beta Firecode.io™

TangoZulu's Solutions Book

1 2 3 4 5

Looping Lists : Space complexity O(n)

Linked Lists Hash-Tables

Given a singly-linked list, implement a method to check if the list has cycles. The space complexity can be **O(n)**. If there is a cycle, return `true` otherwise return `false`. Empty lists should be considered non-cyclic.

Examples:

`1->2->3->4->1 ==> true`

`1->2->3->4 ==> false`

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public Boolean isCyclic(ListNode head) {

    HashSet<ListNode> seen = new HashSet<ListNode>();

    while (head != null)
    {
        if (seen.contains(head))
        {
            return true;
        }
        seen.add(head);
        head = head.next;
    }
    return false;

}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.
```

```
public Boolean isCyclic(ListNode head) {
    // Add your code below this line. Do not modify any other code.
    if (head == null)
        return false;

    ListNode slow = head;
    ListNode fast = head;

    while (fast.next != null && fast.next.next != null) {

        fast = fast.next.next;

        if (slow == fast)
            return true;

        slow = slow.next;
    }

    return false;

    // Add your code above this line. Do not modify any other code.
}
```

Comments

AndrÃ© Pinto - 16 Jul, 2016

For anyone scratching their head trying to figure out how this works: Floyd's cycle-finding algorithm. It's nicely explained here: http://stackoverflow.com/a/6110767/43046

JoÃ£o Lopes - 10 Aug, 2016

This is very smart solution but is actually not following the problem constrains that says "space complexity O(n)". This is better than O(n) space but at the cost of time complexity that I suspect is O(n log(n)).

AndrÃ© Pinto - 10 Aug, 2016

@JoÃ£o. It's O(1) space complexity and O(n) time complexity. The time complexity of this depends on the size of the list and of its loop. See: https://en.wikipedia.org/wiki/Cycle_detection#Tortoise_and_hare

0

1 2 3 4 5

Find a Node in a Binary Tree Without Using Recursion

Trees Queues

Given a binary tree, write a method to find and return the node with data = the input data. Do not use recursion.

```
Example:
       1
      / \
     2   3
    / \ / \
   4  5 6  7

findNode(root, 5) ==> 5
```

Note: Return `null`, if desired node is not found.

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public TreeNode findNode(TreeNode root, int val) {


    if (root == null) return null;
    Queue<TreeNode> q = new LinkedList<TreeNode>();
    q.offer(root);
    while (!q.isEmpty())
    {
        TreeNode n = q.poll();
        if (n.data == val)
        {
            return n;
        }
        if (n.left != null) q.offer(n.left);
        if (n.right != null) q.offer(n.right);
    }

    return null;

}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public TreeNode findNode(TreeNode root, int val) {
    // Add your code below this line. Do not modify any other code.
    Stack<TreeNode> stack = new Stack<TreeNode>();

    while(!stack.isEmpty() || root!=null) {
        if(root!=null) {
            stack.push(root);
            root = root.left;
        } else {
            TreeNode node = stack.pop();
            if(node.data == val) return node;
            root = node.right;
        }
    }
    return null;
    // Add your code above this line. Do not modify any other code.
}
```

1 2 3 4 5

Implement the Djikstra Shortest Path Algorithm in a Graph

Graphs Trees Search Algorithms

Implement a method to compute the shortest path from `source` to `target` in a graph using Djikstra Algorithm. The method should return a `List` of `Vertices` denoting the optimal path. Click "Use Me" to understand the `Vertex` and `Edge` classes.

```
Example:

     V2
    |   \
    |10  \3
    |   5 \    7
    V0 â€”â€”-- V1 â€”â€”â€” V4
      \          /
       \8       /2
        \      /
           V3

 v0 = Rville
 v1 = Bville
 v2 = Gville
 v3 = Oville
 v4 = Pville

Shortest Path to V3 from V0 = [Rville, Oville]
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static List<Vertex> getShortestPath(Vertex source, Vertex target) {
    computePaths(source);
    return getShortestPathTo(target);
}

public static List<Vertex> getShortestPathTo(Vertex target) {
```

```
    List<Vertex> path = new ArrayList<Vertex>();
    for (Vertex vertex = target; vertex != null; vertex = vertex.previous)
        path.add(vertex);
    Collections.reverse(path);
    return path;
}

// Helper function to compute shortest path and store in each vertex
public static void computePaths(Vertex source) {
    source.minDistance = 0.;
    PriorityQueue<Vertex> vertexQueue = new PriorityQueue<Vertex>();
    vertexQueue.add(source);

    while (!vertexQueue.isEmpty()) {
        Vertex u = vertexQueue.poll();
        // Visit each edge exiting u
        for (Edge e : u.adjacencies) {
            Vertex v = e.target;
            double weight = e.weight;
            double distanceThroughU = u.minDistance + weight;
            if (distanceThroughU < v.minDistance) {
                vertexQueue.remove(v);
                v.minDistance = distanceThroughU ;
                v.previous = u;
                vertexQueue.add(v);
            }
        }
    }
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static List<Vertex> getShortestPath(Vertex source, Vertex target) {

    if(source == null || target == null) return null;

    final List<Vertex> listOfVerticesFromSToT = new ArrayList<Vertex>();
    final PriorityQueue<Vertex> queue = new PriorityQueue<Vertex>();
    final Stack<Vertex> stack = new Stack<Vertex>();

    source.minDistance = 0;
    queue.add(source);
    while(!queue.isEmpty())
    {
        Vertex temp = queue.remove();
        final Edge[] edges = temp.adjacencies;
        final double minDist = temp.minDistance;
```

```
        for(int i = 0; i< edges.length; i++)
        {
            final Edge edge = edges[i];
            final Vertex tempTarget = edge.target;
            final double dist = edge.weight + minDist;
            if(dist < tempTarget.minDistance)
            {
                queue.remove(tempTarget);
                tempTarget.minDistance = dist;
                tempTarget.previous = temp;
                queue.add(tempTarget);
            }
        }
    }

    Vertex minPrevious = target.previous;
    stack.push(target);
    while(minPrevious != null)
    {
        stack.push(minPrevious);
        minPrevious = minPrevious.previous;
    }

    while(!stack.isEmpty())
    {
        listOfVerticesFromSToT.add(stack.pop());
    }

    return listOfVerticesFromSToT;
}
```

1 2 3 4 5

Largest Square

Multi Dimensional Arrays | Dynamic Programming

Given a two dimensional matrix made up of **0**'s and **1**'s, find the largest square containing all **1**'s and return its 'area'. The 'area' is simply the sum of all integers enclosed in the square.

Example:

```
Input Matrix :

  1101
  1101
  1111

Output  : 4
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int largestSquare(char[][] matrix) {

    int rows = matrix.length;
    int cols = matrix[0].length;

    int[][] memo = new int[rows][cols];
    for (int i=0; i<rows; ++i)
    {
        memo[i][0] = Character.getNumericValue(matrix[i][0]);
    }

    for (int j=0; j<cols; ++j)
    {
        memo[0][j] = Character.getNumericValue(matrix[0][j]);
    }

    for (int i=1; i<rows; ++i)
    {
        for (int j=1; j<cols; ++j)
        {
            if (matrix[i][j] == '1')
            {
                int min = Math.min(memo[i-1][j], memo[i-1][j-1]);
```

```
                min = Math.min(min, memo[i][j-1]);
                memo[i][j] = min + 1;
            }
            else
            {
                memo[i][j] = 0;
            }
        }
    }

    int max = 0;
    for (int i=0; i<rows; ++i)
    {
        for (int j=0; j<cols; ++j)
        {
            max = Math.max(max, memo[i][j]);
        }
    }
    return max * max;
}
```

Top voted solution

```
public static int largestSquare(char[][] matrix) {
    // Brute force solution
    int maximumSquareSize = Math.min(matrix.length, matrix[0].length);
    for (int squareSize = maximumSquareSize; squareSize >= 1; squareSize--) {
        for (int cornerX = 0; cornerX <= matrix.length - squareSize; cornerX++) {
            for (int cornerY = 0; cornerY <= matrix[0].length - squareSize; cornerY++) {
                if (allOnes(matrix, cornerX, cornerY, squareSize))
                    return squareSize * squareSize;
            }
        }
    }
    return 0;
}

public static boolean allOnes(char[][] matrix, int cornerX, int cornerY, int squareSize) {
    for (int x = cornerX; x < cornerX + squareSize; x++) {
        for (int y = cornerY; y < cornerY + squareSize; y++) {
            if (matrix[x][y] == '0')
                return false;
        }
    }
    return true;
}
```

Comments

Fooble - 22 Jan, 2016

I wasted so much time debugging this, all because the input is a char matrix, instead of an int matrix like I was expecting.

Jason Banich - 15 Sep, 2016

Same. I guess its one of those "read the problem statement" things, though in an interview the person would correct you instantly, and if you were running the code manually, debugging would help you notice this almost instantly.

1 2 3 4 5

Boggle with Paper Dictionary

Multi Dimensional Arrays | DFS | Search Algorithms | Recursion | Prefix Tree

You're given a 2D **Boggle Board** which contains an m x n matrix of chars - `char[][] board`, and a rudimentary, paper Dictionary in the form of an `ArrayList` of close to 10,000 words. Write a method - `boggleByot` that searches the Boggle Board for words in the dictionary. Your method should return an **alphabetically sorted** `ArrayList` of words that are present on the board as well as in the dictionary. Words on the board can be constructed with **sequentially adjacent** letters, where adjacent letters are horizontal or vertical neighbors (not diagonal). Also, each letter on the Boggle Board must be used only once.

Note:
Your program should run in a reasonable amount of time - about a few milliseconds for each test case.

Example:

```
Input Board :
{
    {A, O, L},
    {D, E, L},
    {G, H, I},
}
Dictionary : [HELLO, HOW, ARE, YOU] (as a Trie)
Output: [HELLO]
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

class TrieNode {
    Character c;

    Boolean isLeaf = false;

    HashMap<Character, TrieNode> children = new HashMap<>();

    public TrieNode(){
    }

    public TrieNode(Character c)
    {
        this.c = c;
    }
}
```

```
class Trie {
    private TrieNode root;

    public Trie()
    {
        this.root = new TrieNode();
    }

    public void insertWord(String word)
    {
        TrieNode current = this.root;
        HashMap<Character, TrieNode> children = current.children;

        for (int i=0; i<word.length(); ++i)
        {
            char c = word.charAt(i);
            if (!children.containsKey(c))
            {
                children.put(c, new TrieNode(c));
            }
            current = children.get(c);
            children = current.children;

            if (i == word.length()-1)
            {
                current.isLeaf = true;
            }
        }
    }

    public Boolean searchWord(String word)
    {
        TrieNode current = this.root;
        HashMap<Character, TrieNode> children = current.children;

        for (int i=0; i<word.length(); ++i)
        {
            char c = word.charAt(i);
            if (!children.containsKey(c))
            {
                return false;
            }
            current = children.get(c);
            children = current.children;
        }

        return current.isLeaf;
    }

    public Boolean searchPrefix(String word)
    {
        TrieNode current = this.root;
        HashMap<Character, TrieNode> children = current.children;
```

```
        for (int i=0; i<word.length(); ++i)
        {
            char c = word.charAt(i);
            if (!children.containsKey(c))
            {
                return false;
            }
            current = children.get(c);
            children = current.children;
        }

        return true;
    }
}


public ArrayList<String> boggleByot(char[][] board, ArrayList<String> dictionary){

    TreeSet<String> foundWords = new TreeSet<String>();
    Trie dict = new Trie();
    for (String s : dictionary)
    {
        dict.insertWord(s);
    }

    int rows = board.length;
    int cols = board[0].length;

    for (int row = 0; row < rows; ++row)
    {
        for (int col = 0; col < cols; ++col)
        {
            mineForWords(board, row, col, "", dict, foundWords);
        }
    }

    return new ArrayList<String>(foundWords);
}

private void mineForWords(char[][] board, int row, int col, String current, Trie dict, TreeSet<String
> foundWords)
{
    if (row < 0 ||
        row >= board.length ||
        col < 0 ||
        col >= board[0].length ||
        !dict.searchPrefix(current) ||
        board[row][col] == '*')
    {
        return;
    }

    char c = board[row][col];
```

```
    board[row][col] = '*';

    current += c;
    if (dict.searchWord(current))
    {
        foundWords.add(current);
    }

    mineForWords(board, row+1, col, current, dict, foundWords);
    mineForWords(board, row-1, col, current, dict, foundWords);
    mineForWords(board, row, col+1, current, dict, foundWords);
    mineForWords(board, row, col-1, current, dict, foundWords);

    board[row][col] = c;

}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ArrayList<String> boggleByot(char[][] board, ArrayList<String> dictionary){
    Trie trie = new Trie();
    for (String word : dictionary) {
        trie.insertWord(word);
    }

    Set<String> wordList = new TreeSet<>();
    int xLength = board.length;
    int yLength = board[0].length;


    for (int x = 0; x < xLength; x++) {
        for (int y = 0; y < yLength; y++) {
            boggleByot(board, x, y, trie, "", wordList);
        }
    }

    return new ArrayList<>(wordList);
}

private void boggleByot(char[][] board, int x, int y, Trie dictionary,
                        String prefix, Set<String> wordList) {
    if (!validIndices(board, x, y) || board[x][y] == '*') {
        return;
    }

    if (dictionary.searchWord(prefix)) {
        wordList.add(prefix);
```

```
    }

    if (dictionary.searchPrefix(prefix)) {
        char c = board[x][y];
        prefix += c;
        board[x][y] = '*';

        boggleByot(board, x + 1, y, dictionary, prefix, wordList);
        boggleByot(board, x - 1, y, dictionary, prefix, wordList);
        boggleByot(board, x, y + 1, dictionary, prefix, wordList);
        boggleByot(board, x, y - 1, dictionary, prefix, wordList);

        board[x][y] = c;
    }

}

private boolean validIndices(char[][] board, int x, int y) {
    if (x < 0 || x >= board.length
            || y < 0 || y >= board[0].length) {
        return false;
    }
    return true;
}

class TrieNode {
    Character c;
    Boolean isLeaf = false;
    HashMap<Character, TrieNode> children = new HashMap<>();

    public TrieNode() {
    }

    public TrieNode(Character c) {
        this.c = c;
    }
}

class Trie {
    private TrieNode root;

    public Trie() {
        root = new TrieNode();
    }

    public void insertWord(String word) {
        TrieNode node = root;

        for (char c : word.toCharArray()) {
            if (!node.children.containsKey(c)) {
                TrieNode newNode = new TrieNode(c);
                node.children.put(c, newNode);
                node = newNode;
            } else {
```

```
                node = node.children.get(c);
            }
        }
        node.isLeaf = true;
    }

    public Boolean searchWord(String word) {
        TrieNode node = root;

        for (char c : word.toCharArray()) {
            node = node.children.get(c);
            if (node == null) {
                return false;
            }
        }
        return node.isLeaf;
    }

    public Boolean searchPrefix(String word) {
        TrieNode node = root;

        for (char c : word.toCharArray()) {
            node = node.children.get(c);
            if (node == null) {
                return false;
            }
        }
        return true;
    }
}
```

1 2 3 4 5

Subset Summation with Number Constraint

Recursion Arrays Numbers

Given an array of integers and a target integer, write a method `groupSumWithNum` to determine if it is possible to choose a group of integers from the array such that the group sums to the given target. An additional constraint is that the summation must include the integer `must_have` if it is present in the array.

Examples:

`groupSumWithNum({1,2,3,6,5},5,10) ==> true`

`groupSumWithNum({1,2,3,6,5},3,7) ==> false`

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static boolean groupSumWithNum(int[] arr, int must_have, int target) {

    return groupSum(arr, 0, must_have, target);

}

private static boolean groupSum(int[] arr, int start_index, int must_have, int target)
{
    if (start_index >= arr.length)
    {
        return (target == 0);
    }

    if (arr[start_index] == must_have)
    {
        return groupSum(arr, start_index + 1, must_have, target - must_have);
    }
    else
    {
        if (groupSum(arr, start_index + 1, must_have, target - arr[start_index]))
        {
            return true;
        }
        if (groupSum(arr, start_index + 1, must_have, target))
        {
            return true;
```

```
        }
        return false;
    }
}
```

Top voted solution

```
public static boolean groupSumWithNumHelper(int[] arr, int from, int target) {
    if (target == 0)
        return true;

    if (from >= arr.length)
        return false;

    return groupSumWithNumHelper(arr, from + 1, target) ||
           groupSumWithNumHelper(arr, from + 1, target - arr[from]);
}

// java.util.* has been imported for this problem.
// You don't need any other imports.

public static boolean groupSumWithNum(int[] arr, int must_have, int target) {
    int from = 0;

    for (int i = 0; i < arr.length; i++) {
        if (arr[i] == must_have) {
            int tmp = arr[0];
            arr[0] = arr[i];
            arr[i] = tmp;
            from = 1;
            target = target - must_have;
            break;
        }
    }

    return groupSumWithNumHelper(arr, from, target);
}
```

1 2 3 4 5

Pascal's Triangle

Arrays Numbers Puzzles Dynamic Programming

Given an input parameter `numRows`, generate the first `numRows` number of rows of **Pascal's** triangle. As a quick refresher - in a Pascal's triangle, each number is equal to the sum of the two directly above it.

Example:

```
Input  : 4
Output :
        [
             [1],
            [1,1],
           [1,2,1],
          [1,3,3,1]

        ]
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static ArrayList<ArrayList<Integer>> generatePascalTriangle(int numRows) {

    ArrayList<ArrayList<Integer>> triangle = new ArrayList<ArrayList<Integer>>();
    if (numRows == 0)
    {
        return triangle;
    }

    ArrayList<Integer> level = new ArrayList<Integer>();
    level.add(1);
    triangle.add(level);

    ArrayList<Integer> previous = null;

    for (int i=2; i<= numRows; ++i)
    {
        previous = level;
        level = new ArrayList<Integer>();
        level.add(1); // first
```

```
        for (int j=0; j < previous.size() - 1; ++j)
        {
            level.add(previous.get(j) + previous.get(j+1));
        }
        level.add(1); // last;
        triangle.add(level);
    }

    return triangle;


}
```

Top voted solution

```
public static ArrayList<ArrayList<Integer>> generatePascalTriangle(int numRows) {
    ArrayList<ArrayList<Integer>> answer = new ArrayList<ArrayList<Integer>>();
    if (numRows < 1)
        return answer;
    // The first row is [1].
    ArrayList<Integer> prevRow = new ArrayList<Integer>();
    prevRow.add(1);
    answer.add(prevRow);
    // Make the remaining rows.
    for (int row = 1; row < numRows; row++) {
        ArrayList<Integer> curRow = new ArrayList<Integer>();
        curRow.add(1);
        for (int col = 1; col < prevRow.size(); col++)
            curRow.add(prevRow.get(col - 1) + prevRow.get(col));
        curRow.add(1);
        answer.add(curRow);
        prevRow = curRow;
    }
    return answer;
}
```

1 2 3 4 5

Find the Maximum Contiguous Subsequence in an Array

Recursion Arrays Numbers

Given an array of integers consisting of both positive and negative numbers, find the contiguous subsequence that has the **maximum sum** among all subsequences in the array (click the red text to learn more about subsequences). Write a method that takes in an array of integers `arr` and returns an array `res` containing 3 integers in the following format:

```
res[0] = max sum
res[1] = starting index of the subsequence
res[2] = ending index of the subsequence
```

Examples:

`maxContSequence({-1,-2,3,4,5}) ==> {12,2,4}`

`maxContSequence({1,2,3,-2,5}) ==> {6,0,2}`

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int[] maxContSequence(int[] arr) {

    int[] res = new int[3];

    if (arr == null || arr.length == 0)
    {
        // special case for empty array.
        res[0] = 0;
        res[1] = 0;
        res[2] = -1;
        return res;
    }

    int maxSum = arr[0];
    int maxSoFar = 0;
    int maxSoFarStart = 0;
    int maxSoFarEnd = 0;
```

```
    int maxStart = 0;
    int maxEnd = 0;

    // motified Kadane's algorithm
    for(int i=0; i<arr.length; ++i)
    {
        int sum = maxSoFar + arr[i];
        if (arr[i] > sum)
        {
            maxSoFar = arr[i];
            maxSoFarStart = i;
            maxSoFarEnd = i;
        }
        else
        {
            maxSoFar = sum;
            maxSoFarEnd = i;
        }

        if (maxSoFar > maxSum)
        {
            maxSum = maxSoFar;
            maxStart = maxSoFarStart;
            maxEnd = maxSoFarEnd;
        }
    }

    res[0] = maxSum;
    res[1] = maxStart;
    res[2] = maxEnd;
    return res;
}
```

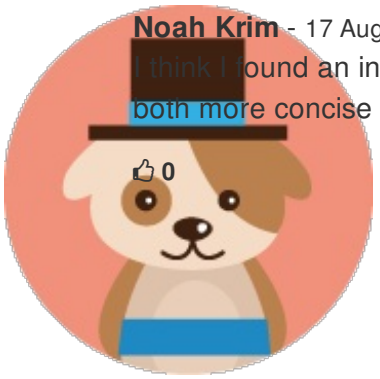
Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int[] maxContSequence(int[] arr) {
    // Edge case (empty list)
    if(arr.length == 0)
        return new int[] { 0, 0, -1 };

    // Initialzie groups
    ArrayList<RangeSum> groups = new ArrayList<>();
    int sum = arr[0];
    int start = 0;
    for(int i=1; i<arr.length; i++) {
        int x = arr[i];
        if(Math.signum(sum) != Math.signum(x)) {
```

```
            groups.add(new RangeSum(sum, start, i-1));
            sum = 0;
            start = i;
        }
        sum += x;
    }
    groups.add(new RangeSum(sum, start, arr.length-1));

    // Edge case (only negatives)
    if(groups.size() == 1 && groups.get(0).sum < 0) {
        int max = arr[0];
        int maxIndex = 0;
        for(int i=1; i<arr.length; i++) {
            if(arr[i] > max) {
                max = arr[i];
                maxIndex = i;
            }
        }
        return new int[] { max, maxIndex, maxIndex };
    }

    // Trim front and ends of negative groups
    if(groups.get(0).sum < 0)
        groups.remove(0);
    if(groups.get(groups.size()-1).sum < 0)
        groups.remove(groups.size()-1);

    // Brute force for max grouping by merging groups until best merge is left
    return mergeForMax(groups).toArray();
}

private static RangeSum mergeForMax(ArrayList<RangeSum> groups) {
    if(groups.size() == 0)
        return null;
    if(groups.size() == 1)
        return groups.get(0);

    RangeSum pos = groups.remove(0);
    RangeSum neg = groups.remove(0);
    int mergedSum = pos.sum + neg.sum;
    if(mergedSum > 0) {
        RangeSum merge = groups.get(0);
        merge.sum += mergedSum;
        merge.start = pos.start;
        return mergeForMax(groups);
    }
    RangeSum ret = mergeForMax(groups);
    return pos.sum >= ret.sum ? pos : ret;
}

private static class RangeSum {
    int sum;
    int start;
    int end;
```

```
    public RangeSum(int sum, int start, int end) {
        this.sum = sum;
        this.start = start;
        this.end = end;
    }

    public int[] toArray() {
        return new int[] { this.sum, this.start, this.end };
    }
}
```

Comments



Noah Krim - 17 Aug, 2016

I think I found an interesting (yet verbose) solution to the problem, but I see that most other solutions are both more concise and more efficient, so I was wondering if anyone saw any merit in my solution?

0

1 2 3 4 5

Implement the Breadth First Search Algorithm for a Graph

Graphs Trees Search Algorithms

Implement a method to find a node in a graph using Breadth First Search. Click 'Use me!' to inspect the `Node` class and its methods.

Example:

```
          apple
         /     \
     banana    mango
    /      \     /      Find
 peach    strawberry
    \      /
     cherry

cherry ==> True
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public boolean breadthFirstSearch(Node rootNode, String data){

    if (rootNode == null) return false;
    Queue<Node> q = new LinkedList<Node>();

    q.offer(rootNode);
    while (!q.isEmpty())
    {
        Node n = q.poll();
        n.visited = true;

        if (n.data.equals(data))
        {
            return true;
        }
```

```
        for (Node child : n.adjacentNodes)
        {
            if (!child.visited)
            {
                q.offer(child);
            }
        }
    }
    return false;

}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public boolean breadthFirstSearch(Node rootNode, String data){
    // Add your code below this line. Do not modify any other code.
    if(rootNode == null || data == null || data.length() ==0) return false;
    final Queue<Node> queue = new LinkedList<Node>();
    queue.add(rootNode);

    while(!queue.isEmpty())
    {
        Node temp = queue.remove();
        if(!temp.visited && data.equals(temp.data)) return true;
        if(!temp.visited && !temp.adjacentNodes.isEmpty()) queue.addAll(temp.adjacentNodes);
        temp.visited = true;
    }
    return false;
    // Add your code above this line. Do not modify any other code.
}
```

1 2 3 4 5

Even Split

Recursion Arrays Numbers

Given an array of integers, determine if it is possible to split
the array into two parts such that the sum of all elements in each part is the same.

Examples:

```
splitArray({1,2,3,4}) ==> true
splitArray({1,2,4}) ==> false
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static boolean splitArray(int[] arr) {

    if (arr == null || arr.length == 0) return false;
    int sum = 0;
    for (int i=0; i<arr.length; ++i)
    {
        sum += arr[i];
    }

    if ((sum % 2) != 0)
    {
        // sum is odd.  Cannot split
        return false;
    }
    // see if combination of array values can sum to sum/2
    return CanSumToValue(arr, 0, sum / 2);
}

private static boolean CanSumToValue(int[] arr, int start_index, int target)
{
    if (start_index >= arr.length)
    {
        return (target == 0);
    }

    // Need to consider using current value AND not using the current value.
    // O(2^n) though...look for a more efficient way
    if (CanSumToValue(arr, start_index + 1, target - arr[start_index]))
    {
```

```
        return true;
    }
    if (CanSumToValue(arr, start_index + 1, target))
    {
        return true;
    }
    return false;
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static boolean splitArray(int[] arr) {
    //Add your code below this line. Do not modify any other code.
    return splitArray(arr, 0,0,0);
    //Add your code above this line. Do not modify any other code.
}

private static boolean splitArray(int arr[], int leftVal, int rightVal, int position) {
    if(position<arr.length) {
        return splitArray(arr, leftVal+arr[position], rightVal, position+1) ||
            splitArray(arr, leftVal, rightVal+arr[position], position+1);
    } else if(position==arr.length && leftVal==rightVal) {
        return true;
    } else  {
        return false;
    }
}
```

1 2 3 4 5

Minimum Sum Path

Multi Dimensional Arrays | Dynamic Programming

Given an **m x n** matrix filled with **non-negative** integers, find the minimum sum along a path from the top-left of the grid to the bottom-right which minimizes the sum of all numbers along it. Return this minimum sum. The direction of movement is limited to right and down.

Example:

```
Input Matrix :

    1 2 3
    4 5 6
    7 8 9

Output  : 21
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int minWeightedPath(int[][] grid) {

    int rows = grid.length;
    int cols = grid[0].length;
    int[][] memo = new int[rows][cols];

    memo[0][0] = grid[0][0];

    for (int row = 1; row < rows; ++row)
    {
        memo[row][0] = memo[row-1][0] + grid[row][0];
    }

    for (int col = 1; col < cols; ++col)
    {
        memo[0][col] = memo[0][col-1] + grid[0][col];
    }

    for (int row = 1; row < rows; ++row)
    {
        for (int col = 1; col < cols; ++col)
        {
```

```
            int min = Math.min(memo[row-1][col], memo[row][col-1]);
            memo[row][col] = grid[row][col] + min;
        }
    }

    return memo[rows-1][cols-1];



}
```

Top voted solution

```
public static int minWeightedPath(int[][] grid) {
    // Top
    for (int x = 1; x < grid.length; x++)
        grid[x][0] = grid[x - 1][0] + grid[x][0];
    // Left
    for (int y = 1; y < grid[0].length; y++)
        grid[0][y] = grid[0][y - 1] + grid[0][y];
    // Remainder
    for(int x = 1; x < grid.length; x++) {
        for(int y = 1; y < grid[0].length; y++) {
            int left = grid[x - 1][y];
            int above = grid[x][y - 1];
            grid[x][y] = grid[x][y] + Math.min(above, left);
        }
    }
    return grid[grid.length - 1][grid[0].length - 1];
}
```

1 2 3 4 5

Iterative Preorder Traversal

Trees Stacks

Given a binary tree, write a method to iteratively traverse the tree in the preorder manner. Mark a node as visited by adding its `data` to a list - `Arraylist <Integer> preorderedList`. Return this list.

```
Example:
     1
    / \
   2   3      ==> 1245367
  / \ / \
 4  5 6  7
```

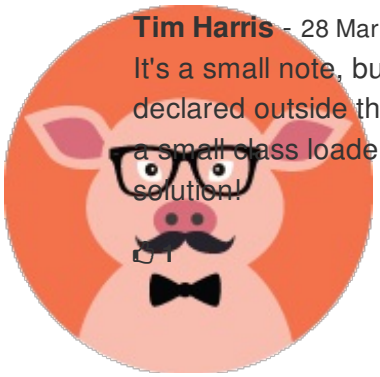
Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ArrayList<Integer> preorderItr(TreeNode root) {

    Stack<TreeNode> s = new Stack<TreeNode>();
    ArrayList<Integer> out = new ArrayList<Integer>();
    if (root == null)
    {
        return out;
    }

    s.push(root);
    while (!s.isEmpty())
    {
        TreeNode n = s.peek();
        s.pop();
        out.add(n.data);

        if (n.right != null) s.push(n.right);
        if (n.left != null) s.push(n.left);
    }

    return out;



}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ArrayList<Integer> preorderItr(TreeNode root) {
    // Add your code below this line. Do not modify any other code.

    Stack<TreeNode> stack = new Stack<TreeNode>();

    ArrayList<Integer> returnList = new ArrayList<Integer>();
    if (root == null) return returnList;

    stack.push(root);

    while (!stack.isEmpty()) {
        TreeNode current = stack.pop();
        returnList.add(current.data);
        if (current.right != null) {
            stack.push(current.right);
        }
        if (current.left != null) {
            stack.push(current.left);
        }
    }
    return returnList;

    // Add your code above this line. Do not modify any other code.
}
```

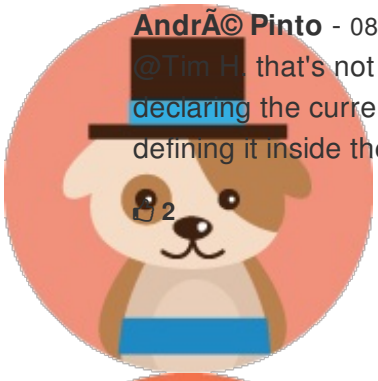
Comments



Tim Harris - 28 Mar, 2016

It's a small note, but for maximum performance your current object (TreeNode current) should be declared outside the while loop and should be set inside the while loop. The reason for this is that there is a small class loader overhead if the Type is declared every time. It's a JVM quirk essentially. But I like your solution!

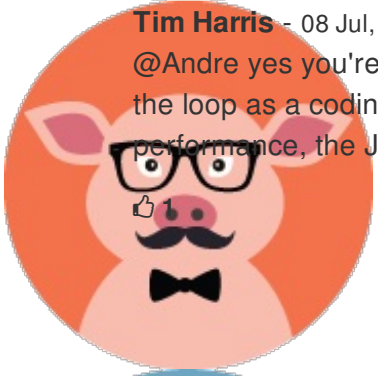
1



AndrÃ© Pinto - 08 Jul, 2016

@Tim H. that's not true. From the bytecode perspective there's absolutely no difference between declaring the current variable outside the while loop and assigning it inside the loop, and declaring and defining it inside the loop like Lupe did. The Java compiler is much smarter than that.

2



Tim Harris - 08 Jul, 2016

@Andre yes you're right. After scoping Stack Overflow I realized that it's probably better to declare it within the loop as a coding practice as well as that limits the scope of the variable to its usage block. As for performance, the JVM does produce identical bytecode.

1

Shapan Dashore - 12 Jul, 2016

@Andre can you help where i was wrong, because my code was very similar to Lupe's but i had an error for infinite loop

0

1 2 3 4 5

Graph Depth First Search

Graphs Trees Search Algorithms

Implement a method to find a node in a graph using Depth First Search.

```
Example:
          apple
         /    \
     banana   mango
    /     \    /
 peach   strawberry
    \     /
     cherry

Find cherry ==> true
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public boolean depthFirstSearch(Node rootNode, String data){

    if (rootNode == null)
    {
        return false;
    }

    rootNode.visited = true;

    if (rootNode.data == data)
    {
        return true;
    }

    for (Node n : rootNode.adjacentNodes)
    {
        if (!n.visited)
        {
            return depthFirstSearch(n, data);
        }
```

```
    }

    return false;

}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public boolean depthFirstSearch(Node rootNode, String data){
    // Add your code below this line. Do not modify any other code.

    if(rootNode == null || data == null) return false;

    final Stack<Node> stack = new Stack<Node>();
    stack.push(rootNode);

    while(!stack.isEmpty())
    {
        Node temp = stack.pop();
        if(!temp.visited)
        {
            temp.visited = true;
            if(data.equals(temp.data)) return true;
            for(Node aj: temp.adjacentNodes)
            {
                stack.push(aj);
            }
        }
    }
    return false;
    // Add your code above this line. Do not modify any other code.
}
```

1 2 3 4 5

Longest Palindromic Substring

Multi Dimensional Arrays | Dynamic Programming

Given a `String`, write a method - `longestPalSubstr` that finds and returns the longest substring which is also a Palindrome. Try and accomplish this in at most **O(n^2)** runtime.

Examples :

```
"bccb" => "bccb"
"bccd" => "cc"
"bccc" => "ccc"
```

Your solution

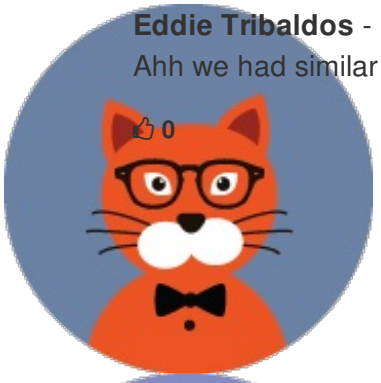
```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public String longestPalSubstr(String str){

    int start = 0, end = 0;
    for (int i = 0; i < str.length(); i++)
    {
        int len1 = expandAroundCenter(str, i, i);
        int len2 = expandAroundCenter(str, i, i + 1);
        int len = Math.max(len1, len2);
        if (len > end - start)
        {
            start = i - (len - 1) / 2;
            end = i + len / 2;
        }
    }

    if (end-start == 0)
    {
        // special case when the palindrome is a single character. Return first.
        // I disagree entirely with the online judge that the acceptable answer is the
        // first character, when in reality, any character in the string could be correct.  My soluti
on
        // minus this special exception returns the last character.
        return str.substring(0, 1);
    }
    return str.substring(start, end+1);
}


private static int expandAroundCenter(String s, int left, int right)
```

```
{
    int L = left, R = right;
    while (L >= 0 && R < s.length() && s.charAt(L) == s.charAt(R))
    {
        L--;
        R++;
    }
    return R - L - 1;
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public String longestPalSubstr(String str) {
    int n = str.length();
    String longest = str.substring(0, 1);

    for (int i = 0; i < n - 1; i++) {
        String s1 = expandAroundCentre(str, i, i);
        if (s1.length() > longest.length()) {
            longest = s1;
        }

        String s2 = expandAroundCentre(str, i, i + 1);
        if (s2.length() > longest.length()) {
            longest = s2;
        }
    }
    return longest;
}

private String expandAroundCentre(String str, int l, int r) {
    int n = str.length();

    while (l >= 0 && r < n && str.charAt(l) == str.charAt(r)) {
        l--;
        r++;
    }
    return str.substring(l + 1, r);
}
```

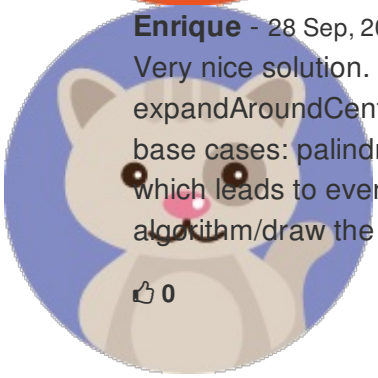
Comments



Eddie Tribaldos - 19 Aug, 2016

Ahh we had similar ideas. I like your execution better though.

0



Enrique - 28 Sep, 2016

Very nice solution. For anybody reading this, if you don't understand the need for the 2 calls to expandAroundCentre you are not alone! Correct me if I'm mistakes, but I understand it has to do with both base cases: palindrome of length 1, which leads to odd-length palindromes, and palindrome of length 2, which leads to even-length palindromes. That way we can cover all possibilities. I suggest you trace the algorithm/draw the spans of the different palindromes to see it more clearly.

0

1 2 3 4 5

Minimum Sum Path in a Triangle

Arrays Puzzles Multi Dimensional Arrays

Given a 'triangle' as an `ArrayList` of `ArrayList`s of integers, with each list representing a level of the triangle, find the **minimum sum** achieved by following a top-down path and adding the integer at each level along the path. Movement is restricted to adjacent numbers from the top to the bottom.

Note:
- You can only traverse through adjacent nodes while moving up or down the triangle.
- An adjacent node is defined as a node that is reached by moving down and left or down and right from a level. For eg, in the triangle shown below, if you are at the digit 3 in the second row, its adjacent nodes are 5 and 6

Example:

```
Input Triangle:
[   [1],
   [2, 3],
  [4, 5, 6],
[7, 8, 9, 10],
]

Output : 14 (1->2->4->7)
Note: [..] denotes an ArrayList
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int minTriangleDepth(ArrayList<ArrayList<Integer>> input) {
    int height = input.size();
    int[] outBuffer = new int[input.get(height-1).size()];

    for(int i = 0; i < input.get(height-1).size(); i++){
        outBuffer[i] = input.get(height-1).get(i);
    }

    for(int r = height-2; r >= 0; r--){
        ArrayList<Integer> row = input.get(r);
        for(int i = 0; i < row.size(); i++){
            outBuffer[i] = row.get(i) + Math.min(outBuffer[i], outBuffer[i+1]);
        }
    }
    return outBuffer[0];
}
```

Top voted solution

```
public static int minTriangleDepth(ArrayList<ArrayList<Integer>> input) {
    return helper(input, 0, 0);
}

public static int helper(ArrayList<ArrayList<Integer>> input, int row, int column) {
    if (row >= input.size())
        return 0;
    int curValue = input.get(row).get(column);
    int leftSum = helper(input, row + 1, column);
    int rightSum = helper(input, row + 1, column + 1);
    return curValue + Math.min(leftSum, rightSum);
}
```

Comments

Fooble - 22 Jan, 2016

I'm pretty sure this is O(N^3) because it visits some locations multiple times. It is very concise though.

1 2 3 4 5

Steal the Node

Trees

Write a method to delete a node from a given binary search tree.

```
Example:
   4                      4
  / \                    / \
 2   8   delete 10 =>   2   8
    / \                    /
   5  10                  5
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public TreeNode delete(TreeNode root, int data) {

    if (root == null) return null;
    if (data < root.data)
    {
        root.left = delete(root.left, data);
    }
    else if (data > root.data)
    {
        root.right = delete(root.right, data);
    }
    else
    {
        if (root.left == null && root.right == null)
        {
            root = null;
        }
        else if (root.left == null || root.right == null)
        {
            root = (root.left == null) ? root.right : root.left;
        }
        else
        {
            // full children.  Replace own value with min from right subTree, and and
            // remote the min node from the right tree;
```

```
            TreeNode min = findMin(root.right);
            root.data = min.data;
            root.right = delete(root.right, min.data);
        }
    }


    return root;


}
```

Top voted solution

```
public TreeNode delete(TreeNode root, int data) {
        // Add your code below this line. Do not modify any other code.
        if(root == null) {
            return null;
        } else if(data < root.data) {
            root.left = delete(root.left, data);
        } else if(data > root.data) {
            root.right = delete(root.right, data);
        } else { //element found
            if(root.left != null && root.right != null) { //full node case
                root.data = findMax(root).data; //find right most node (max) *must give this method
                root.left = delete(root.left, root.data);
            } else if(root.left == null && root.right == null) {
                root = null;
            }
            else if(root.left == null) {
                root = root.right;
            } else if(root.right == null) {
                root = root.left;
            }
        }
        return root;
        // Add your code above this line. Do not modify any other code.
    }
```

1 2 3 4 5

Word Similarity - Edit Distance

Multi Dimensional Arrays | Dynamic Programming | Strings

Edit distance is a classic algorithm that is used in many applications, including Spell Correction, DNA Sequencing and Natural Language Processing. Given two Strings, `a` and `b`, write a method - `editDistance` that returns the **minimum number of operations** needed to transform `a` into `b`. The following character operations are allowed :

a) Replace character
b) Insert character
c) Delete character

Examples :

```
editDistance("sale", "sales") => 1

Operations :
1) Insert "s"
```

```
editDistance("sale", "sold") => 2

Operations :
1) Replace "a" with "o"
2) Replace "e" with "d"
```

```
editDistance("sa", "s") => 1

Operations :
1) Delete "a"
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public int editDistance(String a, String b){
    int lenA = a.length(), lenB = b.length();
    int[][] memo = new int[lenA+1][lenB+1];
    // Prefill first row and column
    for(int i = 1; i <= lenA; i++) memo[i][0] = i;
    for(int j = 1; j <= lenB; j++) memo[0][j] = j;
```

```
    // Traverse and fill cells
    for(int i = 1; i <= lenA; i++){
        char cA = a.charAt(i-1);
        for(int j = 1; j <= lenB; j++){
            char cB = b.charAt(j-1);
            if(cA == cB){
                memo[i][j] = memo[i-1][j-1];
            }
            else {
                int replaceDist = 1 + memo[i-1][j-1];
                int insertDist = 1 + memo[i][j-1];
                int deleteDist = 1 + memo[i-1][j];
                int minDist = Math.min(replaceDist,Math.min(insertDist, deleteDist));
                memo[i][j] = minDist;
            }
        }
    }
    return memo[lenA][lenB];
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public int editDistance(String word1, String word2){

int len1 = word1.length();
 int len2 = word2.length();

 // len1+1, len2+1, because finally return dp[len1][len2]
 int[][] dp = new int[len1 + 1][len2 + 1];

 for (int i = 0; i <= len1; i++) {
  dp[i][0] = i;
 }

 for (int j = 0; j <= len2; j++) {
  dp[0][j] = j;
 }

 //iterate though, and check last char
 for (int i = 0; i < len1; i++) {
  char c1 = word1.charAt(i);
  for (int j = 0; j < len2; j++) {
   char c2 = word2.charAt(j);

   //if last two chars equal
   if (c1 == c2) {
    //update dp value for +1 length
```

```
    dp[i + 1][j + 1] = dp[i][j];
   } else {
    int replace = dp[i][j] + 1;
    int insert = dp[i][j + 1] + 1;
    int delete = dp[i + 1][j] + 1;

    int min = replace > insert ? insert : replace;
    min = delete > min ? min : delete;
    dp[i + 1][j + 1] = min;
   }
  }
 }

 return dp[len1][len2];
}
```

1 2 3 4 5

Mobile Game Range Module - Inserting Ranges

Sorting Algorithms | Arrays | Numbers

A Range Module is a module that tracks ranges of numbers. Range modules are used extensively when designing scalable online game maps with millions of players. Your task is to write a method - `insertRange` that takes in an `ArrayList` of **sorted, non-overlapping integer** `Interval` s (aka ranges) and a new `Interval` - `insert`, and returns an `ArrayList` of **sorted** `Interval` s where `insert` has been added to the `ArrayList` in the correct spot and the required overlapping ranges have been merged. The `Interval` class is available by clicking **Use Me**. Target a time complexity of **O(n)**.

Note:

a) [1,3] represents an interval that includes 1, 2 and 3.

b) Intervals should be sorted based on the value of `start`

c) The words **Range** and **Interval** are used interchangeably

Examples:

Inputs: [[1,3], [7,10]] & [2,6], Output: [[1,6], [7,10]]

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static ArrayList<Interval> insertRange(ArrayList<Interval> intervalsList, Interval insert) {
    ArrayList<Interval> out = new ArrayList<Interval>();

    for (int index=0; index<intervalsList.size(); ++index)
    {
        Interval i = intervalsList.get(index);
        if (i.end < insert.start)
        {
            // less than insert
            out.add(i);
        }
        else if (i.start > insert.end)
        {
            // greater than insert
            out.add(insert);
            insert = i;
        }
        else if ((insert.end >= i.start && insert.end <= i.end) ||
                 (insert.start >= i.start && insert.start <= i.end) ||
                 (insert.start >= i.start && insert.end <= i.end))
        {
```

```
            insert = new Interval(Math.min(insert.start, i.start), Math.max(insert.end, i.end));
        }
    }

    out.add(insert);
    return out;
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static ArrayList<Interval> insertRange(ArrayList<Interval> intervalsList, Interval insert) {
    ArrayList<Interval> result = new ArrayList<Interval>();

        if (intervalsList.isEmpty()) {
            result.add(insert);
            return result;
        }


        int start = Integer.MIN_VALUE;
        int end = Integer.MIN_VALUE;

        for (Interval i : intervalsList) {
            int relationship = compare(i, insert);

            switch (relationship) {
                case BEFORE:
                    result.add(i);
                    break;
                case PRE_CONTINUATION:
                    start = i.start;
                    break;
                case INCLUDED:
                    if (start == Integer.MIN_VALUE) start = insert.start;
                    break;
                case AFTER:
                    if (start == Integer.MIN_VALUE) {
                        start = insert.start;
                    }
                    if (end == Integer.MIN_VALUE) {
                        end = insert.end;
                        result.add(new Interval(start, end));
                    }
                    result.add(i);
                    break;
                case INCLUDES:
                    start = i.start;
```

```
                    end = i.end;
                    result.add(i);
                    break;
                case POST_CONTINUATION:
                    if (start == Integer.MIN_VALUE)  {
                        start = insert.start;
                    }
                    end = i.end;
                    result.add(new Interval(start, end));
                    break;
            }
        }

        if (end == Integer.MIN_VALUE) result.add(new Interval(start, insert.end));

        return result;
}

static final int BEFORE = 1;
static final int PRE_CONTINUATION = 2;
static final int INCLUDED = 3;
static final int AFTER = 4;
static final int INCLUDES = 5;
static final int POST_CONTINUATION = 6;

static int compare(Interval a, Interval b) {
    if (a.end < b.start) return BEFORE;
    if (a.start <= b.start && a.end >= b.start && a.end <= b.end) return PRE_CONTINUATION;
    if (a.start >= b.start && a.end <= b.end) return INCLUDED;
    if (a.start > b.end) return AFTER;
    if (a.start <= b.start && a.end >= b.end) return INCLUDES;
    if (a.start >= b.start && a.start <= b.end && a.end >= b.end) return POST_CONTINUATION;

    throw new RuntimeException("forgot something!");
}
```

1 2 3 4 5

Recovering IPv4 Addresses

DFS

You are given a `String` containing at least 4 numbers that represented an IPv4 Address, but the separator data - i.e. the dots that separate each Byte in a 4 Byte Ipv4 address, has been lost. Write a method - `generateIPAddrs` that takes in this `String` and returns an `ArrayList` of Strings containing all possible IPv4 Addresses that can be generated from the given sequence of decimal integers.

Note:

- The IP Addresses for this problem are written in the decimal dot notation.
- You must use all the digits in the input String
- The order in which the IP Addresses are returned does not matter
- 0.0.0.1 and 0.0.0.01 may be considered 2 distinct possibilities. i.e. do not ignore leading or trailing 0s.

Examples:

`generateIPAddrs("0001")` ==> `{"0.0.0.1"}`

`generateIPAddrs("0010")` ==> `{"0.0.1.0"}`

`generateIPAddrs("25525511135")` ==> `{"255.255.11.135", "255.255.111.35"}`

Your solution

```
/*
 You are given a String containing at least 4 numbers that represented an IPv4 Address, but the separ
ator data - i.e. the dots that separate each Byte in a 4 Byte Ipv4 address, has been lost. Write a met
hod - generateIPAddrs that takes in this String and returns an ArrayList of Strings containing all po
ssible IPv4 Addresses that can be generated from the given sequence of decimal integers.


 Note:

 - The IP Addresses for this problem are written in the decimal dot notation.
 - You must use all the digits in the input String
 - The order in which the IP Addresses are returned does not matter
 - 0.0.0.1 and 0.0.0.01 may be considered 2 distinct possibilities. i.e. do not ignore leading or tra
iling 0s.

 Examples:

 generateIPAddrs("0001") ==> {"0.0.0.1"}
```

```
 generateIPAddrs("0010") ==> {"0.0.1.0"}

 generateIPAddrs("25525511135") ==> {"255.255.11.135", "255.255.111.35"}

 */

public static ArrayList<String> generateIPAddrs(String input)
{
   class IpLevelNode {
     public int level = 0;
     public String predecessor;
     public String successor;

     public IpLevelNode(int level, String ipToAppend, String predecessor, String successor) {
       this.level = level;
       this.successor = successor;
       if (level == 0) {
           this.predecessor = ipToAppend;
       } else {
           this.predecessor = predecessor + "." + ipToAppend;
       }
     }
   }

   ArrayList<String> out = new ArrayList<>();
   Deque<IpLevelNode> stack = new LinkedList<>();
   // Push 3 possibilities onto the stack
   stack.addFirst(new IpLevelNode(0, input.substring(0,1),"", input.substring(1)));
   stack.addFirst(new IpLevelNode(0, input.substring(0,2),"", input.substring(2)));
   stack.addFirst(new IpLevelNode(0, input.substring(0,3),"", input.substring(3)));

   while(!stack.isEmpty()){
     IpLevelNode node = stack.removeFirst();
     int curlevel = node.level;
     String predecessor = node.predecessor;
     String remaining = node.successor;
     if(curlevel == 3 && remaining.length() == 0){
         out.add(node.predecessor);
         continue;
     }
     int i = 1;
     while(i <= 3){
       if(remaining.length() < i) break;
       String ipToAppend = remaining.substring(0,i);
       String successor = remaining.substring(i);
       if(ipToAppend.length() > 0){
         int numIpToAppend = Integer.parseInt(ipToAppend);
         if(numIpToAppend <= 255) stack.addFirst(new IpLevelNode(curlevel+1,ipToAppend,predecessor,su
ccessor));
       }
       i++;
     }
   }
   return out;
```

```
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static ArrayList<String> generateIPAddrs(String s){
        return generateIPAddrsHelper("", s, 4);
    }

    public static ArrayList<String> generateIPAddrsHelper(String prefix, String s, int level){
        ArrayList<String> reval = new ArrayList<>();
        if(level < 1 || s.isEmpty())
            return reval;
        if(level == 1){
            if(s.length() <= 3 && Integer.parseInt(s) <= 255) {
                reval.add(prefix + "." + s);
                return reval;
            }
            else
                return reval;
        }

        for(int i = 1; i < 4 && i <= s.length(); i++){
                String range = s.substring(0, i);
                if(Integer.parseInt(range) <= 255){
                   String newPrefix = prefix.isEmpty() ? range : prefix + "." + range;
                   reval.addAll(generateIPAddrsHelper(newPrefix, s.substring(i), level - 1));
                }
        }

        return reval;
    }
```

1 2 3 4 5

Find the Lowest Common Ancestor of Two Tree Nodes

Trees

Given a binary tree and two tree nodes, write a method to find LCA (Lowest Common Ancestor) of the two nodes.

```
Example:
       1
      / \
     2   3
    / \ / \
   4  5 6  7
    ==> LCA of 6 and 4 is 1,
    LCA of 4 and 5 is 2.
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public TreeNode findLCA(TreeNode root, TreeNode a, TreeNode b) {

    if (root == null) return null;
    if (root == a || root == b)
    {
        return root;
    }

    TreeNode left = findLCA(root.left, a, b);
    TreeNode right = findLCA(root.right, a, b);


    if (left != null && right != null)
    {
        return root;
    }

    if (left != null)
    {
        return left;
    }

    if (right != null)
```

```
    {
        return right;
    }

    return null;

}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public TreeNode findLCA(TreeNode root, TreeNode a, TreeNode b) {
    // Add your code below this line. Do not modify any other code.
    if(root == null || root == a || root == b) {
        return root;
    }

    TreeNode rightAncestor = findLCA(root.right, a, b);
    TreeNode leftAncestor = findLCA(root.left, a, b);

    if(rightAncestor!=null && leftAncestor!=null) {
        return root;
    }
    return rightAncestor!=null?rightAncestor:leftAncestor;
    // Add your code above this line. Do not modify any other code.
}
```

1 2 3 4 5

Image Manipulation

Multi Dimensional Arrays

You are given an n x n square 2D matrix that represents the pixels of an image.
Rotate it by 90 degrees in the clockwise direction.

Example:

```
Input Matrix :

               1 0
               0 1

Output       :

               0 1
               1 0
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int[][] rotate(int[][] matrix) {

    int size = matrix.length;

    for (int layer=0; layer<size / 2; ++layer)
    {
        int first = layer;
        int last = size - layer - 1;


        for (int i=first; i<last; ++i)
        {
            int offset = i - first;
            int top = matrix[first][i]; // save top

   // left -> top
   matrix[first][i] = matrix[last-offset][first];

   // bottom -> left
   matrix[last-offset][first] = matrix[last][last - offset];
```

```
   // right -> bottom
   matrix[last][last - offset] = matrix[i][last];

   // top -> right
   matrix[i][last] = top; // right <- saved top
        }


    }
    return matrix;

}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int[][] rotate(int[][] matrix) {
    // Write your code below this line. Do not modify any other part of the code.

    int n = matrix.length;
    for (int i = 0; i < n / 2; i++) {
        for (int j = 0; j < (n+1)/ 2; j++) {
            int t = matrix[i][j];
            matrix[i][j] = matrix[n-j-1][i];
            matrix[n-j-1][i] = matrix[n-i-1][n-j-1];
            matrix[n-i-1][n-j-1] = matrix[j][n-i-1];
            matrix[j][n-i-1] = t;
        }
    }

    return matrix;

    // Write your code above this line. Do not modify any other part of the code.
  }
```

1 2 3 4 5

Better Fibonacci

Numbers

The Fibonacci Sequence is the series of numbers: `0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...` The next number is found by adding up the two numbers before it.

Your goal is to write an **optimal** method - `betterFibonacci` that returns the `nth` Fibonacci number in the sequence. `n` is 0 indexed, which means that in the sequence `0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...`, n == 0 should return 0 and n == 3 should return 2. Your method should exhibit a runtime complexity of **O(n)** and use constant **O(1)** space. With this implementation, your method should be able to compute larger sequences where n > 40.

Examples:

`fib(0) ==> 0`

`fib(1) ==> 1`

`fib(3) ==> 2`

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int betterFibonacci(int n) {

    int n_2 = 0;
    int n_1 = 1;
    int temp = n_2 + n_1;

    if (n == 0) return n_2;
    if (n == 1) return n_1;

    for (int i=2; i<= n; i++)
    {
        temp = n_1 + n_2;
        n_2 = n_1;
        n_1 = temp;

    }

    return temp;
```

```
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int betterFibonacci(int n) {
    if(n < 2) return n;

    int [] fibs = new int[n+1];
    fibs[0] = 0;
    fibs[1] = 1;

    for(int i = 2; i < fibs.length; i++)
        fibs[i] = fibs[i-2] + fibs[i-1];


    return fibs[n];
}
```

1 2 3 4 5

Mirror Mirror on the Wall ...

Trees

Write a method to check if the two given binary trees are the **mirror images** of each other. Return `true` if they are, `false` otherwise. What's a binary tree's mirror image? Hold it by the root and rotate all other nodes by 180 degrees!

```
Example:
       1               1
      / \             / \
     2   3     ==    3   2
    / \ / \         / \ / \
   4  5 6  7       7  6 5  4
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public boolean isMirror(TreeNode root1, TreeNode root2) {

    if (root1 == null && root2 == null)
    {
        return true;
    }

    if ((root1 != null && root2 == null) ||
       (root1 == null && root2 != null) ||
       (root1.data != root2.data))
    {
        return false;
    }

    return isMirror(root1.left, root2.right) && isMirror(root1.right, root2.left);


}
```

Top voted solution

```java
// java.util.* has been imported for this problem.
// You don't need any other imports.

public boolean isMirror(TreeNode root1, TreeNode root2) {
    // Add your code below this line. Do not modify any other coode.
    if (root1 == null && root2 == null) return true;
    if (root1 == null || root2 == null) return false;
    if (root1.data != root2.data) return false;
    return isMirror(root1.left, root2.right) && isMirror(root2.left, root1.right);
    // Add your code above this line. Do not modify any other code.
}
```

1 2 3 4 5

Delete the Node at a Particular Position in a Linked List

Linked Lists

Given a singly-linked list, implement a method to delete the node at a given position (starting from 1 as the head position) and return the head of the list. Do nothing if the input position is out of range.

```
Examples:

LinkedList: 1->2->3->4 , Head = 1

deleteAtMiddle(Head,3) ==> 1->2->4
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ListNode deleteAtMiddle(ListNode head, int position) {

    if (head == null)
    {
        return null;
    }

    ListNode current = head;
    ListNode previous = null;
    for (int i=1; i < position; ++i)
    {
        if (current == null)
        {
            // past the end, return head
            return head;
        }
        previous = current;
        current = current.next;
    }

    if (previous == null)
    {
        // deleting first node
```

```
        return current.next;
    }
    else
    {
        previous.next = current.next;
    }

    return head;

}
```

Top voted solution

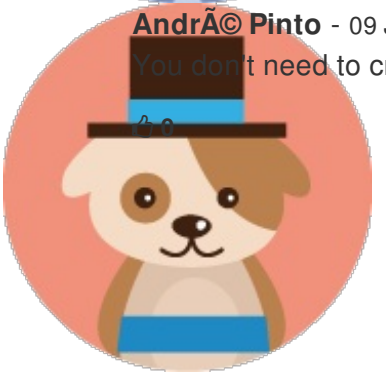
```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ListNode deleteAtMiddle(ListNode head, int position) {
    if (head == null) { return null; }
    if (position == 1) { return head.next; }
    ListNode result = new ListNode(head.data);
    result.next = deleteAtMiddle(head.next, position - 1);
    return result;

}
```

Comments

Darren - 21 Jun, 2016

Really well done

0



AndrÃ© Pinto - 09 Jul, 2016

You don't need to create a new ListNode every time. Changing .next is enough.

0

1 2 3 4 5

Distance between two nodes in a Binary Tree

Trees Strings Queues

Given the root of a Binary Tree and 2 integers that represent the `data` values of any two `TreeNode`s present in the tree, write a method - `getNodeDistance` that returns the distance between the nodes. You can assume that the given keys exist in the tree. The **distance** between two nodes is defined as the minimum number of **edges** that must be traversed to travel between the two nodes.

Example:

```
        1
       / \
      2   3
       \   \
        4   5

getNodeDistance(2,5) => 3
```

Your solution

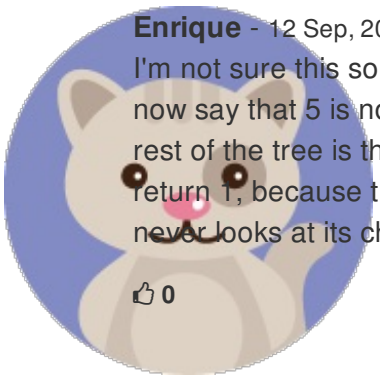
```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public int getNodeDistance(TreeNode root, int n1, int n2) {

    TreeNode lca = getLCA(root, n1, n2);
    ArrayList<TreeNode> n1_path = new ArrayList<TreeNode>();
    ArrayList<TreeNode> n2_path = new ArrayList<TreeNode>();

    if (!findPath(lca, n1_path, n1) || !findPath(lca, n2_path, n2))
    {
        System.out.println("Halp....something is wrong.");
        return -1;
    }

    return n1_path.size() -1 + n2_path.size() -1;


}

private static boolean findPath(TreeNode node, ArrayList<TreeNode> path, int val)
{
    if (node == null)
    {
```

```
        return false;
    }

    path.add(node);

    if (node.data == val)
    {
        return true;
    }

    if (findPath(node.left, path, val) || findPath(node.right, path, val))
    {
        return true;
    }
    else
    {
        // pop back the last node added
        path.remove(path.size()-1);
        return false;
    }

}


private static TreeNode getLCA(TreeNode node, int a, int b)
{
    if (node == null) return null;

    ArrayList<TreeNode> a_path= new ArrayList<TreeNode>();
    ArrayList<TreeNode> b_path = new ArrayList<TreeNode>();

    if (!findPath(node, a_path, a) || !findPath(node, b_path, b))
    {
        return null;
    }

    int i = 0;
    for (; i < a_path.size() && i < b_path.size(); ++i)
    {
        if (a_path.get(i) != b_path.get(i))
        {
            break;
        }
    }

    return a_path.get(i-1);
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public int getNodeDistance(TreeNode root, int n1, int n2) {

    return getNodeDistanceHelper(root, n1, n2, 0);

}

public int getNodeDistanceHelper(TreeNode root, int n1, int n2, int depth) {
    if(root == null)
        return 0;

    if((root.data == n1 || root.data == n2) && depth != 0 )
        return depth;

    int left  = getNodeDistanceHelper(root.left, n1, n2, depth + 1);
    int right = getNodeDistanceHelper(root.right, n1, n2, depth + 1);

    if(left != 0 && right != 0){
        return (left - depth) + (right - depth);
    }

    return Math.max(left, right);

}
```

Comments



Enrique - 12 Sep, 2016

I'm not sure this solves the problem. Let's take the second test case: getNodeDistance(root,2,5) = 1. Let's now say that 5 is not the child of 2, but its grandchild (that is, 2 has a child X, and X is parent of 5). The rest of the tree is the same as the one in the test case. If I understand this correctly, this code would still return 1, because the helper returns as soon as it finds 2 (which is equal to n1 with a depth of 1) and never looks at its children.

0

1 2 3 4 5

Combinations and Permutations

Recursion

Given a string, list all possible combinations and permutations of its characters.

Examples:

`getCombPerms("a") ==> {"a"}`

`getCombPerms("ab") ==> {"a","ab","ba","b"}`

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static ArrayList<String> getCombPerms(String s) {

    if (s == null)
    {
        // error case
        return null;
    }

    ArrayList<String> set = new ArrayList<String>();
    if (s.length() == 1)
    {
        // base case
        set.add(s);
        return set;
    }

    char prefix = s.charAt(0);
    String suffix = s.substring(1);

    set.add(String.valueOf(prefix));

    ArrayList<String> subSets = getCombPerms(suffix);

    for (String str : subSets)
    {
        for (int i=0; i <= str.length(); ++i)
        {
            set.add(insertCharToIndex(str, prefix, i));
        }
    }
```

```
    set.addAll(subSets);

    return set;
}

private static String insertCharToIndex(String s, char c, int i)
{
    return s.substring(0, i) + c + s.substring(i);
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static ArrayList<String> getCombPerms(String s) {
    if (s == null) {
        return null;
    }

    ArrayList<String> combPerms = new ArrayList<>();
    getCombPerms("", s, combPerms);
    return combPerms;
}

private static void getCombPerms(String prefix, String s, List<String> combPerms) {
    if (!prefix.isEmpty()) {
        combPerms.add(prefix);
    }

    for (int i = 0; i < s.length(); i++) {
        getCombPerms(prefix + s.charAt(i), s.substring(0, i) + s.substring(i + 1), combPerms);
    }
}
```

1 2 3 4 5

Find the Level that has the Maximum Sum

Trees Queues

Given a binary tree, write a method to return the level that has the maximum sum. In case the tree is empty, return **-1**

```
Example:
       1
      / \
     2   3
    / \ / \
   4  5 6  7
  /
 8

Output ==> 2
```

Note: Assume that root is at level 0.

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public int findMaxSumLevel(TreeNode root) {

    int maxLevel = -1;

    // error case
    if (root == null) return maxLevel;

    ArrayList<Integer> levelSum = new ArrayList<Integer>();
    InorderTraverseAndAdd(root, 0, levelSum);

    int maxVal = Integer.MIN_VALUE;

    for (int i=0; i<levelSum.size(); ++i)
    {
        if (levelSum.get(i) > maxVal)
        {
            maxLevel = i;
            maxVal = levelSum.get(i);
```

```
        }
    }
    return maxLevel;
}

private static void InorderTraverseAndAdd(TreeNode root, int level, ArrayList<Integer> levelSum)
{

    if (root == null)
    {
        return;
    }

    if (level == levelSum.size())
    {
        levelSum.add(level, 0);
    }

    InorderTraverseAndAdd(root.left, level+1, levelSum);

    int prevSum = levelSum.get(level);
    levelSum.set(level, prevSum + root.data);

    InorderTraverseAndAdd(root.right, level+1, levelSum);
}
```

Top voted solution

```
public int findMaxSumLevel(TreeNode root) {
    if (root == null)
        return -1;
    ArrayList<Integer> sums = new ArrayList();
    Queue<TreeNode> nodeQueue = new LinkedList();
    Queue<Integer> levelQueue = new LinkedList(); // The level of nodes in the queue
    nodeQueue.add(root);
    levelQueue.add(0);
    while (!nodeQueue.isEmpty()) {
        TreeNode curNode = nodeQueue.remove();
        int level = levelQueue.remove();
        if (curNode == null)
            continue;
        nodeQueue.add(curNode.left);
        levelQueue.add(level + 1);
        nodeQueue.add(curNode.right);
        levelQueue.add(level + 1);
        // Add to the sum for this level.
        if (sums.size() == level)
            sums.add(0);
        sums.set(level, sums.get(level) + curNode.data);
    }
```

```
    int maxLevel = 0;
    for (int level = 1; level < sums.size(); level++) {
        if (sums.get(level) > sums.get(maxLevel))
            maxLevel = level;
    }
    return maxLevel;
}
```

1 2 3 4 5

Check Balanced Parentheses

Recursion Stacks

Write a method to **recursively** check whether an equation has a balanced number of left and right parentheses and brackets - (including `(,),[,],{,}`).

Examples:

`isBalanced("() [] ()") ==> true`

`isBalanced("([)]") ==> false`

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static boolean isBalanced(String input) {

    HashMap<Character, Character> mapping = new HashMap<Character, Character>();
    mapping.put('(', ')');
    mapping.put('[', ']');
    mapping.put('{', '}');

    Stack<Character> s = new Stack<Character>();

    for (int i=0; i<input.length(); ++i)
    {
        Character c = input.charAt(i);
        if (mapping.containsKey(c))
        {
            s.push(c);
        }
        else
        {
            if (s.isEmpty())
            {
                // should have a match
                return false;
            }
            Character ch = s.peek();
            s.pop();
            Character expected = mapping.get(ch);
            if (c != expected)
            {
```

```
                return false;
            }
        }
    }


    return s.isEmpty();

}
```

Top voted solution

```
public static boolean isBalanced(String input) {
    if (input == null)
        return true;
    int originalLength = input.length();
    if (originalLength == 0)
        return true;
    // Remove all occurences of (), [], or {}.
    input = input.replaceAll("(\\(\\)|\\[\\]|\\{\\})", "");
    if (input.length() == originalLength) // No change made
        return false;
    return isBalanced(input);
}
```

Comments

Sergey Tychinin - 05 Aug, 2016

I seems to me that it won't work if string contains other characters than parenthesis, like "(asdf)", or the string from example "() [] ()".

1 2 3 4 5

Reverse Level Order Traversal

Trees Stacks Queues

Traverse a given binary tree in the Reverse Level Order. Mark a node as visited by adding its data to an `ArrayList` which will be returned.

```
Example:
     1
    / \
   2   3
  / \ / \
 4  5 6  7

Output => 4567231
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ArrayList<Integer> levelorderRev(TreeNode root) {

    ArrayList<Integer> list = new ArrayList<Integer>();
    if (root == null)
    {
        return list;
    }

    Queue<TreeNode> q = new LinkedList<TreeNode>();
    Stack<Integer> s = new Stack<Integer>();

    q.offer(root);
    while (!q.isEmpty())
    {
        TreeNode n = q.poll();

        if (n.right != null)
        {
            q.offer(n.right);
        }
        if (n.left != null)
        {
            q.offer(n.left);
        }
```

```
        s.push(n.data);
    }

    while (!s.isEmpty())
    {
        list.add(s.peek());
        s.pop();
    }


    return list;


}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ArrayList<Integer> levelorderRev(TreeNode root) {
    // Add your code below this line. Do not modify any other code.
    Queue<TreeNode> queue = new LinkedList<TreeNode>(Arrays.asList(root));
    ArrayList<Integer> result = new ArrayList<Integer>();
    if(root!=null) {
        while(!queue.isEmpty()) {
            TreeNode node = queue.remove();
            if(node.right!=null) queue.add(node.right);
            if(node.left!=null) queue.add(node.left);
            result.add(0, node.data);
        }
    }
    return result;
    // Add your code above this line. Do not modify any other code.
}
```

1 2 3 4 5

1-800-PROBLEM

Hash-Tables DFS Search Algorithms

Given a `String` that represents the digits pressed on a classic cell phone keypad - return all possible letter combinations that the numbers could represent in an `ArrayList` of `String`s. Check out the keypad and mapping below for reference.

Note:

a) You can assume that the input String contains only numbers between 2 and 9.

b) The order of the combinations in the output does not matter.

Mapping:

2 -> "abc"

3 -> "def"

4 -> "ghi"

5 -> "jkl"

6 -> "mno"

7 -> "pqrs"

8 -> "tuv"

9 -> "wxyz"

Example:

```
Input  : "34"
Output : [dg, dh, di, eg, eh, ei, fg, fh, fi]
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static  ArrayList<String> getStringsFromNums(String digits)
{

       HashMap<Character, String> mapping = new HashMap<Character, String>();
        mapping.put('2', "abc");
        mapping.put('3', "def");
        mapping.put('4', "ghi");
        mapping.put('5', "jkl");
        mapping.put('6', "mno");
        mapping.put('7', "pqrs");
        mapping.put('8', "tuv");
        mapping.put('9', "wxyz");
```

```
        class PhoneNode
        {
            String word;
            int digitCount;
            PhoneNode(String w, int c)
            {
                word = w;
                digitCount = c;
            }
        }

        ArrayList<String> out = new ArrayList<String>();
        Stack<PhoneNode> stack = new Stack<PhoneNode>();
        int len = digits.length();

        for (Character c : mapping.get(digits.charAt(0)).toCharArray())
        {
            stack.push(new PhoneNode(String.valueOf(c), 1));
        }

        while (!stack.isEmpty())
        {
            PhoneNode node = stack.peek();
            stack.pop();

            if (node.digitCount == len)
            {
                out.add(node.word);
            }
            else
            {
                for (Character ch : mapping.get(digits.charAt(node.digitCount)).toCharArray())
                {
                    stack.push(new PhoneNode(node.word + ch, node.digitCount + 1));
                }
            }
        }

        return out;

}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static  ArrayList<String> getStringsFromNums(String digits) {
    ArrayList<String> out = new ArrayList<>();
    if(digits == null || digits.length() == 0) return out;
```

```
    HashMap<Integer, String> map = new HashMap<>();
    map.put(2,"abc");
    map.put(3,"def");
    map.put(4,"ghi");
    map.put(5,"jkl");
    map.put(6,"mno");
    map.put(7,"pqrs");
    map.put(8,"tuv");
    map.put(9,"wxyz");
    StringBuilder sb = new StringBuilder(); // Reusable StringBuilder
    search(sb, map, out, 0, digits);
    return out;
}

public static void search(StringBuilder sb, HashMap<Integer, String> map, ArrayList<String> out, int
index, String digits){
   if(sb.length() == digits.length()){
       out.add(sb.toString());
       if(sb.length() > 0) sb.deleteCharAt(sb.length()-1); // Backtrack Cleanup
       return;
   }
   int digit = Character.getNumericValue(digits.charAt(index));
   String letters = map.get(digit);
   for(int i = 0; i < letters.length(); i++){
       char c = letters.charAt(i);
       sb.append(c);
       search(sb, map, out, index+1, digits);
   }
   if(sb.length() > 0) sb.deleteCharAt(sb.length()-1); // Backtrack Cleanup
   return;
}
```

1 2 3 4 5

Isomorphic Strings

Arrays Strings Hash-Tables

Given two strings - input1 and input2, determine if they are **isomorphic**.
Two strings are **isomorphic** if the letters in one string can be **remapped** to get the second string. Remapping a letter means replacing **all occurrences** of it with another letter. The ordering of the letters remains unchanged. You can also think of isomorphism as it is used in chemistry - i.e. having the **same form or overall shape**. Target linear time and space complexity with your solution.

Examples:

```
Input 1 : css
Input 2 : dll
Output  : true

Input 1 : css
Input 2 : dle
Output  : false

Input 1 : abcabc
Input 2 : xyzxyz
Output  : true

Input 1 : abcabc
Input 2 : xbexyz
Output  : false
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static boolean isIsomorphic(String input1, String input2) {

    if (input1.length() != input2.length())
    {
        return false;
    }

    HashMap<Character, Integer> map1 = new HashMap<Character, Integer>();
    HashMap<Character, Integer> map2 = new HashMap<Character, Integer>();

    for (int i=0; i<input1.length(); ++i)
    {
        map1.put(input1.charAt(i), i);
```

```
        map2.put(input2.charAt(i), i);
    }

    for (int i=0; i<input1.length(); ++i)
    {
        if (map1.get(input1.charAt(i)) != map2.get(input2.charAt(i)))
        {
            return false;
        }
    }
    return true;
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static boolean isIsomorphic(String input1, String input2) {
    if (input1 == null || input2 == null || input1.length() != input2.length())
        return false;

    Map<Character, Character> m = new HashMap<>();
    Set<Character> rev = new HashSet<>();

    for (int i = 0; i < input1.length(); i++) {
        char c1 = input1.charAt(i);
        char c2 = input2.charAt(i);
        Character mapped = m.get(c1);
        if (mapped == null) {
            if (rev.contains(c2))
                return false;
            else {
                m.put(c1, c2);
                rev.add(c2);
            }
        } else if (mapped != c2)
            return false;
    }

    return true;
}
```

1 2 3 4 5

Bit Swapping

Bit Manipulation

Given a 32 bit **integer** input `x`, swap its odd and even bits and return the resulting **integer**. (e.g., bit 0 and bit 1 are swapped, bit 2 and bit 3 are swapped, and so on).

For example:

```
x = 5, Binary code = 0101
```

`swapOddEvenBits(5) --> 10`

```
Binary representation of 10 = 1010
```

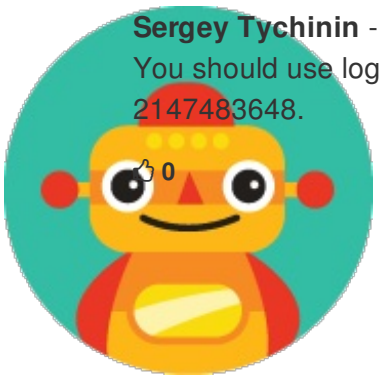
Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public int swapOddEvenBits(int x) {

    int oddMask = 0xAA;  // 1010101010
    int evenMask = 0x55; // 0101010101

    int oddVal = oddMask & x;
    int evenVal = evenMask & x;

    oddVal >>= 1;
    evenVal <<= 1;

    return oddVal | evenVal;



}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public int swapOddEvenBits(int x) {
    //Add your code below this line

    return (((x & 0xAAAAAAAA) >> 1) |
            ((x & 0x55555555) << 1));
}
```

Comments



Sergey Tychinin - 07 Aug, 2016

You should use logical right shift (>>>), not arithmetic. This solution wouldn't work for x = -2147483648.

1 2 3 4 5

Bit Conversion

Bit Manipulation

Given two input integers `a` and `b`, write a method to determine the number of bits required to be swapped to convert `a` to `b`.

Example:

```
a = 21, Binary code = 10101
b = 31, Binary code = 11111
```

`bitSwapRequired(a,b) --> 2`

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public int bitSwapRequired(int a, int b) {

    int diff = 0;

    while (a > 0 && b > 0)
    {
        if ((a & 1) != (b & 1))
        {
            diff++;
        }

        a >>= 1;
        b >>= 1;
    }

    while (a > 0)
    {
        if ((a & 1) == 1)
        {
            diff++;
        }
        a >>= 1;
    }

    while (b > 0)
    {
        if ((b & 1) == 1)
```

```
        {
            diff++;
        }
        b >>= 1;
    }

    return diff;

}
```

Comments

TangoZulu - 17 Aug, 2016

First thing that came to mind is to simply check for non equal LSB, and keep shifting down.

0

Top voted solution

```
public int bitSwapRequired(int a, int b) {
    if (a < 0 || b < 0)
        return -1;
    int diffCount = 0;
    while (a > 0 || b > 0) {
        if (a % 2 != b % 2)
            diffCount++;
        a /= 2;
        b /= 2;
    }
    return diffCount;
}
```

1 2 3 4 5

Check a Linked List for Loops or Cycles With O(1) Space Complexity

Linked Lists

Check if a given linked list has cycles. Try to achieve O(n) runtime with a space complexity of O(1). If there is a cycle, return `true` otherwise return `false`. Consider empty lists as non cyclic.

Examples:

`1->2->3->4->1 ==> true`

`1->2->3->4 ==> false`

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public Boolean isCyclic(ListNode head) {

    ListNode slower = head;
    ListNode faster = head;

    while (faster != null && faster.next != null)
    {
        faster = faster.next.next;
        slower = slower.next;

        if (faster == slower)
        {
            return true;
        }
    }

    return false;


}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public Boolean isCyclic(ListNode head) {
    // Add your code below this line. Do not modify any other code.

    if (head == null || head.next == null || head.next.next == null)
        return false;

    ListNode slow = head.next;
    ListNode fast = head.next.next;

    while (fast.next != null && fast.next.next != null) {
        if (fast == slow) {
            return true;
        }
        fast = fast.next.next;
        slow = slow.next;
    }

    return false;

    // Add your code above this line. Do not modify any other code.
}
```

1 2 3 4 5

Matrix Max Sum Path with Dynamic Programming

Multi Dimensional Arrays | Dynamic Programming

Given an **m x n** matrix filled with **non-negative** integers, use dynamic programming techniques to find the maximum sum along a path from the top-left of the grid to the bottom-right. Return this maximum sum. The direction of movement is limited to right and down.

Example:

```
Input Matrix :

    1 2 3
    4 5 6
    7 8 9

Output  : 1 + 4 + 7 + 8 + 9 = 29
```

Note:

You may have previously solved the DFS variant of this problem. That won't work for large sized matrices - just consider the size of the recursion tree for a 100x100 matrix! Dynamic Programming should afford a better solution.

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int matrixMaxSumDP(int[][] grid) {
    int rows = grid.length;
    int cols = grid[0].length;
    int[][] memo = new int[rows][cols];

    memo[0][0] = grid[0][0];

    for (int row = 1; row < rows; ++row)
    {
        memo[row][0] = memo[row-1][0] + grid[row][0];
    }

    for (int col = 1; col < cols; ++col)
    {
        memo[0][col] = memo[0][col-1] + grid[0][col];
    }
```

```
    for (int row = 1; row < rows; ++row)
    {
        for (int col = 1; col < cols; ++col)
        {
            memo[row][col] = grid[row][col] + Math.max(memo[row-1][col], memo[row][col-1]);
        }
    }

    return memo[rows-1][cols-1];
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int matrixMaxSumDP(int[][] grid) {
    int[][] memo = new int[grid.length][grid[0].length];

    //memo[0][0] = grid[0][0];
    int up=0;
    int left =0;

    for(int i= 0; i< grid.length; i++){
        for(int j=0; j<grid[i].length; j++){

            if(i-1>=0)
                up = memo[i-1][j];
            if(j-1>=0)
                left = memo[i][j-1];

            memo[i][j] = Math.max(up,left) + grid[i][j];
            up = 0;
            left = 0;
        }
    }


    return memo[grid.length-1][grid[0].length-1];
}
```

1 2 3 4 5

Recursive String Permutation

Recursion Strings

String permutations are the various possible strings made by the **rearrangement** of the characters in the original String.

For example, the permutations of `car` are

```
car, cra, acr, arc, rac, rca
```

Write a **recursive** method `getPermutations()` that returns all permutations of an input `String` in an `ArrayList`. Define a helper method if needed. For the sake of simplicity, assume that all characters in the input `String` are unique.

Examples:

`getPermutations("") -> ArrayList -> []`

`getPermutations("c") -> ArrayList -> ["c"]`

`getPermutations("cat") -> ArrayList -> ["cat","cta","act","atc","tca","tac"]` *

*Order does not matter.

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static ArrayList<String> getPermutations(String s) {

    System.out.println(s);

    if (s == null)
    {
        return null;
    }

    ArrayList<String> perms = new ArrayList<String>();
    if (s.length() == 0)
    {
        perms.add("");
        return perms;
    }
```

```
    char prefix = s.charAt(0);
    String suffix = s.substring(1);
    ArrayList<String> words = getPermutations(suffix);
 for (String word : words) {
  for (int i = 0; i <= word.length(); i++) {
   String p = insertChar(word, prefix, i);
   perms.add(p);
  }
 }

    return perms;

}

private static String insertChar(String s, char c, int index)
{
    System.out.println(s + ", " + c + ", " + index);
    String str = s.substring(0, index) + c + s.substring(index);
    System.out.println(str);
    return str;
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static ArrayList<String> getPermutations(String s) {
  // Add your code below this line. Do not modify any other code.
    ArrayList<String> perms = new ArrayList<String>();
    if (s == null)
        return null;
    if (s.equals(""))
        return perms;
    if (s.length() == 1) {
        perms.add(s);
        return perms;
    }
    for (int i = 0; i < s.length(); i++) {
        char removed = s.charAt(i);
        String sub = s.substring(0, i) + s.substring(i+1);
        ArrayList<String> subperms = getPermutations(sub);
        for (String subperm : subperms) {
            perms.add(removed + subperm);
        }
    }
    return perms;
  // Add your code above this line. Do not modify any other code.
}
```

1 2 3 4 5

Boggle Search

Multi Dimensional Arrays | DFS | Search Algorithms | Recursion

You're given a 2D **Boggle Board** which contains an m x n matrix of chars - `char[][] board`, and a String - `word`. Write a method - `boggleSearch` that searches the Boggle Board for the presence of the input `word`. Words on the board can be constructed with **sequentially adjacent** letters, where adjacent letters are horizontal or vertical neighbors (not diagonal). Also, each letter on the Boggle Board must be used only once.

Example:

```
Input Board :
{
    {A, O, L},
    {D, E, L},
    {G, H, I},
}
Word: "HELLO"
Output: true
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static boolean boggleSearch(char[][] board, String word){

    for (int row = 0; row < board.length; ++row)
    {
        for (int col = 0; col < board[0].length; ++col)
        {
            if (search(board, row, col, word, ""))
            {
                return true;
            }
        }
    }

    return false;
}

private static boolean search(char[][]board, int row, int col, String word, String prefix)
{
    int maxRow = board.length - 1;
    int maxCol = board[0].length - 1;
```

```
    if (row < 0 ||
        row > maxRow ||
        col < 0 ||
        col > maxCol ||
        !word.contains(prefix) ||
        board[row][col] == '*')
    {
        return false;
    }

    char c = board[row][col];
    prefix += c;

    if (word.equals(prefix))
    {
        return true;
    }

    board[row][col] = '*';
    boolean result = search(board, row + 1, col, word, prefix) ||
                     search(board, row - 1, col, word, prefix) ||
                     search(board, row, col + 1, word, prefix) ||
                     search(board, row, col - 1, word, prefix);
    board[row][col] = c;
    return result;
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static boolean boggleSearch(char[][] board, String word){
    if(board.length == 0){
        return false;
    }
    else if(word.length() > (board.length * board[0].length)){
        return false;
    }


    for(int row = 0; row < board.length; row++){
        for(int col = 0; col < board[row].length; col++){
            if(tryNext(board, word, row, col, new boolean[board.length][board[0].length])){
                return true;
            }
        }
    }

    return false;
```

```
}

private static boolean tryNext(char[][] board, String word, int row, int col, boolean[][] visited){
        if(word.length() == 0){
            return true;
        }

        if(board[row][col] != word.charAt(0)){
            return false;
        }

        visited[row][col] = true;

        boolean found = false;

        //Northern Side
        //check Northwest
        if(row - 1 >= 0){
            if(col - 1 >= 0 && !visited[row - 1][col - 1]){
                found = tryNext(board, word.substring(1), row - 1, col - 1, visited);
            }

            //Check North
            if(!found && !visited[row - 1][col]){
                found = tryNext(board, word.substring(1), row - 1, col, visited);
            }

            //Check Northeast
            if(!found && col + 1 < visited[0].length && !visited[row - 1][col + 1]){
                found = tryNext(board, word.substring(1), row - 1, col + 1, visited);
            }
        }

        //Check East
        if(!found && col + 1 < visited[0].length && !visited[row][col + 1]){
            found = tryNext(board, word.substring(1), row, col + 1, visited);
        }

        //Check Southern Side
        if(!found && row + 1 < visited.length){
            //Check Southeast
            if(col + 1 < visited[0].length && !visited[row + 1][col + 1]){
                found = tryNext(board, word.substring(1), row + 1, col + 1, visited);
            }
            //Check South
            if(!found && !visited[row + 1][col]){
                found = tryNext(board, word.substring(1), row + 1, col, visited);
            }
            //Check Southwest
            if(!found && col - 1 >= 0 && !visited[row + 1][col - 1]){
                found = tryNext(board, word.substring(1), row + 1, col - 1, visited);
            }
```

```
        }
        //Check West
        if(!found && col - 1 >= 0){
            found = tryNext(board, word.substring(1), row, col - 1, visited);
        }

        return found;
}
```

1 2 3 4 5

Boggle with Electronic Dictionary

Multi Dimensional Arrays | DFS | Search Algorithms | Recursion | Prefix Tree

You're given a 2D **Boggle Board** which contains an m x n matrix of chars - `char[][] board`, and a fast, electronic Dictionary in the form of a Prefix Tree or Trie. Write a method - `boggleSearchWithDict` that searches the Boggle Board for words in the dictionary. Your method should return an **alphabetically sorted** `ArrayList` of words that are present on the board as well as in the dictionary. Words on the board can be constructed with **sequentially adjacent** letters, where adjacent letters are horizontal or vertical neighbors (not diagonal). Also, each letter on the Boggle Board must be used only once. Your program should run in a reasonable amount of time (at max about 50 ms for each test case) and shouldn't time out.

Note: The Trie has two built-in methods that you'll find useful for this problem - `searchWord(String s)` and `searchPrefix(String s)`. These will return `true` if the complete word or prefix are found in the dictionary, respectively.

Example:

```
Input Board :
{
    {A, O, L},
    {D, E, L},
    {G, H, I},
}
Dictionary : [HELLO, HOW, ARE, YOU] (as a Trie)
Output: [HELLO]
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ArrayList<String> boggleSearchWithDict(char[][] board, Trie dictionary){

    TreeSet<String> matchedWords = new TreeSet<String>();

    for(int row = 0; row < board.length; ++row)
    {
        for (int col = 0; col < board[0].length; ++col)
        {
            search(board, row, col, "", dictionary, matchedWords);
        }
    }
```

```
    return new ArrayList<String>(matchedWords);
}

private static void search(char[][] board, int row, int col, String word, Trie dictionary, TreeSet<St
ring> matchedWords)
{
    int maxRows = board.length - 1;
    int maxCol = board[0].length - 1;

    if (row < 0 ||
        row > maxRows ||
        col < 0 ||
        col > maxCol ||
        !dictionary.searchPrefix(word) ||
        board[row][col] == '*')
    {
        return;
    }

    char c = board[row][col];
    word += c;

    if (dictionary.searchWord(word))
    {
        matchedWords.add(word);
    }

    board[row][col] = '*';

    search(board, row + 1, col, word, dictionary, matchedWords);
    search(board, row - 1, col, word, dictionary, matchedWords);
    search(board, row, col + 1, word, dictionary, matchedWords);
    search(board, row, col - 1, word, dictionary, matchedWords);

    board[row][col] = c;
    return;
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ArrayList<String> boggleSearchWithDict(char[][] board, Trie dictionary){
    boolean[][] visited = new boolean[board.length][board[0].length];
    SortedSet<String> reval = new TreeSet<>();
    for(int row = 0; row < board.length; row++){
        for(int col = 0; col < board[0].length; col++){
            reval.addAll(boggleSearchWithDictHelper(board, visited, dictionary, row, col, ""));
        }
```

```
    }

    return new ArrayList<String>(reval);
}

 public SortedSet<String>  boggleSearchWithDictHelper(char[][] board, boolean[][] visited, Trie dicti
onary, int row, int col, String prefix){

     SortedSet<String> reval = new TreeSet<String>();


     if(row < 0 || row > board.length - 1 || col < 0 || col > board[0].length - 1)
        return reval;

     if(visited[row][col])
        return reval;
    visited[row][col] = true;
    prefix = prefix + board[row][col];
    if(dictionary.searchPrefix(prefix)){
        if(dictionary.searchWord(prefix))
            reval.add(prefix);
        reval.addAll(boggleSearchWithDictHelper(board, visited, dictionary, row + 1, col, prefix));
        reval.addAll(boggleSearchWithDictHelper(board, visited, dictionary, row , col + 1, prefix));
        reval.addAll(boggleSearchWithDictHelper(board, visited, dictionary, row - 1, col, prefix));
        reval.addAll(boggleSearchWithDictHelper(board, visited, dictionary, row , col - 1, prefix));

    }
    visited[row][col] = false;
    return reval;


 }
```

1 2 3 4 5

Matrix Max Sum Path with DFS

Multi Dimensional Arrays DFS

Given an **m x n** matrix filled with **non-negative** integers, use depth first search to find the maximum sum along a path from the top-left of the grid to the bottom-right. Return this maximum sum. The direction of movement is limited to right and down.

Example:

```
Input Matrix :

    1 2 3
    4 5 6
    7 8 9

Output  : 1 + 4 + 7 + 8 + 9 = 29
```

Note:

This problem has a more efficient solution based on Dynamic Programming techniques. We'll be exploring those in future problems - so don't fret just yet!

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.



public static int matrixMaxSumDfs(int[][] grid) {

    class TravelNode
    {
        int row;
        int col;
        int nodeSum;
        TravelNode(int r, int c, int sum, int[][] grid)
        {
            row = r;
            col = c;
            nodeSum = sum + grid[r][c];
        }
    }

    int maxPath = 0;
```

```
    Stack<TravelNode> s = new Stack<TravelNode>();
    s.push(new TravelNode(0, 0, 0, grid));
    int lastRow = grid.length-1;
    int lastCol = grid[0].length-1;
    while (!s.isEmpty())
    {
        TravelNode n = s.peek();
        s.pop();

        if (n.row == lastRow && n.col == lastCol)
        {
            maxPath = Math.max(maxPath, n.nodeSum);
        }

        if (n.row < lastRow)
        {
            s.push(new TravelNode(n.row + 1, n.col, n.nodeSum, grid));
        }
        if (n.col < lastCol)
        {
            s.push(new TravelNode(n.row, n.col + 1, n.nodeSum, grid));
        }
    }

    return maxPath;
}
```

Comments

TangoZulu - 15 Aug, 2016

Tried doing this exact same thing recursively and passing back the max sum in a dummy class, and it never worked. Drove me nuts, so I gave up for the iterative solution.

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int matrixMaxSumDfs(int[][] grid) {
    return matrixMaxSumDfsHelper(grid, 0, 0);
}
public static int matrixMaxSumDfsHelper(int[][] grid, int i, int j){
    if(i>grid.length-1 || j>grid[0].length-1)
        return 0;
```

```
    else{
        return grid[i][j] + Math.max(matrixMaxSumDfsHelper(grid,i+1,j),matrixMaxSumDfsHelper(grid,i,
j+1));
    }
}
```

1 2 3 4 5

Subset Summation

Recursion Arrays

Given an array of integers and a target number, determine if it is possible to choose a group of integers from the array, such that the numbers in the group sum to the given target.

Examples:

`groupSum({1,2,3,6,5},10) ==> true`

`groupSum({1,2,3,6,5},18) ==> false`

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static boolean groupSum(int[] arr, int target) {

    int maxCombinations = 1 << arr.length;

    for (int i=0; i<maxCombinations; ++i)
    {
        int sum = computeSum(arr, i);
        if (sum == target)
        {
            return true;
        }
    }

    return false;

}

private static int computeSum(int[] arr, int encodingBits)
{
    int sum = 0;
    int index = 0;
    while (encodingBits != 0)
    {
        if ((encodingBits & 1) == 0)
        {
            sum += arr[index];
        }
```

```
        encodingBits >>= 1;
        index++;
    }
    return sum;
}
```

Top voted solution

```
public static boolean groupSum(int[] arr, int target) {
    return groupSum(arr, target, 0);
}

public static boolean groupSum(int[] arr, int target, int curIndex) {
    if (target == 0)
        return true;
    if (curIndex >= arr.length)
        return false;
    // Try with and without arr[curIndex].
    boolean answer = groupSum(arr, target - arr[curIndex], curIndex + 1);
    answer = answer || groupSum(arr, target, curIndex + 1);
    return answer;
}
```

1 2 3 4 5

Reverse a Linked List in Pairs

Linked Lists Recursion

Given a singly-linked list, reverse the list in pairs.

Example:

`Given 1->2->3->4,`

`reverseInPairs(1) ==> 2->1->4->3`

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ListNode reverseInPairs(ListNode head) {

    ListNode current = head;
    while (current != null && current.next != null)
    {
        ListNode next = current.next;
        int tmp = next.data;
        next.data = current.data;
        current.data = tmp;
        current = current.next.next;
    }

    return head;
}
```

Top voted solution

```
public ListNode reverseInPairs(ListNode head) {
    if (head == null || head.next == null)
        return head;
    ListNode prev = null;
    ListNode a = head;
    ListNode b = head.next;
    // The first swap will change the head.
    a.next = b.next;
    b.next = a;
    head = b;
    // Remaining swaps
```

```
    while (true) {
        // Move all pointers 2 nodes forwards.
        prev = a;
        a = a.next;
        if (a == null || a.next == null)
            return head;
        b = a.next;
        // Swap a and b.
        prev.next = b;
        a.next = b.next;
        b.next = a;
    }
}
```

Comments

Fooble - 29 Nov, 2015

This would have been much easier if I realized I could swap the data instead of the nodes themselves.

AndrÃ© Pinto - 21 Jul, 2016

Swapping the data is kind of cheating anyway xD. It only works with this simplistic Node class (just 1 data field) and when data mutability (instead of collection mutability) is allowed.

1 2 3 4 5

Recursively Merge Two Sorted Linked Lists

Linked Lists Recursion

Given two sorted singly-linked lists, recursively merge them into a new sorted singly-linked list in O(n) runtime. Do not allocate any extra space!
You can assume that both the given lists are already sorted in ascending order.

Examples:

`1->2->3->4 + 5->6->7->8 ==> 1->2->3->4->5->6->7->8`

`1->2->3->4 + 1->2->7->9 ==> 1->1->2->2->3->4->7->9`

`1->2->3->4 + null ==> 1->2->3->4`

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ListNode mergeTwoSortedList(ListNode l1, ListNode l2) {

    if (l1 == null)
    {
        return l2;
    }
    if (l2 == null)
    {
        return l1;
    }

    ListNode node = null;
    if (l1.data < l2.data)
    {
        node = l1;
        l1 = l1.next;
    }
    else
    {
        node = l2;
        l2 = l2.next;
    }

    node.next = mergeTwoSortedList(l1, l2);
    return node;
```

```
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ListNode mergeTwoSortedList(ListNode l1, ListNode l2) {
    // Add your code below this line. Do not modify any other code.
    if(l1==null && l2==null){
        return null;
    }else if(l1==null){
        return l2;
    }else if(l2==null){
        return l1;
    }else{
        if(l1.data<l2.data){
            l1.next=mergeTwoSortedList(l1.next,l2);
            return l1;
        }else{
            l2.next=mergeTwoSortedList(l1,l2.next);
            return l2;
        }
    }

    // Add your code above this line. Do not modify any other code.
}
```

1 2 3 4 5

Merge k Sorted Linked Lists

Linked Lists Queues

Write a method to merge k **Sorted** Linked Lists. Why would you ever want to do that? Well, if you're dealing with a list of over 200 Million `Integers` that needs to be sorted, an efficient approach might involve splitting up the massive list into `k` smaller lists, sorting each list in memory and then combining the sorted lists to re-create the original list, albeit sorted.

```
Example:
Inputs Lists :
LinkedList1: 1->2->13->20
LinkedList2: 1->20->35->40
LinkedList3: 5->6->12->18

Output List:
LinkedList: 1->1->2->5->6->12->13->18->20->20->35->40
```

Note:
`mergeKLists` takes in an `ArrayList` of `ListNode`s - `lists`, where each `ListNode` is the head of a custom Linked List structure.

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.
public static class PQsort implements Comparator<ListNode>
{
 public int compare(ListNode one, ListNode two) {
  return one.data - two.data;
 }
};

public ListNode mergeKLists(ArrayList<ListNode> lists) {



    PriorityQueue<ListNode> q = new PriorityQueue<ListNode>(lists.size(), new PQsort());

    ListNode dummy = new ListNode(0);

    for (int i=0; i<lists.size(); ++i)
    {
        q.offer(lists.get(i));
    }
```

```
    ListNode current = dummy;
    while (!q.isEmpty())
    {
        ListNode n = q.poll();
        current.next = n;
        current = current.next;
        if (n.next != null)
        {
            q.offer(n.next);
        }
    }

    return dummy.next;

}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ListNode mergeKLists(ArrayList<ListNode> lists) {
    ListNode merged = new ListNode(Integer.MIN_VALUE);
    ListNode prev = merged;

    while(true){
        int min = Integer.MAX_VALUE;
        int pos = -1;
        for(int i = 0; i < lists.size(); i++){
            ListNode node = lists.get(i);
            if(node == null){
                continue;
            }

            if(node.data < min){
                min = node.data;
                pos = i;
            }
        }

        if(pos == -1){ // We've consumed all of the lists
            break;
        }

        ListNode curr = lists.get(pos);
        prev.next = curr;
        lists.set(pos, curr.next);
        prev = curr;
    }
```

```
    return merged.next;
}
```

1 2 3 4 5

Making Change

Recursion Strings Arrays

Given an integer array containing the available denominations of coins in descending order, write a method to compute the number of possible ways of representing a monetary `amount` in cents.

For simplicity, assume that there are an infinite number of coins available for each coin denomination in the array.

Examples:

`makeChange({25,10,5,1},10) ==> 4`

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int makeChange(int[] coins, int amount) {
    return doMakeChange(amount, coins, 0);

}

private static int doMakeChange(int amount, int[] coins, int index)
{
    if (index >= coins.length-1) return 1;  // one denom remaining = one way to do Iterable
    int denomAmount = coins[index];

    int ways = 0;
    for (int i=0; i * denomAmount <= amount; ++i)
    {
        int amountRemaining = amount - i * denomAmount;
        ways += doMakeChange(amountRemaining, coins, index + 1);
    }
    return ways;
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int makeChange(int[] coins, int amount) {
```

```
    if (amount<0) return 0;

    int[][] memo = new int[coins.length+1][amount+1];


    for (int i=0;i<=coins.length;i++)
        memo[i][0] = 1;


    for (int i=1;i<=coins.length;i++)
    {
        for (int j=1;j<=amount;j++)
        {
            if (coins[i-1]>j)
                memo[i][j] = memo[i-1][j];
            else
                memo[i][j] = memo[i][j-coins[i-1]] + memo[i-1][j];


        }
    }

    return memo[coins.length][amount];


}
```

Remove Duplicates from a List of Words

Strings Arrays Miscellaneous

Given a `List` of `String`s, write a method `removeDuplicates` that removes duplicate words from the `List` and returns an `ArrayList` of all the unique words. The returned `ArrayList` should be lexically alphabetically.

```
Input: [Hi, Hello, Hey, Hi, Hello, Hey]

Output: [Hello, Hey, Hi]
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static ArrayList<String> removeDuplicates(List<String> input) {
    TreeSet<String> dedup = new TreeSet<String>();
    for (String s : input)
    {
        dedup.add(s);
    }

    return new ArrayList<String>(dedup);

}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static ArrayList<String> removeDuplicates(List<String> input) {
    return new ArrayList<>(new TreeSet<>(input));
}
```

Comments

Tim Harris - 06 Jun, 2016

There's just too much one line awesomeness going on here :) This should be the official answer to this problem.

1 2 3 4 5

Add Two Linked List-ish Numbers

Linked Lists

Given two integers represented as linked-lists, find their sum and return it in the form of linked list.

Note: You can assume that the input integers are non negative and the digits stored in the linked lists are in the reverse order. Take a look at the examples to understand the format.

Example:

```
Input 1: 1->2->3
Input 2: 1->2->3
Output : 2->4->6

Input 1: 9->9
Input 2: 9->8
Output : 8->8->1
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static ListNode sumTwoLinkedLists(ListNode input1, ListNode input2)
{
    return addWithCarry(input1, input2, 0);
}

private static ListNode addWithCarry(ListNode l1, ListNode l2, int carry)
{
    if (l1 == null && l2 == null && carry == 0)
    {
        return null;
    }

    int a = (l1 != null) ? l1.data : 0;
    int b = (l2 != null) ? l2.data : 0;
    int sum = carry + a + b;

    carry = sum / 10;
    ListNode node = new ListNode(sum % 10);
    node.next = addWithCarry(l1 != null ? l1.next : null,
                        l2 != null ? l2.next : null,
```

```
                        carry);

    return node;
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static ListNode sumTwoLinkedLists(ListNode input1, ListNode input2) {

    ListNode put = new ListNode(0);

    ListNode temp1 = input1;
    ListNode temp2 = input2;
    ListNode temp3 = put;

    int carry = 0;

    while(temp1!=null || temp2!=null)
    {
        if(temp1!=null)
        {
            carry+=temp1.data;
            temp1 = temp1.next;
        }

        if(temp2!=null)
        {
            carry+=temp2.data;
            temp2 = temp2.next;
        }

        temp3.next = new ListNode(carry%10);
        carry/=10;
        temp3 = temp3.next;
    }

    if(carry==1)
        temp3.next = new ListNode(1);

    return put.next;


}
```

1 2 3 4 5

Longest Non-Repeating Substring

Strings Hash-Tables Arrays

Given a `String` input, find the length of the longest `substring` that is made up of non-repeating characters. For ex, the longest substrings without repeated characters in â€œBCEFGHBCFGâ€ are â€œCEFGHBâ€ and â€œEFGHBCâ€, with length = 6. In the case of "FFFFF", the longest substring is "F" with a length of 1.

Example:

```
Input : aaabbbabcde
Output: 5
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int longestNRSubstringLen(String input) {

    HashSet<Character> window = new HashSet<Character>();
    int longest = 0;
    int head = 0;
    int tail = 0;

    while (head < input.length())
    {
        Character c = input.charAt(head);
        if (!window.contains(c))
        {
            window.add(c);
            head++;
            longest = Math.max(longest, window.size());
        }
        else
        {
            Character r = input.charAt(tail);
            window.remove(r);
            tail++;
        }
    }

    return longest;
```

```
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int longestNRSubstringLen(String s) {
    if(s==null || s.length()==0)
        return 0;

    HashSet<Character> set = new HashSet<Character>();

    int max=0;

    int i=0;
    int start=0;
    while(i<s.length()){
        char c = s.charAt(i);
        if(!set.contains(c)){
            set.add(c);
        }else{
            max = Math.max(max, set.size());

            while(start<i&&s.charAt(start)!=c){
                set.remove(s.charAt(start));
                start++;
            }
            start++;
        }

        i++;
    }

    max = Math.max(max, set.size());

    return max;




}
```

1 2 3 4 5

Full Tree Decompression

Trees Strings Queues

Given a String that represents a Binary Tree, write a method - `decompressTree` that decompresses that tree (reconstructs the tree) and returns the root `TreeNode`. The compression algorithm included traversing the tree level by level, from the left to the right. The `TreeNode`'s `data` values were appended to the `String`, delimited by commas. Also, `null` `TreeNode`s were denoted by appending an asterisk - `*`. The input `String` denotes the structure of a **Full Binary Tree** - i.e. a tree that is structurally balanced. However, the reconstructed tree may not be a full tree as the String included `*` characters, which represent `null` `TreeNode`s.

Note:
You can assume that if a Binary Tree contains **k** levels, the compressed String will contain 2^k-1 elements - either numbers or *.

Examples:

```
Compressed String : "1,2,3"
Output Tree:
     1
    / \
   2   3

Compressed String : "1,2,3,4,*,6,*"
Output Tree:
     1
    / \
   2   3
  /   /
 4   6

Compressed String : "1,*,2,*,*,*,3"
Output Tree:
     1
      \
       2
        \
         3
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.
```

```
public TreeNode decompressTree(String str){
       if (str == null || str.isEmpty() || str.equals("*")) return null;

    ArrayList<Integer> nums = new ArrayList<Integer>();
    String[] strArr = str.split(",");
    for(String s : strArr)
    {
        nums.add(s.equals("*") ? null : Integer.valueOf(s));
    }

    TreeNode head = new TreeNode(nums.get(0));
    Queue<TreeNode> parents = new LinkedList<TreeNode>();
    parents.offer(head);

    int i=0;
    int size = nums.size();
    while (i < size)
    {
        TreeNode n = parents.poll();
        if (n == null)
        {
            i += 2;
        }
        else
        {
            Integer leftVal = (i+1 < size) ? nums.get(i+1) : null;
            Integer rightVal = (i+2 < size) ? nums.get(i+2) : null;
            TreeNode left = leftVal != null ? new TreeNode(leftVal) : null;
            TreeNode right = rightVal != null ? new TreeNode(rightVal) : null;
            n.left = left;
            n.right = right;
            parents.offer(left);
            parents.offer(right);
            i += 2;
        }
    }

    return head;
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public TreeNode decompressTree(String str){

    String[] arr= str.split(",");
    TreeNode[] nodeArr = new TreeNode[arr.length];
```

```
    for(int i=0; i< arr.length;i++)
    {
        if("*".equals(arr[i]))
         nodeArr[i]=null;

        else
         nodeArr[i]=new TreeNode(Integer.parseInt(arr[i]), null, null);
    }

    for(int i=0;i<arr.length;i++)
    {
        if(2*i+1>=arr.length) break;

        if("*".equals(arr[i])) continue;

       nodeArr[i].left=nodeArr[2*i+1];
       nodeArr[i].right=nodeArr[2*i+2];

       //if(2*i+2>=arr.length) break;
    }
    return nodeArr[0];

}
```

1 2 3 4 5

Iterative Inorder Traversal

Trees Stacks

Given a binary tree, write a method to perform the inorder traversal **iteratively**. Append the data of each node visited to an ArrayList. Return an empty Arraylist in the case of an empty tree.

```
Example:
     1
    / \
   2   3      ==> 4251637
  / \ / \
 4  5 6  7
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ArrayList<Integer> inorderItr(TreeNode root) {

    ArrayList<Integer> list = new ArrayList<Integer>();
    if (root == null) return list;

    Stack<TreeNode> s = new Stack<TreeNode>();

    while (true)
    {
        while (root != null)
        {
            s.push(root);
            root = root.left;
        }

        if (s.isEmpty())
        {
            break;
        }

        root = s.peek();
        s.pop();

        list.add(root.data);
```

```
        root = root.right;
    }

    return list;

}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ArrayList<Integer> inorderItr(TreeNode root) {
    // Add your code below this line. Do not modify any other code.
    Stack<TreeNode> stack = new Stack<TreeNode>();
    ArrayList<Integer> list = new ArrayList<Integer>();
    if(root==null)
        return list;

    TreeNode currentNode = root;
    while(currentNode!=null || !stack.isEmpty())
    {
        if(currentNode!=null)
        {
            stack.push(currentNode);
            currentNode=currentNode.left;
        }
        else
        {
            TreeNode t = stack.pop();
            list.add(t.data);
            currentNode = t.right;
        }
    }
    return list;
    // Add your code above this line. Do not modify any other code.
}
```

Comments

Jean Paul Crescent Mugizi - 14 Apr, 2016

I definitely like this solution better than mine :)

1 2 3 4 5

Find the Maximum Number of Repetitions

Arrays

Given an Array of integers, write a method that will return the integer with the maximum number of repetitions. Your code is expected to run with **O(n)** time complexity and **O(1)** space complexity. The elements in the array are between **0** to **size(array) - 1** and the array will not be empty.

```
f({3,1,2,2,3,4,4,4}) --> 4
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int getMaxRepetition(int[] a) {

    int largest = Integer.MIN_VALUE;
    int largestIndex = 0;
    for (int i=0; i < a.length; ++i)
    {
        a[a[i]%a.length] += a.length;
    }

    for (int i=0; i<a.length; ++i)
    {
        if (a[i] > largest)
        {
            largest = a[i];
            largestIndex = i;
        }
    }

    return largestIndex;
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.
```

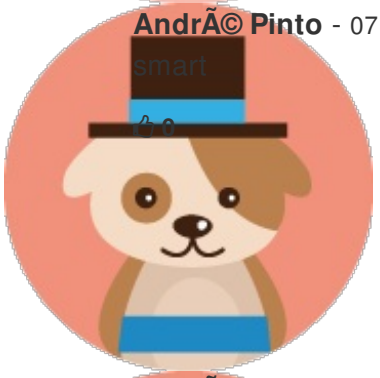
```
public static int getMaxRepetition(int[] a) {
    // Add your code below this line. Do not modify any other code.

    int k = a.length;
    for (int i = 0; i < k; i++) {
        a[a[i] % k] += k;
    }

    int imax = 0;
    for (int i = 1; i < k; i++) {
        if (a[i] > a[imax]) imax = i;
    }

    return imax % k;

    // Add your code above this line. Do not modify any other code.
}
```

Comments

Tim Harris - 03 Jun, 2016

Of all the solutions I like this one the best!

0



AndrÃ© Pinto - 07 Jul, 2016

smart

0

AndrÃ© Pinto - 12 Jul, 2016

You can just return imax though. No need for % k, as imax refers to the index in a, which is guaranteed to be < k.

1

1 2 3 4 5

Find the Nth Node from the end without using extra memory - Linked List

Linked Lists

Given a singly-linked list, implement the method that returns Nth node from the end of the list without using extra memory (constant space complexity).

Examples:

`1->2->3->4->5->6, n=2 ==> 5`

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ListNode findNthNodeFromEnd(ListNode head, int n) {

    ListNode current = head;
    for (int i=0; i<n; i++)
    {
        if (current == null)
        {
            return null;
        }
        current = current.next;
    }

    ListNode nth = head;
    while (current != null)
    {
        current = current.next;
        nth = nth.next;
    }

    return nth;



}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ListNode findNthNodeFromEnd(ListNode head, int n) {
    // Add your code below this line. Do not modify any other code.
    if(head == null || n<1){
        return null;
    }
    ListNode current = head;
    ListNode nGap = head;
    int i;
    for(i=1;i<n&&current.next!=null;i++){
        current = current.next;
    }
    while(current.next!=null){
        current = current.next;
        nGap = nGap.next;
    }
    if(i<n){
        return null;
    }else{
        return nGap;
    }
    // Add your code above this line. Do not modify any other code.
}
```

1 2 3 4 5

Rotate Linear Array

Arrays

Rotate an array to the **left** by `k` positions **without** using extra space. `k` can be greater than the size of the array.

Example:
`rotateLeft({1,2,3,4,5},2) --> {3,4,5,1,2}`

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int[] rotateLeft(int[] arr, int k) {

    int shiftLength = k % arr.length;
    reverse(arr, 0, arr.length-1);

    reverse(arr, 0, arr.length - shiftLength-1);

    reverse(arr, arr.length - shiftLength, arr.length - 1);
    return arr;
}

private static void reverse(int[] arr, int start, int end)
{
    while (start < end)
    {
        int temp = arr[start];
        arr[start] = arr[end];
        arr[end] = temp;
        start++;
        end--;
    }
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int[] rotateLeft(int[] arr, int k) {
```

```
    if (arr == null || arr.length == 0) {
        return null;
    }

    int length = arr.length;

    for (int i = 0; i < k; i++) {
        int temp = arr[0];
        for (int j = 0; j < length-1; j++) {
            arr[j] = arr[j+1];
        }
        arr[length-1] = temp;
    }

    return arr;
}
```

Comments

Enrique - 23 Jul, 2016

This is not O(n), but O(kn).

1

1 2 3 4 5

Number of Half Nodes in a Binary Tree

Trees

Write a function to find the total number of half nodes in a binary tree. A half node is a node which has exactly one child node. If there are no half nodes, return `0`.

```
Example:
       1
      / \
     2   3
    / \ / \
   4  5 6  7
  /
 8

Half nodes count => 1
```

Your solution

```java
// java.util.* has been imported for this problem.
// You don't need any other imports.

public int numberOfHalfNodes(TreeNode root) {
    if (root == null) return 0;

    int left = numberOfHalfNodes(root.left);
    int right = numberOfHalfNodes(root.right);

    if ((root.left != null && root.right == null) ||
        (root.left == null && root.right != null))
        {
            return left + right + 1;
        }
        else
        {
            return left + right;
        }
```

```
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public int numberOfHalfNodes(TreeNode root) {
    // Add your code below this line. Do not modify any other code.
    if (root == null) return 0;

    int children = 0;
    if (root.left != null) children++;
    if (root.right != null) children++;

    return numberOfHalfNodes(root.left) + numberOfHalfNodes(root.right) + (children == 1 ? 1 : 0);


    // Add your code above this line. Do not modify any other code.
}
```

Comments

Tim Harris - 06 Jun, 2016

Nice and concise

0

1 2 3 4 5

Mobile Game Range Module - Merging Ranges

Sorting Algorithms Arrays Numbers

A Range Module is a module that tracks ranges of numbers. Range modules are used extensively when designing scalable online game maps with millions of players. Your task is to write a method - `mergeIntervals` that takes in an `ArrayList` of integer `Interval` s (aka ranges), and returns an `ArrayList` of **sorted** `Interval` s where all overlapping intervals have been merged. The `Interval` class is available by clicking **Use Me**.

Note:

a) [1,3] represents an interval that includes 1, 2 and 3.

b) Intervals should be sorted based on the value of `start`

Examples:

Input: [[1,3], [2,5]], Output: [[1,5]]

Input: [[3,5], [1,2]], Output: [[1,2], [3,5]]

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static ArrayList<Interval> mergeIntervals(ArrayList<Interval> intervalsList) {

    if (intervalsList.size() == 0)
    {
        return intervalsList;
    }
    Collections.sort(intervalsList, new Comparator<Interval>()
    {
        @Override
        public int compare(Interval int1, Interval int2)
        {
            return Integer.compare(int1.start, int2.start);

        }
    });

    ArrayList<Interval> merged = new ArrayList<Interval>();

    Interval prev = intervalsList.get(0);
    for (int i=1; i<intervalsList.size(); ++i)
    {
        Interval cur = intervalsList.get(i);
        if (cur.start <= prev.end)
```

```
        {
            prev = new Interval(prev.start, Math.max(cur.end, prev.end));
        }
        else
        {
            merged.add(prev);
            prev = cur;
        }
    }

    merged.add(prev);

    return merged;
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static ArrayList<Interval> mergeIntervals(ArrayList<Interval> intervalsList) {
    if (intervalsList.size() < 2 || intervalsList == null) {
        return intervalsList;
    }

    ArrayList<Interval> output = new ArrayList<Interval>();

    Collections.sort(intervalsList, new Comparator<Interval>() {
        @Override
        public int compare(Interval o1, Interval o2) {
            return Integer.compare(o1.start, o2.start);
        }
    });

    Interval prev = intervalsList.get(0);

    for (int i = 1; i < intervalsList.size(); i++) {
        Interval cur = intervalsList.get(i);

        if (cur.start <= prev.end) {
            prev = new Interval(prev.start, Math.max(cur.end, prev.end));
        } else {
            output.add(prev);
            prev = cur;
        }
    }

    output.add(prev);
    return output;
}
```

Comments

Tim Harris - 03 Jun, 2016

Yep, that's pretty much the best solution I could think of. Though this problem could be solved with an interval tree .. I guess that'll be in the harder section.

1 2 3 4 5

Is this List a Palindrome?

Linked Lists

Given a singly-linked list, write a method `isListPalindrome` to determine if the list is a palindrome. A palindrome is a sequence that reads the same backward as forward.
Examples:

`1->2->3->2->1 ==> true`

`1->2->2->3 ==> false`

`1 ==> true`

`null ==> true`

Your Notes

```
12321

f=1->3->1
s=1->2->3
st=1->2

1221

f=1->2->null
s=1->2->2
st=1->
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public Boolean isListPalindrome(ListNode head) {

    if (head == null) return true;

    ListNode fast = head;
    ListNode slow = head;

    Stack<ListNode> s = new Stack<ListNode>();
```

```
    while (fast != null && fast.next != null)
    {
        fast = fast.next.next;
        s.push(slow);
        slow = slow.next;
    }


    // if odd, advance slow
    if (fast != null)
    {
        slow = slow.next;
    }

    while (slow != null)
    {
        ListNode n = s.peek();
        s.pop();
        if (slow.data != n.data)
        {
            return false;
        }
        slow = slow.next;
    }

    return true;

}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public Boolean isListPalindrome(ListNode head) {
    ListNode reverse = reverse(copy(head));

    while(reverse != null || head != null){
        if(reverse.data != head.data){
            return false;
        }
        reverse = reverse.next;
        head = head.next;
    }

    return true;
}

public ListNode reverse(ListNode head){
    if(head == null){
```

```
        return null;
    }

    ListNode previous = null;

    while(head != null){
        ListNode nextNodeToFlip = head.next;
        head.next = previous;
        previous = head;
        head = nextNodeToFlip;
    }

    return previous;
}

public ListNode copy(ListNode orig){
    if(orig == null){
        return null;
    }

    ListNode head = new ListNode(orig.data);
    ListNode curr = head;

    while(orig.next != null){
        curr.next = new ListNode(orig.next.data);
        curr = curr.next;
        orig = orig.next;
    }

    return head;
}
```

1 2 3 4 5

Is this Integer a Palindrome?

Arrays Miscellaneous

Write a method that checks if a given integer is a palindrome - without allocating additional heap space

---

Examples:

`-1 ==> false`

`0 ==> true`

`1221 ==> true`

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static Boolean isIntPalindrome(int x) {

    if (x < 0)
    {
        return false;
    }

    int base = 1;
    while (base < (x / 10))
    {
        base *= 10;
    }

    while (x > 0)
    {
        int lmd = x / base;
        int rmd = x % 10;
        if (lmd != rmd)
        {
            return false;
        }

        x %= base;
        x /= 10;
        base /= 100;
    }

    return true;

}
```

Top voted solution

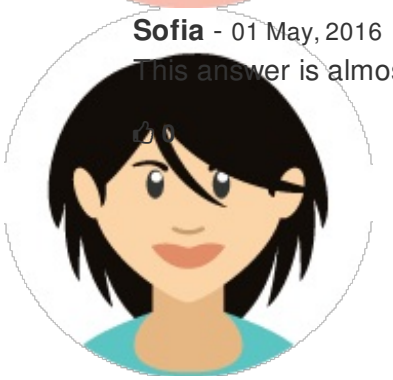
```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static Boolean isIntPalindrome(int x) {
    // Add your code below this line. Do not modify any other code.

    if (x < 0) return false;

    int rev = 0;
    int copy = x;

    while (copy > 0) {
        rev = rev * 10 + copy % 10;
        copy = copy / 10;
    }

    return rev == x;

    // Add your code above this line. Do not modify any other code.
}
```

Comments

Tim Harris - 17 Apr, 2016

Nice answer!

0



Sofia - 01 May, 2016

This answer is almost similar to Firecode's solution as well.

0

TangoZulu - 06 Aug, 2016

Only thing I would call out as an interviewer is this solution does not handle potential overflow of reversing x.

0

1 2 3 4 5

Remove the "Nth from the end" Node from a Singly-Linked List

Linked Lists

Given a singly-linked list, remove its Nth from the end node.

Examples:

```
1->2->3->4->5, n=3 ==> 1->2->4->5
1->2->3->4->5, n=1 ==> 1->2->3->4
1->2->3->4->5, n=5 ==> 2->3->4->5
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ListNode removeNthFromEnd(ListNode head, int n) {

    ListNode current = head;
    for (int i=0; i<n; ++i)
    {
        if (current == null)
        {
            return head;
        }

        current = current.next;
    }

    ListNode tail = head;
    ListNode prev = null;
    while (current != null)
    {
        current = current.next;
        prev = tail;
        tail = tail.next;

    }

    // tail should now be pointing at the nth from the extends

    if (tail == head)
    {
        head = head.next;
    }
```

```
    else
    {
        prev.next = tail.next;
    }
    return head;

}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ListNode removeNthFromEnd(ListNode head, int n) {
    // Add your code below this line. Do not modify any other code.
    if(head==null||n<=0) return head;

    ListNode current = head;
    ListNode previous = head;

    while(current!=null) {
        if(n--<0) {
            previous = previous.next;
        }
        current = current.next;
    }
    if(n>0) return head;
    if(n==0) return head.next;

    previous.next = previous.next.next;
    return head;
    // Add your code above this line. Do not modify any other code.
}
```

1 2 3 4 5

Print a Binary Tree Level by Level

Trees Queues

Given a binary tree, write a method to print the tree level by level. Return your output in an `ArrayList>`.

```
Example:
     1
    / \
   2   3      ==>  [1][2, 3][4, 5, 6, 7]
  / \ / \
 4  5 6  7
```

Note: Each item in the list is an ArrayList of the format [A[], B,[] .....], where A[],B[].... are the nodes at a particular level, stored in an `ArrayList`.

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ArrayList<ArrayList<Integer>> printLevelByLevel(TreeNode root) {

    ArrayList<ArrayList<Integer>> levels = new ArrayList<ArrayList<Integer>>();
    inorderTraverse(root, 0, levels);
    return levels;
}

private static void inorderTraverse(TreeNode root, int level, ArrayList<ArrayList<Integer>> levels)
{
    if (root == null) return;

    if (levels.size() == level)
    {
        ArrayList<Integer> innerList = new ArrayList<Integer>();
        levels.add(innerList);
    }

    levels.get(level).add(root.data);

    inorderTraverse(root.left, level + 1, levels);
    inorderTraverse(root.right, level + 1, levels);
```

```
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ArrayList<ArrayList<Integer>> printLevelByLevel(TreeNode root) {
    ArrayList<ArrayList<Integer>> solution = new ArrayList<>();
    if (root == null) {
        return solution;
    }

    Queue<TreeNode> q = new LinkedList<>();
    q.add(root);

    while (!q.isEmpty()) {
        ArrayList<Integer> level = new ArrayList<>();
        int size = q.size();

        while (size-- > 0) {
            root = q.remove();
            level.add(root.data);

            if (root.left != null) {
                q.add(root.left);
            }
            if (root.right != null) {
                q.add(root.right);
            }
        }
        solution.add(level);
    }
    return solution;
}
```

1 2 3 4 5

Binary Representation

Recursion Bit Manipulation

Write a method to compute the binary representation of a positive integer. The method should return a string with 1s and 0s.

```
computeBinary(6) ==> "110"
computeBinary(5) ==> "101"
```

Note: Use the minimum number of binary digits needed for the representation (Truncate unnecessary trailing 0s).

```
computeBinary(5) ==> "0101" (incorrect)
computeBinary(5) ==> "101" (correct)
```

Your solution

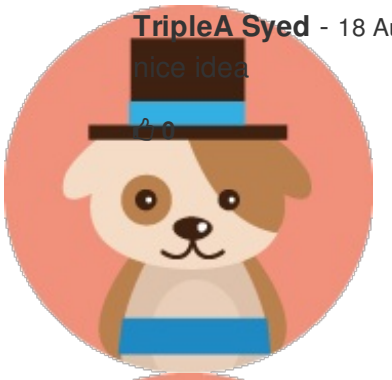
```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static String computeBinary(int val) {
    if (val == 0) return "0";
    StringBuilder sb = new StringBuilder();
    while (val > 0)
    {
        sb.append((val & 1) == 1 ? "1" : "0");
        val >>= 1;
    }
    return sb.reverse().toString();

}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static String computeBinary(int val) {
    // Add your code below this line. Do not modify any other code.
    if (val == 0) return "0";

    StringBuilder binStr = new StringBuilder();
    while (val > 0) {
```

```
        binStr.insert(0, val & 1);
        val = val >> 1;
    }

    return binStr.toString();
    // Add your code above this line. Do not modify any other code.
}
```

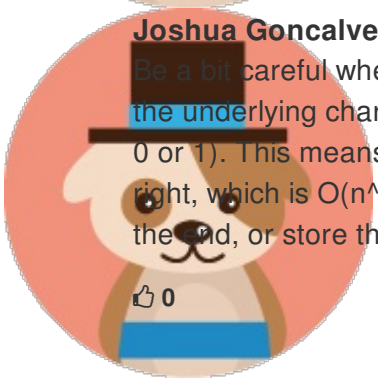
Comments



TripleA Syed - 18 Aug, 2016

nice idea

0



Joshua Goncalves - 02 Sep, 2016

Be a bit careful when inserting to the first position, the code for StringBuilder basically moves everything in the underlying char array forward by the size of the string you're inserting (1 in this case since its either a 0 or 1). This means that every time you insert in the first position, you're moving your entire array to the right, which is O(n^2). Would be better to add everything normally and then reverse the StringBuilder at the end, or store them in an array and construct the string manually.

0

1 2 3 4 5

Count Paths on a Game Board

Dynamic Programming | Multi Dimensional Arrays

You're given a game board that has m x n squares on it, represented by an m x n array. Write a method - `countPaths` that takes in `m` and `n` and returns the number of possible paths from the top left corner to the bottom right corner. Only **down** and **right** directions of movement are permitted.

Note:
Your method should output the result in a reasonable amount of time for large values of m and n. If you're thinking of using DFS, consider the tree depth and branching factor for `m` and `n` > 15!

`m` = number of rows, `n` = number of columns

```
Example:
countPaths(m = 2, n = 2) => 2

as on the following 2x2 Board, the two paths are A->C->D and A->B->D

A B
C D
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public int countPaths(int m, int n){
    int[][] memo = new int[m][n];

    for (int i=0; i<m; ++i)
    {
        memo[i][0] = 1;
    }

    for (int i=0; i<n; ++i)
    {
        memo[0][i] = 1;
    }

    for (int row = 1; row < m; ++row)
    {
        for (int col = 1; col < n; ++col)
        {
            memo[row][col] = memo[row-1][col] + memo[row][col-1];
```

```
            }
        }

        return memo[m-1][n-1];
    }
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public int countPaths(int m, int n){
    if(m <= 0 || n <= 0)
        return 0;
    int min = Math.min(m,n)-1;
    int max = Math.max(m,n)-1;
    double prod = 1;
    for(int i=0; i<min; i++) {
        prod *= (min+max-i)/((double)min-i);
    }
    return (int)Math.round(prod);
}
```

Comments

Enrique - 29 Aug, 2016

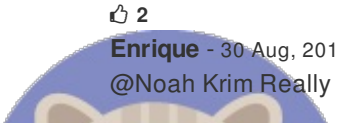
Cool solution - could you explain a little bit about how it works, or point me to any relevant material? Thanks!

1

Noah Krim - 30 Aug, 2016

@Enrique this is just the math behind the problem itself, using the formula for permutations. The idea is that with the `m` x `n` matrix, we can perform any permutation containing `m` down movements and `n` right movements, as you will always move down and to the right the same number of times to reach the end, just in different orderings. If this is the case then the full formula to solve this is (m+n)!/(m!*n!). If you try to purely compute the factorials before the division you will get obscenely large numbers (and hence slow execution), so I tried to reduce the formula to a multiplicative "summation", pre-reducing to speed it up and make it so that there's an equal number of numerators to denominators to work with. This can be done by noticing that if, for example, the matrix was 4 x 3, then the full formula is (4+3)!/(4!*3!) which can be expanded to (7*6*5*4*3*2*1)/((4*3*2*1)*(3*2*1)) so you can reduce the max(4, 3) (`m` and `n`), to make it (7*6*5)/(3*2*1), and that is essentially what's happening in that for loop, i'm doing, in this example, (3+4-i)/(3-i), where `i` goes from [0,3), and multiplying each iteration up you can see how this will produce the same result as the reduced formula above, without the gargantuan numbers of computing factorials, because I think computing even 13! will overflow in 32 bit ints.

2



Enrique - 30 Aug, 2016

@Noah Krim Really clever! Thanks for the explanation, it was very clear.


1

1 2 3 4 5

Binary Tree Serialization

Trees File and I/O

In Computer Science, serialization is the process of converting objects or data structures into a sequence (or series) of characters that can be stored easily in a file / database table or transmitted across a network. Serialized objects need to be de-serialized to create a semantically identical clone of the original object, before being used in programs. You're given the root node of a binary tree - `TreeNode root` in the method `serializeTree`. This method should serialize the binary tree and output a String str, which is then used as an input parameter for the method `restoreTree`. `restoreTree` should create a Binary Tree that is structurally identical to the one you serialized and return the root node of the tree. Your task is to fill in the logic for these 2 methods. Don't worry about passing the serialized String to `restoreTree` - that will be done automatically when you run your code. Feel free to use any notation you prefer when serializing the binary tree. The choice of traversal algorithm is also open - but try and limit the time complexity of both methods to **O(n)**.

Note:
Your serialized String will be used to restore the tree. Be sure to use the same format and notation in `restoreTree` that you use to serialize in `serializeTree`.

```
Example:
     1
    / \
   2   3

Serialization :
Output => "1,2,3"

Restoring Tree from "1,2,3" :
Output ->

     1
    / \
   2   3
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public String serializeTree(TreeNode root)
{
    StringBuilder sb = new StringBuilder();
    serializeHelper(root, sb);
    return sb.toString();
}

private static void serializeHelper(TreeNode root, StringBuilder sb)
{
```

```
    if (root == null)
    {
        sb.append(",null");
        return;
    }

    sb.append("," + Integer.toString(root.data));

    serializeHelper(root.left, sb);
    serializeHelper(root.right, sb);
}

public TreeNode restoreTree(String str)
{
    String[] nodeParse = str.split(",");
    LinkedList<String> nodeList = new LinkedList<String>();
    for (String s : nodeParse)
    {
        if (!s.equals(""))
        {
            nodeList.add(s);
        }
    }

    return restoreHelper(nodeList);
}

private static TreeNode restoreHelper(LinkedList<String> head)
{
    if (head.isEmpty()) return null;
    String strVal = head.remove(0);

    if (strVal.equals("null"))
    {
        return null;
    }

    int data = Integer.parseInt(strVal);
    TreeNode node = new TreeNode(data);
    node.left = restoreHelper(head);
    node.right = restoreHelper(head);
    return node;
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public String serializeTree(TreeNode root){
```

```
    StringBuilder sb = new StringBuilder();
    serializeTreeHelper(root, sb);

    if (sb.length() > 0) {
        sb.deleteCharAt(0);
    }

    return sb.toString();
}

private StringBuilder serializeTreeHelper(TreeNode t, StringBuilder sb) {
    if (t == null) {
        sb.append(",null");
    } else {
        sb.append("," + t.data);
        serializeTreeHelper(t.left, sb);
        serializeTreeHelper(t.right, sb);
    }

    return sb;
}

public TreeNode restoreTree(String str){
    String[] nodesSplit = str.split(",");
    LinkedList<String> nodesList = new LinkedList<>(Arrays.asList(nodesSplit));
    return restoreTreeHelper(nodesList);
}

private TreeNode restoreTreeHelper(LinkedList<String> nodes) {
    String nodeDataStr = nodes.remove();
    if (nodeDataStr.equals("null")) {
        return null;
    }

    TreeNode t = new TreeNode(Integer.valueOf(nodeDataStr));
    t.left = restoreTreeHelper(nodes);
    t.right = restoreTreeHelper(nodes);
    return t;
}
```

1 2 3 4 5

Find the kth Smallest Node in a BST

Trees

Given a binary search tree and an integer k, implement a method to find and return the kth smallest node.

```
Example:
       4
      / \
     2   8
        / \
       5  10

K = 2, Output = 4
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public TreeNode findKthSmallest(TreeNode root, int k) {

    if (root == null) return null;

    int leftSize = 0;
    if (root.left != null)
    {
        leftSize = size(root.left);
    }

    if (leftSize == k-1)
    {
        return root;
    }

    if (leftSize >= k)
    {
        return findKthSmallest(root.left, k);
    }
    else
    {
        return findKthSmallest(root.right, k - leftSize - 1);
    }
```

```
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public TreeNode findKthSmallest(TreeNode root, int k) {
    // Add your code below this line. Do not modify any other code.
    if(root == null){
        return null;
    }

    ArrayList<TreeNode> inorderedList = new ArrayList<TreeNode>();
    Stack<TreeNode> stack = new Stack<TreeNode>();

    boolean done = false;
    while(!done){
        if(root!=null){
            stack.push(root);
            root = root.left;
        }else{
            if(stack.isEmpty()){
                done = true;
            }else{
                TreeNode tmp = stack.pop();
                inorderedList.add(tmp);
                root = tmp.right;
            }
        }
    }

    return k>inorderedList.size()?null:inorderedList.get(k-1);
    // Add your code above this line. Do not modify any other code.
}
```

Comments

Gabriel - 25 Nov, 2015

Bad. O(N) space overhead.

0

1 2 3 4 5

Find the Maximum BST Node

Trees

Given a Binary Search Tree, return the node with the maximum data.

```
Example:
       4
      / \
     2   8
        / \
       5  10

Output ==> 10
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public TreeNode findMax(TreeNode root) {


    if (root == null) return null;

    TreeNode current = root;
    while (current.right != null)
    {
        current = current.right;
    }

    return current;




}
```

Top voted solution

```
public TreeNode findMax(TreeNode root) {
```

```
    if(root==null)return null;
    if(root.right==null)return root;
    return findMax(root.right);
}
```

Comments

Tim Harris - 30 Mar, 2016

You can definitely avoid recursion in this problem - a simple while loop to traverse to the bottom right node will do the trick and yield the most efficient answer!

3

Sachin Nambiar - 30 Mar, 2016

Yes, absolutely! public TreeNode findMax(TreeNode root) { TreeNode temp=root; while(temp!=null){ if(temp.right==null)return temp; temp=temp.right; } return null; }

0

1 2 3 4 5

Iterative BST Validation

Trees BFS

Given the root node of a **Binary Tree**, write a method - `validateBSTItr` to **iteratively** determine if it is a Binary **Search** Tree.

A BST must satisfy the following conditions :

* The left subtree of a node contains nodes with data < its data.
* The right subtree of a node contains nodes data > its data.
* A node's left and right subtrees follow the above two conditions.

Examples:

```
        20
       /   \
     15     30
    /  \
  14   18

  output ==> true

        20
       /   \
     30     15
    /  \
  14   18

 output ==> false
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static boolean validateBSTItr(TreeNode root) {

    class MinMaxNode
    {
        TreeNode node;
        Integer minValue;
        Integer maxValue;

        MinMaxNode(TreeNode n, Integer min, Integer max)
        {
            node = n;
            minValue = min;
```

```
                maxValue = max;
            }
        }

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

        Queue<MinMaxNode> q = new LinkedList<MinMaxNode>();
        q.offer(new MinMaxNode(root, null, null));

        while (!q.isEmpty())
        {
            MinMaxNode n = q.poll();
            TreeNode node = n.node;

            if ((n.minValue != null && node.data < n.minValue) ||
                (n.maxValue != null && node.data > n.maxValue))
            {
                return false;
            }

            if (node.left != null)
            {
                q.offer(new MinMaxNode(node.left, n.minValue, node.data));
            }
            if (node.right != null)
            {
                q.offer(new MinMaxNode(node.right, node.data, n.maxValue));
            }

        }

        return true;
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static boolean validateBSTItr(TreeNode root) {
    if (root == null) {
        return true;
    }

    Stack<TreeNode> s = new Stack<TreeNode>();
    TreeNode curr = root;
    int prevNodeVal = Integer.MIN_VALUE;
```

```
    while (!s.empty() || curr!=null){
        if (curr != null) {
            s.push(curr);

            if (curr.left != null && curr.data < curr.left.data) {
                return false;
            }

            curr = curr.left;
        } else {
            curr = s.pop();

            if (prevNodeVal < curr.data) {
                prevNodeVal = curr.data;
            } else {
                return false;
            }

            curr = curr.right;
        }
    }

    return true;
}
```

1 2 3 4 5

Max Gain

Arrays Numbers

Given an array of integers, write a method - `maxGain` - that returns the **maximum gain**. Maximum Gain is defined as the maximum difference between 2 elements in a list such that the larger element appears **after** the smaller element. If no gain is possible, return 0.

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int maxGain(int[] a) {

    int smallest = a[0];
    int maxGain = 0;

    for (int i=1; i<a.length; ++i)
    {
        maxGain = Math.max(maxGain, a[i] - smallest);
        if (a[i] < smallest)
        {
            smallest = a[i];
        }
    }

    return maxGain;
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int maxGain(int[] a) {

    if(a.length == 1)
        return 0;

    int max = 0, min = a[0];
    for(int i = 1; i < a.length; i++)
    {
        min = Math.min(min, a[i]);
        max = Math.max(max, a[i] - min);
    }
```

```
    return max;
}
```

1 2 3 4 5

Distance of a node from the root

Trees | Search Algorithms | DFS

Given the root of a Binary Tree and an integer that represents the `data` value of a `TreeNode` present in the tree, write a method - `pathLengthFromRoot` that returns the distance between the root and that node. You can assume that the given key exists in the tree. The **distance** is defined as the minimum number of **nodes** that must be traversed to reach the target node.

Example:

```
        1
       / \
      2   3
       \   \
        4   5

pathLengthFromRoot(root,5) => 3
pathLengthFromRoot(root,1) => 1
pathLengthFromRoot(root,3) => 2
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public int pathLengthFromRoot(TreeNode root, int n1) {

    class DepthNode
    {
        TreeNode node;
        int depth;
        DepthNode(TreeNode n, int d)
        {
            node = n;
            depth = d;
        }
    }

    if (root == null) return 0;

    Queue<DepthNode> q = new LinkedList<DepthNode>();
    q.offer(new DepthNode(root, 1));

    while (!q.isEmpty())
    {
        DepthNode n = q.poll();

        if (n.node.data == n1)
        {
```

```
            return n.depth;
        }

        if (n.node.left != null)
        {
            q.offer(new DepthNode(n.node.left, n.depth + 1));
        }
        if (n.node.right != null)
        {
            q.offer(new DepthNode(n.node.right, n.depth + 1));
        }
    }

    return 0;

}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public int pathLengthFromRoot(TreeNode root, int n1) {
    if (root == null) return 0;
    if (root.data == n1) return 1;
    int leftLength = pathLengthFromRoot(root.left, n1);
    int rightLength = pathLengthFromRoot(root.right, n1);
    if (leftLength == 0 && rightLength == 0)
        return 0;
    return leftLength > rightLength ? 1 + leftLength : 1 + rightLength;
}
```

1 2 3 4 5

Print Paths

Multi Dimensional Arrays | DFS | Recursion

You're given a 2D board which contains an m x n matrix of chars - `char[][] board`. Write a method - `printPaths` that prints all possible paths from the top left cell to the bottom right cell. Your method should return an `ArrayList` of Strings, where each String represents a path with characters appended in the order of movement. You're only allowed to move **down** and **right** on the board. The order of `String` insertion in the `ArrayList` does not matter.

Example:

```
Input Board :
{
    {A, X},
    {D, E}
}
Output: ["ADE", "AXE"]
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ArrayList<String> printPaths(char[][] board){
    ArrayList<String> paths = new ArrayList<String>();
    String current = "";

    if (board.length == 0 || board[0].length == 0)
    {
        return paths;
    }
    generateAllPaths(board, 0, 0, current, paths);

    return paths;
}

public void generateAllPaths(char[][] board, int row, int col, String current, ArrayList<String> path
s)
{
    int endRow = board.length-1;
    int endCol = board[0].length-1;

    if (row == endRow && col == endCol)
    {
        current += board[row][col];
        paths.add(current);
        return;
    }
```

```
    current += board[row][col];

    if (row < endRow)
    {
        generateAllPaths(board, row+1, col, current, paths);
    }
    if (col < endCol)
    {
        generateAllPaths(board, row, col+1, current, paths);
    }
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ArrayList<String> printPaths(char[][] board){
    ArrayList<String> out = new ArrayList<>();
    StringBuilder sb = new StringBuilder();
    search(0,0,board,sb,out);
    return out;
}

private void search(int r, int c, char[][] board, StringBuilder sb, ArrayList<String> out){
    int rows = board.length;
    int cols = board[0].length;
    if(r > rows-1 || c > cols-1) return;
    char ch = board[r][c];
    sb.append(ch);
    if(r == rows-1 && c == cols-1){
        out.add(sb.toString());
        sb.deleteCharAt(sb.length()-1);
        return;
    }
    search(r+1,c,board,sb,out);
    search(r,c+1,board,sb,out);
    sb.deleteCharAt(sb.length()-1);
    return;
}
```

1 2 3 4 5

Introduction to Tries

Prefix Tree

A Trie or Prefix Tree an efficient data lookup structure - often used to store large collections of words or dictionaries. With a Trie, search complexities can be reduced to O(k) where k is the key or word length. The autocorrect on your iOS or Android keyboard uses a Trie of the most commonly used words along with fuzzy match algorithms to autocorrect and autosuggest words as you type. You're given a completed `TrieNode` class that represents one node of a `Trie`, and a partially complete `Trie` class. Your task is to complete the `insertWord`, `searchWord` and `searchPrefix` methods on the `Trie` class. Take a look at the examples below to see what each of these do.

```
Example:
trie.inserWord("AB")
trie.inserWord("ABS")
trie.inserWord("ADS")
trie.inserWord("ADSD")
trie.inserWord("ACS")

Internal Trie Structure:


    A
  / | \
 B  C  D
 |  |  |
 S  S  S
       |
       D
```

Note:
In the above example, underlined letters represent word boundaries. Word boundaries are important when differentiating between words and prefixes. For example, `searchPrefix("AC")` should return `true`, but since C is not a word boundary, `searchWord("AC")` should return `false`. The TrieNode class has a `Boolean` - `isLeaf` that is used to denote if the node is a word boundary.

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

class TrieNode {
    Character c;
    Boolean isLeaf = false;
    HashMap<Character, TrieNode> children = new HashMap<>();
    public TrieNode() {}
    public TrieNode(Character c) {
        this.c = c;
    }
}

class Trie {
    private TrieNode root;
```

```
// Implement these methods :
public Trie() {
    this.root = new TrieNode();
}
public void insertWord(String word)
{
     if (word == null || word.isEmpty()) return;

     TrieNode current = this.root;


     for (int i=0; i<word.length(); ++i)
     {
         HashMap<Character, TrieNode> children = current.children;
         char c = word.charAt(i);
         if (children.containsKey(c))
         {
             current = children.get(c);
         }
         else
         {
            TrieNode n = new TrieNode(c);
            children.put(c, n);
            current = n;
         }
     }

     current.isLeaf = true;
}

public Boolean searchWord(String word)
{
   if (word == null || word.length() == 0)
   {
       return false;
   }

   TrieNode current = this.root;

   for (int i=0; i<word.length(); i++)
   {
       HashMap<Character, TrieNode> children = current.children;
       char c = word.charAt(i);
       if (!children.containsKey(c))
       {
           return false;
       }
       current = children.get(c);
       children = current.children;
   }

   return current.isLeaf;
}
```

```
    public Boolean searchPrefix(String word)
    {
        if (word == null || word.length() == 0)
       {
           return false;
       }

       TrieNode current = this.root;

       for (int i=0; i<word.length(); i++)
       {
           HashMap<Character, TrieNode> children = current.children;
           char c = word.charAt(i);
           if (!children.containsKey(c))
           {
               return false;
           }
           current = children.get(c);
           children = current.children;
       }

       return true;
    }


}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

class TrieNode {
    Character c;
    Boolean isLeaf = false;
    HashMap<Character, TrieNode> children = new HashMap<>();
    public TrieNode() {}
    public TrieNode(Character c) {
        this.c = c;
    }
}

class Trie {
    private TrieNode root;

    // Implement these methods :
    public Trie() {
        this.root = new TrieNode();
    }
```

```
    public void insertWord(String word) {
        if (word == null || word.length() < 1) {
            return;
        }
        TrieNode curr = root;
        HashMap<Character, TrieNode> children = curr.children;

        for (int i = 0; i < word.length(); i++) {
            char c = word.charAt(i);

            if (children.containsKey(c)) {
                curr = children.get(c);
            } else {
                TrieNode newNode = new TrieNode(c);
                children.put(c, newNode);
                curr = newNode;
            }

            children = curr.children;

            if (i == word.length()-1) {
                curr.isLeaf = true;
            }
        }

    }
    public Boolean searchWord(String word) {
        TrieNode curr = root;
        HashMap<Character, TrieNode> children = curr.children;

        for (int i = 0; i < word.length(); i++) {
            char c = word.charAt(i);

            if (children.containsKey(c)) {
                curr = children.get(c);
                children = curr.children;
            } else {
                return false;
            }
        }
        return curr.isLeaf;
    }
    public Boolean searchPrefix(String word) {
        TrieNode curr = root;
        HashMap<Character, TrieNode> children = curr.children;

        for (int i = 0; i < word.length(); i++) {
            char c = word.charAt(i);

            if (children.containsKey(c)) {
                curr = children.get(c);
                children = curr.children;
            } else {
                return false;
```

```
            }
        }
        return true;
    }
}
```

1 2 3 4 5

Rotate a Square Image Clockwise

Multi Dimensional Arrays

You are given a square 2D image matrix where each integer represents a pixel. Write a method `rotateSquareImageCW` to rotate the image clockwise - **in-place**. This problem can be broken down into simpler sub-problems you've already solved earlier! Rotating an image clockwise can be achieved by taking the transpose of the image matrix and then flipping it on its vertical axis.

Source: en.wikipedia.org/wiki/Transpose

Example: Input image :

1 0
1 0

Modified to :

1 1
0 0

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static void rotateSquareImageCW(int[][] matrix) {

    int n = matrix.length;
    for (int layer = 0; layer < n / 2; ++layer)
    {
  int first = layer;
  int last = n - 1 - layer;
  for(int i = first; i < last; ++i)
  {
   int offset = i - first;
   int top = matrix[first][i]; // save top

   // left -> top
   matrix[first][i] = matrix[last-offset][first];

   // bottom -> left
   matrix[last-offset][first] = matrix[last][last - offset];

   // right -> bottom
   matrix[last][last - offset] = matrix[i][last];

   // top -> right
   matrix[i][last] = top; // right <- saved top
  }
 }

}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static void rotateSquareImageCW(int[][] matrix) {
    if(matrix == null){
        return;
    }

    flipOnVertical(transpose(matrix));
}

/*
1   2   3
4   5   6
7   8   9

*/


public static int[][] transpose(int[][] matrix){
    int n = matrix.length - 1;

    for(int row = 0; row <= n; row++){
        for(int col = row + 1; col <= n; col++){
            int temp = matrix[row][col];
            matrix[row][col] = matrix[col][row];
            matrix[col][row] = temp;
        }
    }

    return matrix;
}

public static void flipOnVertical(int[][] matrix){
    int n = matrix.length - 1;

    for(int row = 0; row <= n; row++){
        for(int col = 0; col <= n / 2; col++){
            int temp = matrix[row][col];
            matrix[row][col] = matrix[row][n - col];
            matrix[row][n - col] = temp;
        }
    }
}
```

1 2 3 4 5

Stock Market Oracle

Arrays Numbers Puzzles

You've recently acquired market prediction superpowers that let you predict the closing stock price of a Acme Inc. 's stock a month into the future! To get the most out of this superpower, you need to write a method called `maxProfit` that takes in an array of integers representing the close out stock price on a given day. This method should return the maximum profit you can make out of trading Acme Inc.'s stock. There are a few limitations however :

1) You must sell your current holding before buying another - i.e. You may not buy and then buy again. It needs to be a buy - sell - buy - sell ... pattern.

2) You may complete as many transactions as you like. You're using an awesome service like Robinhood, and so there are no transaction costs!

3) If you're enormously unlucky (or karma takes over) and no profit can be made, return 0.

Examples:

`[50,100,20,80,20] => 110`

`[50,100] => 0`

`[50,100,50,100,50] => 100`

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int maxProfit(int[] a) {

    int profit = 0;

    if (a == null || a.length < 2)
    {
        return profit;
    }

    for (int i=1; i<a.length; i++)
    {
        if (a[i] > a[i-1])
        {
            profit += a[i] - a[i-1];
        }
    }

    return profit;
}
```

Top voted solution

```java
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int maxProfit(int[] a) {
    if(a.length < 2)
        return 0;
    int sum = 0;

    for(int i = 1; i < a.length; i++){
        if(a[i-1] < a[i])
            sum += a[i] - a[i-1];
    }

    return sum;


}
```

1 2 3 4 5

Find the Diameter of a BST

Trees

Given a BST, write a function to return its diameter. The diameter of a Binary Tree is defined as the "Number of nodes on the longest path between two leaf nodes".

```
Example:
      20
    /    \
   15     30
  /  \      \  diameter ==> 7
 14  18     35
    /  \    /
   17  19  32
```

Check out **Use Me** section to learn about the helper methods.

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public int diameter(TreeNode root) {

    if (root == null) return 0;

    int leftH = findHeight(root.left);
    int rightH = findHeight(root.right);

    int leftD = diameter(root.left);
    int rightD = diameter(root.right);

    return Math.max(leftH + rightH + 1, Math.max(leftD, rightD));



}
```

Top voted solution

```
public int[] diameterAndHeight(TreeNode root) {
  int heightDiameter[] = { 0, 0 };          // initialize the diameter and height
```

```
  if (root != null) {
   int[] leftResult = diameterAndHeight(root.left);
   int[] rightResult = diameterAndHeight(root.right);
   int height = Math.max(leftResult[1], rightResult[1]) + 1;
   int leftDiameter = leftResult[0];
   int rightDiameter = rightResult[0];
   int rootDiameter = leftResult[1] + rightResult[1] + 1;
   int finalDiameter = Math.max(rootDiameter, Math.max(leftDiameter, rightDiameter));
   heightDiameter[0] = finalDiameter;
   heightDiameter[1] = height;
  }
  return heightDiameter;
}

public int diameter(TreeNode root) {
    int[] result = diameterAndHeight(root);
    return result[0];
}
```

Comments

AndrÃ© Pinto - 10 Jul, 2016

Best solution. O(N) time complexity, instead of O(N^2) in almost every other solution. Nice idea with returning the array.

1

AndrÃ© Pinto - 10 Jul, 2016

heightDiameter should probably be called diameterHeight though.

0

1 2 3 4 5

Find the kth Largest Node in a BST

Trees

Given a Binary Search Tree and an integer k, implement a method to find and return its kth largest node

Example:

```
       4
      / \
     2   8
        / \
       5  10

K = 2, Output = 8
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public TreeNode findKthLargest(TreeNode root, int k) {

    if (root == null) return null;

    int rightSize = 0;
    if (root.right != null)
    {
        rightSize = size(root.right);
    }

    if (rightSize + 1 == k)
    {
        return root;
    }

    if (k < rightSize)
    {
        return findKthLargest(root.right, k);
    }
    else
    {
        return findKthLargest(root.left, k - rightSize - 1);
    }
```

```
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public TreeNode findKthLargest(TreeNode root, int k) {
    // Add your code below this line. Do not modify any other code.

    Stack<TreeNode> stack = new Stack<TreeNode>();
    TreeNode current = root;

    while(!stack.empty() || current != null) {
        if (current != null) {
            stack.push(current);
            current = current.right;
        } else {
            current = stack.pop();
            if (k-- == 1) break;
            current = current.left;
        }
    }

    return current;
    // Add your code above this line. Do not modify any other code.
}
```

Comments

Anonymous - 22 Mar, 2016

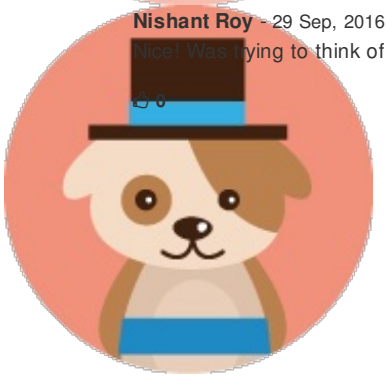
Pretty awesome solution!

15

Tim Harris - 17 Apr, 2016

Clever!

0



Nishant Roy - 29 Sep, 2016

Nice! Was trying to think of how to use a stack.

0

1 2 3 4 5

Merge Two Sorted Arrays

Sorting Algorithms | Arrays | Numbers

The idea behind the classic `Mergesort` algorithm is to divide an array in half, sort each half, and then use a `merge()` method to merge the two halves into a single sorted array.

Implement the `merge()` method that takes in
two **sorted** arrays and returns a third **sorted** array that contains elements of both the input arrays.

You can assume
that the input arrays will always be sorted in ascending order and can have different sizes.

Examples:

```
merge({2,5,7,8,9},{9}) -> {2,5,7,8,9,9}
merge()({7,8},{1,2}) -> {1,2,7,8}
merge()({2},{}) -> {2}
   {} -> [Empty] Array
```

Your solution

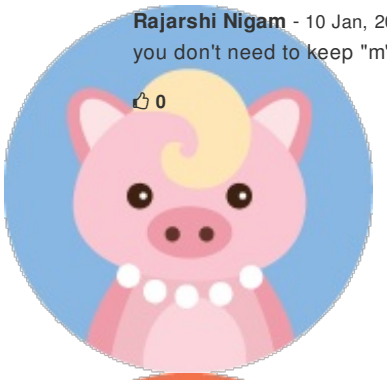
```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int[] merge(int[] arrLeft, int[] arrRight){

    int[] merged = new int[arrLeft.length + arrRight.length];

    int left = arrLeft.length - 1;
    int right = arrRight.length - 1;

    int end = arrLeft.length + arrRight.length - 1;

    while (left >= 0 && right >= 0)
    {
        if (arrLeft[left] > arrRight[right])
        {
            merged[end--] = arrLeft[left--];
        }
        else
        {
            merged[end--] = arrRight[right--];
        }
    }

    while (left >= 0) merged[end--] = arrLeft[left--];
    while (right >= 0) merged[end--] = arrRight[right--];

    return merged;
```

```
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int[] merge(int[] arrLeft, int[] arrRight){
  // Add your code below this line. Do not modify any other code.
    int[] merged = new int[arrLeft.length + arrRight.length];
    int l = 0, r = 0, m = 0;
    while (m < merged.length) {
        if (l < arrLeft.length && arrLeft[l] <= arrRight[r])
            merged[m++] = arrLeft[l++];
        else
            merged[m++] = arrRight[r++];
    }
    return merged;
  // Add your code above this line. Do not modify any other code.
}
```

Comments



Rajarshi Nigam - 10 Jan, 2016

you don't need to keep "m" since it is simply "l + r"

0

Tim Harris - 13 Apr, 2016

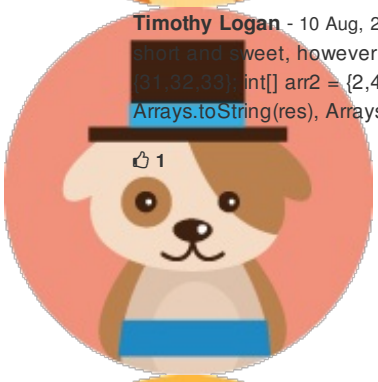
Don't you need to make sure that 'r' never exceeds the index bounds in your else condition? This solution may fail some more test cases

0

Ghassane Adnani - 11 Jun, 2016

Great one! Short and to the point!

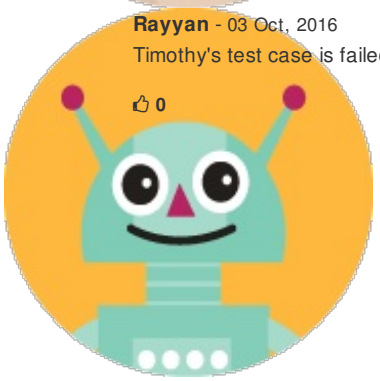
0



Timothy Logan - 10 Aug, 2016

short and sweet, however needs some work --assumes my test is correct. Below test case fails - @Test public void testMerge3() { int[] arr1 = {31,32,33}; int[] arr2 = {2,4,7,12,32}; int [] res = Challenges.merge(arr1, arr2); assertTrue("Expected {2,4,7,12,31,32,32,33} and got " + Arrays.toString(res), Arrays.equals(res, new int[]{2,4,7,12,31,32,32,33})); }

1



Rayyan - 03 Oct, 2016

Timothy's test case is failed by this code..

0

1 2 3 4 5

Find the Sum of all Elements in a Binary Tree

Trees Recursion

Given a binary tree, write a method to find and return the sum of all the elements using recursion. For an empty tree the sum is 0.

```
Example:
        1
       / \
      2   3
     / \ / \
    4  5 6  7
   /
  8
   ==> sum of all nodes = 36
   (1+2+3+4+5+6+7+8)
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public int sum(TreeNode root) {

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

    return root.data + sum(root.left) + sum(root.right);


}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public int sum(TreeNode root) {
```

```
    // Add your code below this line. Do not modify any other code.
    if(root == null){
        return 0;
    }
    return root.data + sum(root.left) + sum(root.right);
    // Add your code above this line. Do not modify any other code.
}
```

1 2 3 4 5

Print all Nodes in the Range a .. b in a given BST

Trees

Given a Binary Search Tree and two numbers - `a` & `b`, return all the nodes in the tree that lie in the range `[a .. b]`. Your method should return an `ArrayList` with the data of the qualifying nodes inserted in **ascending** order.

```
Example:
       4
      / \
     2   8
        / \
       5  10

Range (2,8) ==> [2, 4, 5, 8]

Range includes 2 & 8
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public  ArrayList<Integer> rangeList = new ArrayList<Integer>();
public void printRange(TreeNode root, int a, int b) {

    if (root == null) return;

    printRange(root.left, a, b);

    if (root.data >= a && root.data <= b)
    {
        rangeList.add(root.data);
    }

    printRange(root.right, a, b);
```

```
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public  ArrayList<Integer> rangeList = new ArrayList<Integer>();
public void printRange(TreeNode root, int a, int b) {
   if(root == null) return;
   if(root.data > a) printRange(root.left,a,b);
   if(root.data >=a && root.data <=b) rangeList.add(root.data);
   if(root.data < b) printRange(root.right,a,b);

   }
```

1 2 3 4 5

Snake

Multi Dimensional Arrays

Let's have some fun with 2D Matrices! Write a method `findSpiral` to traverse a 2D matrix of `int`s in a **clockwise spiral order** and append the elements to an output `ArrayList` if `Integer`s.

Example:

```
Input Matrix :

    {1, 2, 3}

    {4, 5, 6}

    {7, 8, 9}

Output ArrayList:[1, 2, 3, 6, 9, 8, 7, 4, 5]
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static ArrayList<Integer> findSpiral(int[][] arr) {

    ArrayList<Integer> list = new ArrayList<Integer>();
    int i = 0;
    int k = 0;
    int l = 0;
    int m = arr.length;
    int n = arr[0].length;
    /*  k - starting row index
        m - ending row index
        l - starting column index
        n - ending column index
        i - iterator
    */

    while (k < m && l < n)
    {
        /* Print the first row from the remaining rows */
        for (i = l; i < n; ++i)
        {
```

```
            list.add(arr[k][i]);
        }
        k++;

        /* Print the last column from the remaining columns */
        for (i = k; i < m; ++i)
        {
            list.add(arr[i][n-1]);
        }
        n--;

        /* Print the last row from the remaining rows */
        if ( k < m)
        {
            for (i = n-1; i >= l; --i)
            {
                list.add(arr[m-1][i]);
            }
            m--;
        }

        /* Print the first column from the remaining columns */
        if (l < n)
        {
            for (i = m-1; i >= k; --i)
            {
                list.add(arr[i][l]);
            }
            l++;
        }
    }

    return list;
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static ArrayList<Integer> findSpiral(int[][] arr) {

    int left=0;
    int right=arr[0].length-1;
    int top=0;
    int bottom=arr.length-1;

    ArrayList<Integer> res = new ArrayList<Integer>();

    while(true)
```

```
    {


    //print left to right
    for(int i=left; i<=right; i++)
      res.add(arr[top][i]);

    top++;
    if(top > bottom || left > right) break;

    //print otp to bottom
    for(int i=top; i<=bottom; i++)
      res.add(arr[i][right]);

    right--;
    if(top > bottom || left > right) break;

    //right to left
    for(int i=right; i>=left; i--)
      res.add(arr[bottom][i]);

    bottom--;
    if(top > bottom || left > right) break;

    //bottom to top
    for(int i=bottom; i>=top; i--)
      res.add(arr[i][left]);

    left++;
    if(top > bottom || left > right) break;
    }
    return res;
}
```

Comments

Shapan Dashore - 08 Sep, 2016
Good solution, easy to understand.
0

1 2 3 4 5

Find the Minimum BST Node

Trees

Given a Binary Search Tree, return the node with the minimum data.

```
Example:
       4
      / \
     2   8
        / \
       5  10

Output ==> 2
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public TreeNode findMin(TreeNode root) {

    TreeNode leftMost = root;
    while (leftMost != null && leftMost.left != null)
    {
        leftMost = leftMost.left;
    }

    return leftMost;
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public TreeNode findMin(TreeNode root) {

    if (root == null)
        return null;

    TreeNode res = root;
```

```
    while (res.left != null) {
        res = res.left;
    }

    return res;
}
```

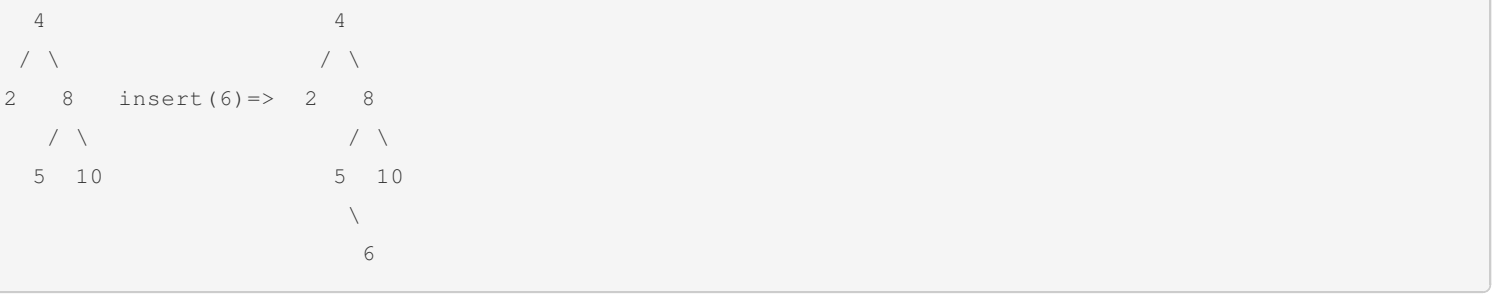
1 2 3 4 5

Jam into a BST

Trees

Implement a method to insert a node into a Binary Search Tree. Return the root of the modified tree.

Example:



Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public TreeNode insert(TreeNode root, int data) {

    TreeNode node = new TreeNode(data);
    if (root == null)
    {
        return node;
    }

    TreeNode current = root;
    while (current != null)
    {
        if (data < current.data)
        {
            if (current.left == null)
            {
                current.left = node;
                break;
            }
            else
            {
                current = current.left;
            }
        }
        else
        {
            if (current.right == null)
```

```
            {
                current.right = node;
                break;
            }
            else
            {
                current = current.right;
            }
        }
    }


    return root;



}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public TreeNode insert(TreeNode root, int data) {
    // Add your code below this line. Do not modify any other code.

    final TreeNode newNode = new TreeNode(data, null, null);
    if(root == null) return newNode;

    TreeNode current = root;
    while(current != null)
    {
        int tempData = current.data;
        if(tempData == data) return root;
        if(tempData > data)
        {
            if(current.left == null)
            {
                current.left = newNode;
                break;
            }
            current = current.left;
        }
        else
        {
            if(current.right == null)
            {
                current.right = newNode;
                break;
            }
```

```
            current = current.right;
        }
    }

    return root;

    // Add your code above this line. Do not modify any other code.
}
```

Comments

Tim Harris - 06 Jun, 2016

Like the fact that you didn't use recursion for this. It's a pretty simple problem without it as well.

0

1 2 3 4 5

Convert a Binary Tree to its Mirror Image

Trees

Write a function to convert a binary tree into its mirror image and return the root node of the mirrored tree.

```
Example:
       1
      / \
     2   3
    / \ / \
   4  5 6  7

       |
       V
  Mirror Form
       1
      / \
     3   2
    / \ / \
   7  6 5  4

Output = Level Order:[1, 3, 2, 7, 6, 5, 4]
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public TreeNode mirror(TreeNode root) {


    if (root == null) return null;
    inorder(root);
    return root;

}

private static void inorder(TreeNode node)
{
    if (node == null) return;

    TreeNode tmp = node.left;
```

```
    node.left = node.right;
    node.right = tmp;

    inorder(node.left);
    inorder(node.right);
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public TreeNode mirror(TreeNode root) {

    if (root == null)
        return null;

    TreeNode left = root.left;
    root.left = mirror(root.right);
    root.right = mirror(left);
    return root;
}
```

1 2 3 4 5

Fill in the Ancestors of the Node in a Binary Tree

Trees Search Algorithms

Given a binary tree's `root` node, an empty `ArrayList` and an integer `nodeData`, write a method that finds a target node - N with `data` = `nodeData` and populates the `ArrayList` with the `data` of the **ancestor nodes** of N - added from the bottom - up.

```
Example:

       1
      / \
     2   3
    / \ / \
   4  5 6  7

Node: 5 ==> [2, 1]
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

//Populate the list of ancestors from bottom to top in the below list.
public ArrayList<Integer> ancestorsList = new ArrayList<Integer>();
public boolean printAncestors(TreeNode root, int nodeData) {

    if (root == null) return false;

    if (printAncestors(root.left, nodeData))
    {
        ancestorsList.add(root.data);
        return true;
    }

    if (printAncestors(root.right, nodeData))
    {
        ancestorsList.add(root.data);
        return true;
    }

    return root.data == nodeData;
```

```
}
```

Top voted solution

```
public ArrayList<Integer> ancestorsList = new ArrayList<Integer>();
public boolean printAncestors(TreeNode root, int nodeData) {
        // Add your code below this line. Do not modify any other code.
        if(root==null)
        return false;
        if(nodeData == root.data)
            return true;
        if(printAncestors(root.left, nodeData) || printAncestors(root.right, nodeData))
        {
            ancestorsList.add(root.data);
            return true;
        }
        return false;
        // Add your code above this line. Do not modify any other code.
    }
```

1 2 3 4 5

Are these Binary Trees Identical?

Trees

Given two binary trees, determine if they are identical. If they are, return true otherwise return false.

```
Example:
       1               1
      / \             / \
     2   3    !=     2   3
    / \ / \         / \ / \
   4  5 6  7       4  5 6  7
  /
 8
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public boolean isIdentical(TreeNode root1, TreeNode root2) {

    if (root1 == null && root2 == null) return true;
    if (root1 != null && root2 == null) return false;
    if (root2 != null && root1 == null) return false;
    if (root1.data != root2.data) return false;

    return isIdentical(root1.left, root2.left) && isIdentical(root1.right, root2.right);


}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public boolean isIdentical(TreeNode root1, TreeNode root2) {
    // Add your code below this line. Do not modify any other code.
    if (root1 == null && root2 == null) { return true; }
```

```
    else if (root1 == null || root2 == null) { return false; }
    boolean dataEqual = root1.data == root2.data;
    boolean leftSubtreesEqual = isIdentical(root1.left, root2.left);
    boolean rightSubtreesEqual = isIdentical(root1.right, root2. right);
    return  dataEqual && leftSubtreesEqual && rightSubtreesEqual;
    // Add your code above this line. Do not modify any other code.
}
```

Comments

Noah Krim - 12 Aug, 2016

This unnecessarily descends down the subtrees once you reach the first instance of two nodes having unequal data values, instead you should see if dataEqual is false and return there, otherwise descend the subtrees and return their logical AND.

0

1 2 3 4 5

Find the Sum of all Elements in a Binary Tree Iteratively

Trees Queues

Given a binary tree, write a method to find and return the sum of all nodes of the tree **iteratively**.

```
Example:
       1
      / \
     2   3
    / \ / \
   4  5 6  7
  /
 8
  ==> sum of all nodes = 36
  (1+2+3+4+5+6+7+8)
```

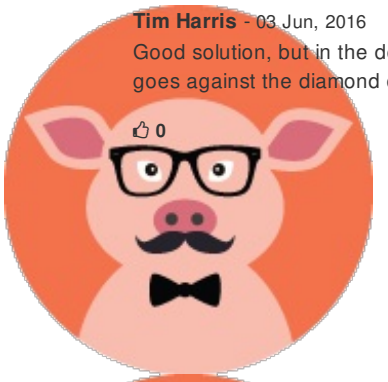
Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public int sumItr(TreeNode root) {

    int sum = 0;

    if (root == null) return 0;
    Queue<TreeNode> q = new LinkedList<TreeNode>();
    q.offer(root);

    while (!q.isEmpty())
    {
        TreeNode n = q.poll();

        sum += n.data;
        if (n.left != null) q.offer(n.left);
        if (n.right != null) q.offer(n.right);
    }

    return sum;

}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public int sumItr(TreeNode root) {
    // Add your code below this line. Do not modify any other code.
    if (root == null) { return 0; }
    int sum = 0;
    Stack<TreeNode> frontier = new Stack<TreeNode>();
    frontier.add(root);
    while (!frontier.isEmpty()) {
        TreeNode currentNode = frontier.pop();
        sum += currentNode.data;
        if (currentNode.left != null) { frontier.push(currentNode.left); }
        if (currentNode.right != null) { frontier.push(currentNode.right); }
    }
    return sum;
    // Add your code above this line. Do not modify any other code.
}
```

Comments



Tim Harris - 03 Jun, 2016

Good solution, but in the declaration of Stack frontier = new Stack(); you could definitely avoid the in the new Stack section. As far as I know that goes against the diamond declaration pattern in Java.

0

Tim Harris - 03 Jun, 2016

I meants <TreeNode> . For some reason the comment isn't displaying properly

0

1 2 3 4 5

Maximum Sum Path

Trees Recursion

Given a binary tree consisting of nodes with **positive integer** values, write a method - `maxSumPath` that returns the **maximum sum** of `data` values obtained by traversing nodes along a path between any 2 nodes of the tree. The path must originate and terminate at 2 different nodes of the tree, and the maximum sum is obtained by summing all the `data` values of the nodes traversed along this path.

```
Example:

     1
    / \
   2   3      => 18
  / \ / \
 4  5 6  7

Path: 5 -> 2 -> 1 -> 3 -> 7
Max Sum = 5+2+1+3+7 = 18
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int maxSumPath(TreeNode root) {

    int[] maxCumulativeValue = new int[1];
    maxSumPathBottomUp(root, maxCumulativeValue);
    return maxCumulativeValue[0];

}

private static int maxSumPathBottomUp(TreeNode root, int[] maxCumulativeValue)
{
    if (root == null) return 0;

    int leftSum = maxSumPathBottomUp(root.left, maxCumulativeValue);
    int rightSum = maxSumPathBottomUp(root.right, maxCumulativeValue);

    int nodeSum = Math.max(root.data + leftSum, root.data + rightSum);

    maxCumulativeValue[0] = Math.max(leftSum + root.data + rightSum, maxCumulativeValue[0]);
    return nodeSum;
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int maxSumPath(TreeNode root) {
    if (root == null) {
        return 0;
    }

    return root.data + maxSumPathChild(root.left) + maxSumPathChild(root.right);
}

private static int maxSumPathChild(TreeNode root) {
    if (root == null) {
        return 0;
    }

    return root.data + Math.max(maxSumPathChild(root.left), maxSumPathChild(root.right));
}
```

Comments

JoÃ£o Lopes - 28 Aug, 2016

I think this will fail if the maxsumpath doesn't include the root node.

0

1 2 3 4 5

Number of Full Nodes in a Binary Tree

Trees Queues

Write a function to iteratively determine the total number of "full nodes" in a binary tree. A full node contains left and right child nodes. If there are no full nodes, return `0`.

```
Example:
       1
      / \
     2   3
    / \ / \
   4  5 6  7
  / \
 8   9

Full nodes count ==> 4
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public int numberOfFullNodes(TreeNode root) {

    if (root == null ) return 0;

    int left = numberOfFullNodes(root.left);
    int right = numberOfFullNodes(root.right);
    int current = (root.left != null && root.right != null) ? 1 : 0;

    return left + right + current;

}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.
```

```java
public int numberOfFullNodes(TreeNode root) {

    if(root == null) return 0;


    int sum = 0;
    if(root.left != null && root.right != null)
        sum = 1;

    return sum + numberOfFullNodes(root.left) + numberOfFullNodes(root.right);
}
```

1 2 3 4 5

The Deepest Node

Trees Queues

Given a binary tree, write a method to find and return its deepest node. Return null for an empty tree.

```
Example:
       1
      / \
     2   3      ==> deepest = 9
    / \ / \
   4  5 6  7
  / \
 8   9
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public TreeNode findDeepest(TreeNode root) {

    if (root == null) return null;
    Queue<TreeNode> q = new LinkedList<TreeNode>();

    TreeNode deepestNode = root;
    q.offer(root);
    while (!q.isEmpty())
    {
        TreeNode n =  q.poll();
        deepestNode = n;
        if (n.left != null) q.offer(n.left);
        if (n.right != null) q.offer(n.right);
    }

    return deepestNode;

}
```

Top voted solution

```java
// java.util.* has been imported for this problem.
// You don't need any other imports.

public TreeNode findDeepest(TreeNode root) {
    // Add your code below this line. Do not modify any other code.
    if(root == null){
        return null;
    }
    Queue<TreeNode> q = new LinkedList<TreeNode>();
    q.add(root);
    while(!q.isEmpty()){
        root = q.remove();
        if(root.left != null){
            q.add(root.left);
        }
        if(root.right != null){
            q.add(root.right);
        }
    }
    return root;

    // Add your code above this line. Do not modify any other code.
}
```

1 2 3 4 5

Generate Combinations of Parentheses

Recursion

Write a method to return all valid combinations of n-pairs of parentheses.

The method should return an `ArrayList` of strings, in which each string represents a valid combination of parentheses.

The order of the strings in the ArrayList does not matter.

Examples:
`combParenthesis(2) ==> {"(())","()()"}`

Note: Valid combination means that parentheses pairs are not left open. ")()(" is not a valid combination.

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static ArrayList<String> combParenthesis(int pairs) {

    ArrayList<String> validCombos = new ArrayList<String>();
    String temp = "";
    doParenCombo(0, 0, temp, validCombos, pairs);
    return validCombos;



}

private static void doParenCombo(int open, int closed, String temp, ArrayList<String> validCombos, in
t pairs)
{
    if (open == pairs && closed == pairs)
    {
        validCombos.add(temp);
    }

    if (open < pairs)
    {
        doParenCombo(open + 1, closed, temp + "(", validCombos, pairs);
    }
    if (closed < open)
    {
        doParenCombo(open, closed + 1, temp + ")", validCombos, pairs);
    }
```

```
}
```

Top voted solution

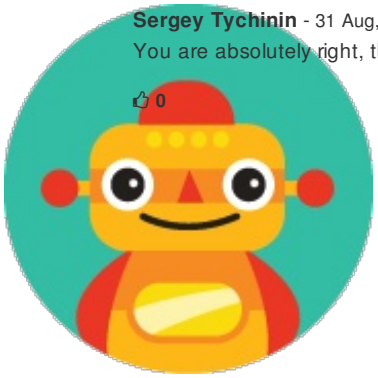
```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static ArrayList<String> combParenthesis(int pairs) {

    ArrayList<String> res = new ArrayList<String>();
    if (pairs <= 0)
        return res;

    if (pairs == 1) {
        res.add("()");
        return res;
    }

    for (String s : combParenthesis(pairs - 1)) {
        res.add("(" + s + ")");
        String s1 = "()" + s;
        String s2 = s + "()";
        res.add(s1);
        if (s1.equals(s2) == false)
            res.add(s2);
    }

    return res;
}
```

Comments

Noah Krim - 30 Aug, 2016

I don't think this would work in cases where pairs > 3, because at 3 you will get the two distinct strings "()(())" and "(())()" in the recursive call, and you aren't handling for when doing "()(())"+"()" and "()"+"(())()" in two separate loop iterations, you'll get a repeated string. Some sort of set like a hashset would fix this.

1



Sergey Tychinin - 31 Aug, 2016

You are absolutely right, there will be repeated strings in the results. There is really not enough test cases on this website.

0

1 2 3 4 5

Levelorder Traversal

Trees Queues

Given a binary tree, write a method to perform a levelorder traversal and return an `ArrayList` of integers containing the data of the visited nodes in the correct order.

```
Example:
     1
    / \
   2   3      ==> 1234567
  / \ / \
 4  5 6  7
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ArrayList<Integer> levelorder(TreeNode root) {

    ArrayList<Integer> levelOrder = new ArrayList<Integer>();
    Queue<TreeNode> q = new LinkedList<TreeNode>();

    if (root == null) return levelOrder;

    q.offer(root);
    while (!q.isEmpty())
    {
        TreeNode n = q.poll();

        levelOrder.add(n.data);

        if (n.left != null) q.offer(n.left);
        if (n.right != null) q.offer(n.right);

    }

    return levelOrder;



}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ArrayList<Integer> levelorder(TreeNode root) {
        ArrayList<Integer> list = new ArrayList<>();
        if(root == null){
            return list;
        }

        Deque<TreeNode> queue = new LinkedList<>();
        queue.offer(root);

        while(queue.peek() != null){
            root = queue.poll();
            list.add(root.data);

            if(root.left != null){
                queue.offer(root.left);
            }

            if(root.right != null){
                queue.offer(root.right);
            }
        }

        return list;

}
```

1 2 3 4 5

Delete the Node at the Specific Position in a Doubly Linked List

Linked Lists | Multi-link Linked Lists

Given a doubly-linked list, write a method to delete the node at a given position (starting from 1 as the head position) and return the modified list's head. Do nothing if the input position is out of range.

Examples:

`1<=>2<=>3<=>4, pos=6 ==> 1<=>2<=>3<=>4`

`1<=>2<=>3<=>4, pos=3 ==> 1<=>2<=>4`

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public DoublyLinkedNode deleteAtPos(DoublyLinkedNode head, int pos)
{
    if (head == null) return null;

    DoublyLinkedNode current = head;
    DoublyLinkedNode prev = null;

    while (current != null && pos > 1)
    {
        pos--;
        prev = current;
        current = current.next;
    }

    if (current == null)
    {
        // trying to delete past tail
        return head;
    }

    if (current == head)
    {
        // special case for head
        current = head.next;
        head.next = null;
        if (current != null)
        {
            current.prev = null;
        }
        head = current;
    }
```

```
    else if (current.next == null)
    {
        // last node
        prev.next = null;
        current.prev = null;
    }
    else
    {
        prev.next = current.next;
        current.next.prev = prev;
        current.prev = null;
        current.next = null;
    }


    return head;

}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public DoublyLinkedNode deleteAtPos(DoublyLinkedNode head, int pos) {
    DoublyLinkedNode cur = head;
    if(cur == null){
        return null;
    }
    int count = 0;
    while(cur != null){
        count++;
        if(count == pos){
            if(cur.next != null){
                cur.next.prev = cur.prev;
            }
            if(cur.prev != null){
                cur.prev.next = cur.next;
            }else{
                head = cur.next;
            }
        }
        cur = cur.next;
    }
    return head;
```

```
}
```

1 2 3 4 5

Minimum Depth of a Tree

Trees BFS

Write a non-recursive method `minTreeDepth` that takes in the root node of a Binary Tree and returns the **minimum depth** of the tree. The minimum depth is defined as the least number of node traversals needed to reach a leaf from the root node. Your method should run in linear **O(n)** time and use at max **O(n)** space.

```
Example:

      1
     / \
    2   3
   / \
  4   5

Output : 2
1 -> 3 is the minimum depth, with a 2 node traversal.
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public class DepthNode
{
    public TreeNode node;
    public int depth;

    public DepthNode(TreeNode n, int d)
    {
        this.node = n;
        this.depth = d;
    }
}

public int minTreeDepth(TreeNode root) {

    if (root == null) return 0;

    Queue<DepthNode> q = new LinkedList<DepthNode>();

    q.offer(new DepthNode(root, 1));

    while(!q.isEmpty())
    {
```

```
        DepthNode dn = q.poll();
        TreeNode n = dn.node;
        int depth = dn.depth;

        if (n.left == null && n.right == null)
        {
            return depth;
        }


        if (n.left != null)
        {
            q.offer(new DepthNode(n.left, depth + 1));
        }

        if (n.right != null)
        {
            q.offer(new DepthNode(n.right, depth + 1));
        }
    }

    return 0;
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public int minTreeDepth(TreeNode root) {
   if (root == null) return 0;

   int depth = 1;

   Queue<TreeNode> currLevel = new LinkedList<>();
   Queue<TreeNode> nextLevel = new LinkedList<>();

   currLevel.add(root);

   while (!currLevel.isEmpty()) {
      TreeNode node = currLevel.poll();
      if (node.left == null && node.right == null) return depth;
      if (node.left != null) nextLevel.add(node.left);
      if (node.right != null) nextLevel.add(node.right);
      if (currLevel.isEmpty()) {
          currLevel = nextLevel;
          nextLevel = new LinkedList<>();
          depth++;
      }
   }
```

```
    return depth;
}
```

1 2 3 4 5

Insert a Node at the Tail of a Circular Linked List

Linked Lists | Multi-link Linked Lists

Given a circular linked list, write a method to insert a node at its tail. Return the list's head.

Examples:

*x = indicates head node

```
Insert 1 ==> *1
Insert 2 ==> 1->2->*1
Insert 3 ==> 1->2->3->*1
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ListNode insertAtTail(ListNode head, int data) {


    ListNode current = head;

    ListNode node = new ListNode(data);

    if (head == null)
    {
        node.next = node;
        return node;
    }

    while (current.next != head)
    {
        current = current.next;
    }

    current.next = node;
    node.next = head;

    return head;

}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ListNode insertAtTail(ListNode head, int data) {
    // Add your code below this line. Do not modify any other code.
    ListNode node = new ListNode(data);
    if(head==null) {
        node.next = node;
        return node;
    }

    ListNode current = head;
    while(current.next!=head) {
        current = current.next;
    }
    current.next = node;
    node.next = head;
    return head;
    // Add your code above this line. Do not modify any other code.
}
```

1 2 3 4 5

Insert a Node at the Specified Position in Doubly Linked List

Linked Lists | Multi-link Linked Lists

In doubly linked list, implement a method to insert a node at specified position and return the list's head. Do nothing if insertion position is outside the bounds of the list.

Examples:

`insertAtPos(1<=>2<=>3,4,2) ==> 1<=>4<=>2<=>3`

`insertAtPos(1,4,3) ==> 1`

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public DoublyLinkedNode insertAtPos(DoublyLinkedNode head, int data, int pos) {

    DoublyLinkedNode node = new DoublyLinkedNode(data);
    if (head == null)
    {
        return (pos == 1) ? node : null;
    }

    DoublyLinkedNode current = head;
    DoublyLinkedNode prev = null;
    int i = 1;
    while (current != null && i != pos)
    {
        prev = current;
        current = current.next;
        i++;
    }

    // check if beyond the end and i != pos
    if (current == null)
    {
        System.out.println("pos: " + pos + ", i:" + i);
        if (Math.abs(pos - i) >= 1)
        {
            // trying to insert beyond the last node
            return head;
        }
        else
        {
            // insert after last node
            prev.next = node;
```

```
            node.prev = prev;
        }
    }
    else
    {

        // insert node before current.
        current.prev = node;
        node.next = current;
        if (head == current)
        {
            // special case for inserting before head (no previous)
            head = node;
        }
        else
        {
            // connect previous
            prev.next = node;
            node.prev = prev;
        }
    }

    return head;

}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public DoublyLinkedNode insertAtPos(DoublyLinkedNode head, int data, int pos) {
    // Add your code below this line. Do not modify any other code.
    if(pos<=1) {
        DoublyLinkedNode node = new DoublyLinkedNode(data);
        if(head!=null) {
            node.next = head;
            node.prev = head.prev;
            head.prev = node;
        }
        head = node;
    } else if(head!=null) {
        head.next = insertAtPos(head.next, data, pos-1);
    }
    return head;
    // Add your code above this line. Do not modify any other code.
}
```

Comments

Bungolio - 30 Aug, 2016

For this test case: Doubly Linked List: 1<=>2<=>3, Node to be inserted: 4, Position: 4 the calls will be as follows call 0 - insertAtPos(1<=>2<=>3, 4, 4) call 1 - insertAtPos(2<=>3, 4, 3) call 2 - insertAtPos(3, 4, 2) call 3 - insertAtPos(null, 4, 1) in call 3, as head is null, it is just setting head = node then returning head, the new nodes prev does not appear to be getting set, or am I missing something?

0

Bungolio - 30 Aug, 2016

To summarize my previous comment, when a new node inserting at the end, it doesn't appear to be setting the previous link back to the previous end node

0

1 2 3 4 5

Insert a Node at the Head in a Doubly Linked List

Linked Lists | Multi-link Linked Lists

Given a doubly linked list, implement a method to insert a node at its head. Return the head of the list.

Examples:

```
Insert 1 ==> 1
Insert 2 ==> 2<=>1
Insert 3 ==> 3<=>2<=>1
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public DoublyLinkedNode insertAtHead(DoublyLinkedNode head, int data) {


    DoublyLinkedNode node = new DoublyLinkedNode(data);

    if (head == null)
    {
        head = node;
        return head;
    }

    node.next = head;
    head.prev = node;
    head = node;

    return head;

}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public DoublyLinkedNode insertAtHead(DoublyLinkedNode head, int data) {
    // Add your code below this line. Do not modify any other code.
    DoublyLinkedNode node = new DoublyLinkedNode(data);
```

```
    node.next = head;
    if(head!=null) head.prev = node;
    return node;
    // Add your code above this line. Do not modify any other code.
}
```

1 2 3 4 5

Remove Duplicate Nodes

Linked Lists Hash-Tables

Given a singly-linked list, remove duplicates in the list and return head of the list. Target a worst case space complexity of O(n).

Examples:

```
1->2->2->4->3->1 ==> 1->2->4->3
1 ==> 1
"" ==> ""
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ListNode removeDuplicates(ListNode head) {

    ListNode dummy = new ListNode(0);
    HashSet<Integer> seen = new HashSet<Integer>();
    ListNode current = dummy;

    while (head != null)
    {
        // need to break the link
        ListNode next = head.next;
        head.next = null;
        if (!seen.contains(head.data))
        {
            seen.add(head.data);
            current.next = head;
            current = current.next;
        }

        head = next;
    }

    return dummy.next;



}
```

Top voted solution

```java
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ListNode removeDuplicates(ListNode head) {
    if (head == null || head.next == null) {
        return head;
    }

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

    ListNode prev = head;
    ListNode iter = head.next;
    map.put(head.data, head);
    while (iter != null) {
        if (map.containsKey(iter.data)) {
            prev.next = iter.next;
        } else {
            map.put(iter.data, iter);
        }
        prev = iter;
        iter = iter.next;
    }

    return head;
}
```

1 2 3 4 5

Find the Nth Node from the End - Linked List

Linked Lists Hash-Tables

Given a singly-linked list, implement the method that returns Nth node from the end of the list. You are allowed to use extra memory for this implementation.

Examples:

`1->2->3->4->5->6, n=2 ==> 5`

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ListNode findNthNodeFromEnd(ListNode head, int n) {

    if (head == null) return null;

    ListNode current = head;
    for (int i=0; i<n; ++i)
    {
        if (current == null)
        {
            return null;
        }
        current = current.next;
    }

    ListNode front = head;

    while (current != null)
    {
        current = current.next;
        front = front.next;
    }

    return front;



}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ListNode findNthNodeFromEnd(ListNode head, int n) {
    // Add your code below this line. Do not modify any other code.
    ListNode previous = head;
    while(head!=null) {
        if(n--<=0) {
            previous = previous.next;
        }
        head = head.next;
    }
    return n<=0 ? previous : null;
    // Add your code above this line. Do not modify any other code.
}
```

Comments

AndrÃ© Pinto - 16 Jul, 2016

smart

1 2 3 4 5

Reverse a Singly Linked List

Linked Lists

Given the head pointer of a singly linked list, implement a method to reverse the list and return the new head.

Example:

```
1->2->3 ==> 3->2->1
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ListNode reverseList(ListNode head) {

    ListNode current = head;
    ListNode previous = null;

    while (current != null)
    {
        ListNode next = current.next;
        current.next = previous;
        previous = current;
        current = next;
    }
    return previous;


}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ListNode reverseList(ListNode head) {
    // Add your code below this line. Do not modify any other code.
    ListNode output = null;
    while (head != null) {
        ListNode next = head.next;
        head.next = output;
```

```
        output = head;
        head = next;
    }
    return output;
    // Add your code above this line. Do not modify any other code.
}
```

1 2 3 4 5

Insert a Node at a specified position in a Linked List

Linked Lists

Given a singly-linked list, implement a method to insert a node at a specific position and return the head of the list.

If the given position is greater than the list size, simply insert the node at the end.

Examples:

`Input List: 1->2->3`

`insertAtPosition(1,4,2) ==> 1->4->2->3`

***position=2** means the 2nd node in the list

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ListNode insertAtPosition(ListNode head, int data, int pos) {

    ListNode current = head;
    ListNode prev = null;
    for (int i=1; i<pos; ++i)
    {
        if (current == null)
        {
            return head;
        }
        prev = current;
        current = current.next;
    }

    ListNode node = new ListNode(data);

    if (prev == null)
    {
        node.next = head;
        head = node;
    }
    else
    {
        prev.next = node;
        node.next = current;
    }
```

```
    return head;

}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ListNode insertAtPosition(ListNode head, int data, int pos) {
    // Add your code below this line. Do not modify any other code.
    if (head == null) {
        return new ListNode(data);
    }
    if (pos == 1) {
        ListNode newNode = new ListNode(data);
        newNode.next = head;
        return newNode;
    }
    head.next = insertAtPosition(head.next, data, pos-1);
    return head;
    // Add your code above this line. Do not modify any other code.
}
```

1 2 3 4 5

Happy Numbers!

Arrays Puzzles

Write a method to determine whether a postive number is **Happy**.

A number is **Happy** (Yes, it is a thing!) if it follows a sequence that ends in 1 after following the steps given below :
Beginning with the number itself, replace it by the sum of the squares of its digits until either the number becomes 1 or loops endlessly in a cycle that does not include 1.

For instance, **19** is a happy number. **Sequence**:
$1^2 + 9^2 = 82$
$8^2 + 2^2 = 68$
$6^2 + 8^2 = 100$
$1^2 + 0^2 + 0^2 = 1$

Example:

```
Input : 19
Output: true
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static boolean isHappyNumber(int n) {

    HashSet<Integer> seen = new HashSet<Integer>();
    seen.add(n);

    while (n != 1)
    {
        int newN = 0;
        while (n != 0)
        {
            int d = n % 10;
            n /= 10;
            newN += d*d;
        }

        n = newN;

        if (seen.contains(n))
        {
            return false;
        }
```

```
        else
        {
            seen.add(n);
        }
    }

    return true;

}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static boolean isHappyNumber(int n) {
    // Write your code below this line.Do not modify any other part of the code.
    TreeSet<Integer> visited = new TreeSet<Integer>();

    while(n!=1 && !visited.contains(n)) {
        visited.add(n);
        n = newHappyNumber(n);
    }
    return n==1;
    // Write your code above this line. Do not modify any other part of the code.
  }

private static int newHappyNumber(int n) {
    int result = 0;
    while(n!=0) {
        int remainder = n%10;
        result += remainder*remainder;
        n /= 10;
    }
    return result;
}
```

1 2 3 4 5

Iteratively, find the Max Element in a Give Binary Tree

Trees Queues

Write a method to find the maximum element in a binary tree iteratively.

```
Examples:
     1
    / \
   2   3      ==>  7
  / \ / \
 4  5 6  7
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public int findMaxItr(TreeNode root) {

    int max = root.data;
    Queue<TreeNode> q = new LinkedList<TreeNode>();
    q.offer(root);

    while (!q.isEmpty())
    {
        TreeNode n = q.poll();
        max = Math.max(max, n.data);
        if (n.left != null) q.offer(n.left);
        if (n.right != null) q.offer(n.right);
    }
    return max;
}
```

Comments

TangoZulu - 08 Sep, 2016

I'd normally just use BFS to solve this. Just wanted to use in order DFS traversal for the heck of it.

0

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public int findMaxItr(TreeNode root) {
    if (root == null) {
        return 0;
    }

    Stack<TreeNode> stack = new Stack<>();

    stack.push(root);

    int max = Integer.MIN_VALUE;
    while (!stack.isEmpty()) {
        TreeNode cur = stack.pop();
        if (max < cur.data) {
            max = cur.data;
        }

        if (cur.right != null) {
            stack.push(cur.right);
        }

        if (cur.left != null) {
            stack.push(cur.left);
        }
    }

    return max;
}
```

1 2 3 4 5

Anagrams

Strings

Write a method `isAnagram` that checks if two **lowercase** input `String`s are anagrams of each other. An anagram of a String is a String that is formed by simply re-arranging its letters, using each letter exactly once. Your algorithm should run in linear **O(n)** time and use constant **O(1)** space.

Examples:

`isAnagram ("abc","cab")` => `true`
`isAnagram ("b","b")` => `true`
`isAnagram ("bd","cb")` => `false`

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static boolean isAnagram(String input1, String input2) {

    if (input1 == null || input2 == null || input1.length() != input2.length()) return false;
    input1 = input1.toLowerCase();
    input2 = input2.toLowerCase();
    int[] count = new int[26];

    for (int i=0; i<input1.length(); ++i)
    {
        count[input1.charAt(i) - 'a']++;
        count[input2.charAt(i) - 'a']--;
    }

    for (int i=0; i<count.length; ++i)
    {
        if (count[i] != 0)
        {
            return false;
        }
    }
    return true;
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.
```

```
public static boolean isAnagram(String input1, String input2) {
    if(input1==null || input2==null || input1.length()!=input2.length()) return false;

    int bitArray = 0;
    for(int i=0;i<input1.length();i++){
        bitArray |= 1<<(input1.charAt(i)-'a');
    }
    for(int i=0;i<input2.length();i++){
        bitArray &= ~(1<<(input2.charAt(i)-'a'));
    }

    return bitArray == 0 ? true : false;
}
```

Comments

Miguel - 12 Jul, 2016

@Enrique @Ashish what do you think about my solution?

0

1 2 3 4 5

Delete the Head Node of a Circular Linked List

Linked Lists | Multi-link Linked Lists

Given a circular linked list, implement a method to delete its head node. Return the list's new head node.

---

*x = indicates head node

`1->2->3->4->*1 ==> 2->3->4->*2`

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ListNode deleteAtHead(ListNode head) {



    if (head == null || head.next == head)
    {
        return null;
    }

    ListNode current = head;

    while (current.next != head)
    {
        current = current.next;
    }

    current.next = head.next;
    head.next = null;
    head = current.next;
    return head;


}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.
```

```
public ListNode deleteAtHead(ListNode head) {
    // Add your code below this line. Do not modify any other code.

    if (head == null || head == head.next) return null;
    ListNode current = head.next;

    while (current.next != head) {
        current = current.next;
    }

    current.next = head.next;
    return head.next;

    // Add your code above this line. Do not modify any other code.
}
```

1 2 3 4 5

Find the Transpose of a Square Matrix

Multi Dimensional Arrays

You are given a square 2D image matrix where each integer represents a pixel. Write a method `transposeMatrix` to transform the matrix into its **Transpose** - **in-place**. The transpose of a matrix is a matrix which is formed by turning all the rows of the source matrix into columns and vice-versa in the following manner :

Source : wikipedia.org/wiki/Transpose

Example: Input image :
1 0
1 0
Modified to :
1 1
0 0

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static void transposeMatrix(int[][] matrix) {
    int temp = 0;

    for (int row = 0; row < matrix.length; ++row)
    {
        for (int col = row+1; col < matrix[0].length; ++col)
        {
            temp = matrix[row][col];
            matrix[row][col] = matrix[col][row];
            matrix[col][row] = temp;
        }
    }
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static void transposeMatrix(int[][] matrix) {
    for(int i=0; i<matrix.length;i++){
        for(int j=i+1;j<matrix[i].length;j++){
            int temp = matrix[i][j];
            matrix[i][j] = matrix[j][i];
            matrix[j][i] = temp;
```

```
            }
        }
}
```

1 2 3 4 5

Insert Stars

Recursion Strings

Given a string, recursively compute a new string where the identical adjacent characters in the original string are separated by a "*".

Examples:

`insertPairStar("cac") ==> "cac"`

`insertPairStar("cc") ==> "c*c"`

Your solution

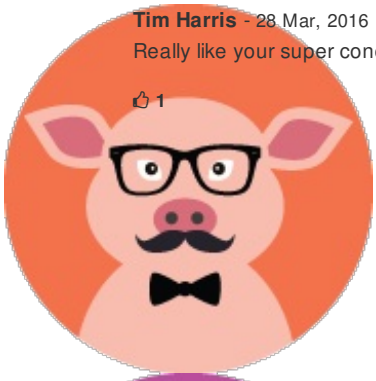
```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static String insertPairStar(String s) {

    if (s == null || s.length() < 2)
    {
        return s;
    }

    if (s.charAt(0) == s.charAt(1))
    {
        return s.substring(0, 1) + "*" + insertPairStar(s.substring(1));
    }
    else
    {
        return s.substring(0, 1) + insertPairStar(s.substring(1));
    }



}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static String insertPairStar(String s) {
    // Add your code below this line. Do not modify any other code.
    if(s==null || s.length()<=1) return s;
```

```
    return s.charAt(0) +
            (s.charAt(0)==s.charAt(1) ? "*" : "") +
            insertPairStar(s.substring(1));
    // Add your code above this line. Do not modify any other code.
}
```

Comments



Tim Harris - 28 Mar, 2016

Really like your super concise return statement Lukasz!

1

Yash - 13 Jul, 2016

Smart approach.

1

1 2 3 4 5

Count the Leaves!

Trees Queues

Write a function to find the total number of leaf nodes in a binary tree. A node is described as a leaf node if it doesn't have any children. If there are no leaf nodes, return `0`.

```
Example:
       1
      / \
     2   3
    / \ / \
   4  5 6  7
  / \
 8   9
==> no. of leaves = 5
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public int numberOfLeaves(TreeNode root) {

    if (root == null) return 0;

    int leftLeaf = numberOfLeaves(root.left);
    int rightLeaf = numberOfLeaves(root.right);

    if (root.left == null && root.right == null)
    {
        return leftLeaf + rightLeaf + 1;
    }
    else
    {
        return leftLeaf + rightLeaf;
    }


}
```

Top voted solution

```java
// java.util.* has been imported for this problem.
// You don't need any other imports.

public int numberOfLeaves(TreeNode root) {
    // Add your code below this line. Do not modify any other code.

    // Add your code above this line. Do not modify any other code.

    if (root==null){
        return 0;
    }
    if(root.left==null && root.right ==null){
        return 1;
    }

    return numberOfLeaves(root.left)+numberOfLeaves(root.right);
}
```

1 2 3 4 5

POW!

Numbers Miscellaneous Recursion

Write a method - pow(x,n) that returns the value of x raised to the power of n (x^n). n can be negative!

Examples:

```
pow(2,3) ==> 8.0
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static double pow(double x, int n) {

    if (n == 0) return 1.0;
    if (n == 1) return x;
    if (x == 1) return 1.0;
    if (x == 0) return 0.0;

    if (n < 0)
    {
        x = 1 / x;
        if (n == Integer.MIN_VALUE)
        {
            n = Integer.MAX_VALUE;
        }
        else
        {
            n = -n;
        }
    }

    if ((n % 2) == 0)
    {
        return pow(x*x, n/2);
    }
    else
    {
        return x*pow(x*x, n/2);
    }

}
```

Comments

TangoZulu - 01 Sep, 2016

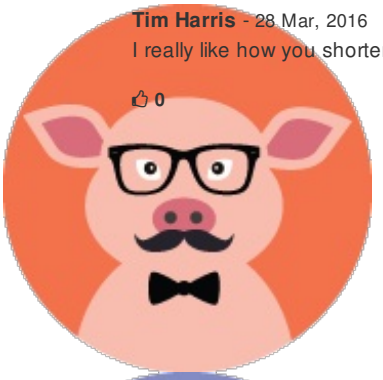
The check for -INT_MAX here is needed to handle that special edge cause, otherwise you get the overflow with negating. Otherwise, this is a O(log n) solution that's similar to the sample answer.

0

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static double pow(double x, int n) {
    // Add your code below this line. Do not modify any other code.

    if (n == 0) return 1;
    if (n == 1) return x;

    return n < 0
        ? 1 / (x * pow(x, -n - 1))
        : x * pow(x, n - 1);

    // Add your code above this line. Do not modify any other code.
}
```

Comments



Tim Harris - 28 Mar, 2016

I really like how you shortened the return statement. The only comment I have though is that you don't need the if(n==1) condition.

0

Enrique - 28 Jul, 2016

Nice solution, but it's linear instead of logarithmic.

2

1 2 3 4 5

Recursive Preorder Traversal

Trees Queues

Given a binary tree, write a method to recursively traverse the tree in the preorder manner. Mark a node as visited by adding its `data` to the list - `Arraylist <Integer> preorderedList`.

```
Example:
     1
    / \
   2   3      ==> 1245367
  / \ / \
 4  5 6  7
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

//Populated the elements of the tree  in the below list in preorder format
ArrayList<Integer> preorderedList = new ArrayList<Integer>();
public void preorder(TreeNode root) {

    if (root == null) return;

    preorderedList.add(root.data);
    preorder(root.left);
    preorder(root.right);
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

//Populated the elements of the tree  in the below list in preorder format
ArrayList<Integer> preorderedList = new ArrayList<Integer>();
public void preorder(TreeNode root) {
    // Add your code below this line. Do not modify any other code.
    if (root == null)
```

```
        return;

    preorderedList.add(root.data);
    preorder(root.left);
    preorder(root.right);
    // Add your code above this line. Do not modify any other code.
}
```

1 2 3 4 5

Height of a Binary Tree

Trees Recursion

Given a binary tree, write a method to find its height `recursively`. An empty tree has a height of 0.

```
Example:
     1
    / \
   2   3      ==> height=3
  / \ / \
 4  5 6  7
```

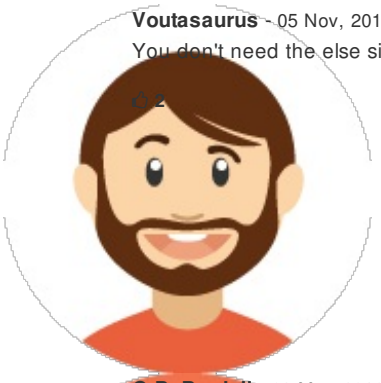
Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public int findHeight(TreeNode root) {

    if (root == null) return 0;
    return 1 + Math.max(findHeight(root.left), findHeight(root.right));

}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public int findHeight(TreeNode root) {
    // Add your code below this line. Do not modify any other code.
    if (root == null) { return 0; }
    else { return Math.max(findHeight(root.left), findHeight(root.right)) + 1; }
    // Add your code above this line. Do not modify any other code.
}
```

Comments



Voutasaurus - 05 Nov, 2015

You don't need the else since you've already returned in the body of the if.

2

G.P. Burdell - 02 Mar, 2016

Great answer!

0

Tim Harris - 28 Mar, 2016

@Voutasaurus Even though the else is not required it provides better readability, and in certain cases can lead to smarter optimizations during the compilation phase.

0

Pratibha Rawat - 06 Jun, 2016

I have a question, ideally an empty tree has a height of -1. Do you guys think we should return -1 or 0?

0

Tim Harris - 06 Jun, 2016

@Pratibha, in this case the problem mentions that "An empty tree has a height of 0". Since the return type is a primitive int, it makes sense. If it was object - Integer or TreeNode, it should be null. I haven't seen cases where empty trees have height = -1. Maybe in C ?

0

Pratibha Rawat - 06 Jun, 2016

@TimHarris yes you are right. C++ and C. My bad.

0

Pratibha Rawat - 06 Jun, 2016

@TimHarris yes you are right. C++ and C. My bad.

0

1 2 3 4 5

Selection Sort

Sorting Algorithms | Arrays | Numbers

Selection sort offers improved performance over bubble sort, especially for arrays with a large number of elements. Where bubble sort accumulated the largest elements towards the end of the array, selection sort accumulates the smallest elements at the beginning of the array.

Write a method that uses the selection sort algorithm to sort an input array of integers. See the hints and click the red colored links for additonal details on the algorithm.

```
Examples:

selectionSortArray({1,5,2}) -> {1,2,5}

selectionSortArray({11}) -> {11}

selectionSortArray({}) -> {}
    {} -> [Empty] Array
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int[] selectionSortArray(int[] arr){

    int i, j, minIndex, tmp;
    int n = arr.length;
    for (i = 0; i < n - 1; i++)
    {
        minIndex = i;
        for (j = i + 1; j < n; j++)
        {
            if (arr[j] < arr[minIndex])
            {
                minIndex = j;
            }
        }
        if (minIndex != i)
        {
            tmp = arr[i];
            arr[i] = arr[minIndex];
            arr[minIndex] = tmp;
        }
    }

    return arr;
```

```
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int[] selectionSortArray(int[] arr){
    // Add your code below this line. Do not modify any other code.
    for(int i=0;i<arr.length-1;i++){
        int minElem = i;
        for(int j=i+1;j<arr.length;j++){
            if(arr[minElem] > arr[j]){
                minElem = j;
            }
        }
        int tmp = arr[i];
        arr[i] = arr[minElem];
        arr[minElem] = tmp;
    }
    // Add your code above this line. Do not modify any other code.
    return arr;
}
```

1 2 3 4 5

Couple Sum

Arrays Hash-Tables

Given an `array` of integers, find two numbers such that they sum up to a specific target.
The method `coupleSum` should return the indices of the two numbers in the array, where index1 must be less than index2.
Please note that the indices are **not** zero based, and you can assume that each input has **exactly one** solution. Target linear runtime and space complexity.

Example:

```
Input Array : {2, 3, 4, 7}
Target      : 7
Output      : {2, 3}
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int[] coupleSum(int[] numbers, int target)
{

    HashMap<Integer, Integer> sums = new HashMap<Integer, Integer>();
    int[] sum = new int[2];

    for (int i=0; i<numbers.length; ++i)
    {
        int n = numbers[i];
        int diff = target - n;
        if (sums.containsKey(diff))
        {
            int j = sums.get(diff);
            sum[0] = Math.min(i+1, j+1);
            sum[1] = Math.max(i+1, j+1);
        }
        else
        {
            sums.put(n, i);
        }
    }

    return sum;
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int[] coupleSum(int[] numbers, int target) {
    HashMap<Integer,Integer> map = new HashMap<Integer,Integer>();
    for(int i=0 ; i<numbers.length ; i++){
        map.put(target - numbers[i] , i);
    }
    for(int i=0 ; i<numbers.length ; i++){
        if(map.containsKey(numbers[i])){
            return new int[]{i+1 , map.get(numbers[i])+1};
        }
    }

    return new int[]{};
}
```

1 2 3 4 5

Find the size of the Binary Tree

Trees

Given a binary tree, write a method to return its size. The size of a tree is the number of nodes it contains.

```
Examples:
     1
    / \
   2   3      ==>  7
  / \ / \
 4  5 6  7
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public int size(TreeNode root) {

    if (root == null) return 0;
    return 1 + size(root.left) + size(root.right);




}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public int size(TreeNode root) {
    // Add your code below this line. Do not modify any other code.

    if (root == null) return 0;
    return 1 + size(root.left) + size(root.right);
```

```
    // Add your code above this line. Do not modify any other code.
}
```

1 2 3 4 5

Even or Odd?

Linked Lists

Given a singly-linked list, check whether its length is even or odd in a single pass. An Empty list has 0 nodes which makes the number of nodes in it even.

Examples:
`1->2->3->4 == true` `1->2->3->4->5 == false`

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public Boolean isListEven(ListNode head) {

    ListNode current = head;
    int count = 0;
    while (current != null)
    {
        current = current.next;
        count++;
    }

    return (count % 2) == 0;


}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public Boolean isListEven(ListNode head) {
    // Add your code below this line. Do not modify any other code.
    boolean even = true;
    while(head != null) {
        even = !even;
        head = head.next;
    }
```

```
    return even;
    // Add your code above this line. Do not modify any other code.
}
```

1 2 3 4 5

Reverse an Integer

Arrays Miscellaneous Numbers

Implement a method that reverses an integer - without using additional heap space

---

Examples:

`-123 ==> -321`

`123 ==> 321`

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int reverseInt(int x) {

    int sign = 1;
    if (x < 0)
    {
        sign = -1;
        x = -x;
    }

    int rev = 0;
    while (x > 0)
    {
        int lmd = x % 10;
        x /= 10;
        rev *= 10;
        rev += lmd;
    }

    return rev * sign;


}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static int reverseInt(int x) {
```

```
    // Add your code below this line. Do not modify any other code.

    int rev = 0;

    while (x != 0) {
        rev = rev * 10 + x % 10;
        x = x / 10;
    }

    return rev;

    // Add your code above this line. Do not modify any other code.
}
```

1 2 3 4 5

Power of 2

Bit Manipulation Numbers

Write a method - `isPowOfTwo` to test whether or not a given positive integer is a power of 2. Your method should run in constant **O(1)** time and use **O(1)** space.

Examples:

```
isPowOfTwo(5) ==> false
isPowOfTwo(8) ==> true
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static boolean isPowOfTwo(int num) {
    return ((num - 1) & num) == 0;
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static boolean isPowOfTwo(int num) {
    int mask = 1;
    do{
        if((num^mask) == 0) return true;
        mask = mask<<1;
    }while(mask!=(1<<31));
    return false;
}
```

1 2 3 4 5

Delete a Circular-Linked List's Tail Node

Linked Lists | Multi-link Linked Lists

Given a circular-linked list, write a function to delete its tail node and return the modified list's head.

---

*x = indicates head node

`1->2->3->4->*1 ==> 1->2->3->*1`

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ListNode deleteAtTail(ListNode head) {

    ListNode current = head;
    ListNode previous = null;

    while (current.next != head)
    {
        previous = current;
        current = current.next;
    }

    previous.next = head;
    return head;

}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public ListNode deleteAtTail(ListNode head) {
    if(head == null || head.next == head)
        return null;
    ListNode p = head;
    ListNode q = head.next;

    while(q.next != head){
```

```
            p = p.next;
            q = q.next;
        }
        p.next = p.next.next;
        q.next = null;
        return p.next;

}
```

1 2 3 4 5

Find the Max Element in a Binary Tree Recursively

Trees

Given a binary tree, write a recursive method to return the maximum element.

```
Examples:
     1
    / \
   2   3      ==>  7
  / \ / \
 4  5 6  7
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public int findMax(TreeNode root) {

    if (root == null)
    {
        return Integer.MIN_VALUE;
    }

    int leftMax = findMax(root.left);
    int rightMax = findMax(root.right);
    return Math.max(root.data, Math.max(leftMax, rightMax));
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public int findMax(TreeNode root) {
    // Add your code below this line. Do not modify any other code.
    if (root == null) return Integer.MIN_VALUE;

    return Math.max(root.data, Math.max(findMax(root.left), findMax(root.right)));
```

```
    // Add your code above this line. Do not modify any other code.
}
```

1 2 3 4 5

String Compression

Strings

Compress a sorted String by replacing instances of repeated characters with the character followed by the count of the character.

```
compressString("aaabbbbbcccc") --> a3b5c4
compressString("aabbbbccc") --> a2b4c3
compressString("abc") --> abc
```

Note: This kind of compression will only be effective when the count of consecutive identical characters is greater than 1.

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static String compressString(String text) {

    if (text == null || text.length() < 3)
    {
        return text;
    }

    StringBuilder sb = new StringBuilder();

    int current = 0;
    int len = text.length();
    int count = 1;

    while (current < len)
    {
        count = 1;
        int next = current + 1;
        while (next < len && text.charAt(next) == text.charAt(current))
        {
            next++;
            count++;
        }


        sb.append(text.charAt(current));
        if (count > 1)
        {
            sb.append(count);
            count = 1;
        }
```

```
        current = next;

    }

    if (count > 1)
    {
        sb.append(text.charAt(current));
        sb.append(count);

    }

    return sb.toString();




}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static String compresString(String text) {
    // Add your code below this line. Do not modify any other code.

    if (text == null || text.length() < 2) return text;

    String head = text.substring(0, 1);
    String tail = text.substring(1);

    int count = 1;

    while (tail.length() > 0 && head.equals(tail.substring(0,1))) {
        count++;
        tail = tail.substring(1);
    }

    return count > 1
        ? head + Integer.toString(count) + compresString(tail)
        : head + compresString(tail);

    // Add your code above this line. Do not modify any other code.
}
```

Comments

Pratibha Rawat - 03 Jun, 2016

Wow that's an impressive solution. I dived into HashMap for this.

0

Shapan Dashore - 10 Jul, 2016

a very clean solution...

0

TangoZulu - 04 Oct, 2016

I like the elegance of this solution. But too much string creation and stack overhead is going to impact performance. Generally, I think interviewers will be looking for a 2 pointer implementation and O(n) runtime and O(1) space.

0

1 2 3 4 5

Permutations!

Strings

Implement a method that checks if two strings are permutations of each other.

```
permutation("CAT","ACT") --> true
permutation("hello","aloha") --> false
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static boolean permutation(String str1, String str2) {

    if (str1 == null || str2 == null || str1.length() != str2.length())
    {
        return false;
    }

    int[] map = new int[256];

    for (int i=0; i < str1.length(); ++i)
    {
        map[str1.charAt(i)]++;
        map[str2.charAt(i)]--;
    }

    for (int c : map)
    {
        if (c != 0)
        {
            return false;
        }
    }

    return true;

}
```

Top voted solution

```
// java.util.* has been imported for this problem.
```

```
// You don't need any other imports.

public static boolean permutation(String str1, String str2) {
    if(str1.length() != str2.length())
        return false;

    HashMap<Character,Integer> map = new HashMap<Character,Integer>();
    for(int i=0 ; i<str1.length() ; i++){
        char c = str1.charAt(i);
        if( map.containsKey(c)){
            map.put(c, map.get(c)+1);
        }else{
            map.put(c, 1);
        }
    }
    for(int i=0 ; i<str2.length() ; i++){
        char c = str2.charAt(i);
        if(map.containsKey(c)){
            if(map.get(c) >1)
                map.put(c, map.get(c)-1);
            else
                map.remove(c);
        }else
            return false;
    }


    return true;
}
```

1 2 3 4 5

BST Validation

Trees Recursion

Given the root node of a **Binary Tree**, determine if it is a Binary **Search** Tree.

Examples:

```
          20
        /   \
      15     30
     /  \
    14   18

    output ==> true

          20
        /   \
      30     15
     /  \
    14   18

   output ==> false
```

Your solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static boolean validateBST(TreeNode root) {

    return doValidateBST(root, null, null);

}

private static boolean doValidateBST(TreeNode root, Integer min, Integer max)
{
    if (root == null)
    {
        return true;
    }

    if (min != null && root.data < min)
    {
        return false;
```

```
    }

    if (max != null && root.data > max)
    {
        return false;
    }

    return doValidateBST(root.left, min, root.data) &&
           doValidateBST(root.right, root.data, max);
}
```

Top voted solution

```
// java.util.* has been imported for this problem.
// You don't need any other imports.

public static boolean validateBST(TreeNode root) {
    return validateBST(root, Integer.MIN_VALUE, Integer.MAX_VALUE);
}

public static boolean validateBST(TreeNode root, int min, int max) {
    if (root == null) {
        return true;
    }

    if (root.data <= min || root.data >= max) {
        return false;
    }

    return validateBST(root.left, min, root.data) && validateBST(root.right, root.data, max);
}
```