vladyslav horbachenko

BASIC INFO

Email: vladyslav-horbachenko@lambdastudents.com
Test: Lambda School - Whiteboard Fitness Assessment I
Solved: 1/3
Score: 129/175
Finished On: 18 Aug 2020
Time Taken: 58m/3h

Labels: -

Task	Solve Time	Score	Similarity
condense_linked_list	21min	50/50	none
tree_paths_sum	-	29/50	-
merge_packages	-	50/75	-

Task details: **condense_linked_list**

Description:

Given a linked list of integers, remove any nodes from the linked list that have values that have previously occurred in the linked list. Your function should return a reference to the head of the updated linked list.

Example:
Input: `(3) -> (4) -> (3) -> (2) -> (6) -> (1) -> (2) -> (6) -> N`
Output: `(3) -> (4) -> (2) -> (6) -> (1) -> N`
Explanation: The input list contains redundant nodes `(3)`, `(6)`, and `(2)`, so those should be removed from the list.

Solution (main.js):

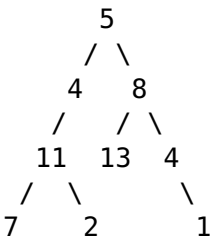
```
// Singly-linked lists are already defined with this interface:
// function ListNode(x) {
//   this.value = x;
//   this.next = null;
// }
//
function condense_linked_list(node) {
    if(!node)
        return node;
  var cur=node;
  var prev=new ListNode(null);
  var set=new Set();
  while(cur!=null)
  {
      if(set.has(cur.value))
      {
          prev.next=cur.next;
      }
      else
      {
          set.add(cur.value);
          prev=cur;
      }
      cur=cur.next;
  }
  return node;
}
```

Task details: **tree_paths_sum**

Description:

Given the root of a binary tree where each node contains an integer, determine the sum of all of the integer values in the tree.

Example:



The expected output given the above tree is `5 + 4 + 8 + 11 + 13 + 4 + 7 + 2 + 1`, so your function should return `55`.

Solution (main.js):

```
//
// Binary trees are already defined with this interface:
// function Tree(x) {
//   this.value = x;
//   this.left = null;
//   this.right = null;
// }
function tree_paths_sum(root) {
    let sum = Number.NEGATIVE_INFINITY;
    const dfs = (root) => {
        if(!root) {
            return 0;
        }

        const sumL = Math.max(0, dfs(root.left, sum));
        const sumR = Math.max(0, dfs(root.right, sum));
        sum = Math.max(sum, (sumL+ sumR + root.value));
        return root.value + Math.max(sumR, sumL);
    };
    dfs(root);
    return sum;
}
```

Task details: **merge_packages**

Description:

Given a package with a weight limit `limit` and an array of integers `items` of where each integer represents the weight of an item, implement a function `merge_packages` that finds the first two items in the `items` array whose sum of weights equals the given weight limit `limit`.

Your function should return a pair `[i, j]` of the indices of the item weights, ordered such that `i > j`. If such a pair doesn't exist, return an empty array.

Examples:

```
Input: items = [4, 6, 10, 15, 16], limit = 21
Output: [3, 1]
Explanation: The weight of the items at indices 3 and 1 sum up to the specified limit.
```

Solution (main.js):

```
function merge_packages(items, limit) {
    const complementObj = {};
  let len = items.length;
  let i;
  let val;
  let complementIndex;
  for (i = 0; i < len; i++) { // Check1
    val = items[i];
    complementIndex = complementObj[limit - val]
    if (complementIndex !== undefined) {
      return [i, complementIndex];
    } else {
      complementObj[val] = i;
    }
  }
  return -1;
}
```