Range queries 3

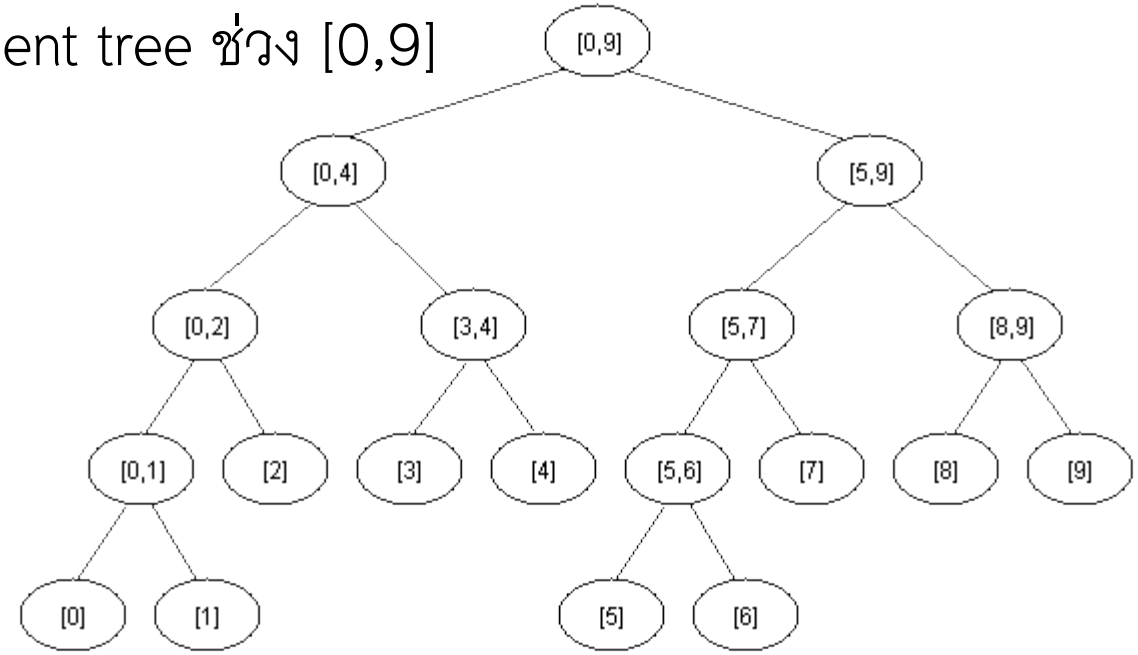
Segment tree

Segment tree มีการดำเนินการหลักๆ สามอย่าง

- **build tree** เริ่มต้นสร้างต้นไม้ **O(N lg(N))**
- **update tree** เพิ่มค่าภายในช่วง [i, j] ด้วยค่า val **O(lg(N+k))**
- **query tree** สืบค้นค่า**<u>มาก</u>**สุดภายในช่วง [i, j] **O(lg(N+k))**
- K = จำนวนของช่วงที่ถูกสืบค้น

ตัวอย่างของ segment tree

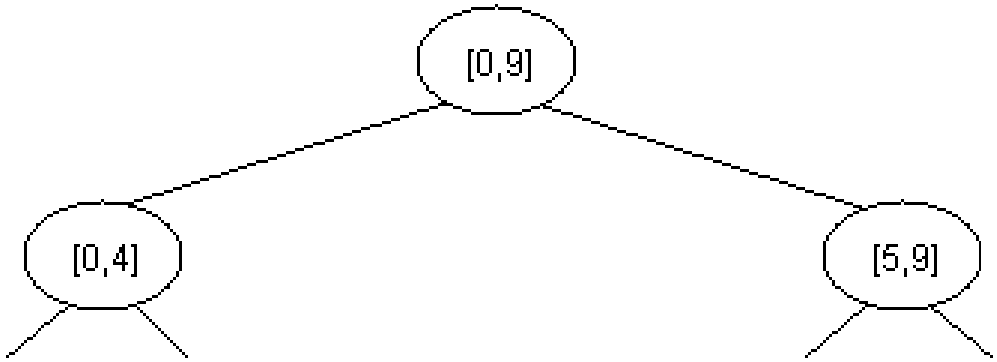
- โหนดแรกจะเก็บข้อมูลทั้งหมดของช่วง [i, j]
- ถ้า i < j ลูกทางซ้ายและลูกทางขวาจะเก็บข้อมูลของช่วง [i, (i+j)/2] และ [(i+j)/2+1, j]
- สังเกตว่าความสูงของ segment tree ของช่วงที่มี N ตัวคือ [logN+1]
- ตัวอย่าง segment tree ช่วง [0,9]



```
#define N 20
#define MAX (1+(1<<6)) #define inf
0x7fffffff

int arr[N];
int tree[MAX];
```

- `build_tree(1, 0, N-1);`

สร้าง tree โดยเริ่มที่ root (1) และช่วงที่ดูแลคือ 0 ถึง N-1



เช่นช่วง 0 ถึง N-1 เราก็จะขอคำตอบของลูกทางซ้ายและลูกทางขวาแล้วก็จะแบ่งครึ่งไปทำต่อที่ [0,4] กับ [5,9] แล้วทำจนถึงโหนดลูก โดยโหนดลูกคือโหนดที่ขอบเขตทางซ้ายและขวาเท่ากัน จากนั้นจึงค่อยกำหนดค่าของ array ของเราลงไป

```
void build_tree(int node, int a, int b) {
        // Out of range
        if(a > b) return;
        // Leaf node

        if(a == b) {
                tree[node] = arr[a];
                return;
        }
        // Init left and right child

        build_tree(node*2, a, (a+b)/2);
        build_tree(node*2+1, 1+(a+b)/2, b);
        // Init node value
        tree[node] = max(tree[node*2], tree[node*2+1]);
}
```

- `update_tree(1, 0, N-1, 0, 6, 5);`

- update ต้นไม้เริ่มต้นที่ root(1) ขอบเขต [0, N-1] โดยการบวกเป็นช่วง [i, j] เช่น ช่วง [0,6] ด้วยค่า 5
- เริ่มที่ root ลงไปทำทั้ง tree ขอค่า max ของลูกทางซ้ายและขวา เพื่อ update ตัวเอง ถ้าไม่ใช่ช่วง [i, j] ที่สอบถามก็ไม่ update
  ถ้าเป็นโหนดลูก ก็ให้เพิ่มค่า

- การทำงานคล้ายกับ build_tree

```
void update_tree(int node, int a, int b, int i, int j, int value)
{
        // Current segment is not within range [i, j]
       if(a > b || a > j || b < i)
              return;
       // Leaf node
       if(a == b) {
              tree[node] += value;
              return;
       }
       // Updating left and right child
       update_tree(node*2, a, (a+b)/2, i, j, value);
       update_tree(1+node*2, 1+(a+b)/2, b, i, j, value);
       // Updating root with max value
       tree[node] = max(tree[node*2], tree[node*2+1]);
}
```

- `query_tree(1, 0, N-1, 5, N-1)`

- การสืบค้นเช่นสืบค้นช่วง 5 ถึง n-1 ก็ทำงานคล้ายกับ build_tree และ update_tree นั่นคือ เราจะเริ่มต้นที่ root (1) จากนั้นก็ไปถามลูกทางซ้ายและลูกทางขวา
  - ถ้าไม่อยู่ในช่วง คืนค่า –infinity เพราะว่าเราหาค่า max
  - ถ้าอยู่ในช่วงพอดีเลย คืนค่า tree[node]
  - ถ้าช่วงคาบเกี่ยวก็ไปถามลูกทางซ้าย ลูกทางขวาให้ไปหาต่อ แล้วเราก็เอาค่ามากสุด

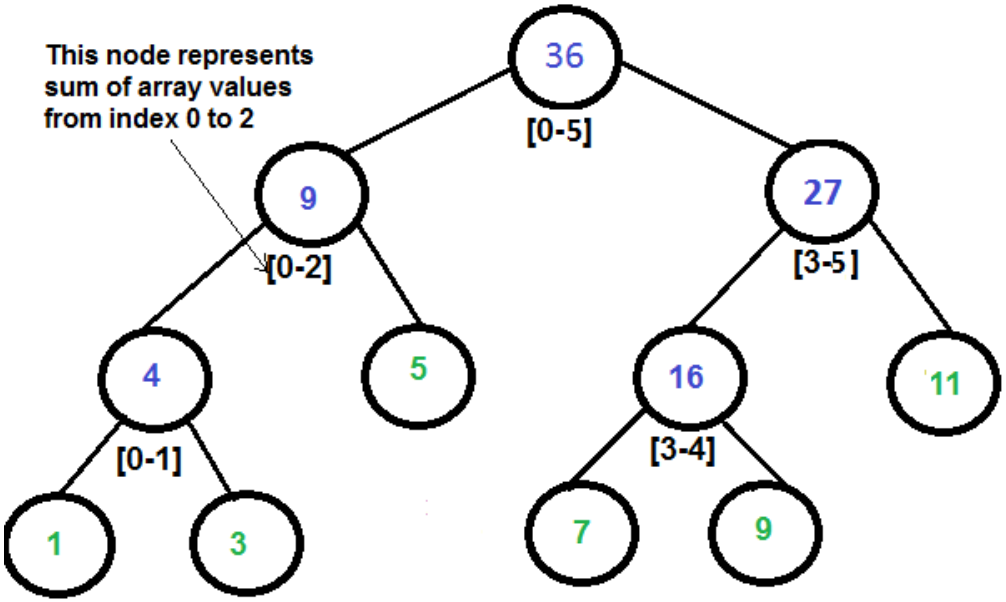
```
int query_tree(int node, int a, int b, int i, int j) {
      // Out of range
      if(a > b || a > j || b < i)
            return -inf;
      // Current segment is totally within range [i, j]
      if(a >= i && b <= j)
            return tree[node];
      // Query left and right child
      int q1 = query_tree(node*2, a, (a+b)/2, i, j);
      int q2 = query_tree(1+node*2, 1+(a+b)/2, b, i,
j);
      int res = max(q1, q2);

      return res;
}
```

```
int main() {
        //Init array arr
       for(int i = 0; i < N; i++)
              arr[i] = 1;
       build_tree(1, 0, N-1);
       // Increment range [0, 6] by 5
       update_tree(1, 0, N-1, 0, 6, 5);
       // Increment range [7, 10] by 12
       update_tree(1, 0, N-1, 7, 10, 12);
       // Increment range [10, N-1] by 100
       update_tree(1, 0, N-1, 10, N-1, 100);
       // Get max element in range [0, N-1]
       cout << query_tree(1, 0, N-1, 0, N-1) << endl;
}
```

Lazy propagation

- ทุกอย่างดูดีไปหมดจนกระทั่งงงงง...
- ในบางครั้ง segment tree ถ้ามีการดำเนินการ update บ่อยๆ สิ่งที่เกิดขึ้นถ้าเป็นแบบเดิมคือ update ครั้งหนึ่ง เหมือนทำทั้ง tree ใหม่ทีนึงเลย
- ถ้ามีการ update บ่อยๆ ในช่วงหนึ่ง จะทำอย่างไรให้เลื่อนการ update ทั้งหมด และทำเมื่อจำเป็นได้

- เนื่องจากว่า node ใน segment tree นั้นเก็บค่า ของช่วงนั้นไว้
- ถ้าช่วงของโหนด อยู่ภายในช่วงที่ต้อง update แล้วทุกโหนดด้านล่างของโหนดนั้นต้องถูก update



Segment Tree for input array {1, 3, 5, 7, 9, 11}

- ตัวอย่างเช่น พิจารณา โหนดที่เก็บค่า 27 ในรูปก่อนหน้า โหนดนี้เก็บผลรวมของช่วง 3 ถึง 5
- ถ้าการ update ของเราทำในช่วง 2 ถึง 5, แล้วเราต้อง update โหนดนี้และทุกโหนดใต้โหนดนี้
- ใน Lazy propagation, เราจะ update โหนดที่เก็บค่า 27 และเลื่อนการ update ของโหนดลูกโดยเก็บการ update ข้อมูลนี้ในโหนดต่างหาก ที่เรียกว่า lazy nodes
- เราจะสร้าง array lazy[] ที่แทน lazy node โดยขนาดของ lazy[] เป็นขนาดเดียวกับ array ของ segment tree หรือ tree[] นั่นเอง

- แนวคิดคือ
- เริ่มต้นให้ทุกค่าใน lazy[] เป็น 0
- โดยค่า 0 ใน lazy[i] เป็นตัวบอกว่ามีการ update ที่รออยู่ไหมในโหนด i ใน segment tree
- ค่าที่ไม่เป็น 0 (non-zero value) ของ lazy[i] หมายความว่าค่านี้จำเป็นที่จะต้องถูกเพิ่มให้กับโหนด i ใน segment tree ก่อนที่จะมีการ query อื่นกับโหนดนี้
- ต้องคิดว่าถ้า sum max min จะ update lazy แบบไหนหรือคืนค่าแบบไหน

- เริ่มต้นเพิ่ม lazy

```
int arr[N];
int tree[MAX];
int lazy[MAX];

void build_tree(int node, int a, int b)
```

เหมือนเดิม

```
void update_tree(int node, int a, int b, int i, int j, int value) {
   if(lazy[node] != 0) { // This node needs to be updated
      tree[node] += lazy[node]; // Update it
      if(a != b) {
         lazy[node*2] += lazy[node]; // Mark child as lazy
         lazy[node*2+1] += lazy[node]; // Mark child as lazy
      }
      lazy[node] = 0; // Reset it
   }
   if(a > b || a > j || b < i)
      return;
   if(a >= i && b <= j) {
      tree[node] += value;
      if(a != b) {
         lazy[node*2] += value;
         lazy[node*2+1] += value;
      }
      return;
   }
   update_tree(node*2, a, (a+b)/2, i, j, value);
   update_tree(1+node*2, 1+(a+b)/2, b, i, j, value);
   tree[node] = max(tree[node*2], tree[node*2+1]);
}
```

```
int query_tree(int node, int a, int b, int i, int j) {
	if(a > b || a > j || b < i) return -inf;
	if(lazy[node] != 0) { // This node needs to be updated
		tree[node] += lazy[node]; // Update it
		if(a != b) {
			lazy[node*2] += lazy[node];
			lazy[node*2+1] += lazy[node];

		}
		lazy[node] = 0; // Reset it
	}
	if(a >= i && b <= j)
		return tree[node];
	int q1 = query_tree(node*2, a, (a+b)/2, i, j);
	int q2 = query_tree(1+node*2, 1+(a+b)/2, b, i, j);
	int res = max(q1, q2); // Return final result
	return res;
}
```

```
int main() {
        for(int i = 0; i < N; i++) arr[i] = 1;

        build_tree(1, 0, N-1);

        memset(lazy, 0, sizeof lazy);

        update_tree(1, 0, N-1, 0, 6, 5);
        update_tree(1, 0, N-1, 7, 10, 12);
        update_tree(1, 0, N-1, 10, N-1, 100);

        cout << query_tree(1, 0, N-1, 0, N-1) << endl;
}
```