ds-and-algos

Data Structures And Algorithms

View on GitHub

Data Structures and Algorithms

Linked List Problems

Problem #	Problem Title
1	GitHub Flavored Markdown
Content Cell	Content Cell

Binary Tree Problems

Problem #	Problem Title
1	Insert Into Binary Tree

Binary Search Tree Problems

Problem #	Problem Title
1	GitHub Flavored Markdown
Content Cell	Content Cell

Dynamic Programming Problems

Problem #	Problem Title
1	GitHub Flavored Markdown
Content Cell	Content Cell

LeetCode Problems

Problem #	Problem Title
1	GitHub Flavored Markdown
Content Cell	Content Cell

String Problems

Problem #	Problem Title
1	GitHub Flavored Markdown
Content Cell	Content Cell

Array Problems

Problem #	Problem Title
1	Find pair with given sum in the array

Linked List Problems Solutions

1) Insert into Linked List

```
/**
 *
 * @param {ListNode} node
 * prints the linked list with given node
 */
function printLL(node) {
  console.log(JSON.stringify(node), null, 2);
}

/**
 *
 * @param {*} val
 */
function ListNode(val) {
```

```
  this.key = val;
  this.next = null;
}

/**
 *
 * @param {ListNode} node
 * @param {Number} val
 * @return {ListNode} final head of the linked list
 */
function insertNode(node, val) {
  if (node === null) {
    return new ListNode(val);
  } else {
    node.next = insertNode(node.next, val);
  }
  return node;
}

// prepare linked list
let values = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
let head = null;
for (let value of values) {
  head = insertNode(head, value);
}
printLL(head);
```

2) Insert Node at Front of Linked List

```
/**
 *
 * @param {ListNode} node
 * prints the linked list with given node
 */
function printLL(node) {
  console.log(JSON.stringify(node, null, 2));
}

/**
 *
 * @param {*} val
 */
function ListNode(val) {
  this.key = val;
  this.next = null;
}
```

```
/**
 * inserts node at Front of the linked list
 * @param {ListNode} node
 * @param {Number} val
 * @return {ListNode} final head of the linked list
 */
function insertNodeAtFront(node, val) {
  if (node === null) {
    return new ListNode(val);
  }
  let newNode = new ListNode(val);
  newNode.next = node;
  return newNode;
}

// prepare linked list
let values = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
let head = null;
for (let value of values) {
  head = insertNodeAtFront(head, value);
}
printLL(head);
```

3) Insert element at given position

```
/**
 *
 * @param {ListNode} node
 * prints the linked list with given node
 */
function printLL(node) {
  console.log(JSON.stringify(node, null, 2));
}

/**
 *
 * @param {*} val
 */
function ListNode(val) {
  this.key = val;
  this.next = null;
}

/**
 *
```

```
 * @param {ListNode} node
 * @param {Number} val
 * @return {ListNode} final head of the linked list
 */
function insertNode(node, val) {
  if (node === null) {
    return new ListNode(val);
  } else {
    node.next = insertNode(node.next, val);
  }
  return node;
}

/**
 * inserts node at Front of the linked list
 * @param {ListNode} node
 * @param {Number} val
 * @return {ListNode} final head of the linked list
 */
function insertNodeAtGivenPosition(node, val, pos) {
  // if given node is null return;
  if (node === null) {
    return;
  }
  // create a new pointer which points to the node
  // and use this to traverse through the node
  let temp = node;
  // minimum position should be 1 and if position is less than 1, then it is invalid
  if (pos < 1) {
    return node;
  }
  // traverse the linked list while the position is greater than 1 and
  // there are nodes present in the linked list
  while (pos > 1 && temp.next) {
    temp = temp.next;
    pos--;
  }
  // if the position is still greater than 1 after traversing all nodes
  // it means we have reached the end point and there are no nodes further
  // so this is an inavlid position
  // in this case we return the given node.
  if (pos > 1) {
    return node;
  }
  let newNode = new ListNode(val);
  newNode.next = temp.next;
  temp.next = newNode;
  return node;
}
```

```
// prepare linked list
let values = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
let head = null;
for (let value of values) {
  head = insertNode(head, value);
}
head = insertNodeAtGivenPosition(head, 100, -1);
head = insertNodeAtGivenPosition(head, 100, 1);
head = insertNodeAtGivenPosition(head, 200, 100);
printLL(head);
```

4) Delete Last node of linked list

```
/**
 *
 * @param {ListNode} node
 * prints the linked list with given node
 */
function printLL(node) {
  console.log(JSON.stringify(node, null, 2));
}

/**
 *
 * @param {*} val
 */
function ListNode(val) {
  this.key = val;
  this.next = null;
}

/**
 *
 * @param {ListNode} node
 * @param {Number} val
 * @return {ListNode} final head of the linked list
 */
function insertNode(node, val) {
  if (node === null) {
    return new ListNode(val);
  } else {
    node.next = insertNode(node.next, val);
  }
  return node;
}
```

```
/**
 * deletes the last node of the linked list
 * @param {ListNode} node
 * @return {ListNode} final head of the linked list
 */
function deleteLastNode(node) {
  // if the nodes next is null
  // it is the last node in the linked list
  // so this is our base case and we return null if the next node of the current Nod
  if (node === null || node.next === null) {
    return null;
  }
  // recursively calling the function to get the next node for the current node
  node.next = deleteLastNode(node.next);
  // return node finally
  return node;
}

// prepare linked list
let values = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
let head = null;
for (let value of values) {
  head = insertNode(head, value);
}
printLL(deleteLastNode(head));
```

Binary Tree Problems Solutions

Binary Search Tree Problems Solutions

Array Problems Solutions

1) Find pair with given sum in the array

```
/**
 * @param {Array} A This is input array
 * @param {Number} sum this is equal to target sum
 * @return {Array} with indexes in array which will be equal to given sum
 */
const findPairWithGivenSum = (A, sum) => {
```

```
  let map = {};
  let index = 0;
  let diff;
  for (let num of A) {
    // take the diff of sum and num
    diff = sum - num;
    // if an element in the array is found with that value
    // a pair is found which makes the current sum
    if (map[diff] !== undefined) {
      return [map[diff], index];
    }
    // storing the num as key and index as value to keep track of the index
    // of the number we have found during traversal
    map[num] = index;
    // increment index by 1 for each iteration
    index++;
  }
};
```

2) Find pair with given sum in the array

```
/**
 * @param {Array} A This is input array
 * @param {Number} sum this is equal to target sum
 * @return {Array} with indexes in array which will be equal to given sum
 */
const findPairWithGivenSum = (A, sum) => {
  let map = {};
  let index = 0;
  let diff;
  for (let num of A) {
    // take the diff of sum and num
    diff = sum - num;
    // if an element in the array is found with that value
    // a pair is found which makes the current sum
    if (map[diff] !== undefined) {
      return [map[diff], index];
    }
    // storing the num as key and index as value to keep track of the index
    // of the number we have found during traversal
    map[num] = index;
    // increment index by 1 for each iteration
    index++;
  }
};
```

LeetCode Problems

Problem #	Problem Title
1	Two Sum
100	Same Tree
125	Valid Palindrome
199	Binary Tree Right Side View
242	Valid Anagram
344	Reverse String
349	Intersection of Two Arrays
350	Intersection of Two Arrays II
657	Robot Return to Origin
700	Search in a Binary Search Tree
704	Binary Search
876	Middle of the Linked List
917	Reverse Only Letters
1046	Last Stone Weight
1119	Remove Vowels from a String
1185	Day of the Week
1213	Intersection of Three Sorted Arrays
242	Valid Anagram
242	Valid Anagram
242	Valid Anagram
242	Valid Anagram

Problem #	Problem Title
242	Valid Anagram
242	Valid Anagram
242	Valid Anagram
242	Valid Anagram

1. Two Sum

```
/**
 * https://leetcode.com/problems/two-sum/
 * @param {number[]} nums
 * @param {number} target
 * @return {number[]}
 * Time Compexity: O(n) because we traverse each element once in worst case
 * Space Complexity: O(n) for map
 */
var twoSum = function (nums, target) {
  let map = {};
  let index = 0;
  for (let num of nums) {
    let currDiff = target - num;
    if (map[currDiff] !== undefined) {
      return [map[currDiff], index];
    }
    map[num] = index;
    index++;
  }
};

console.log(twoSum([2, 7, 11, 15], 9));
console.log(twoSum([3, 2, 4], 6));
```

100. Same Tree

```
/**
 * Definition for a binary tree node.
 * @param {TreeNode} left
 * @param {TreeNode} right
 * @param {Number} val
 *
```

```
 */
function TreeNode(val, left, right) {
  this.val = val === undefined ? 0 : val;
  this.left = left === undefined ? null : left;
  this.right = right === undefined ? null : right;
}

/**
 * @param {TreeNode} p
 * @param {TreeNode} q
 * @return {boolean}
 * Time Complexity: O(n) because we travserse all nodes
 * Space Complexity: O(n) for recursive call stack.
 */
var isSameTree = function (p, q) {
  if (p === null && q === null) return true;
  if ((p === null) | (q === null)) return false;
  return (
    p.val === q.val &&
    isSameTree(p.left, q.left) &&
    isSameTree(p.right, q.right)
  );
};
```

125. Valid Palindrome

```
/**
 * @param {string} s
 * @return {boolean}
 */
var isPalindrome = function (s) {
  s = s
    .trim()
    .replace(/[^a-zA-Z0-9]/g, '')
    .toLowerCase();
  if (s.trim().length === 0) return true;
  let left = 0;
  let right = s.length - 1;
  while (left < right) {
    if (s[left] === s[right]) {
      left++;
      right--;
    } else {
      return false;
    }
  }
```

```
  return true;
};
```

199. Binary Tree Right Side View

```
/**
 * @param {TreeNode} root
 * @return {number[]}
 */
var rightSideView = function (root) {
  let map = {};
  helper(root, 0, map);
  return Object.values(map);
};

var helper = function (root, level, map) {
  if (root === null) {
    return;
  }
  map[level] = root.val;
  helper(root.left, level + 1, map);
  helper(root.right, level + 1, map);
};
```

242. Valid Anagram

```
/**
 * @param {string} s
 * @param {string} t
 * @return {boolean}
 */
var isAnagram = function (s, t) {
  if (s.length !== t.length) return false;
  let a1 = new Array(26).fill(0);
  let a2 = new Array(26).fill(0);
  for (let char of s) {
    a1[char.charCodeAt(0) - 97]++;
  }
  for (let char of t) {
    if (a1[char.charCodeAt(0) - 97] === 0) {
      return false;
    }
    a1[char.charCodeAt(0) - 97]--;
    if (a1[char.charCodeAt(0) - 97] < 0) {
```

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

344. Reverse String

```
/**
 * @param {character[]} s
 * @return {void} Do not return anything, modify s in-place instead.
 */
var reverseString = function (s) {
  let left = 0;
  let right = s.length - 1;
  while (left < right) {
    let temp = s[left];
    s[left] = s[right];
    s[right] = temp;
    left++;
    right--;
  }
};
```

349. Intersection of Two Arrays

```
/**
 * @param {number[]} nums1
 * @param {number[]} nums2
 * @return {number[]}
 */
var intersection = function (nums1, nums2) {
  let set1 = new Set();
  let resultSet = new Set();
  for (let num of nums1) {
    set1.add(num);
  }
  for (let num of nums2) {
    if (set1.has(num)) {
      resultSet.add(num);
    }
  }
  return Array.from(resultSet);
};
```

350. Intersection of Two Arrays II

```
/**
 * @param {number[]} nums1
 * @param {number[]} nums2
 * @return {number[]}
 */
var intersect = function (nums1, nums2) {
  let hashMap = {};
  let results = [];
  for (let num of nums1) {
    hashMap[num] = hashMap[num] + 1 || 1;
  }
  for (let num of nums2) {
    if (hashMap[num] > 0) {
      results.push(num);
      hashMap[num]--;
    }
  }
  return results;
};
```

657. Robot Return to Origin

```
/**
 * @param {string} moves
 * @return {boolean}
 */
var judgeCircle = function (moves) {
  let left = 0;
  let right = 0;
  let up = 0;
  let down = 0;
  let total = 0;
  for (let move of moves) {
    if (move === 'R') {
      right++;
    } else if (move === 'L') {
      left++;
    } else if (move === 'U') {
      up++;
    } else {
      down++;
    }
  }
```

```
  return left === right && up === down;
};
```

700. Search in a Binary Search Tree

```
/**
 * @param {TreeNode} root
 * @param {number} val
 * @return {TreeNode}
 */
var searchBST = function (root, val) {
  if (root === null) {
    return null;
  }
  if (root.val === val) {
    return root;
  }
  if (root.val > val) {
    return searchBST(root.left, val);
  } else {
    return searchBST(root.right, val);
  }
};
```

704. Binary Search

```
/**
 * @param {number[]} nums
 * @param {number} target
 * @return {number}
 */
var search = function (nums, target) {
  let left = 0;
  let right = nums.length - 1;
  while (left <= right) {
    let mid = Math.floor((left + right) / 2);
    if (nums[mid] === target) {
      return mid;
    }
    if (nums[mid] >= target) {
      right = right - 1;
    } else {
      left = left + 1;
    }
```

```
  }
  return -1;
};
```

876. Middle of the Linked List

```
/**
 * https://leetcode.com/problems/middle-of-the-linked-list/
 * @param {ListNode} head
 * @return {ListNode}
 */
var middleNode = function (head) {
  let slow = head;
  let fast = head;
  while (fast && fast.next) {
    slow = slow.next;
    fast = fast.next.next;
  }
  return slow;
};
```

917. Reverse Only Letters

```
/**
 * @param {string} S
 * @return {string}
 */
var reverseOnlyLetters = function (S) {
  S = S.split('');
  let left = 0;
  let right = S.length - 1;
  while (left < right) {
    if (S[left].match(/[a-zA-Z]/) && S[right].match(/[a-zA-Z]/)) {
      let temp = S[left];
      S[left] = S[right];
      S[right] = temp;
      left++;
      right--;
    } else if (!S[left].match(/[a-zA-Z]/)) {
      left++;
    } else {
      right--;
    }
  }
```

```
  return S.join('');
};
```

1046. Last Stone Weight

```
/**
 * @param {number[]} stones
 * @return {number}
 */
var lastStoneWeight = function (stones) {
  let maxHeap = new MaxBinaryHeap();
  for (let stone of stones) {
    maxHeap.insert(stone);
  }
  console.log(maxHeap);
  while (maxHeap.values.length > 1) {
    let stone1 = maxHeap.extractMax();
    let stone2 = maxHeap.extractMax();
    console.log(stone1, stone2);
    if (stone1 !== stone2) {
      maxHeap.insert(Math.abs(stone1 - stone2));
    }
  }
  if (maxHeap.values.length) {
    return maxHeap.values[0];
  } else {
    return 0;
  }
};

class MaxBinaryHeap {
  constructor() {
    this.values = [];
  }

  insert(val) {
    this.values.push(val);
    this.bubbleUp();
  }

  bubbleUp() {
    let currVal = this.values[this.values.length - 1];
    let currIndex = this.values.length - 1;
    let parentIndex = Math.floor((this.values.length - 2) / 2);
    while (this.values[parentIndex] < currVal) {
      if (currVal > this.values[parentIndex]) {
```

```
        this.values[currIndex] = this.values[parentIndex];
        this.values[parentIndex] = currVal;
        currIndex = parentIndex;
      }
      parentIndex = Math.floor(currIndex - 1 / 2);
    }
  }

  extractMax() {
    let ans = this.values[0];
    this.sinkDown();
    return ans;
  }

  sinkDown() {
    this.values[0] = this.values[this.values.length - 1];
    let currSinkDownVal = this.values.pop();
    let currParentIndex = 0;
    let child1Idx = 2 * currParentIndex + 1;
    let child2Idx = 2 * currParentIndex + 2;
    while (
      currSinkDownVal < this.values[child2Idx] ||
      currSinkDownVal < this.values[child1Idx]
    ) {
      if (this.values[child2Idx] > this.values[child1Idx]) {
        this.values[currParentIndex] = this.values[child2Idx];
        this.values[child2Idx] = currSinkDownVal;
        currParentIndex = child2Idx;
      } else {
        this.values[currParentIndex] = this.values[child1Idx];
        this.values[child1Idx] = currSinkDownVal;
        currParentIndex = child1Idx;
      }
      child1Idx = 2 * currParentIndex + 1;
      child2Idx = 2 * currParentIndex + 2;
    }
  }
}
```

1119. Remove Vowels from a String

```
/**
 * @param {string} S
 * @return {string}
 */
var removeVowels = function (S) {
```

```
  return S.replace(/[aeiou]/g, '');
};
```

1185. Day of the Week

```
/**
 * @param {number} day
 * @param {number} month
 * @param {number} year
 * @return {string}
 */
var dayOfTheWeek = function (day, month, year) {
  var d = new Date(year, month - 1, day);
  console.log(d);
  var weekday = new Array(7);
  weekday[0] = 'Sunday';
  weekday[1] = 'Monday';
  weekday[2] = 'Tuesday';
  weekday[3] = 'Wednesday';
  weekday[4] = 'Thursday';
  weekday[5] = 'Friday';
  weekday[6] = 'Saturday';
  return weekday[d.getDay()];
};
```

1213. Intersection of Three Sorted Arrays

```
/**
 * @param {number[]} arr1
 * @param {number[]} arr2
 * @param {number[]} arr3
 * @return {number[]}
 */
var arraysIntersection = function (arr1, arr2, arr3) {
  let result = [];
  const hashMap = {};

  for (let i = 0; i < arr1.length; i++) {
    hashMap[arr1[i]] = false;
  }

  for (let i = 0; i < arr2.length; i++) {
    if (hashMap[arr2[i]] === false) {
      hashMap[arr2[i]] = true;
```

```
    }
  }

  for (let i = 0; i < arr3.length; i++) {
    if (hashMap[arr3[i]] === true) {
      result.push(arr3[i]);
    }
  }

  return result;
};
```

---

ds-and-algos is maintained by skmuddamsetty.

This page was generated by GitHub Pages.