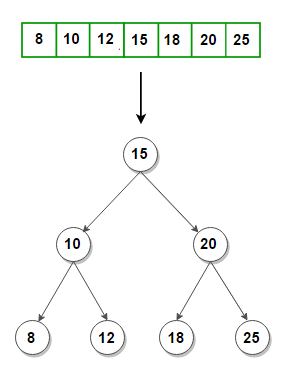
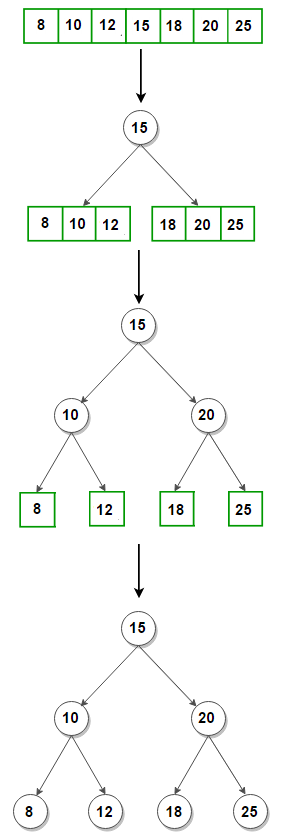
1. **Construct a height-balanced BST from a sorted doubly linked list**

Given a sorted doubly linked list, [**in-place**](https://www.techiedelight.com/in-place-vs-out-of-place-algorithms/) convert it into a height-balanced binary search tree (BST). The difference between the height of the left and right subtree for every node of a height-balanced BST is never greater than 1.

The conversion should be done such that the previous child pointer of a doubly-linked list node should act as a left pointer for a binary tree node, and the next child pointer should act as the right pointer for a binary tree node. The conversion should also be done by only exchanging the pointers without allocating any memory for the BST nodes.

For example,





class NodeWrapper

{

public TreeNode TreeNode;

NodeWrapper(TreeNode TreeNode) {

this.TreeNode = TreeNode;

}

}

public class AssignDLLtoBST {

public static TreeNode push(TreeNode head, int data)

{

// allocate a new TreeNode and link it at the beginning

TreeNode TreeNode = new TreeNode(data);

TreeNode.right = head;

// change `prev` of the existing head TreeNode to point to the new TreeNode

if (head != null) {

head.left = TreeNode;

}

// update head pointer

head = TreeNode;

return head;

}

// Function to print and count the total number of nodes in a doubly-linked list

public static int printAndCountNodes(TreeNode TreeNode)

{

int counter = 0;

while (TreeNode != null)

{

System.out.print(TreeNode.data + " ");

TreeNode = TreeNode.right;

counter++;

}

System.out.println();

return counter;

}

// Function to print preorder traversal of the BST

public static void preorder(TreeNode root)

{

if (root == null) {

return;

}

System.out.print(root.data + " ");

preorder(root.left);

preorder(root.right);

}

// Recursive function to construct a height-balanced BST from a sorted doubly

// linked list. It takes a reference to the head TreeNode of the doubly linked

// list and the total number of nodes in it as an argument

public static TreeNode convertSortedDLLToBalancedBST(NodeWrapper head, int n)

{

// base case

if (n <= 0) {

return null;

}

// recursively construct the left subtree

TreeNode leftSubTree = convertSortedDLLToBalancedBST(head, n/2);

// `head` now points to the middle TreeNode of the sorted DDL

// make the middle TreeNode of the sorted DDL as the root TreeNode of the BST

TreeNode root = head.TreeNode;

// update left child of the root TreeNode

root.left = leftSubTree;

// update the head reference of the doubly linked list

head.TreeNode = head.TreeNode.right;

// recursively construct the right subtree with the remaining nodes

root.right = convertSortedDLLToBalancedBST(head, n - (n/2 + 1));

/\* +1 for the root TreeNode \*/

// return the root TreeNode

return root;

}

public static void main(String[] args)

{

// points to the head of a doubly linked list

TreeNode head = null;

// construct a doubly linked list from sorted keys

int[] keys = { 25, 20, 18, 15, 12, 10, 8 };

for (int key: keys) {

head = push(head, key);

}

// print the list and count the total number of nodes

System.out.print("Doubly Linked List: ");

int n = printAndCountNodes(head);

// construct a height-balanced BST from a sorted doubly linked list

// wrap the `head` TreeNode, so its reference can be changed

TreeNode root = convertSortedDLLToBalancedBST(new NodeWrapper(head), n);

System.out.print("Preorder traversal of the constructed BST: ");

preorder(root);

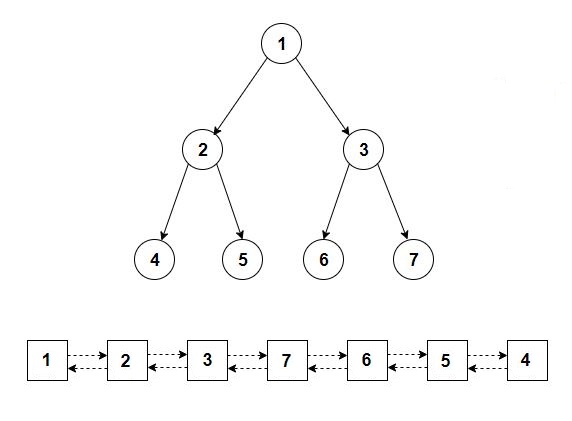
}

}

2. **Convert a binary tree into a doubly-linked list in spiral order**

Given a binary tree, [**in-place**](https://www.techiedelight.com/in-place-vs-out-of-place-algorithms/) convert it into a doubly-linked list following the spiral order.

The conversion should be done so that the left child pointer of a binary tree node should act as a previous pointer for a doubly-linked list node, and the right child pointer should act as the next pointer for a doubly-linked list node. The conversion should also be done by only exchanging the pointers without allocating any memory for the doubly linked list’s nodes.  
For example,



public static void printDoublyLinkedList(TreeNode TreeNode)

{

while (TreeNode != null)

{

System.out.print(TreeNode.data + " —> ");

TreeNode = TreeNode.right;

}

System.out.println("null");

}

// Insert a tree TreeNode at the front of the doubly linked list

public static TreeNode push(TreeNode TreeNode, TreeNode head)

{

// initialize head pointer of the doubly linked list

if (head == null)

{

head = TreeNode;

head.left = head.right = null;

return head;

}

// insert the given TreeNode at the front of the doubly linked list

head.left = TreeNode;

TreeNode.right = head;

// update left child pointer to be null

TreeNode.left = null;

// update head pointer to point to the given TreeNode

head = TreeNode;

return head;

}

// Traverse the tree in a preorder fashion and store nodes in a map

// corresponding to their level

public static void preorder(TreeNode root, int level, Map<Integer, Deque<TreeNode>> map)

{

// base case: empty tree

if (root == null) {

return;

}

// insert the current TreeNode and its level into the map

map.putIfAbsent(level, new ArrayDeque<>());

// if the level is odd, insert at front; otherwise, search at the back

if ((level & 1) == 1) {

map.get(level).addFirst(root);

}

else {

map.get(level).addLast(root);

}

// recur for the left and right subtree with a level increased by 1

preorder(root.left, level + 1, map);

preorder(root.right, level + 1, map);

}

// Recursive function to convert a binary tree into a doubly-linked list

// using hashing

public static void convert(TreeNode root)

{

// create an empty map to store nodes between given levels

Map<Integer, Deque<TreeNode>> map = new HashMap<>();

// traverse the tree and insert its nodes into the map

// corresponding to their level

preorder(root, 0, map);

// iterate through the map in decreasing order of level and

// push nodes of each level into the doubly linked list

int n = map.size();

TreeNode head = null;

for (int i = n - 1; i >=0; i--)

{

for (TreeNode TreeNode: map.get(i)) {

head = push(TreeNode, head);

}

}

}

public static void main(String[] args)

{

/\* Construct the following tree

1

/ \

/ \

2 3

/ \ / \

/ \ / \

4 5 6 7

\*/

TreeNode root = new TreeNode(1);

root.left = new TreeNode(2);

root.right = new TreeNode(3);

root.left.left = new TreeNode(4);

root.left.right = new TreeNode(5);

root.right.left = new TreeNode(6);

root.right.right = new TreeNode(7);

convert(root);

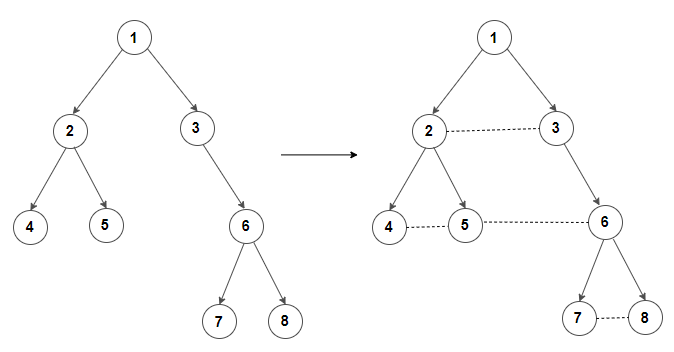
printDoublyLinkedList(root);

}

3. **Link nodes present in each level of a binary tree in the form of a linked list**

Given the root of a special binary tree with each node containing an additional next pointer, link nodes at the same level using the next pointer in the form of a linked list like structure.

For example, the binary tree on the left should be converted into a binary tree on the right.



class Node

{

int data;

Node left = null, right = null, next = null;

Node(int data) {

this.data = data;

}

}

public class AssignBstToEachLevelAsLinkToDLL {

public static void printList(Node head)

{

while (head != null)

{

System.out.print(head.data + " —> ");

head = head.next;

}

System.out.println("null");

}

// Function to perform inorder traversal on a given binary tree where nodes

// at the same level are linked together in the form of a linked list

public static void inorder(Node root)

{

if (root == null) {

return;

}

inorder(root.left);

// print current node and its next node

System.out.print(root.data + "—>");

if (root.next != null) {

System.out.println(root.next.data);

}

else {

System.out.println("null");

}

inorder(root.right);

}

// Recursive function to find the first node in the next level of a given root node

public static Node findNextNode(Node root)

{

// base case

if (root == null || root.next == null) {

return null;

}

// if the left child of the root's next node exists, return it

if (root.next.left != null) {

return root.next.left;

}

// if the right child of the root's next node exists, return it

if (root.next.right != null) {

return root.next.right;

}

// if root's next node is a leaf node, recur for root's next node

return findNextNode(root.next);

}

// Recursive function to link nodes present in each level of a binary tree

// in the form of a linked list

public static void linkNodes(Node root)

{

// base case

if (root == null) {

return;

}

// ensure that the nodes of the current level are linked before the

// next level nodes

linkNodes(root.next);

// Update the next pointer of root's left child to root's right child.

// If the right child doesn't exist, link it to the first node in the

// next level.

if (root.left != null) {

root.left.next = (root.right != null)? root.right: findNextNode(root);

}

// update the next pointer of the root's right child to the first node

// in the next level

if (root.right != null) {

root.right.next = findNextNode(root);

}

// recur for the left and right subtree

linkNodes(root.left);

linkNodes(root.right);

}

public static void main(String[] args)

{

/\* Construct the following tree

1

/ \

2 3

/ \ \

4 5 6

\ /

7 8

\*/

Node root = new Node(1);

root.left = new Node(2);

root.right = new Node(3);

root.left.left = new Node(4);

root.left.right = new Node(5);

root.right.right = new Node(6);

root.left.left.right = new Node(7);

root.right.right.left = new Node(8);

// link nodes at the same level

linkNodes(root);

// print the nodes

Node node = root;

while (node != null)

{

// print the current level

printList(node);

// find the leftmost node in the next level

if (node.left != null) {

node = node.left;

}

else if (node.right != null) {

node = node.right;

}

else {

node = findNextNode(node);

}

}

// inorder(root);

}

}

4. Path with Maximum Gold

You are given a grid representing a gold mine, and each cell in the grid has a certain amount of gold. You can start from any cell containing gold, move in 4 directions, and collect the gold in each cell you visit. Find the maximum amount of gold you can collect.

Example1:

Input:

{0, 6, 0},

{5, 8, 7},

{0, 9, 0}

Output:24

public int getMaximumGold(int[][] grid) {  
 int maxGold = 0;  
 int rows = grid.length;  
 int cols = grid[0].length;  
  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < cols; j++) {  
 if (grid[i][j] != 0) {  
 maxGold = Math.*max*(maxGold, dfs(grid, i, j));  
 }  
 }  
 }  
  
 return maxGold;  
}  
  
private int dfs(int[][] grid, int row, int col) {  
 if (row < 0 || row >= grid.length || col < 0 || col >= grid[0].length || grid[row][col] == 0) {  
 return 0;  
 }  
  
 int originalValue = grid[row][col];  
 grid[row][col] = 0; *// Mark the cell as visited* int maxGold = 0;  
 int[] rowOffsets = {-1, 1, 0, 0};  
 int[] colOffsets = {0, 0, -1, 1};  
  
 for (int i = 0; i < 4; i++) {  
 int newRow = row + rowOffsets[i];  
 int newCol = col + colOffsets[i];  
 maxGold = Math.*max*(maxGold, dfs(grid, newRow, newCol));  
 }  
  
 grid[row][col] = originalValue; *// Restore the cell's value* return maxGold + originalValue;  
}  
  
public static void main(String[] args) {  
 AssignGoldMaxPath solution = new AssignGoldMaxPath();  
 int[][] grid = {  
 {0, 6, 0},  
 {5, 8, 7},  
 {0, 9, 0}  
 };  
 System.*out*.println(solution.getMaximumGold(grid)); *// Output: 24* int[][] grid1 = {  
 {0, 6, 0},  
 {5, 8, 7},  
 {0, 9, 0}  
 };  
 System.*out*.println(solution.getMaximumGold(grid1)); *// Output: 24* int[][] grid2 = {  
 {1, 0, 7},  
 {2, 0, 6},  
 {3, 4, 5},  
 {0, 3, 0},  
 {9, 0, 20}  
 };  
 System.*out*.println(solution.getMaximumGold(grid2)); *// Output: 28* int[][] grid3 = {  
 {0, 0, 0},  
 {0, 0, 0},  
 {0, 0, 0}  
 };  
 System.*out*.println(solution.getMaximumGold(grid3)); *// Output: 0* int[][] grid4 = {  
 {1, 2, 3},  
 {4, 5, 6},  
 {7, 8, 9}  
 };  
 System.*out*.println(solution.getMaximumGold(grid4)); *// Output: 45*}

5. Alien Dictionary

Given a sorted dictionary of an alien language, find the order of the characters. The dictionary is given in the form of a list of words where the order of characters is lexicographically increasing.

Input1:

words1 = {"wrt", "wrf", "er", "ett", "rftt"};

Output:

wertf

zx

public String alienOrder(String[] words) {  
 Map<Character, Set<Character>> graph = new HashMap<>();  
 for (char c : words[0].toCharArray()) {  
 graph.putIfAbsent(c, new HashSet<>());  
 }  
  
 *// Build the graph* for (int i = 1; i < words.length; i++) {  
 String prev = words[i - 1];  
 String curr = words[i];  
 int minLength = Math.*min*(prev.length(), curr.length());  
  
 for (int j = 0; j < minLength; j++) {  
 char prevChar = prev.charAt(j);  
 char currChar = curr.charAt(j);  
  
 if (prevChar != currChar) {  
 graph.putIfAbsent(prevChar, new HashSet<>());  
 graph.get(prevChar).add(currChar);  
 break;  
 }  
 }  
 }  
  
 *// Perform topological sorting using DFS* Stack<Character> stack = new Stack<>();  
 Set<Character> visited = new HashSet<>();  
 for (char c : graph.keySet()) {  
 if (!visited.contains(c) && !dfs(c, graph, stack, visited, new HashSet<>())) {  
 return "";  
 }  
 }  
  
 StringBuilder result = new StringBuilder();  
 while (!stack.isEmpty()) {  
 result.append(stack.pop());  
 }  
  
 return result.toString();  
}  
  
private boolean dfs(char curr, Map<Character, Set<Character>> graph,  
 Stack<Character> stack, Set<Character> visited, Set<Character> visiting) {  
 if (visiting.contains(curr)) {  
 return false; *// Detected a cycle* }  
  
 visiting.add(curr);  
  
 for (char neighbor : graph.getOrDefault(curr, new HashSet<>())) {  
 if (!visited.contains(neighbor)) {  
 if (!dfs(neighbor, graph, stack, visited, visiting)) {  
 return false;  
 }  
 }  
 }  
  
 visiting.remove(curr);  
 visited.add(curr);  
 stack.push(curr);  
 return true;  
}  
  
public static void main(String[] args) {  
 AssignAlientDictionary solution = new AssignAlientDictionary();  
  
 String[] words1 = {"wrt", "wrf", "er", "ett", "rftt"};  
 System.*out*.println(solution.alienOrder(words1)); *// Output: "wertf"* String[] words2 = {"z", "x"};  
 System.*out*.println(solution.alienOrder(words2)); *// Output: "zx"* String[] words3 = {"z", "x", "z"};  
 System.*out*.println(solution.alienOrder(words3)); *// Output: ""*}