TREES 5

Good Evening

Need full attention for these topics

Today's content → (a) TreeMap

(b) Nearest 1

(c) Serialization of tree & Deserialization

TreeMap

01. In treemap, data is sorted in increasing order of keys
02. Each insertion / deletion takes O(log N) time, where N is no. of keys

Treeset

01. In treeset, data is sorted in ascending order of keys.
02. Each insertion / deletion takes O(log N) time, where N is no. of keys

Q You are given an array having all ele = 0 & Q queries

2 Type of queries

Type 1 :- Flip data at i^{th} index

Type 2 :- Get nearest idx from i which has value 1

→ if (ar[i] == 1) print i

→ if 2 indices are nearest, print min idx

→ if no index exist with val 1, print -1

Eg:- ar[10] = { 0 0 ~~0~~ 0 0 0 ~~0~~ ~~0~~ ~~0~~ 0 }

(indices: 0 1 2 3 4 5 6 7 8 9)

1 1 1 +

0

Queries

	Type	Index	
→	1	2	
	1	8	
	1	7	
	2	4	→ ans = 2
	1	8	
	2	9	→ ans = 7
	1	6	
	2	6	→ ans = 6
	2	4	→ ans = 2

l = idx of closest 1 on LHS

r = idx of closest 1 on RHS

TC : O(Q * N)

SC : O(1)

Idea

Type 1 → flip the data at i^th idx

ar[i] = 1 − ar[i]

Type 2 →

```
if (ar[i] == 1) print i
else
    Go to left    ld = ∞
    l ←—— i
    ld = i − l
    Go to right   rd = ∞
    i ——→ r
    rd = r − i
    if (ld ≤ rd) print(l)
    else print(r)
```

* Idea 2 → Using treeSet

Eg: ar[10] = { 0 0 0 0 0 0 0 0 0 0 }

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

Queries

TreeSet = { 2, 6, ~~8~~, 7 }

	Type	Index			
→	1	2			
→	1	8			
→	1	7			
→	2	4	→ floor(4) = 2	ld = 4-2 = 2	return 2
			ceiling(4) = 7	rd = 7-4 = 3	
→	1	8			
→	2	9	→ floor(9) = 7	ld = 9-7 = 2	return 7
			ceil(9) = ∞	rd = ∞-9 = ∞	
→	1	6			
→	2	6	→ floor(6) = 6	ld = 6-6 = 0	return 6
			ceiling(6) = 6	rd = 6-6 = 0	
→	2	4			

→ floor(4) = 2 ld = 4-2 = 2 return 2

ceiling(4) = 6 rd = 6-4 = 2

* TreeSet<Integer> set = new TreeSet<>();

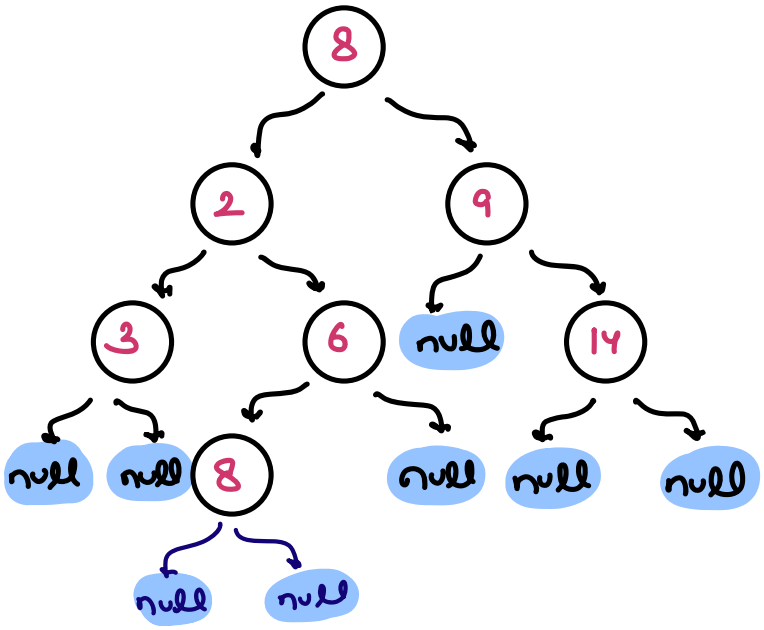
```
for ( i = 0 ; i < Queries.length ; i++ ) {
        int type = Queries[i][0]
        int idx = Queries[i][1]

        if (type == 1)
        {
            ar[idx] = 1 - ar[idx];
            if (set.contains(idx) == true) { set.remove(idx) }
            else { set.add(idx); }
        }
        else {
            if (set.contains(idx) == true) print(idx);
            else {
                int ld = ∞ , int rd = ∞
                if (set.floor(idx) != null) {
                    ld = idx - set.floor(idx);
                }
                if (set.ceiling(idx) != null) {
                    rd = set.ceiling(idx) - idx;
                }
                if (ld == ∞ && rd == ∞) print(-1);
                if (ld ≤ rd) print(set.floor(idx))
                else print(set.ceiling(idx))
            }
        }
}
```

TC: O(Q*log n)

SC: O(N)

* Serialization → Converting tree to 1D list
  ↳ Using level order

* Purpose → Retrace to get tree back (Deserialization)



(DLR)
Preorder = 8 2 3 6 8 9 14

Inorder = 3 2 8 6 8 9 14
(LDR)

* Duplicates are present, won't get correct answer

01. pop ←
02 work ←
03 insert ← child

-1 marker

Queue 8 2 9 3 6 N 14 N N 8 N N N N N

1D AL = {8 2 9 3 6 -1 14 -1 -1 8 -1 -1 -1 -1 -1}

01. If we have null as child, we are going to add it in queue

02. If we are removing null, store -1 in list

```
list<Integer> l

Queue<Node> q

q.add(root);

while(q.size()>0)
{
    Node rem = q.remove();

    if(rem==null){

        l.add(-1);
    }
    else{

        l.add(rem.data);

        q.add(rem.left);

        q.add(rem.right);

    }
}
```

TC: O(n)

SC: O(n)

* Deserialisation → Level order Traversal

list = {

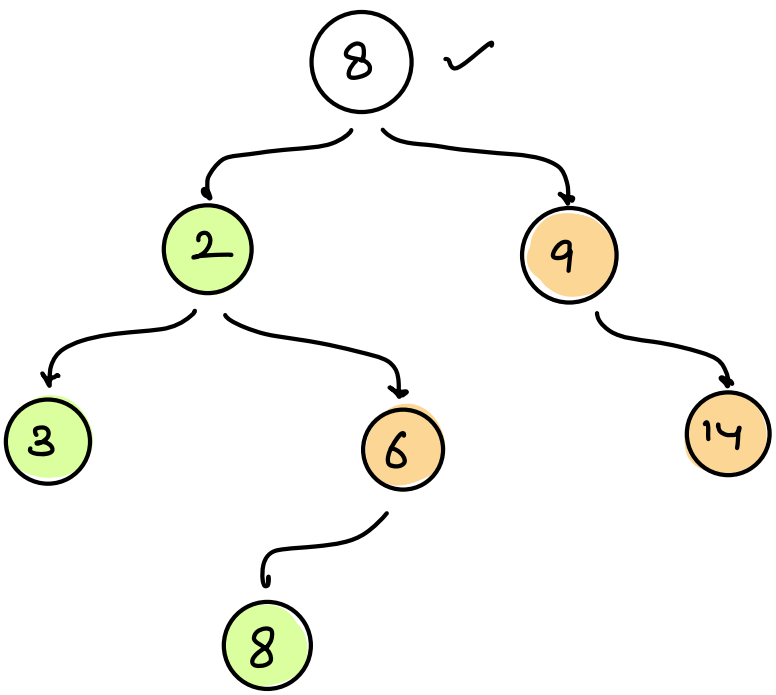
8	2	9	3	6	-1	14	-1	-1	8	-1	-1	-1	-1	-1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

}

i

Queue

~~8~~ ~~2~~ ~~9~~ ~~3~~ ~~6~~ ~~14~~ ~~8~~



```
Node deserialize (list<int> l)
{
    Node root = new Node (l[0]);

    Queue<Node> q;
     q.add (root);

     i = 1

    while (q.size() > 0)
    {
        Node rem = q.remove();
          // left child will be at l[i]
          // right child at l[i+1]

         if (l[i] != -1){
            rem.left = new Node (l[i]);
             q.add(rem.left);
         }
         if (l[i+1] != -1)
         {
            rem.right = new Node (l[i+1]);
             q.add (rem.right);
         }
        i = i+2;
    }
    return root;
}
```

TC: O(n)
SC: O(n)