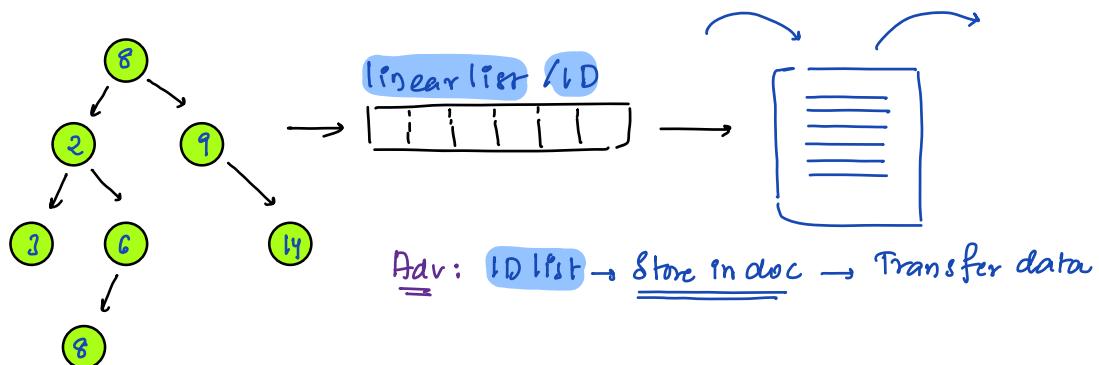


Today's Content:

- Serialization
- deserialization
- Max sum path
- diameter

Serialization: We will store every node, & child information as well



Adv: ID list → Store in doc → Transfer data

Idea: Inorder[] & preOrder[] construct tree?

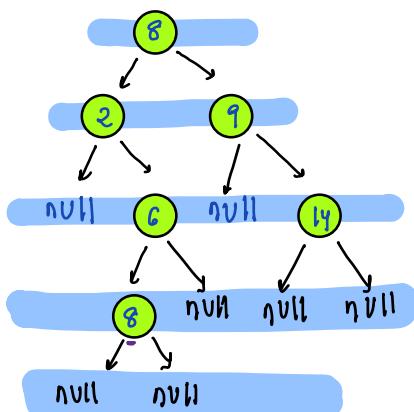
preOrder[]: 8 2 3 6 8 9 14 } Note: If data repeats, we
cannot construct tree using
Inorder[]: 3 2 8 6 8 9 14 } pre[] & in[]

Serialization: Converting Tree → ID list : Note:

→ preOrder / postOrder / levelOrder / Inorder *? [TODO] ↳ Explain in coming

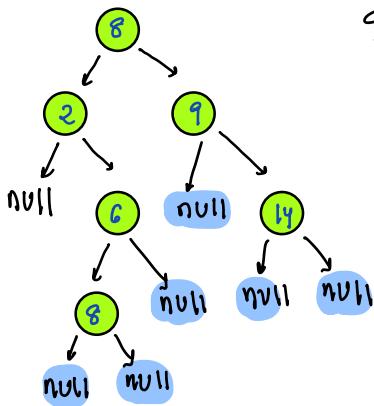
Serialization level order: Store info about nodes & children

8 2 9 null 6 null 14 8 null null null null



→ Say store list<int> : Assignments / Interviewbit

- : Storing null is not possible
 - : Instead of storing null, store -1
 - : If data is the
- Say store list<string> : Real world
- : Storing "null" is possible
 - : Storing number as strings is possible



Obsr: When we pop a node
 → if node not null
 a) push data in lfst
 b) Children in Queue
 → if node is null
 a) push -1 in lfst

Qs: 8 | 2 9 | null 6 | null 14 | 8 null null null null }

Li: 8 2 9 -1 6 -1 14 8 -1 -1 -1 -1 -1 }

} { lfst <--> }

list <--> Serializatn level order (Node root){ TC:O(N) SC:O(N) }

lfst <--> li

Queue <--> q;

q.enqueue(root)

while(q.size() > 0) {

 Node tmp = q.front();

 q.dequeue();

 if(tmp != null) {

 li.insert(tmp.data);

 q.enqueue(tmp.left);

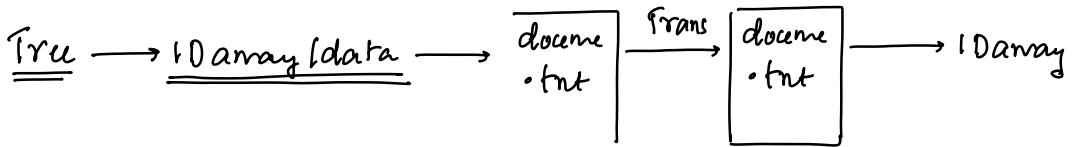
 q.enqueue(tmp.right);

 } else {

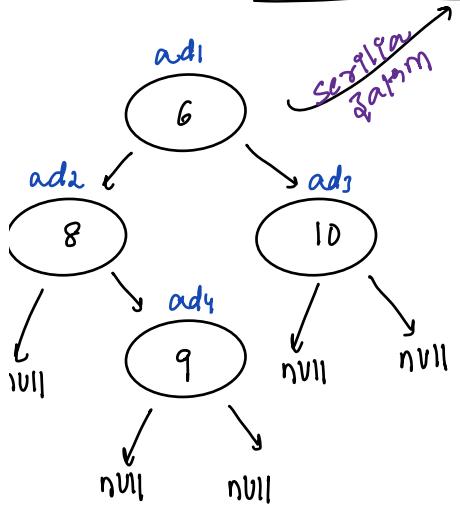
 li.insert(-1); // instead of null we are pushing -1

}

return li



6 8 10 null 9 null null null null



seg1
datm

desg1
t4m

cd1

cd2

cd3

cd4

cd5

cd6

cd7

cd8

cd9

cd10

cd11

cd12

cd13

cd14

cd15

cd16

cd17

cd18

cd19

cd20

cd21

cd22

cd23

cd24

cd25

cd26

cd27

cd28

cd29

cd30

cd31

cd32

cd33

cd34

cd35

cd36

cd37

cd38

cd39

cd40

cd41

cd42

cd43

cd44

cd45

cd46

cd47

cd48

cd49

cd50

cd51

cd52

cd53

cd54

cd55

cd56

cd57

cd58

cd59

cd60

cd61

cd62

cd63

cd64

cd65

cd66

cd67

cd68

cd69

cd70

cd71

cd72

cd73

cd74

cd75

cd76

cd77

cd78

cd79

cd80

cd81

cd82

cd83

cd84

cd85

cd86

cd87

cd88

cd89

cd90

cd91

cd92

cd93

cd94

cd95

cd96

cd97

cd98

cd99

cd100

cd101

cd102

cd103

cd104

cd105

cd106

cd107

cd108

cd109

cd110

cd111

cd112

cd113

cd114

cd115

cd116

cd117

cd118

cd119

cd120

cd121

cd122

cd123

cd124

cd125

cd126

cd127

cd128

cd129

cd131

cd132

cd133

cd134

cd135

cd136

cd137

cd138

cd139

cd140

cd141

cd142

cd143

cd144

cd145

cd146

cd147

cd148

cd149

cd150

cd151

cd152

cd153

cd155

cd156

cd157

cd158

cd159

cd160

cd161

cd162

cd163

cd164

cd165

cd166

cd167

cd168

cd169

cd170

cd171

cd172

cd173

cd174

cd175

cd176

cd177

cd178

cd179

cd180

cd181

cd182

cd183

cd184

cd185

cd186

cd187

cd188

cd189

cd190

cd191

cd192

cd193

cd194

cd195

cd196

cd197

cd198

cd199

cd200

cd201

cd202

cd203

cd204

cd205

cd206

cd207

cd208

cd209

cd210

cd211

cd212

cd213

cd214

cd215

cd216

cd217

cd218

cd219

cd220

cd221

cd222

cd223

cd224

cd225

cd226

cd227

cd228

cd229

cd230

cd231

cd232

cd233

cd234

cd235

cd236

cd237

cd238

cd239

cd240

cd241

cd242

cd243

cd244

cd245

cd246

cd247

cd248

cd249

cd250

cd251

cd252

cd253

cd254

cd255

cd256

cd257

cd258

cd259

cd260

cd261

cd262

cd263

cd264

cd265

cd266

cd267

cd268

cd269

cd270

cd271

cd272

cd273

cd274

cd275

cd276

cd277

cd278

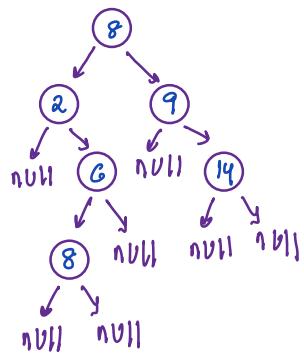
cd279

deserialization Using level order:

	0	1 2	3 9	5 6	7 8	9	10	11	12
i=13	8	2 9	-1 6	-1 14	8 -1	-1 -1	-1 -1		
	level	8	x 9	f 6	y 14	z 8			

Steps:

- Pop node from Queue say: temp
- for temp left child is at index i : insert leftchild in queue if not null
- for temp right child is at index i+1 : insert rightchild in queue if not null
- $i = i + 2$

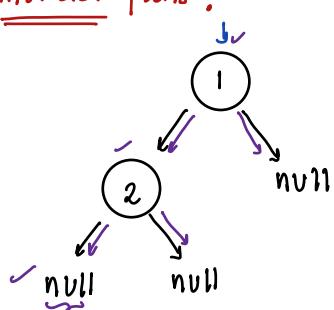


Node deserializeLevelOrder(list<int> l) TC: O(N) SC: O(N)

```
Node root = new Node(l[0])
Queue<Node> que
que.enqueue(root)
i = 1
while(que.size() > 0) {
    Node tmp = que.front()
    que.dequeue()
    if(l[i] != -1) { // left child of tmp at index i
        tmp.left = new Node(l[i])
        que.enqueue(tmp.left)
    }
    if(l[i+1] != -1) { // right child of tmp at index i+1
        tmp.right = new Node(l[i+1])
        que.enqueue(tmp.right)
    }
    i = i + 2
}
return root;
```

10:33 pm 28

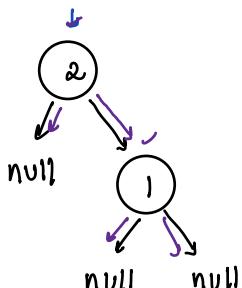
Why inorder fails?



inorder Serializatim:

null 2 null 1 null

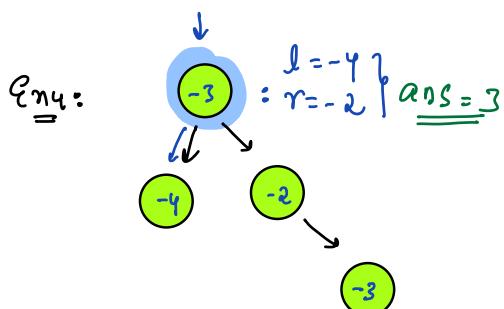
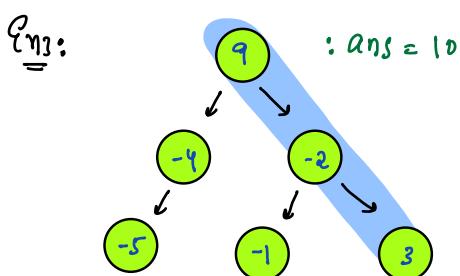
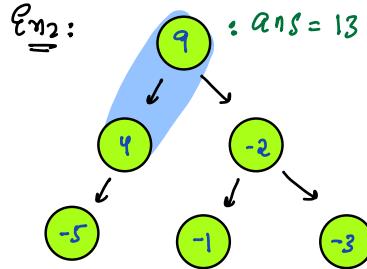
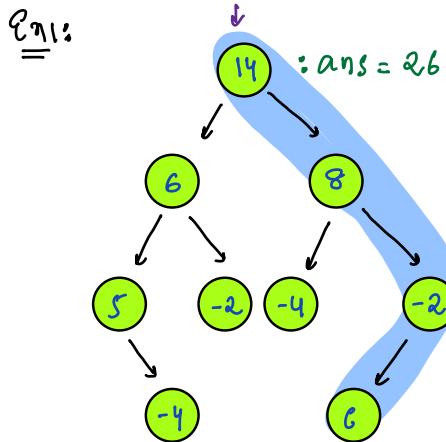
{
Issue: different
tree can have
same inorder
Serializatim,
de-serializatim
not possible}



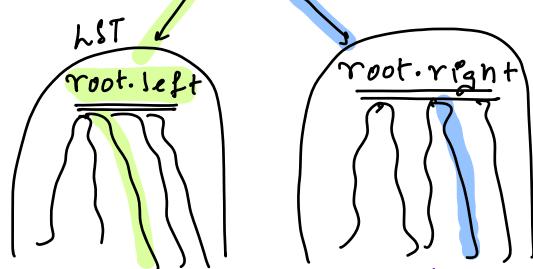
inorder Serializatim:

null 2 null 1 null

Q1: Man path sum starting from **root**? : {Can end anywhere}



Idea: man path sum \rightarrow root.data + man(l, r, o)



// If l < 0, r > 0 ignore

$l = \text{manpathsum in LST start at root.left}$

$r = \text{manpathsum in RST start at root.right}$

Ass: Given tree, return man pathsum starting at root node

int manpathsum startRoot (Node root) { Tc: O(N) Sc: O(H) }

```

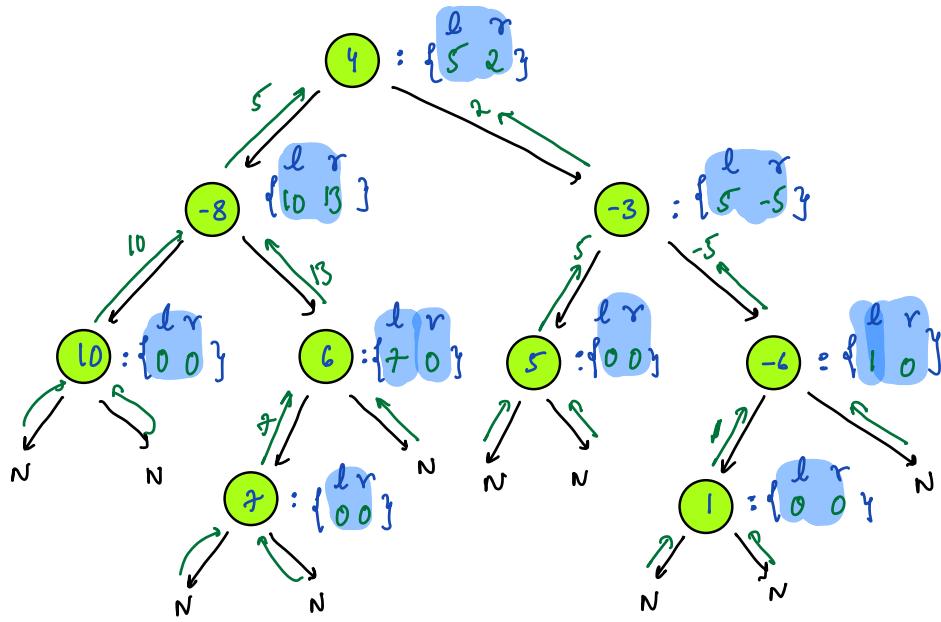
if (root == null) { return 0 }

int l = manpathsum startRoot (root.left)
int r = manpathsum startRoot (root.right)

return root.data + man3(l, r, 0)
  
```

\hookrightarrow // will return man of 3 numbers

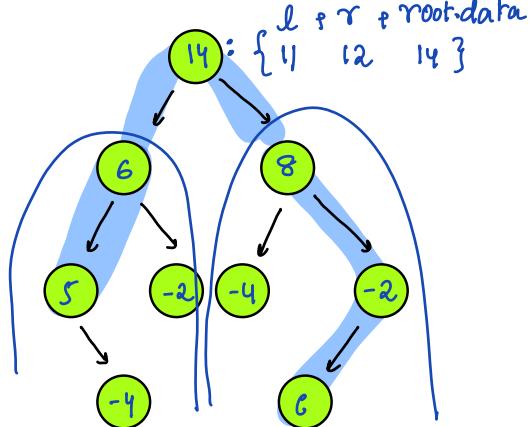
Tracing: l = manpathsum in LST start at root.left
 r = manpathsum in RST start at root.right



Q2) Man path sum passing root node in given Tree
 → path containing root node

E_n1:

$$ans = 37$$

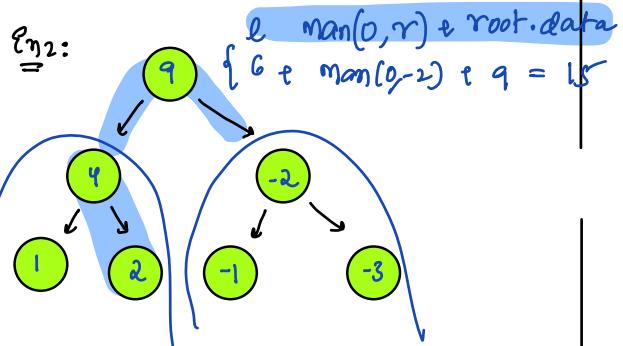


$$ans = 15$$

E_n2:

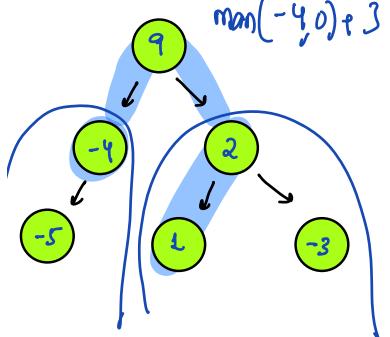
$$l \ man(0, r) + \text{root.data}$$

$$\{6 + \text{man}(0, 2) + 9 = 15\}$$



E_n3: $ans = 12$ $man(0, l) + \text{root.data}$

$$man(-4, 0) + 3 + 9 = 12$$

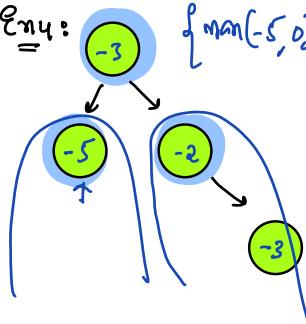


$$ans = -3$$

$$man(l, 0) + man(r, 0) + \text{root.data}$$

$$\{man(-5, 0) + man(-2, 0) + -3\} = -3$$

E_n4:



// Given root node:

if (root == null) { return 0; }

$l = \text{manPathSumStartRoot}(\&\text{root.left})$ // manPathSumStartRoot of LST

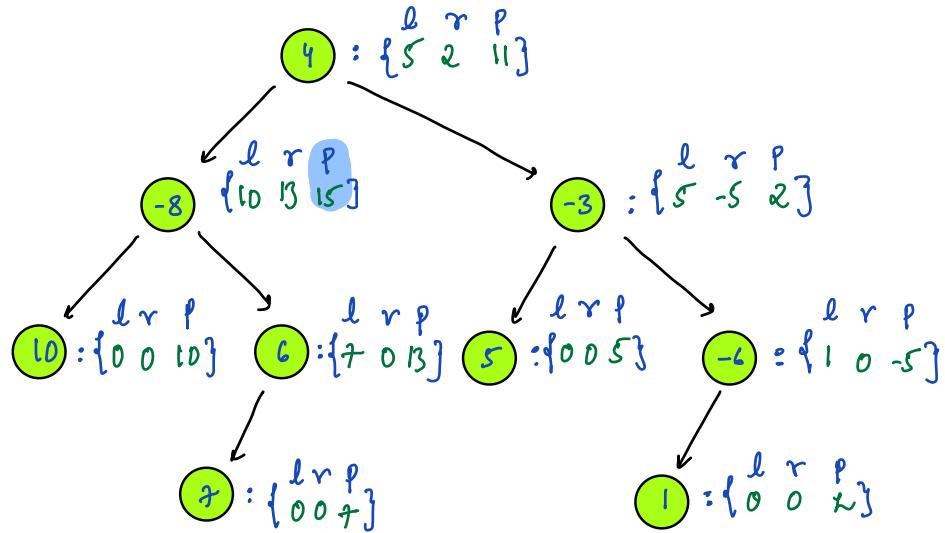
$r = \text{manPathSumStartRoot}(\&\text{root.right})$ // manPathSumStartRoot of RST

$val = \text{root.data} + man(0, l) + man(0, r)$

// val: man path sum containing root node

}

Last Observation: Max sum path passing via root:
 $= \text{man}(l, o) + \text{man}(r, o) + \text{root.data}$



// When we call below function for every node we get max sum path
 Start in LST & Start in RST

{ Max path sum in BT }

int ans = INT_MIN

int manPathSum startRoot (Node root) { TC: O(N) SC: O(H) }

```

if (root == null) { return 0 }

int l = manPathSum startRoot (root.left)
int r = manPathSum startRoot (root.right)

int p = man(l, o) + man(r, o) + root.data // Max sum path passing
                                            across the root node
ans = man(ans, p) // Max path sum in BT
return root.data + man3(l, r, o)
}

```

// Once code is return ans;

Obs: If all data of nodes, we say it is 1, and calculate max

sum path, $\text{manPathInTree} - 1 = \text{diameter of Tree}$ // longest path in Tree

—

—

—

—