i = 2
stack = [2, 5)
result [-1, -1, 2, -1, -1]

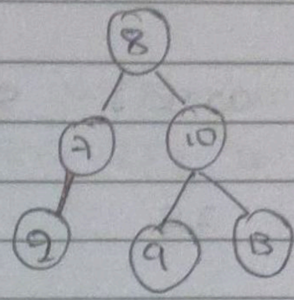
i = 1
stack [2, 5, 8]
res = [-1, 5, 2, -1, -1]

i = 0
stack [2, 4)
res = [2, 5, 2, -1, -1]

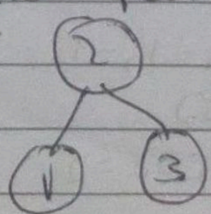
Binary search tree is a binary tree: properties
Left subtree of a node contains only nodes with keys lesser than node's key.
the right subtree of a node contains only nodes with key greater than node's key
the left subtree & right subtree each must also be a BST

problem:
Given a Binary Search Tree that contains unique positive integer values greater than 0 the task is find if the BST contains a dead end. Here Dead End means leaf node at which no other integer can be inserted.

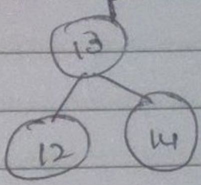
Ex:



Dead is 9 why? why not 2 or 13
first let know when a node become leaf node.?
So the condition low = high become true that node is called as leaf whenever no more values can be inserted!!
for low = node - 1 } to calculate high & low
for high = node + 1
let calculate for 2
low of 2 = 2 - 1 => 1
high of 2 = 2 + 1 => 3
so both 1 & 3 are not present in the tree so 2 can have child further



next for (13)
low for 13 => 13-1 = 12
high for 13 => 13+1 = 14
both 12 & 14 are not present meaning they can be a child of 13 so 13 cannot be the leaf node.



then come 9
low for 9 => 9-1 = 8 return 9
high for 9 => 9+1 = 10 return 9
both 8 & 10 are already there in the tree
if there meaning high = low 9 cannot be divided further. so it becomes dead end.
=> "i.e No no. can be inserted between low & high (9==9)"

Approach

(1) Recursive Range-Based Approach (DFS)

(1) If the node is none return false no dead end found
(2) if low = high return true dead is found
(3) Recursive calls
Recur for left subtree with updated range [low, x-1)
Recur for right subtree with updated range [x+1, high)
(4) If either subtree return True else return false.

Node	low	high	low=high?	left Rec	right Rec	Result
8	1	∞	✗	check 7	check no	✓
7	1	6	✗	check 2	no right	✓
2	1	3	✗	No	NO	✓
10	9	∞	✗	check 9	check 10	✓
9	9	9	✓	—	—	✓
13	12	∞	—	—	—	✗

time complex O(N)
Space complex O(H)

(2) Set-Based Approach (using 2 set)
(1) create 2 sets: all_nodes & leaf_nodes
(2) Traverse the BST to fill these sets
Add every node to all_nodes. If a node is a leaf add it to leaf_nodes

(3) Check for Dead Ends:

for each leaf node x,

check if both x-1 & x+1 exist

in all_nodes

if true, return true (dead end found)

(4) Return false if no dead ends exist.

Ex:

all_nodes = {8, 7, 10, 2, 9, 13}

leaf_nodes = {2, 9, 13}

leaf node	x-1 Exist?	x+1 Exists	Dead End?
2	1 ✗	3 ✗	✗
9	8 ✓	10 ✓	✓
13	12 ✗	14 ✗	✗

Set based is best because It works for all BST structures.