Conclusión:**</u>
- Responder las Q *consultas* usando la propuesta 1 toma: $O(N + N * Q) = O(N * Q)$
- Como $N = 100{,}000$; si Q fuese 10,000; la cantidad de operaciones sería proporcional a 10^9.
- ¡Time Limit Excedeed is coming! ☹
- **¿Se puede evitar recorrer el arreglo en cada *consulta*?**

Motivación

- **Propuesta 2:**
- Usar acumulación de sumas.

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

0	1	6	8	11	19	23	32	39
0	1	2	3	4	5	6	7	8

- Se puede pre-procesar el arreglo en O(*N*).
- Cada *consulta* se responde ahora en O(1).
  - *Consulta*(*x*, *y*) ← dp[*y* + 1] – dp[*x*]
  - *Consulta*(0, 3) ← dp[4] – dp[0]

Motivación

- Responder cada *consulta* nos toma O(*1*).
- Existen *Q consultas*.
- **Conclusión:**
- Responder todas las *consultas* usando la propuesta 2 toma: O(*N + Q*)
- ¡Accepted is coming! ☺
- Pero…
- **¿Qué sucedería si, adicionalmente a las *consultas,* se le pide actualizar el valor de un elemento del arreglo?**

Motivación

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

- Se define la operación *Actualizar (update)*.
  - *Actualizar*(*x*, *val*) = Asigna *val* a v[*x*].
  - Ej. 1. *Actualizar*(0, 7) = Asigna 7 a v[0] (v[0] = 7).
  - Ej. 2. *Consulta*(0, 2) = **14**
- Suponga que las operaciones *Consulta* y *Actualizar* vendrán intercaladamente.
- **¿Se podría adaptar la propuesta 2?**

Motivación

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

- **Propuesta 3:**
- Procesar el arreglo en cada operación *Actualizar*.

- Lamentablemente, podrían venir (vendrán) *Q* operaciones *Actualizar*.
- **Conclusión:**
- ¡Time Limit Excedeed is coming! ☹

Arbol de Segmentos

- Es llamado Segment Tree (en ingles), esta estructura es un tipo especial de árbol binario.

- Este árbol como su nombre lo indica trabaja con segmentos.

- Supongamos que tenemos un arreglo de N elementos (1 <= N <= 1000), sobre el cual se te hará M (1 <= M <= 1000) consultas del tipo

  ¿Cuál es la suma en el rango N a M? (1 <= L <= R <= N)

Arbol de Segmentos

- Tenemos la solución desarrollada.
- Pero también deben existir otras consultas en las que te piden otras operaciones, o que se pueda actualizar L y R.
- Segment Tree podría ser una buena solución óptima.
- Segment Tree resuelve este tipo de problemas donde se da un arreglo de elementos y varias consultas en las que puede surgir actualizaciones de algún elemento o responder preguntas en un rango [L, R].

Arbol de Segmentos

- Las consultas pueden ser como:
  - El mínimo en el rango.
  - El máximo en el rango.
  - Suma de los elementos del rango.
  - Multiplicación de los elementos en el rango.

Construcción

- Se debe crear una estructura que es un árbol, donde debemos guardar la información de un determinado intervalo [L,R] en un nodo de este árbol,

- Por ejemplo el nodo raíz contiene la información de todo el arreglo [0,n], ya sea esta información la suma de un intervalo, calcular mínimos y máximos en un intervalo, o cualquier otra información de importancia.

Construcción

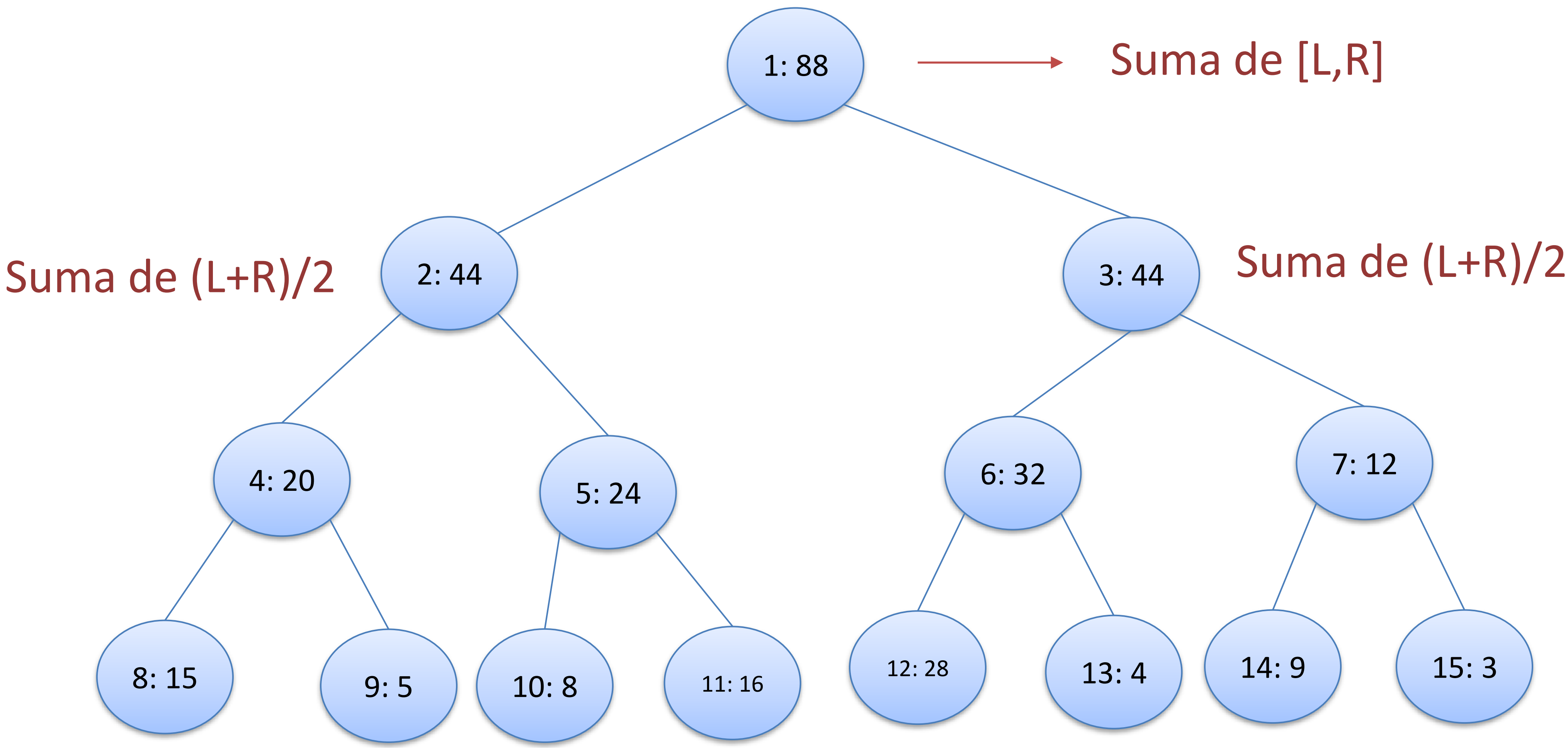
- Construir este tipo de árboles conlleva a un nodo guarda la información de un intervalo [L,R] y el nodo raíz guarda la información de todo el arreglo.

- Ahora diremos que todos los nodos en nuestro árbol tienen 0 o 2 hijos, 0 en el caso de ser nodos hoja.

Indice	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Int Arr[8]	15	5	8	16	28	4	9	3

Construcción

- El árbol sería de la siguiente manera:



Construcción

- Para implementar podemos usar este código:

```
// todos estos valores son constantes en la primera llamada
void init ( int node =1, int l=0, int r=N -1) {
//caso base cuando te encuentras en un nodo hoja
//y no puedes dividir en más intervalos
if(l == r) T[ node ] = a[l];
else {
int mi = (l + r) / 2; // mitad
init (2 * node , l, mi);//llamamos al hijo izquierdo
init (2 * node + 1, mi + 1, r);//llamamos al hijo derecho
T[ node ] = ( T[2 * node ] + T[2 * node + 1] );
//como hemos ido llamando recursivamente
//ya sabremos el valor de T[2*node] y T[2*node+1] en este caso son sumas
//también podrían ser mínimos u otro tipo de datos.
} }
```

Construcción

- A continuación mostramos una tabla donde indica el intervalo que cubre a cada nodo

Nodo	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
L	0	0	4	0	2	4	6	0	1	2	3	4	5	6	7
R	7	3	7	1	3	5	7	0	1	2	3	4	5	6	7

Podemos ver que las hojas contienen el arreglo original.

Update

- Ahora continuamos con las actualizaciones

- Vamos a cambiar el valor de un elemento del arreglo por otro, para esto vamos a usar una función muy parecida a la función previa para construir el árbol.

- Necesitamos dos parámetros más un entero “X” que es la posición que queremos actualizar por un entero “val”, también cambia el caso base de nuestra recursión.

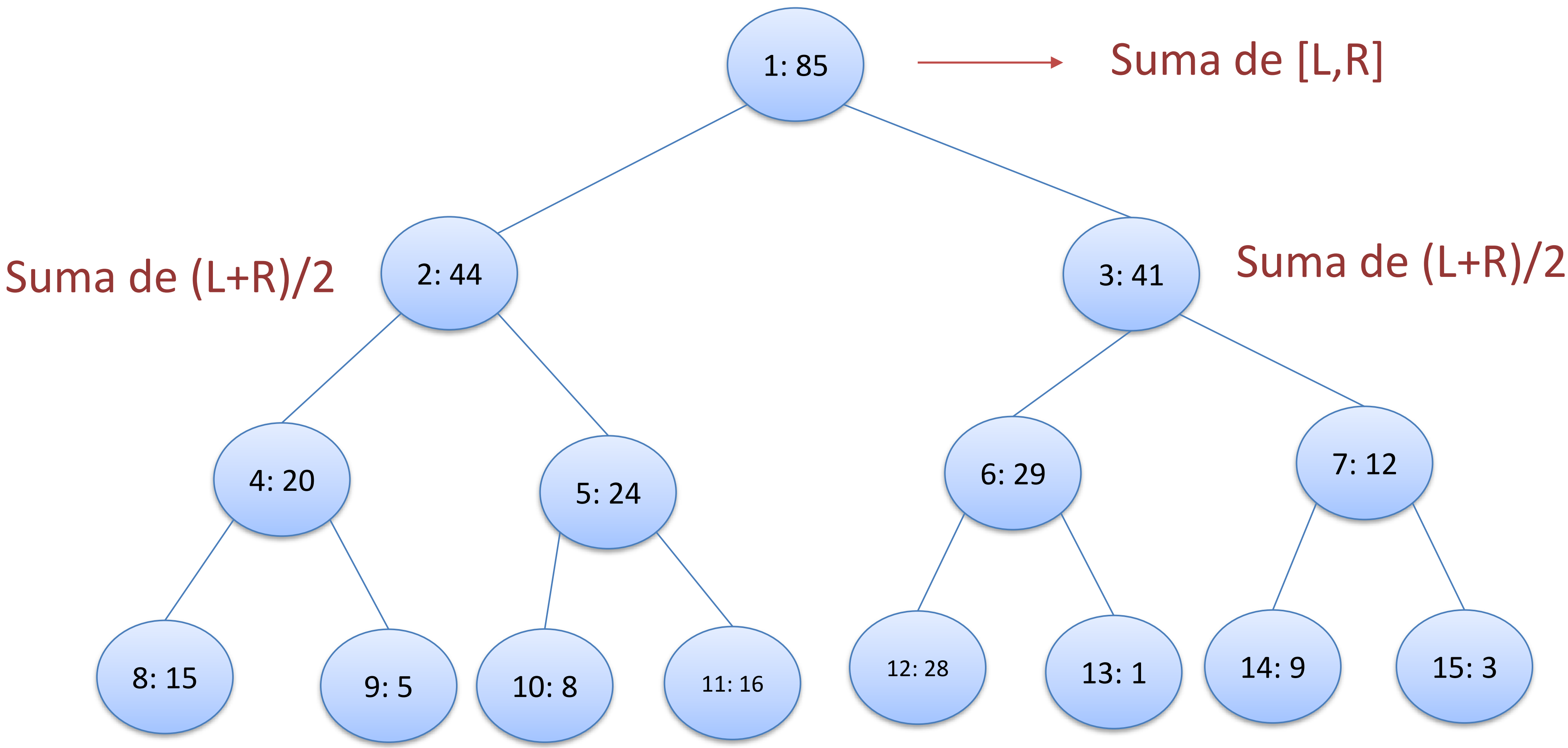
Update

- A continuación vemos el código:

```
void update ( int x, int val , int node =1, int l=0, int r=N -1) {
if(r < x || l > x) return ; //si nos salimos del rango
//actualización cuando hemos llegado al nodo hoja buscado
if(l == r) T[ node ] = val ;
else {
int mi = (l + r) /2;
update (x, val , 2 * node , l, mi);//hijo izquierdo
update (x, val , 2 * node + 1, mi + 1, r);//hijo derecho
//actualización de los otros nodos
T[ node ] = ( T[2 * node ] + T[2 * node + 1] );
}
}
```

Update

- Por ejemplo si actualizamos el arreglo, con Arr[5]: 1



Query

- Ahora podemos realizar las diferentes consultas como la suma, el mínimo, máximo u otra consulta.

```
//inicialización igual pero con el rango x y
int query ( int x, int y, int node =1, int l=0, int r=N -1) {
if(r < x || l > y) return 0;
//caso base en el que un intervalo no debe ser tomado en cuenta en
//este caso retornamos 0 porque sumar algo+0=algo
//si se tratará de hallar el mínimo deberíamos poner un número muy grande
//ya que min(numgrande,otronumero) será el otro número
if(x <= l && r <= y) {
return T[ node ];
//si el rango forma parte de la solución tomamos en cuenta ese nodo
} else {
int mi = (l + r) /2; //la mitad
return query (x, y, 2 * node , l, mi) + query ( x, y, 2 * node
+ 1, mi + 1, r);
//similar a las otras funciones hijo izquierdo e hijo derecho
```

Query

- Segment Tree puede ser extendido a más de una dimensión, por ejemplo en matrices. Se debe tener muy en cuenta la cantidad de espacio que se utiliza para guardar la estructura en estos casos.

- Existen consultas de actualización en un rango en Segment Tree, para esto hay que modificar un poco el algoritmo y usar un algoritmo llamado “lazy propagation”.

- Segment Tree suele acompañar a problemas de geometría, y servir en varios problemas de estructuras de datos.

Muchas Gracias!!!