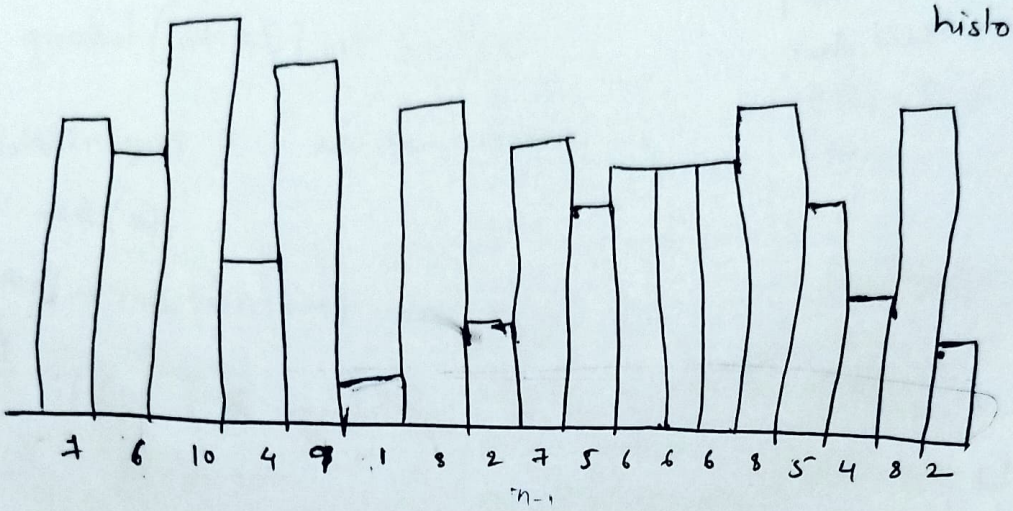
(4) Max rectangular area under histogram



lmin[], rmin[]

ans = max(ans, ((rmin[i] - lmin[i] - 1) * a[i]));

We find the left smallest and right smallest building for current building and find the area of rectangle. For that we precompute them using prev.

Solutions

① N^2, 1

Brute force : for every ele move on its right & left and multiply.

② N+N+N, 3N

A_N: 5 9 1 2 3 6 8 4 1 9 2

K : 4

print max of every subarray of size K

Double ended queue DEQUE (deck)

- Before adding every element, remove all the elements which are lesser than the current element and then append it.

Solutions

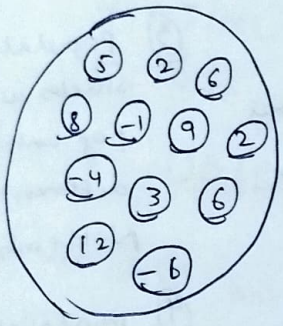
① $(N-K+1)^*K$, 1

→ when we are moving to next window make sure that the left most elements are popped as well.

Linked List

- Two linked lists are joined at one node.

- Hash set solution



Add all nodes of one linked list to a hash set. (ll1)
Now iterate through ll2, and check if the node is in the hash set, the node is start point of joining.

- Stacks soln

① Take 2 stacks, add ll nodes to two stacks, start popping, as long as tops are not equal. The last popped is the ans

② Find length of both ll's and

$$diff = l_2 - l_1$$

Move the head of larger linked list for "diff" times, it is the answer.

LL1: N

5 → 2 → 1 → 8 → -1 → 9 → 2 → 3 → 6 → 12 → 4 → -6

LL2: M: 4 → 8 → 6 → 1 → 9 → 6

Solutions

① N*M, 1

② ~~Hashset~~ N+M, N

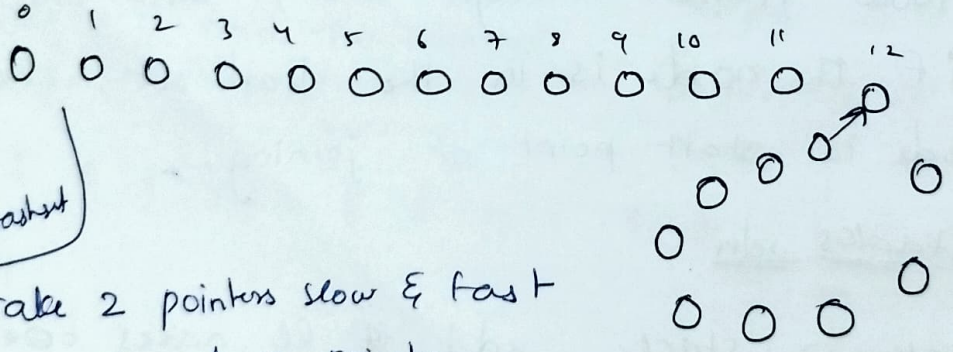
③ Populate two stacks with nodes pop until tops are different
N+M+min(N,M), N+M

④ Find lengths of 2 ll's
find abs diff.
Move head of longer ll for diff times
move heads parallely to find the intersection.

④ min(N,M), 1

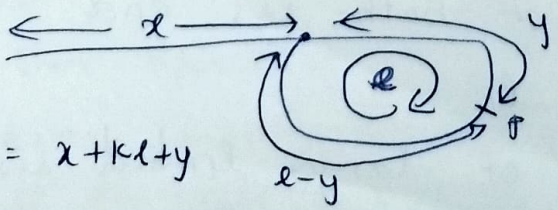
⑤ ~~Fast & slow pointers~~

Cycle detection.



Sol
① N^2, 1
② N, N → Hashset

Take 2 pointers slow & fast
Both will meet at a point.
Now, move one pointer from start and another pointer from point of intersection, then they both will meet at a point, that is the point of cycle



$S = x+y$, $F = x+kl+y$

$F = 2S$

$2(x+y) = (x+y)+kl \Rightarrow kl = x+y \quad (\because x = l-y)$

$x = kl - y$

$x = (k-1)l + (l-y)$

21/11/24

Cache

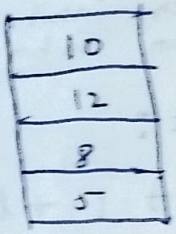
- A Temporary, high speed data storage layer
- Faster retraival of data compare to DB

Applications

1. CPU
2. GPU
3. RAM
4. AI/ML
5. Browsing
6. DB (Redis, mechached) ...

miss miss hit miss hit hit miss miss

5 8 5 12 5 5 10 36 36 36 K=4



when cache is full and it is a miss we evict or replace existing memory

For this we use:

Eviction policies

FIFO/LILO

LFU

LRU

Optimal - performace (Bellady's anomaly)

MFU

MRU

	0	1	2	3	4	5	6	7	8
A[N]:	5	10	5	12	12	8	10	9	6

K: 4

① Maintaining array

② Hash Map

③ Tree Map + Hash Map

3. Miss & Not Full:

$1 + 1 + \log_2 k$

↓ search in hm; ↓ insert in hm; ↓ insert in tm

Hit:

$1 + 1 + \log_2 k + \log_2 k$

↓ search in hm; ↓ update in hm; ↓ delete in tm; ↓ insert in tm

Miss & Full:

$1 + \log_2 k + 1 + 1 + \log_2 k$

↓ search in hm; ↓ delete in tm; ↓ delete in hm; ↓ insert in hm; ↓ insert in tm

$O(N \times \log_2 k, k)$

Solutions

1. Miss & Not full: $O(k)$
   Hit : $O(k)$
   Miss & full : $O(N * (k * N))$

   $O(N + (k * N)), k$

2. Miss & Not full: $O(1)$
   Hit : $O(1)$
   Miss & full : $O(k)$

   $O(N \times k), k$

④ Linked List

⑤ SLL + hash Map

⑥ DLL + hashMap

② 5 10 5 12 12 8 10 9 6

k: 4

5: 02
10: 16
12: 34
8: 5

<num, idx>

when 9 is given the case is Miss & full hence we have iterate through values and remove key with min value

In this iteration over values takes time.

③ 5 10 5 12 12 8 10 9 6

K=4

hm

5:0 2
10:1 6
12:3 4
8:5

tm

0:5 2:5
1:10 6:10
3:12 4:12
5:8

<num, idx> <idx, num>

[treemap sorts the keys, so we want to retrive min idx, so we make idx as key and num as value
We maintain hashmap so that we can check if its a hit and also used to retrive idx in treemap.]

At 9 we get miss and full case.

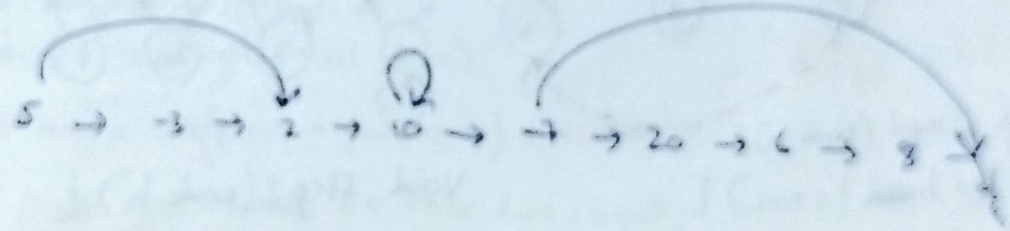
So we remove from both the maps

```
Node dummy = new Node(-1);
Node tail = dummy;
HashMap hm;
~~for(ele in~~
for(int ele : a){
    if(hm.contains(ele))
    {
        Node curr = hm.get(ele);
        if(curr.next == null) continue;
        Curr.prev.next = curr.next;
        curr.next.prev = curr.prev;
        tail.next = curr;
        curr.next = Null;
        curr.prev = tail;
        tail = tail.next;
    }
    else{
        tail.next = new Node(ele);
        hm.put(ele, tail.next);
        tail.next.prev = tail;
        tail = tail.next;
        if(hm.size() == k+1) {
            hm.remove(dummy.next.data);
            dummy.next = dummy.next.next;
        } dummy.next.prev = dummy;
    }
}
```

⑧ No duplicate elements in LL

5 → -3 → 2 → 10 → -1 → 20 → 6 → 8 → X

```
Node clone (Node head){
    Node d = head.Null;
    while (head != null)
    {
        Node n = new Node (head.data);
        d.next = n;
        d = d.next;
    } head = head.next;
}
```



① `int pos = findPos (head, head.rand)`

`newhead.rand = findNode (a, pos)`

$N(N \times N), 1$

for N elements · findPos · findNode

② 

old (32) new

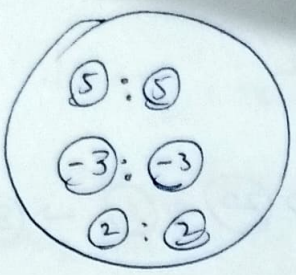
`int pos = hm1.get(h.rand)`

`newhead.rand = hm2.get(pos)`

$N + N + N + N$, $2N$

clone LL · old populate · new populate · for every element connect it rand

(3)



<old add, new add>

newhead.rand = hm.get(head.rand)

N + N + N, N

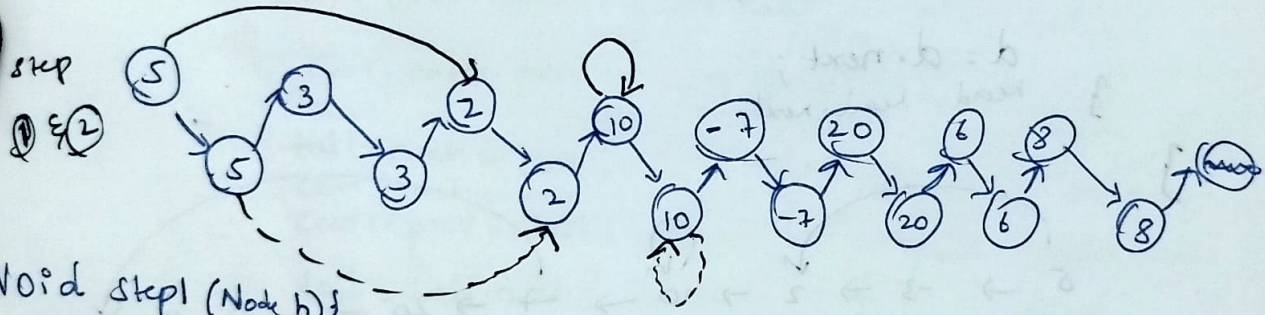
clone SLL → populate hm → access for all elements

(4)

Step 1 : insert new nodes in the given ll

Step 2 : connect randoms

Step 3 : detach the two ll's




```
Void step1 (Node h){
while(h != null) {
    t = h.n;
    h.n = new Node (h.data)
    h.n.n = t;
    h = t;
}
}
```

```
Void step2(Node h) {
while (h != null)
{
    nh = h.n;
    nh.rand = h.rand.n;
    h = h.n.n;
}
}
```

```
Node step3(Node h) {
Node ans = h.next; Node t = ans;
while (h != null){
    h.n = h.n.n;
    t.n = t.n.n;
}
return ans;
}
```

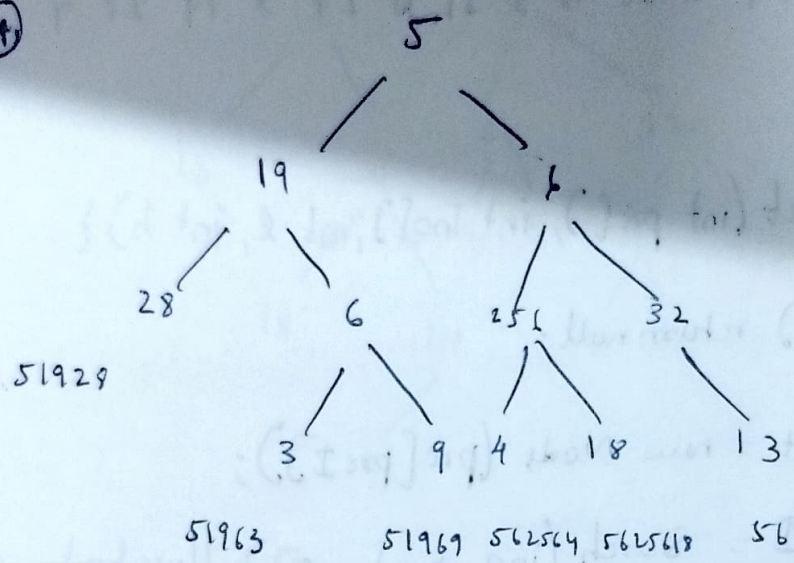
LRU cache

copy list with (LC) rand pointers.

26/11/24

TREES

(1)



find sum of all generated numbers

```
int sum (Node root, int curr){
    if(root==null) return 0;
    int n = NoofDigits(root.data);
    int new = (curr * pow(10,n)) + root.data;
    if(root.right==null && root.left==null){
        return new;
    }
    return sum(root.left, new) + sum(root.right, new);
}
```

(N, 1)

(2) given Inorder, preorder, write post order

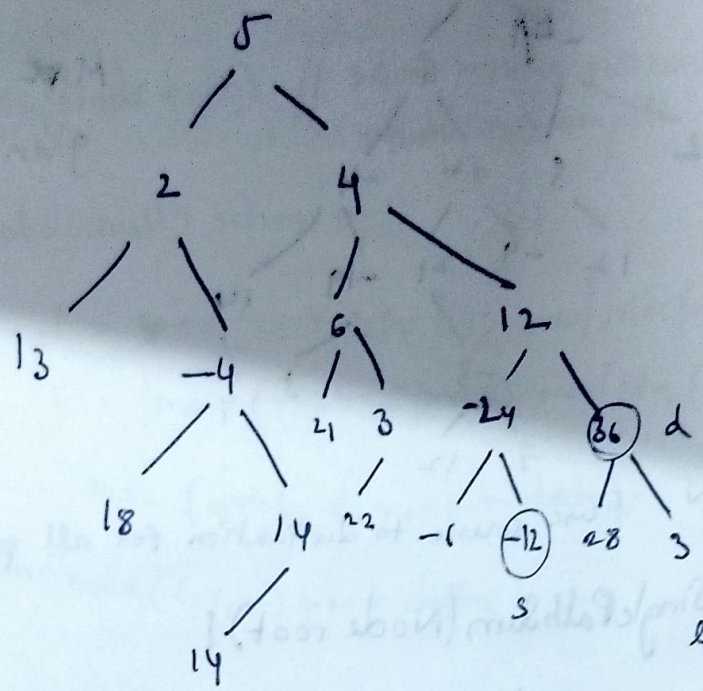
Inorder: 2 24 17 3 13 31 20 15 27 4 19 26 8 9 1

Preorder: 4 19 24 2 20 13 3 31 15 27 8 19 26 9

```
Node construct(int pre[], int ino[], int l, int h){
    if(l > h) return null;
    Node root = new Node(pre[preI]);
    int inoI = search(ino, pre[preI]); //use hashmap
    preI++;
    root.left = construct(pre, ino, l, inoI-1);
    root.right = construct(pre, ino, inoI+1, h);
    return root;
}
```

O(N, N)

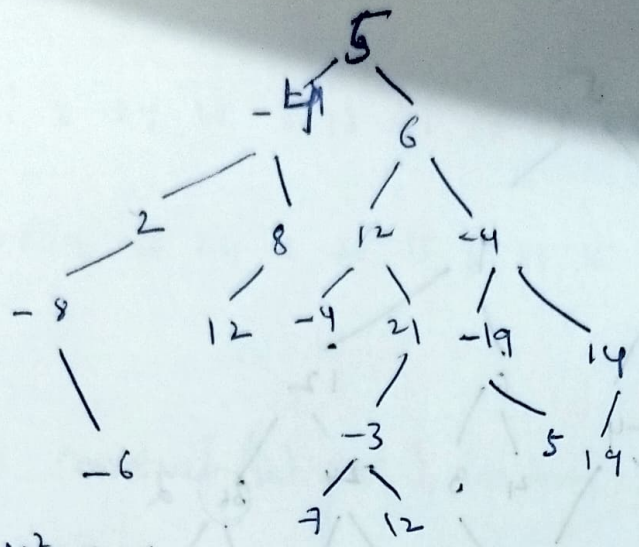
```
//without construction of tree
Node postorder(int pre[], int ino[], int l, int h){
    if(l > h) return;
    int inoI = search(ino, pre[preI]); //use him;
    preI++;
    postorder(pre, ino, l, inoI-1);
    postorder(pre, ino, inoI+1, h);
    print(root.data)
}
```



given 2 nodes source (s) and destination (d) we need to find the distance b/w them.

```
preorder (Node, ar){
        ar.append (node.data);
        preorder (node.left);
        preorder (node.right);
        ar.remove (node.data);
}
```

(+)



Max sum given by any path

① N^2 > N, N (use source to destination for all pairs).

②

```
int maxSinglePathSum(Node root){
    if(root==null) return 0;
    return max(maxSinglePathsum(root.left),
               maxSinglePathsum(root.right), 0)
               + root.data
}
```

$O(N^2, N)$

```
int max = INT_MIN;
void maxSumPath(Node root){
    if(root==null) return;
    int lmax = max(maxSinglePathSum(root.left), 0);
    int rmax = max(maxSinglePathsum(root.right), 0);
    ans = max(ans, lmax + rmax + root.data);
    maxSumPath(root.left);
    maxSumPath(root.right);
}
```

③

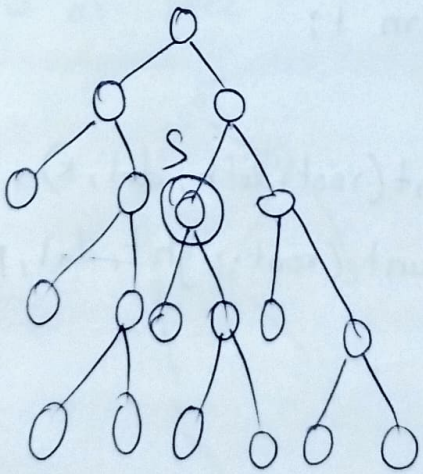
```
int solve(Node root) // return max single path sum
                     // updates max sum path
{
  if(root==null) return 0;

  int l = ~~max(root.left,0)~~; max(solve(root.left),0);
  int r = ~~max(root.right,0)~~; max(solve(root.right),0);

  ans = max(ans, l+r+root.data);

  return max(l,r,0)+root.data;
}
```

$O(N,1)$

④ No of nodes at distance of k from s

k=4 (s is given node)

code (TO-DO)



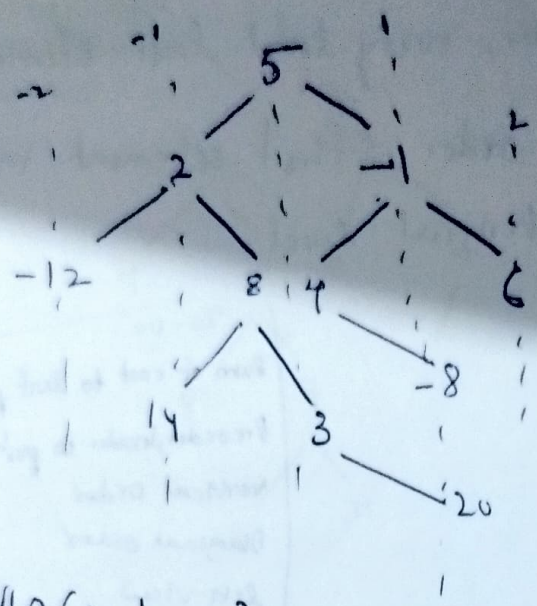
At a distance 'k' from given source node the nodes are below and above also on left and right.

First we count nodes below 's' at distance 'k'

Then we count nodes at distance (k-i) on left, right.

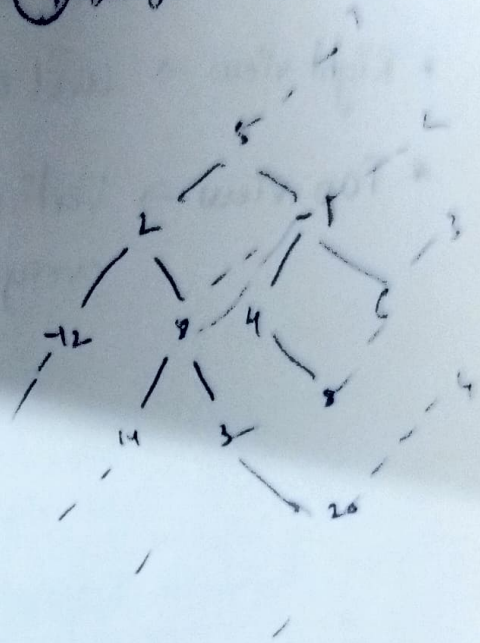
```
count (Node root, int d, int k){
    if (root == null){
        return 0;
    }
    if (d == k){
        return 1;
    }
    return count(root.left, d+1, k) +
           count (root.right, d+1, k)
```

① Vertical Level Order



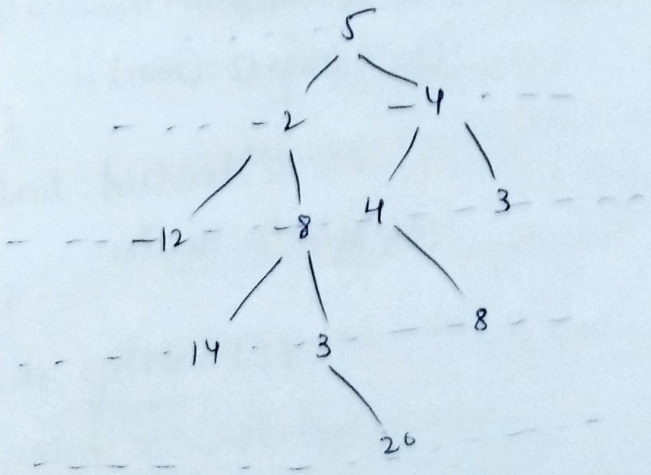

```
VLO(root, l)
{
  //insert in hm
  VLO(root.left, l-1)
  VLO(root.right, l+1)
}
```

② Diagonal level order




```
DLO(root, l)
{
  //insert in hm
  DLO(root.left, l);
  DLO(root.right, l+1);
}
```

③ Left view of Tree



Left view is the first element in horizontal level order.

We use queue to store level order

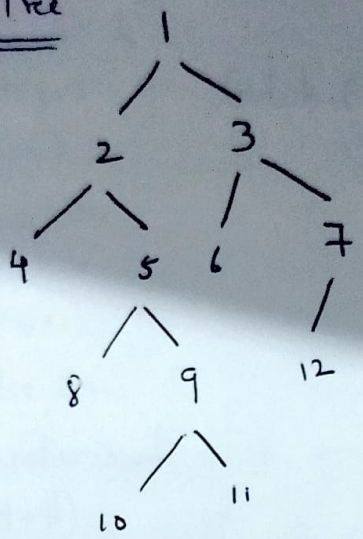
* Left view → level order, every level first element
* Right view → level order, every level last element
* Top view → Vertical order, first element in every vertical level.

TO-DO

Sum of root to leaf path sum
Preorder, inorder to postorder
vertical order
Diagonal order
left view
right view
Top view

29/11/24

Binary Tree



Preorder : 1 2 4 5 8 9 10 11 3 6 7 12

Inorder : 4 2 8 5 10 9 11 1 6 3 12 7

Postorder : 4 8 10 11 9 5 2 6 12 7 3 1

Iterative Preorder :

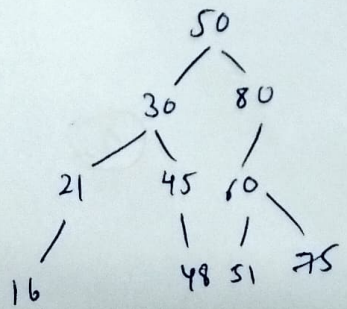
```
class Preorder{
    stack<Node> st;
    preorder(Node root){
        if(root) st.push(root);
    }
    bool hasNext(){
        return st.size>0;
    }
  Node getNext(){
      int x = st.top();
      st.pop();
      if (x.right) st.push(x.right);
      if (x.left) st.push(x.left);
      return x;
    }
}
```

```
main(){
    Preorder obj = new Preorder(root);
    while(obj.hasNext()){
        print(Obj.getNext().data);
    }
```

Iterative Inorder:

```
Class Inorder{
    Stack st;
    bool hasNext(){
        return st.size>0;
    }
    int getNext(root){
        while(root!=null || st.size>0){
            while(root!=null){
                st.push(root);
                root = root.left;
            }
        x = st.top();
        st.pop();
        if(x.right) st.push(x.right);
        root = x.right;
        return x.data;
    }
```

* Given a BST, K check if a+b == K



k=17

Solutions

① For every node, search for k-a) O(N x H), 1

② O(N+N), N

Use a set, search for (K-a)

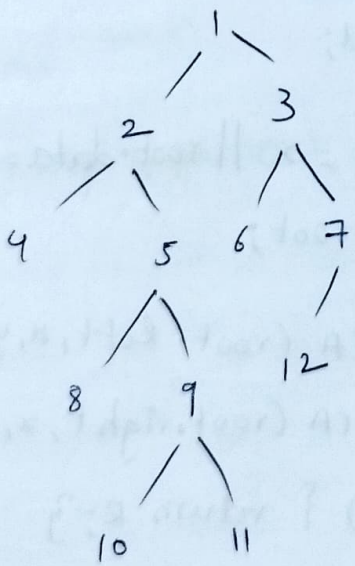
③ Use an array and store the elements in inorder and use 2 pointer technique
O(N+N, N)

④ While adding into set, check if k-a exists O(N, N)

⑤ Two pointer on tree by iterative inorder
O(N, H+H)

⑤
```
Inorder P1 = new Inorder(root);
RevInorder P2 = new RevInorder(root);
P1 = obj1.Next(); P2 = obj2.Next();
while(P1.data <= P2.data){
    int curr = p1.data + p2.data;
    if (curr==k) return true;
    elif(curr>k) P2 = obj2.getNext();
    else         P1 = obj1.getNext();
}
return false;
```

* Least common Ancestor (LCA)



Given a BT, find LCA

LCA(8,11) → 5

LCA(4,7) → 1

LCA(5,10) → 5

Solutions

① Get path of both source and destination nodes from root node and remove all common nodes.

$O(N+N+N, N)$

② If we find a node such that we can find both nodes either on left or right or both, then we can declare it is an ancestor.

```
Node LCA (Node root, int x, int y){
    if (root == null){
        return null;
    }
    if(root.data == x || root.data == y){
        return root;
    }
    Node L = LCA (root.left, x, y);
    Node R = LCA (root.right, x, y);
    if (L == Null ) { return R; }
    if (R == Null) { return L; }
    return root;
}
```

if tree is BST

```
Node LCA (Node root, int x, int y){
    if(root == null) return null;
    if(root.data > x && root.data > y) return LCA (root.left, x, y)
    if(root.data < x && root.data < y) return LCA (root.right, x, y)
}
```

8/12/24

Q given N sorted arrays, merge them into a single sorted array.

Take N pointers and compare them, sort them using minheap.

Store N elements in Min heap and every time, delete min element and store next element.

TC -> O(NxM(logN + logN), N)

Store as

Heap < pair<int, pair<int, int>> >

(2)

A file numbers.txt has bunch of nos to be sorted to sorted.txt file.

The size of file is 10TB

$10TB = 10 \times 10^3 GB$

$= 10^4 \times 10^3 MB$

$10TB = 10^7 MB$

To store integers

$1 \rightarrow 4B$

$10 \rightarrow 40B$

$1000 \rightarrow 4000B = 4kB$

$10^6 \rightarrow 4MB$

$10^7 \rightarrow 40MB$

So in given 10TB space we can divide it as $\frac{10^7}{40} \approx 10^6$ chunks

each chunk is 40MB

We sort each chunk and store it.

Now this turns out to be the matrix with sorted row (here each sorted chunk is a row)

So our min heap always has 10^6 elements, which is a possible size of array (since minheap is stored as array)

Now from min heap we place numbers into sorted.txt

Q

	0	1	2	3	4	5	6	7	8
A_N:	5	9	-1	2	8	10	14	6	-12

for all sub arrays print medians

Solutions

① $N \cdot (N \log N), N/1$ → size 1 if we sort in same array

- for all subsets
- sort each subset
- size N if we dont in other array

② $N * N, 1$

Insertion sort for each element

③ $N(\log N), N$

↓ for operations like insert, push, pop

Check A.size - B.size == 0/1

- → < 0 pop min from right & push on left
- → > 1 pop max from left & push on right

then print max of 'A' which is median

05/12/2024

Summer Challenge.

AN : 11 10 19 23

K = 4

The student can solve almost K problems a day in a contest. print sequence of indices which cmplt first.

Solutions

① ~~T x NlogN~~

T x sum(arr) x N log N ,

for sum(arr) times we sort and update answer.

② T * Sum(arr) $\log_2 N$

for sum(arr) time we update elements in a Max heap (so sorting time decreases)

③

11 10 19 23

11 10 19 19

11 10 15 19

11 10 15 15

⋮

3 2 3 3

instead of continuous subtraction we use % K then we insert in heap