Lu Chang & Zhang Mengjiao

LeetCode Solutions

First Edition

Preface

This project is aimed to accompany my girlfriend @MengjiaoZhang to learn git, programming skills and algorithms.

Here, I want to thank LeetCode providing these problems. It will be better if all problems can be accessed without subscribing. (ಡωಡ)hiahiahia

Contents

Chapter 1

Solutions for Algorithms

"*Perhaps the most important principle for the good algorithm designer is to refuse to be content.*"

— Alfred V. Aho

461. Hamming Distance

Difficulty

Easy

Tags

Bitwise Operation

Description

The Hamming distance between two integers is the number of positions at which the corresponding bits are different.

Given two integers `x` and `y`, calculate the Hamming distance.

Note

$0 \leq x, y < 2^{31}$

Example 1

```
Input: x = 1, y = 4
Output: 2

Explanation:
1   (0 0 0 1)
4   (0 1 0 0)
       ↑   ↑

The above arrows point to positions where the corresponding bits
↪ are different.
```

Analysis

This is an easy problem about Hamming distance and bitwise operation. We can see from the example above that the Hamming distance is the sum of different bits. Therefore, we may easily associate with `xor` operation that detect different bits.

```
1 xor 4 = 0001 xor 0100 = 0101 = 5
```

Then we can just count the bit of value `1` in `5`, i.e. `x xor y`. To perform this operation, we can fetch the last bit of 5 by `and` with `1`, and following shift operation.

Solution

C

```c
int hammingDistance(int x, int y) {
    int mask = x ^ y;  // xor operation
    int number = 0;
    while (mask > 0) {
        number += mask & 1;  // fetch the last bit
        mask >>= 1;  // shift operation
    }
    return number;
}
```

535. Encode and Decode TinyURL

Difficulty

Medium

Tags

Cryptology

Description

TinyURL is a URL shortening service where you enter a URL such as `https://leetcode.com/problems/design-tinyurl` and it returns a short URL such as `http://tinyurl.com/4e9iAk`.

Design the `encode` and `decode` methods for the TinyURL service. There is no restriction on how your encode/decode algorithm should work. You just need to ensure that a URL can be encoded to a tiny URL and the tiny URL can be decoded to the original URL.

Analysis

This is an open problem where numerous solutions can be applied. We can even keep the original url as encode and decode, although it is meaningless.

We should to pay attention to these limitations:

- Correctness: We must make sure that the decoded url is the same as the original url.
- Uniqueness: Each url must have an unique encoded url, and an encoded url must be decoded to a single url.
- Simplicity: The aim to encode a url is to make it easy to share or write, so we need to make the encoded url as simple as possible.

We can use current popular encode/decode algorithms such as `AES`, `DES`, but in this problem, we just design a simpler algorithm to encode and decode a url.

```
function encode(url)
    return hex(current number of urls in hash table)
end

function decode(encoded_url)
    return (hash table).find(encoded_url)
end
```

Solution

C++

```cpp
typedef unordered_map<string, string> Urlmap;

class Solution {
public:

    Urlmap urlmap;

    // Encodes a URL to a shortened URL.
    string encode(string longUrl) {
        size_t size = urlmap.size();
        stringstream encoded;
        encoded << hex << size;
        string encoded_url = encoded.str();
        urlmap.insert(make_pair(encoded_url, longUrl));
        return encoded_url;
    }

    // Decodes a shortened URL to its original URL.
    string decode(string shortUrl) {
    Urlmap::iterator it = urlmap.find(shortUrl);
    if (it == urlmap.end()) return NULL;
        return it->second;
    }
};
```

617. Merge Two Binary Trees

Difficulty

Easy

Tags

Binary Tree, Recursive Algorithm

Description

Given two binary trees and imagine that when you put one of them to cover the other, some nodes of the two trees are overlapped while the others are not.

You need to merge them into a new binary tree. The merge rule is that if two nodes overlap, then sum node values up as the new value of the merged node. Otherwise, the NOT null node will be used as the node of new tree.

Example 1

```
Input:
        Tree 1                     Tree 2
          1                          2
         / \                        / \
        3   2                      1   3
       /                            \   \
      5                              4   7
Output:
Merged tree:
         3
        / \
       4   5
      / \   \
     5   4   7
```

Note

- The merging process must start from the root nodes of both trees.

Analysis

This is a problem about how to traverse two trees in the same time. If two nodes in each tree are both not null, we can simply add the value of Tree 2 to Tree 1. If the left child of node in Tree 1 is null while Tree 2 not, we can let the left child point to Tree 2's left child, which means move Tree 2's left child to Tree 1, and the same as right child. In this way, we don't need to malloc new node. Then we can use a recursive manner to traverse two trees and merge each node.

We assume the size of each tree is `m` and `n` respectively, then

- Time Complexity: $\mathcal{O}(\min(m, n))$
- Space Complexity: $\mathcal{O}(1)$

Solution

C

```c
struct TreeNode* mergeTrees(struct TreeNode* t1, struct TreeNode* t2) {
    if (t1 == NULL) return t2;
    if (t2 == NULL) return t1;
    t1->val += t2->val;
    if (t1->left == NULL && t2->left != NULL) {
        t1->left = t2->left;  // move t2's left to t1, no malloc
    } else if (t1->left != NULL && t2->left != NULL) {
        mergeTrees(t1->left, t2->left);  // go to child node
    }
    if (t1->right == NULL && t2->right != NULL) {
        t1->right = t2->right;
    } else if (t1->right != NULL && t2->right != NULL) {
        mergeTrees(t1->right, t2->right);
    }
    return t1;
}
```

654. Maximum Binary Tree

Difficulty

Medium

Tags

Binary Tree

Description

Given an integer array with no duplicates. A maximum tree building on this array is defined as follow:

1. The root is the maximum number in the array.
2. The left subtree is the maximum tree constructed from left part sub-array divided by the maximum number.
3. The right subtree is the maximum tree constructed from right part sub-array divided by the maximum number.

Construct the maximum tree by the given array and output the root node of this tree.

Example 1

```
Input: [3, 2, 1, 6, 0, 5]
Output: return the tree root node representing the following tree
↪ :

      6
    /   \
   3     5
    \    /
     2  0
       \
        1
```

Note

- The size of the given array will be in the range [1, 1000].

Analysis

At first sight, we can easily know that we can design a recursive algorithm to find the maximum value, take it as root, and do the same step on the left and right array. The worst time complexity is $\mathcal{O}(n^2)$ when array is in order, assuming that the number of array is n.

We can directly build this maximum binary tree in the following steps:

1. Choose the first value as the root;
2. For each successive value, if it is larger that the root, take it as root, and the previous root and its subtree as the left subtree of root (because previous root is the left array). Otherwise go to step 3;
3. If the successive value is less than the root, then compare it with the right node of root (because this value is at the right array). If right node is null, take this value as right node. Otherwise go to step 2.

- Time Complexity: Average is $\mathcal{O}(n\log n)$ because we need to insert each value to a tree. Worst case is $\mathcal{O}(n^2)$, when original array is only descending but not ascending, which has less worst cases than recursive way. In this case, this algorithm degenerates to the insertion sort.
- Space Complexity: $\mathcal{O}(1)$ (We don't need extra space).

Solution

C

```c
struct TreeNode* constructMaximumBinaryTree(int* nums, int numsSize) {
    typedef struct TreeNode TreeNode;
    // we need a root pointer to the real root
    TreeNode *root = (TreeNode *)malloc(sizeof(TreeNode));
    root->right = NULL;
    for (int i = 0; i < numsSize; ++i) {
```

```
        int v = nums[i];
        TreeNode *p = root;
        // move to right and find a node's right child less than current value
        while (p->right != NULL && v < p->right->val) {
            p = p->right;
        }
        TreeNode *node = (TreeNode *)malloc(sizeof(TreeNode));
        node->val = v;
        node->left = p->right;  // null or less than current value should be
        ↪  node's left child
        node->right = NULL;
        p->right = node;  // replace previous right child
    }
    return root->right;  // return real root
}
```

657. Judge Route Circle

Difficulty

Easy

Tags

Math

Description

Initially, there is a Robot at position `(0, 0)`. Given a sequence of its moves, judge if this robot makes a circle, which means it moves back to the original place.

The move sequence is represented by a string. And each move is represent by a character. The valid robot moves are `R` (Right), `L` (Left), `U` (Up) and `D` (down). The output should be true or false representing whether the robot makes a circle.

Example 1

```
Input: "UD"
Output: true
```

Example 2

```
Input: "LL"
Output: false
```

Analysis

Obviously, each point has a coordinate `(x, y)`. `R` means `x + 1`, `L` means `x - 1`, codeU means `y + 1` and `D` means `y - 1`. Therefore, the robot's moving back to the original place means the coordinate is still `(0, 0)` after all moves, assuming original coordinate is `(0, 0)`.

- Time Complexity: $\mathcal{O}(n)$

- Space Complexity: $\mathcal{O}(1)$

Solution

C

```c
bool judgeCircle(char* moves) {
    int x = 0, y = 0;
    char move;
    for (int i = 0; (move = moves[i]) != '\0'; ++i) {
        switch (move) {
            case 'U':
            y += 1;
            break;
            case 'D':
            y -= 1;
            break;
            case 'R':
            x += 1;
            break;
            case 'L':
            x -= 1;
            break;
            default:
            break;
        }
    }
    return (x == 0 && y == 0);
}
```

771. Jewels and Stones

Difficulty

Easy

Tags

Hash Table

Description

You're given strings `J` representing the types of stones that are jewels, and `S` representing the stones you have. Each character in `S` is a type of stone you have. You want to know how many of the stones you have are also jewels.

The letters in `J` are guaranteed distinct, and all characters in `J` and `S` are letters. Letters are case sensitive, so `"a"` is considered a different type of stone from `"A"`.

Example 1

```
Input: J = "aA", S = "aAAbbbb"
Output: 3
```

Example 2

```
Input: J = "z", S = "ZZ"
Output: 0
```

Note

- `S` and `J` will consist of letters and have length at most 50.
- The characters in `J` are distinct.

Analysis

This is an easy problem. All we need to do is to verify wether each letter in `S` exists in `J`.

Therefore, we can use a hash set to store `J`. Specifically, `J` is composed of letters (lower or upper) only, so we can use an array of `char` to store each letter in . After that, we only need to iterate `S` to calculate the number of jewels.

We assume the length of is m, the length of `S` is n, then

- Time complexity: $\mathcal{O}(m + n)$
- Space complexity: $\mathcal{O}(1)$ (256 ASCII chars)

Solution

C

```c
int numJewelsInStones(char* J, char* S) {
    char j_letters[256] = { 0 };  // initialize an array to store letters in J
    char c;
    for (int i = 0; (c = J[i]) != '\0'; ++i) {
        j_letters[c] = 1;  // set j_letters[c] = 1 means c appears in J
    }
    int jewel_number = 0;  // number of jewels
    for (int i = 0; (c = S[i]) != '\0'; ++i) {
        jewel_number += j_letters[c];  // if c appears in J, then we add a
        ↪  number
    }
    return jewel_number;
}
```

797. All Paths From Source to Target

Difficulty

Medium

Tags

Graph

Description

Given a directed, acyclic graph of `N` nodes. Find all possible paths from node `0` to node `N - 1`, and return them in any order.

The graph is given as follows: the nodes are `0, 1, ..., graph.length - 1`. `graph[i]` is a list of all nodes `j` for which the edge `(i, j)` exists.

Example 1

```
Input: [[1, 2], [3], [3], []]
Output: [[0, 1, 3], [0, 2, 3]]

Explanation: The graph looks like this:
0--->1
|    |
v    v
2--->3
There are two paths: 0 -> 1 -> 3 and 0 -> 2 -> 3.
```

Note

- The number of nodes in the graph will be in the range `[2, 15]`.
- You can print different paths in any order, but you should keep the order of nodes inside one path.

Analysis

This is a classic DFS or BFS algorithm. We can easily use recursive method to generate paths.

Let me explain the example first. We first assume that the node number in the graph is `n`, and the node is `0, 1, ..., n - 1`. The input is `[[1, 2], [3], [3], []]`, which means node `0` is connected to `1` and `2` and so forth. In a recursive manner, we first fetch `1` and find `1` is connected to `3`, then these two nodes forms a path. And we next fetch `2`, and find `2` is also connected to `3`, hence, there are two paths. In this manner, we call is Depth-First-Search (DFS), and DFS can be intuitively implemented in a recursive manner.

As for Breadth-First-Search (BFS), BFS is more appropriate to be implemented in a loop manner with a queue. It fetch all nodes in a layer to this queue, dequeue each node, and put successive nodes in queue. In this example, we first fetch `1` and `2`. Then we fetch `3` and `3` as successive nodes of `1` and `2`. Finally we also find there are two paths.

In our solution, we use DFS in a non-recursive manner, just to try more method. We use stack to replace recursion. Similar to BFS, It first fetch all nodes in a layer to the stack, pop from stack and push its successive nodes.

- Time complexity: $\mathcal{O}(n^2)$, when the graph is a complete graph, because node `i` have `n - i` successive nodes.
- Space complexity: $\mathcal{O}(n^2)$, also when it is a complete graph.

Solution

C

```c
// use a linked list to store path temporarily
struct Path {
    int *path;
    int path_len;
    struct Path *next;
};
typedef struct Path Path;
```

```c

int** allPathsSourceTarget(int** graph, int graphRowSize, int *graphColSizes, int** columnSizes, int* returnSize) {
    Path *paths_list = (Path *)malloc(sizeof(Path));
    paths_list->next = NULL;
    Path *paths_list_tail = paths_list;
    int path_number = 0;

    // BFS in a stack manner
    int node_stack[225], top = 0;
    node_stack[0] = 0;
    // this stack is to store the path length in each layer
    // the successive nodes of one node form a layer
    int path_len_stack[225], len_top = 0;
    path_len_stack[0] = 1;

    // a path
    int *current_path = (int *)malloc(15 * sizeof(int));
    int current_path_len;
    while (top >= 0) {
        // fetch next node
        int current_node = node_stack[top--];
        // fetch the coresponding path length for this node
        current_path_len = path_len_stack[len_top--];
        // add this node to current path
        current_path[current_path_len - 1] = current_node;
        if (current_node == graphRowSize - 1) {  // if path exists
            // add path to linked list
            Path *p = (Path *)malloc(sizeof(Path));
            p->path = current_path;
            p->path_len = current_path_len;
            p->next = NULL;
            paths_list_tail->next = p;
            paths_list_tail = p;

            // get a new path based on current path
            current_path = (int *)malloc(15 * sizeof(int));
            memcpy(current_path, p->path, (current_path_len - 1) * sizeof(int));
            ++path_number;
        } else {
            int *current_row = graph[current_node];
            int current_row_size = graphColSizes[current_node];
            // add successive nodes and path lengths to the stack
```

```c
            for (int i = 0; i < current_row_size; ++i) {
                node_stack[++top] = current_row[i];
                path_len_stack[++len_top] = current_path_len + 1;
            }
            // move to next layer
            if (current_row_size > 0) ++current_path_len;
        }
    }

    *returnSize = path_number;
    int **paths = (int **)malloc(path_number * (sizeof(int *)));
    *columnSizes = (int *)malloc(path_number * (sizeof(int *)));
    Path *p = paths_list->next;
    // transform linked list to 2d array
    for (int i = 0; p != NULL; ++i) {
        paths[i] = p->path;
        (*columnSizes)[i] = p->path_len;
        p = p->next;
    }
    return paths;
}
```

804. Unique Morse Code Words

Difficulty

Easy

Tags

Hash Table

Description

International Morse Code defines a standard encoding where each letter is mapped to a series of dots and dashes, as follows: `"a"` maps to `".-"`, `"b"` maps to `"-..."`, `"c"` maps to `"-.-."`, and so on.

For convenience, the full table for the 26 letters of the English alphabet is given below:

```
[".-", "-...", "-.-.", "-..", ".", "..-.", "--.", "....", "..",
↪ ".---", "-.-", ".-..", "--", "-.", "---", ".--.", "--.-",
↪ ".-.", "...", "-", "..-", "...-", ".--", "-..-", "-.--",
↪ "--.."]
```

Now, given a list of words, each word can be written as a concatenation of the Morse code of each letter. For example, `"cab"` can be written as `"-.-.-....-"`, (which is the concatenation `"-.-."` + `"-..."` + `".-"`). We'll call such a concatenation, the transformation of a word.

Return the number of different transformations among all words we have.

Example 1

```
Input: words = ["gin", "zen", "gig", "msg"]
Output: 2
Explanation:
The transformation of each word is:
"gin" -> "--...-."
```

```
"zen" -> "--...-."
"gig" -> "--...--."
"msg" -> "--...--."

There are 2 different transformations, "--...-." and "--...--.".
```

Note

- The length of `words` will be at most `100`.
- Each `words[i]` will have length in range `[1, 12]`.
- `words[i]` will only consist of lowercase letters.

Analysis

This problem is the combination of Morse Code and a hash store. In this problem, we need to check repetition count of the Morse Code for each word in `words`. Therefore we can simply use a hash set to store appeared Morse Codes and check the existence of next.

We assume the size of `words` is m, and we have already known that each `words[i]` will have length in range `[1, 12]`, we assume the max length of a word is n, then

- Time complexity: $\mathcal{O}(mn)$ (We need to iterate all m word in `words` and calculate the hash of this word, while checking the existence is $\mathcal{O}(1)$)

Solution

C++

```cpp
int uniqueMorseRepresentations(vector<string>& words) {
    string morse_table[] = { ".-", "-...", "-.-.", "-..", ".", "..-.", "--.", "....", "..", ".---", "-.-", ".-..", "--", "-.", "---", ".--.", "--.-", ".-.", "...", "-",  "..-", "...-", ".--", "-..-", "-.--", "--.." };
    unordered_set<string> morse_codes(100);  // the max size of words is 100
    for (vector<string>::iterator iter = words.begin(); iter != words.end(); ++iter) {
```

```
        string word = *iter;
        string code;
        code.reserve(50);
        for (string::iterator ch_iter = word.begin(); ch_iter != word.end();
        ↪  ++ch_iter) {
            code += morse_table[*ch_iter - 'a'];  // calculate the morse code of
            ↪  each letter
        }
        morse_codes.insert(code);
    }
    return morse_codes.size();
}
```

814. Binary Tree Pruning

Difficulty

Medium

Tags

Binary Tree, Recursive Algorithm

Description

We are given the head node `root` of a binary tree, where additionally every node's value is either a 0 or a 1.

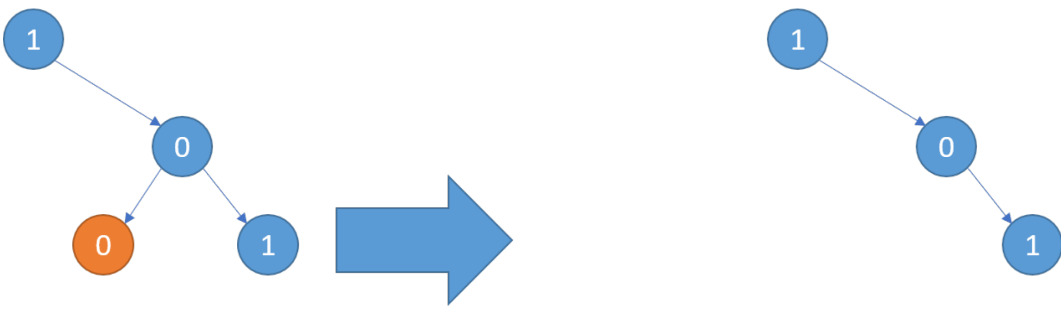
Return the same tree where every subtree (of the given tree) not containing a 1 has been removed.

(Recall that the subtree of a node X is X, plus every node that is a descendant of X.)

Example 1

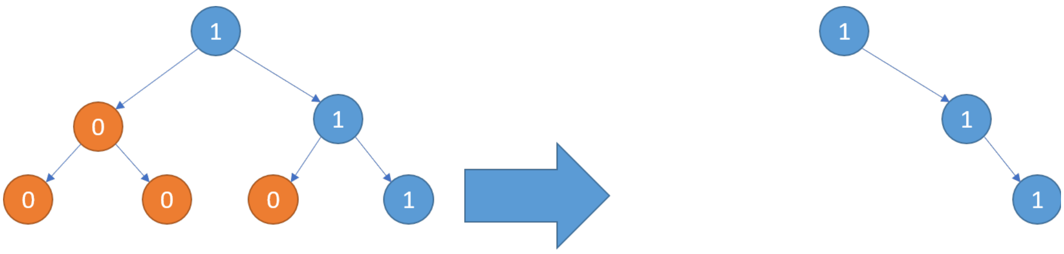
```
Input: [1, null, 0, 0, 1]
Output: [1, null, 0, null, 1]

Explanation:
Only the red nodes satisfy the property "every subtree not
↪ containing a 1".
The diagram on the right represents the answer.
```



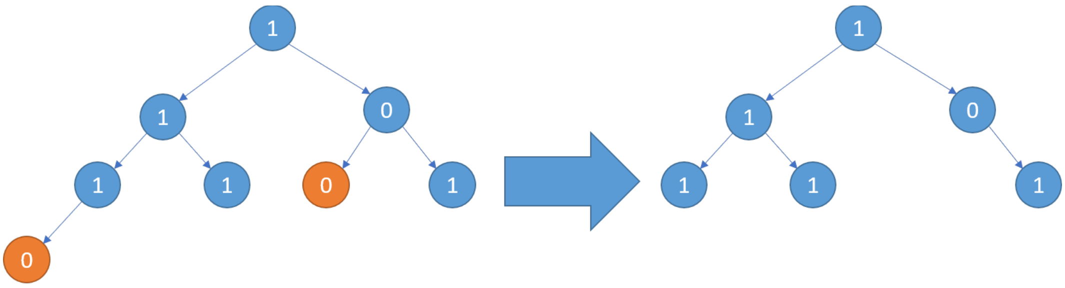
Example 2

```
Input: [1, 0, 1, 0, 0, 0, 1]
Output: [1, null, 1, null, 1]
```



Example 3

```
Example 3:
Input: [1, 1, 0, 1, 1, 0, 1, 0]
Output: [1, 1, 0, 1, 1, null, 1]
```



Note

- The binary tree will have at most `100 nodes`.
- The value of each node will only be `0` or `1`.

Analysis

This is a problem about the binary tree. However, it is not a complicated problem because we only need a preorder traversal and count the number of 1 in the subtree of a node. Then if the count is `0` of the subtree for a node, we just need to make the pointer to `null`.

We assume that the nodes number is n, then

- Time Complexity: $\mathcal{O}(n)$
- Space Complexity: $\mathcal{O}(n)$ (We need to maintain a count for each node)

Solution

C

```c
int count_one(struct TreeNode* r) {
    if (r == NULL) return 0;  // leaf node
    int left_count = count_one(r->left);
    int right_count = count_one(r->right);
    /**
    * I write this comment just to say I remember to free the memory,
    * but in this test, forget it.
    */
    if (left_count == 0) r->left = NULL;
    if (right_count == 0) r->right = NULL;
    return r->val + left_count + right_count;  // consider the value for this
     ↪  node itself.
}

struct TreeNode* pruneTree(struct TreeNode* root) {
    count_one(root);  // only need call this function
    return root;
}
```

Indexes of Solutions for Algorithms

Tags of Solutions for Algorithms

Binary Tree

617. Merge Two Binary Trees

654. Maximum Binary Tree

814. Binary Tree Pruning

Bitwise Operation

461. Hamming Distance

Cryptology

535. Encode and Decode TinyURL

Graph

797. All Paths From Source to Target

Hash Table

771. Jewels and Stones

804. Unique Morse Code Words

Math

657. Judge Route Circle

Recursive Algorithm

617. Merge Two Binary Trees

814. Binary Tree Pruning

Chapter 2

Solutions for Databases

"*Inconsistency of your mind can damage your memory. Remove the inconsistent data and keep the original one only.*"

— Anonym

595. Big Countries

Difficulty

Easy

Tags

Where Condition

Description

There is a table `World`

name	continent	area	population	gdp
Afghanistan	Asia	652230	25500100	20343000
Albania	Europe	28748	2831741	12960000
Algeria	Africa	2381741	37100000	188681000
Andorra	Europe	468	78115	3712000
Angola	Africa	1246700	20609294	100990000

A country is big if it has an area of bigger than 3 million square km or a population of more than 25 million.

Write a SQL solution to output big countries' name, population and area.

For example, according to the above table, we should output:

name	population	area
Afghanistan	25500100	652230
Algeria	37100000	2381741

Analysis

Most basic SQL knowledge on select and where.

Solution

```sql
select name, population, area from World
    where population > 25000000
    or area > 3000000;
```

Indexes of Solutions for Databases

Tags of Solutions for Databases

Where Condition