Problem Solving Session

let us solve problems based on trees and linked list.

Hello Everyone
Very Special Good Evening to all of you 😊😊😊
We will start session from 9:06 PM

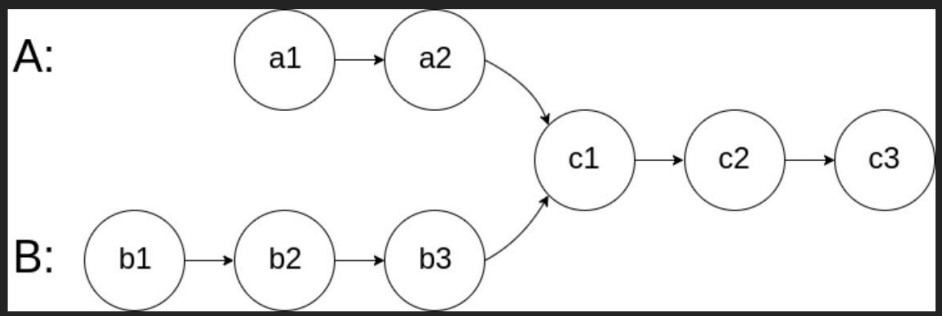
160. Intersection of Two Linked Lists

Solved

Easy | Topics | Companies

Given the heads of two singly linked-lists `headA` and `headB`, return *the node at which the two lists intersect*. If the two linked lists have no intersection at all, return `null`.

For example, the following two linked lists begin to intersect at node `c1`:

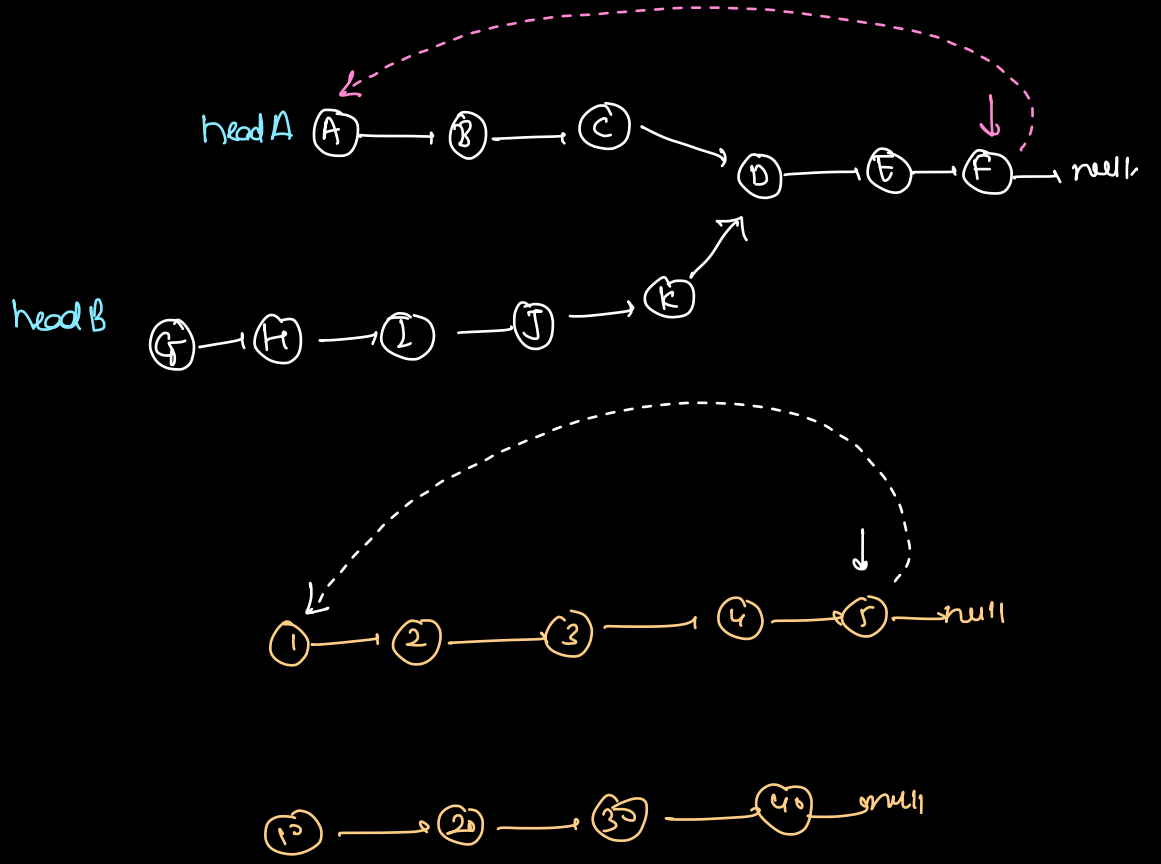


The test cases are generated such that there are no cycles anywhere in the entire linked structure.

Note that the linked lists must **retain their original structure** after the function returns.

Insight

Explanation:



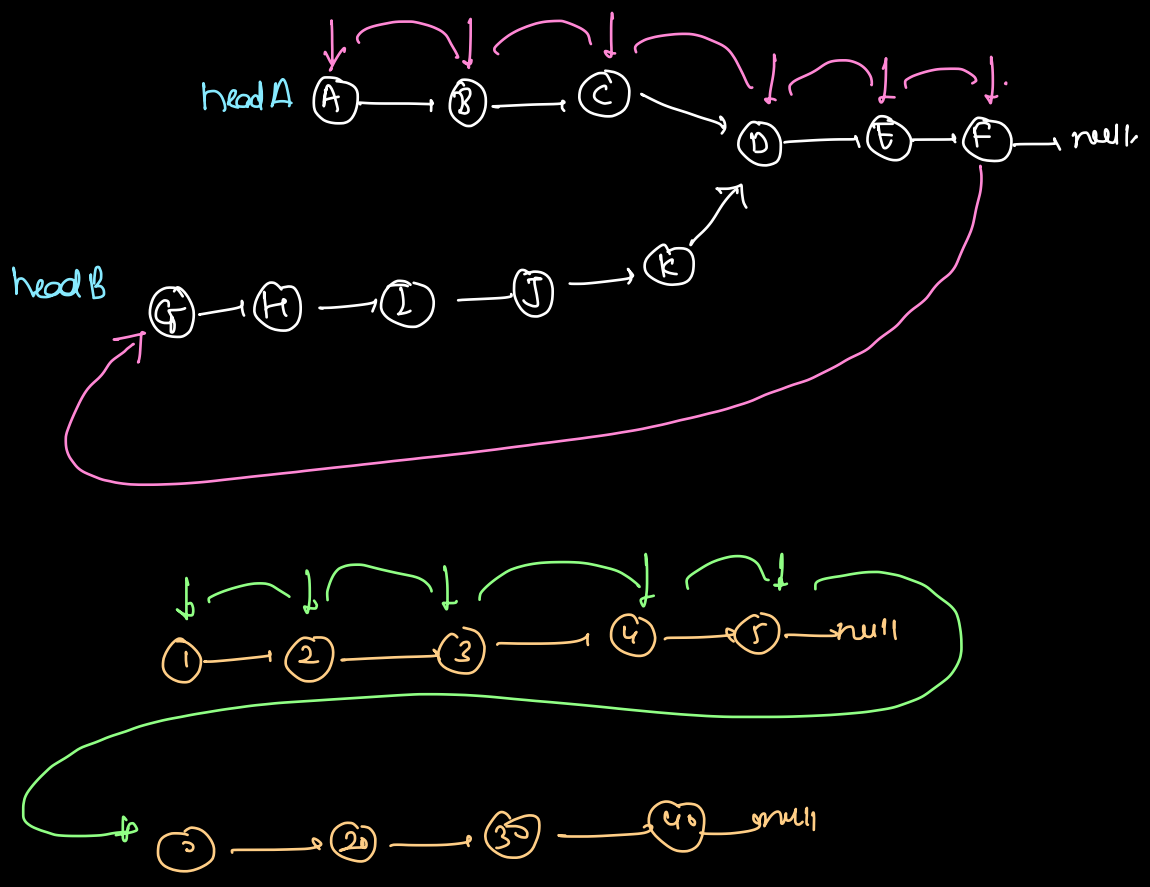
Floyd Cycle Detection

(1) Detect cycle

(2) Using that meeting point find start of cycle.

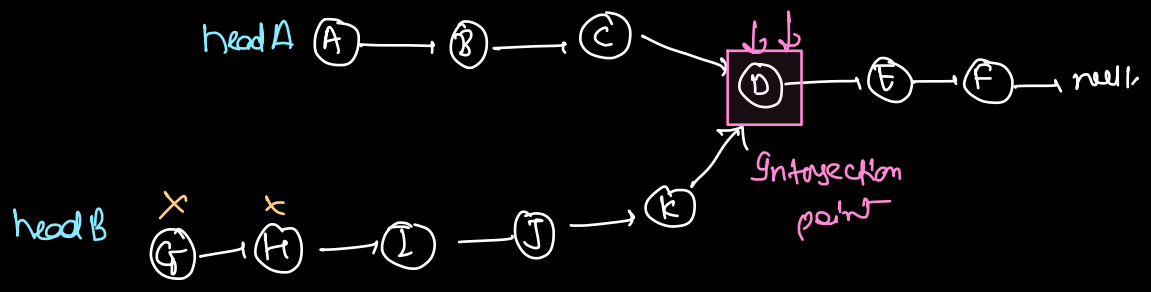
(2) → one.

Using Cyclic Behaviour of LL :



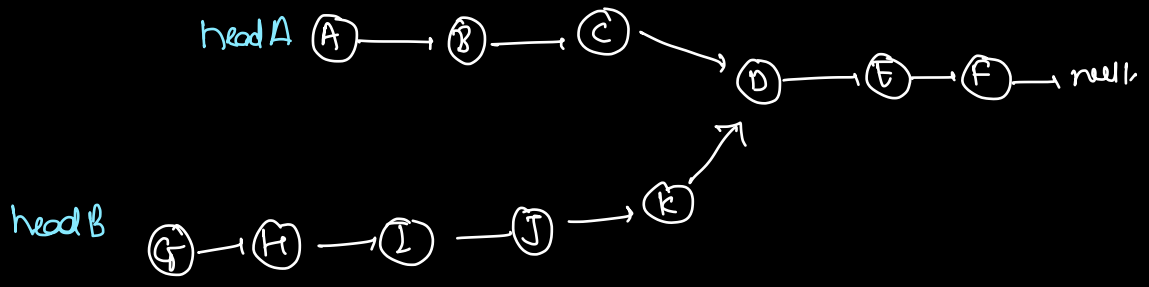
Using length.

length A = 6



length B = 8

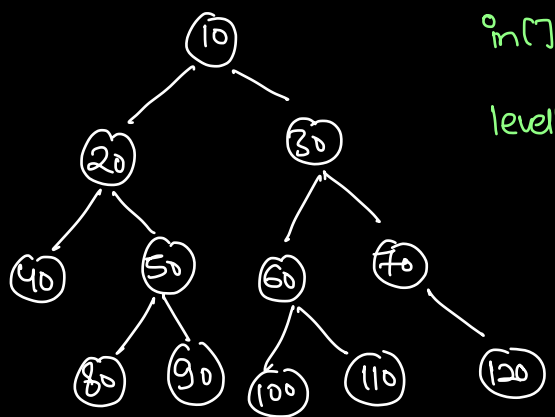
Using HashSet.



fill All nods of headA in set.

Start from headB, and insert nods in Hashset,

if already avl. that means it is Intersection node.

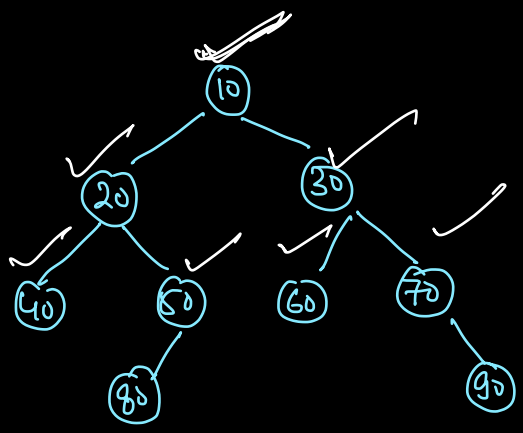


in[]: 40 20 80 50 90 10 100 60 110 30 70 120

level[]: 10 20 30 40 50 60 70 80 90 100 110 120

inorder → left Node Right

level order → Level by level,



in[]: [40, 20, 80, 50, 10, 60, 30, 70, 90]

L[]: [10, 20, 30, 40, 50, 60, 70, 80, 90]

Root

10

in[]: [40, 20, 80, 50]

L[]: [20, 40, 50, 80]

Root

20

in[]: [60, 30, 70, 90]

L[]: [30, 60, 70, 90]

Root

30

in[]: [40]

L[]: [40]

40

in[]: [80, 50]

L[]: [50, 80]

Root

50

in[]: [60]

L[]: [60]

60

in[]: [70, 90]

L[]: [70, 90]

Root

70

in[]: [80]

L[]: [80]

80

in[]: []

L[]: []

in[]: []

L[]: []

in[]: [90]

L[]: [90]

90

in[]: 40 20 80 50 90 10 100 60 110 30 70 120

level[]: 10 20 30 40 50 60 70 80 90 100 110 120

Alignment :) → [Participate in Discussion]

Dates: Topics:

3 → T, 2L

8 → Tries

10 → Backtracking

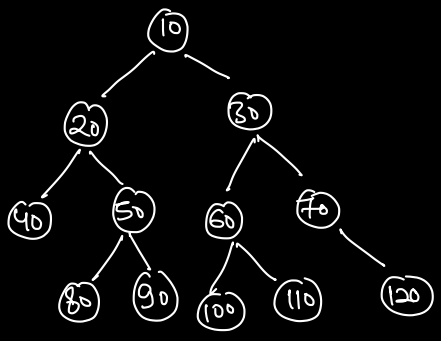
13 → Strings & PM

15 → DP

20 → Graphs

24 → Contest

27 → Discussion



element vs index

40 — 0

20 — 1

80 — 2

50 — 3

90 — 4

10 → 5

:

in[]: 40 20 80 50 90 10 100 60 110 30 70 120

level[]: 10 20 30 40 50 60 70 80 90 100 110 120

10

in[]: [40, 20, 80, 50, 90]

level[]: [20, 40, 50, 80, 90]

Root

20

in[]: [40]

lvl[]: [40]

20

in[]: [80, 50, 90]

lvl[]: [50, 80, 90]

Root

50

in[]: [80]

lvl[]: [80]

80

in[]: [90]

lvl[]: [90]

90

in[]: [100, 60, 110, 30, 70, 120]

lvl[]: [30, 60, 70, 100, 110, 120]

Root

30

in[]: [100, 60, 110]

lvl[]: [60, 100, 110]

Root

60

in[]: [100]

lvl[]: [100]

100

in[]: [110]

lvl[]: [110]

110

in[]: [70, 120]

lvl[]: [70, 120]

Root

70

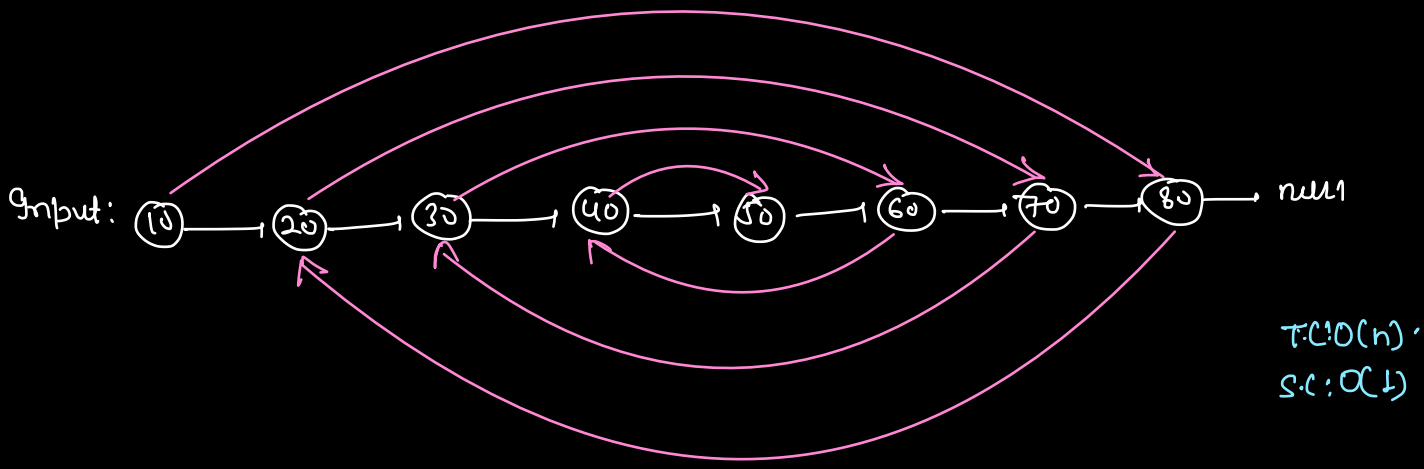
in[]:

lvl[]

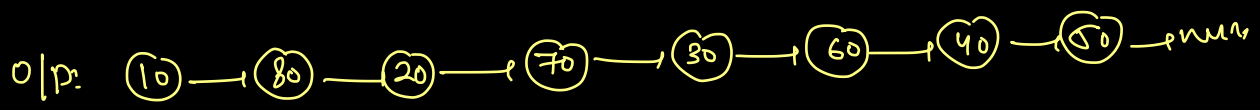
in[120]

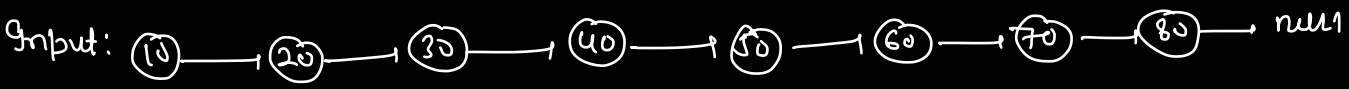
lvl[120]

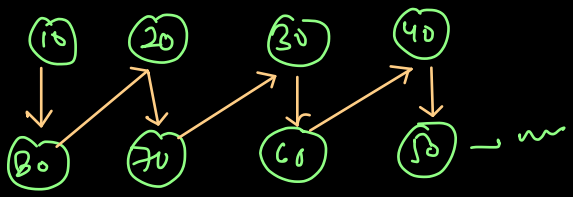
120



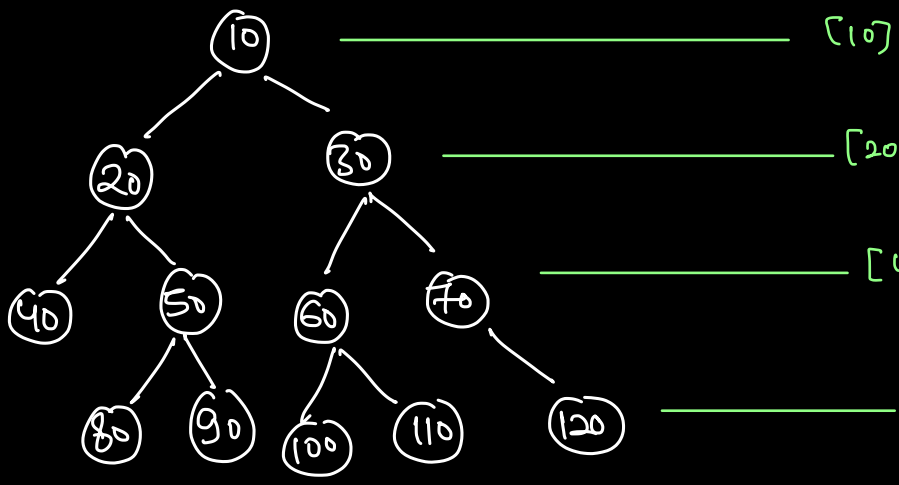
T.C: O(n).
S.C: O(1)







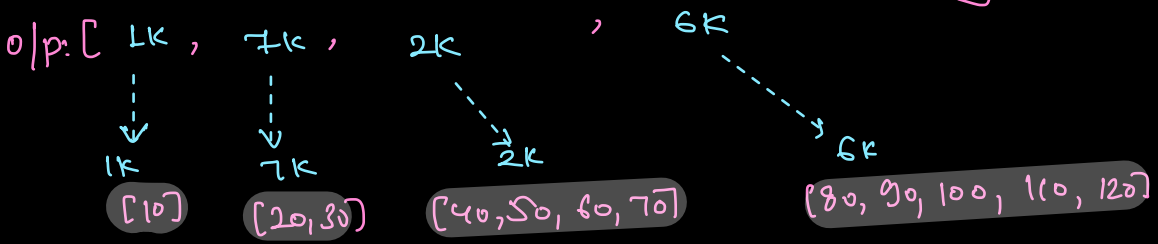
- Split LL in two parts.
- Reverse second part
- Alternate linkage.



o/p: [[80, 90, 100, 110, 120], [40, 50, 60, 70], [20, 30], [10]]

2D - Dynamic Array. ⟶ return type

using space for o/p:



10:18 - 10:28 pm

Break.

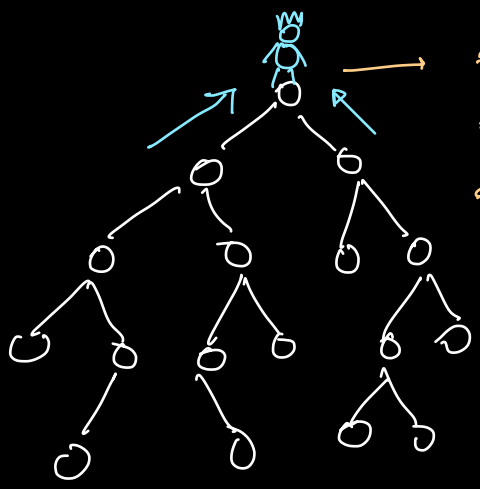
124. Binary Tree Maximum Path Sum

Hard | Topics | Companies

A **path** in a binary tree is a sequence of nodes where each pair of adjacent nodes in the sequence has an edge connecting them. A node can only appear in the sequence **at most once**. Note that the path does not need to pass through the root.

The **path sum** of a path is the sum of the node's values in the path.

Given the `root` of a binary tree, return *the maximum **path sum** of any **non-empty** path*.



* ans can be there in left subtree
* ans can " " " right subtree
* ans is mostly through Root

Expectation from left subtree & Right subtree
→ * It will return me any from left tree.
* max sum from node to Root path.

Diameter of tree.

Root → Prepare our answer which is passing through Root
left Node2Root path + Right Node2Root path + node val.

Vs.

left subtree ans.

Vs.

Right subtree ans.

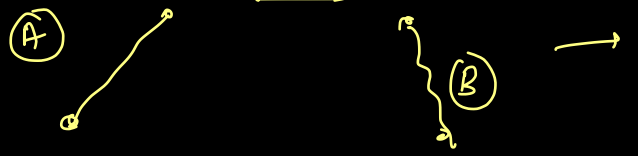
overall ans from the Root

Return → [overall answer, max node2Root path]

(A) int nodeTonodeSum_left = solve([redacted].left);

(B) int nodeTonodeSum-Right = solve(root.right);

max_ = max(A, B) + node.value



ans = max(A + Root + B ,
vs.
ans, ans-available till now
vs
Root
vs.
max_

Return → max(max_ , node.value)