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 for providing these problems. Besides, I also want to thank all contributors of LeetCode for their solutions and discussions.

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

338. Counting Bits

Difficulty

Medium

Tags

Bitwise Operation

Description

Given a non negative integer number **num**. For every numbers i in the range $0 \leq i \leq num$ calculate the number of 1's in their binary representation and return them as an array.
Example:

For $num = 5$ you should return `[0, 1, 1, 2, 1, 2]`.
Follow up:

- It is very easy to come up with a solution with run time $\mathcal{O}(n * \text{sizeof}(integer))$. But can you do it in linear time $\mathcal{O}(n)$ /possibly in a single pass?
- Space complexity should be $\mathcal{O}(n)$.
- Can you do it like a boss? Do it without using any builtin function like `__builtin_popcount` in c++ or in any other language.

Credits:

Special thanks to @syedee for adding this problem and creating all test cases.

Analysis

For a single number, if we want to count the number of bit 1, we have to fetch each bit. But as the description said, we can design an algorithm that requires a linear time, which means the previous result is required.

Take `13 = 1101` and `14 = 1110` as examples, `1101` can be seen as `110` concatenated with `1`, and `1110` is `111` concatenated with `0`. You may have noticed that a number is composed of its previous `n - 1` bits and the last one bit, assuming the length of the binary number is `n`. And the previous `n - 1` bits are exactly this number's right shift by one bit, while the last bit can be calculated by `and` operation with `1`. Therefore, the count can be calculated by the count of right shift and the last bit.

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

Solution

C

```c
int* countBits(int num, int* returnSize) {
    int size = num + 1;
    int *result = (int *)malloc(size * sizeof(int));
    result[0] = 0;
    for (int i = 1; i <= num; ++i) {
        // the last bit and right shift
        result[i] = (i & 1) + result[i >> 1];
    }
    *returnSize = size;
    return result;
}
```

344. Reverse String

Difficulty

Easy

Tags

String & Array

Description

Write a function that takes a string as input and returns the string reversed.
Example:

```
Given s = "hello", return "olleh".
```

Analysis

This is the most primary string and reverse problem which we should dominate when we begin to learn programming. We just need to iterate half of the string and swap the symmetric elements with the middle element as the symcenter.

We assume the length of the string is n, then

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

Solution

C

```c
char* reverseString(char* s) {
    int len = strlen(s);
```

```
    for (int i = 0; i < len / 2; ++i) {
        // swap
        char t = s[i];
        s[i] = s[len - i - 1];
        s[len - i - 1] = t;
    }
    return s;
}
```

419. Battleships in a Board

Difficulty

Medium

Tags

String & Array

Description

Given an 2D board, count how many battleships are in it. The battleships are represented with `'X'`s, empty slots are represented with `'.'`s. You may assume the following rules:

- You receive a valid board, made of only battleships or empty slots.
- Battleships can only be placed horizontally or vertically. In other words, they can only be made of the shape $1 \times N$ (1 row, N columns) or $N \times 1$ (N rows, 1 column), where N can be of any size.
- At least one horizontal or vertical cell separates between two battleships - there are no adjacent battleships.

Example:

```
X..X
...X
...X
```

In the above board there are 2 battleships.

Invalid Example:

```
X..X
...X
...X
```

This is an invalid board that you will not receive - as battleships will always have a cell separating between them.

Follow up:

Could you do it in **one-pass**, using only $\mathcal{O}(1)$ extra memory and **without modifying** the value of the board?

Analysis

This is a Battleship problem. I like playing Battleship, and there are some algorithms helping you improve the possibility of hitting the enemy's ships.

We need to detect the count of ships. You may think that we can iterate the board. If we find an `X`, then we can check the vertical and horizontal line to detect a ship. After validate a ship, we can modify the value of board to another value or use an extra board to record our visits. Or you can use two passes to iterate the row and column respectively to detect the ships. However, these method will consume extra resources.

We should notice that the ships can only be placed horizontally or vertically, which means giving a direction and a start or point, i.e. the head and tail of a ship, a ship can be determined. Therefore, the problem transforms to how to detect the head or tail of a ship. In our solution, we use the head to detect the ship. `board[m][n] = X` is the head only if it's previous point is not an `X`, i.e. `border[m - 1][n] != 'X'` and `board[m][n - 1] != X` or it is the first element in a row or column, otherwise it is in the middle of the ship.

We assume the row number is m and column number is n, then

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

Solution

C

```c
int countBattleships(char** board, int boardRowSize, int boardColSize) {
    int count = 0;
    for (int m = 0; m < boardRowSize; ++m) {
```

```
        for (int n = 0; n < boardColSize; ++n) {
            if ((board[m][n] == 'X')  // the point is an element of a ship
            && (m == 0 || board[m - 1][n] != 'X')  // and it is the head
            && (n == 0 || board[m][n - 1] != 'X')) {
                ++count;
            }
        }
    }
    return count;
}
```

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;
}
```

463. Island Perimeter

Difficulty

Easy

Tags

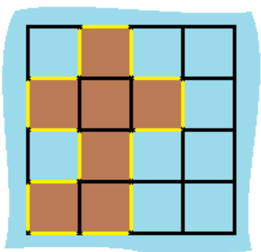
String & Array

Description

You are given a map in form of a two-dimensional integer grid where 1 represents land and 0 represents water. Grid cells are connected horizontally/vertically (not diagonally). The grid is completely surrounded by water, and there is exactly one island (i.e., one or more connected land cells). The island doesn't have "lakes" (water inside that isn't connected to the water around the island). One cell is a square with side length 1. The grid is rectangular, width and height don't exceed 100. Determine the perimeter of the island.

Example 1

```
Input:
[[0,1,0,0],
[1,1,1,0],
[0,1,0,0],
[1,1,0,0]]
Output: 16
Explanation:
The perimeter is the 16 yellow stripes in the image below:
```



Analysis

Maybe we used to meet such problem in primary schools, counting the edge of joint rectangular. We can see from the figure that each land has four edges. And if two land are joint, they will lose two edges. The land in the edge of matrix can't be joint with outer lands. Therefor, we can iterate from top to bottom and left to right, to count the lands and the joint number.

We assume there are `n` rows and `m` columns, then,

- Time complexity: $\mathcal{O}(mn)$
- Space complexity: $\mathcal{O}(1)$

Solution

C

```c
int islandPerimeter(int** grid, int gridRowSize, int gridColSize) {
    int count = 0;
    for (int i = 0; i < gridRowSize; ++i) {
        int *row = grid[i];
        // not the top edge of matrix
        int not_border_top = i != 0;
        for (int j = 0; j < gridColSize; ++j) {
            if (row[j] == 1) {
                count += 4;
                // joint by the top land
                if (not_border_top && grid[i - 1][j] == 1) count -= 2;
                // joint by the left land
                if (j != 0 && row[j - 1] == 1) count -= 2;
            }
        }
    }
    return count;
}
```

476. Number Complement

Difficulty

Easy

Tags

Bitwise Operation

Description

Given a positive integer, output its complement number. The complement strategy is to flip the bits of its binary representation.

Note

1. The given integer is guaranteed to fit within the range of a 32-bit signed integer.
2. You could assume no leading zero bit in the integer's binary representation.

Example 1

```
Input: 5
Output: 2

Explanation: The binary representation of 5 is 101 (no leading
↪ zero bits), and its complement is 010. So you need to output
↪ 2.
```

Example 2

```
Input: 1
Output: 0

Explanation: The binary representation of 1 is 1 (no leading zero
↪  bits), and its complement is 0. So you need to output 0.
```

Analysis

There are many method to solve this problem. For example, you can fetch each bit of this num and calculate its complementary bit. In our solution, we use a different way to calculate. The key point in this problem is how to ignore the leading zero of a number. For example, `5 = 1...11010` and `5 & 7 = 2 = 010`. So, what we need to do is find the smallest $2^n - 1$ that larger than this number. Then we can calculate the complementary number without leading zeros by `num &` $(2^n - 1)$, where $n = \lfloor \log_2 num \rfloor)$. Therefore, this problem transforms to how to fast calculate $\lfloor \log_2 num \rfloor)$. In http://www.graphics.stanford.edu/ seander/bithacks.html, we can find a way with $\mathcal{O}(\log(N))$ time complexity, in which N is the bit length of this number.

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

Solution

C

```c
int findComplement(int num) {
    if (num == 0) return 1;

    int v = num;
    int r;
    int shift;

    // calculate log2(num)
    r = (v > 0xFFFF) << 4; v >>= r;
    shift = (v > 0xFF) << 3; v >>= shift; r |= shift;
    shift = (v > 0xF) << 2; v >>= shift; r |= shift;
    shift = (v > 0x3) << 1; v >>= shift; r |= shift;
    r |= (v >> 1);

    return (~num) & ((2 << r) - 1);
}
```

500. Keyboard Row

Difficulty

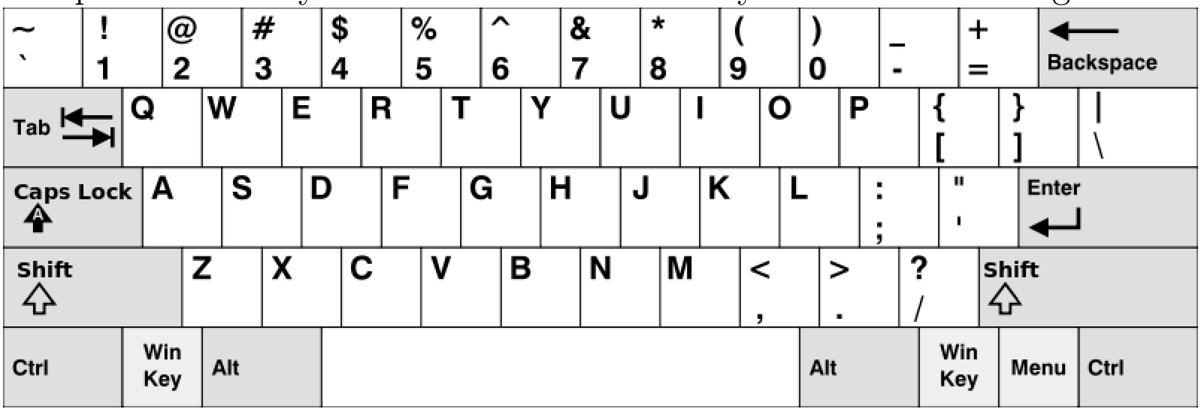
Easy

Tags

String & Array

Description

Given a List of words, return the words that can be typed using letters of alphabet on only one row's of American keyboard like the image below.



Example 1

```
Input: ["Hello", "Alaska", "Dad", "Peace"]
Output: ["Alaska", "Dad"]
```

Note

- You may use one character in the keyboard more than once.
- You may assume the input string will only contain letters of alphabet.

Analysis

Maybe this is a somewhat boring problem. We just need to check whether all letters of a word appear in one row of the keyboard. Therefore, we can pre-calculate the row table of the alphabet, and find in this table when iterate the words.

We assume that the number of words is m and the max length of a words is n, then

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

Solution

C

```c
char** findWords(char** words, int wordsSize, int* returnSize) {
    // row table of each letter from a to z
    int positions[] = { 1, 2, 2, 1, 0, 1, 1, 1, 0, 1, 1, 1, 2,
        2, 0, 0, 0, 0, 1, 0, 0, 2, 0, 2, 0, 2 };
    char **result = (char **)malloc(wordsSize * sizeof(char *));
    int count = 0;
    for (int i = 0; i < wordsSize; ++i) {
        char *word = words[i];
        char c = word[0];
        if (c < 'a') c += 32;  // uppercase
        int pos = positions[c - 'a'];
        for (int k = 1; (c = word[k]) != '\0'; ++k) {
            if (c < 'a') c += 32;
            // check if any letter is not in the same row as the first letter
            if (positions[c - 'a'] != pos) break;
        }
        if (c == '\0') result[count++] = word;
    }
    *returnSize = count;
    return result;
}
```

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;
    }
};
```

537. Complex Number Multiplication

Difficulty

Easy

Tags

Math, String & Array

Description

Given two strings representing two complex numbers.

You need to return a string representing their multiplication. Note $i^2 = -1$ according to the definition.

Example 1

```
Input: "1+1i", "1+1i"
Output: "0+2i"
Explanation: (1 + i) * (1 + i) = 1 + i2 + 2 * i = 2i, and you
↪ need convert it to the form of 0+2i.
```

Example 2

```
Input: "1+-1i", "1+-1i"
Output: "0+-2i"
Explanation: (1 - i) * (1 - i) = 1 + i2 - 2 * i = -2i, and you
↪ need convert it to the form of 0+-2i.
```

Note

- The input strings will not have extra blank.
- The input strings will be given in the form of **a+bi**, where the integer a and b will both belong to the range of [-100, 100]. And **the output should be also in this form**.

Analysis

This is an easy complex multiplication problem with many details. First you should parse the string to a complex number. Secondly you should compute the multiplication, and finally you need to transform it to a string in the required form. To parse the complex number, we can easily think up that the part before `+` is the real and the part between `+` and `i` is the image. And to transform the complex to string, we only need to call `sprintf`. However, if we first find the position of `+` and `i` and call `atoi` to transform string to integers, we will get extra iterations. Therefore, we can implement the transform function by ourselves.

We assume the max length of complex strings is n, then

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

Solution

C

```c
struct Complex {
    int real;
    int image;
};
typedef struct Complex Complex;

int _itoa(int a, char *s) {
    int i = 0, start = 0;
    // zero should also be printed
    if (a == 0) {
        s[0] = '0';
        s[1] = '\0';
        return 1;
    }
    if (a < 0) {
        a = -a;
        s[0] = '-';
        start = i = 1;
    }
```

```
    while (a > 0) {
        s[i++] = (char)((a % 10) + '0');
        a /= 10;
    }
    // reverse number because steps above fetch digit from lower end to higher.
    for (int k = start; k < i / 2 + start; ++k) {
        char c = s[k];
        int pos = i - k - (1 - start);
        s[k] = s[pos];
        s[pos] = c;
    }
    s[i] = '\0';
    return i;
}

void parse_complex(Complex *complex, char *s) {
    char c;
    int val = 0;
    int sign = 1;
    for (int i = 0; (c = s[i]) != '\0'; ++i) {
        switch (c) {
        case '+':
            // val now is the real part
            complex->real = sign == 1 ? val : -val;
            // reset val and sign after real part to calculate image part
            val = 0;
            sign = 1;
            break;
        case 'i':
            // val now is the image part
            complex->image = sign == 1 ? val : -val;
            case '-':
            sign = -1;
            break;
        default:
            val *= 10;
            val += c - '0';
        }
    }
}

char* complexNumberMultiply(char* a, char* b) {
    Complex c1, c2;
```

```
    parse_complex(&c1, a);
    parse_complex(&c2, b);
    int real = c1.real * c2.real - c1.image * c2.image;
    int image = c1.real * c2.image + c1.image * c2.real;
    char *result = (char *)malloc(20 * sizeof(char));
    int pos = _itoa(real, result);
    result[pos++] = '+';
    pos += _itoa(image, result + pos);
    result[pos++] = 'i';
    result[pos] = '\0';
    return result;
}
```

557. Reverse Words in a String III

Difficulty

Easy

Tags

String & Array

Description

Given a string, you need to reverse the order of characters in each word within a sentence while still preserving whitespace and initial word order.

Example 1

```
Input: "Let's take LeetCode contest"
Output: "s'teL ekat edoCteeL tsetnoc"
```

Note

- In the string, each word is separated by single space and there will not be any extra space in the string.

Analysis

The steps in this problems are very clear. We need to find the spaces, and reverse the words between two spaces.

We assume the length of the string is n, then

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

Solution

C

```c
char* reverseWords(char* s) {
    char word[100];
    char c;
    int word_len = 0;
    int s_len = 0;
    for (int i = 0; (c = s[i]) != '\0'; ++i) {
        if (c == ' ') {  // found space
            for (int k = word_len - 1; k >= 0; --k) {
                // reverse word
                s[s_len++] = word[k];
            }
            s[s_len++] = ' ';
            word_len = 0;
        }
        else {
            word[word_len++] = c;
        }
    }
    // remaining word
    for (int k = word_len - 1; k >= 0; --k) {
        s[s_len++] = word[k];
    }
    return s;
}
```

561. Array Partition I

Difficulty

Easy

Tags

String & Array

Description

Given an array of $2n$ integers, your task is to group these integers into n pairs of integer, say $(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)$ which makes sum of $min(a_i, b_i)$ for all i from 1 to n as large as possible.

Example 1

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

Explanation: n is 2, and the maximum sum of pairs is 4 = min(1,
↪ 2) + min(3, 4).
```

Note

- n is a positive integer, which is in the range of $[1, 10000]$.
- All the integers in the array will be in the range of $[-10000, 10000]$.

Analysis

At first sight, you may wonder that what an easy problem it is. All we need to do is sort this array and fetch every other element from 0. In this manner, however, we get a time complexity of $O(n\log n)$, assuming that the number of array is n, because we have to sort the array. After that, we may think a linear method to calculate the sum.

Therefore, we use buckets $b[0..i..n]$ to store the elements. $x[i]$ is the element and $b[i]$ is the count of this element, because we notice that if the count of an element is odd, the remaining one element have to participate the comparison to next elements. Hence, we use a $flag$ to record whether the count of last element is odd or even. If flag is odd, then in the next element, we have

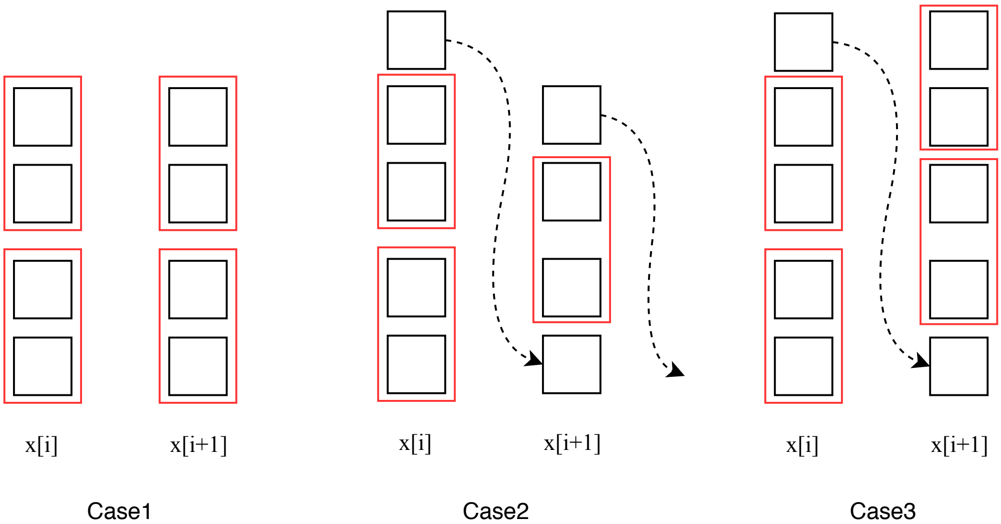
```
sum[i + 1] = sum[i] + (x[i] + ((b[i + 1] — 1) / 2) * x[i + 1]
```

because $x[i+1]$ has to contribute one element to compare with $x[i]$. And when flag is even, we have

```
sum[i + 1] = sum[i] + (x[i] + (b[i + 1] / 2) * x[i + 1]
```

And then we need to check $b[x+1]$ to set the flag.

The cases are shown in the figure below:



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

Solution

C

```c
int arrayPairSum(int* nums, int numsSize) {
    int distinct_nums[20001] = { 0 };
    for (int i = 0; i < numsSize; ++i) {
        ++distinct_nums[nums[i] + 10000];
    }
    int result = 0;
    // flag to record the count of element is odd or even
    int flag = 0;
    // previous element
    int prev;
    for (int i = 0, x = -10000; i <= 20000; ++i, ++x) {
        int count = distinct_nums[i];
        if (count == 0) continue;
        // last count is even
        if (flag == 0) {
            result += x * (count >> 1);
            // remaining element
            if ((count & 1) == 1) {
                flag = 1;
                prev = x;
            } else {
                flag = 0;
            }
        } else {  // last count is odd
            int a = count & 1;
            result += x * ((count - 1) >> 1) + prev;
            // remaining element
            if ((count & 1) == 0) {
                flag = 1;
                prev = x;
            } else {
                flag = 0;
            }
        }
    }
    return result;
}
```

575. Distribute Candies

Difficulty

Easy

Tags

Hash Table, String & Array

Description

Given an integer array with **even** length, where different numbers in this array represent different **kinds** of candies. Each number means one candy of the corresponding kind. You need to distribute these candies **equally** in number to brother and sister. Return the maximum number of **kinds** of candies the sister could gain.

Example 1

```
Input: candies = [1,1,2,2,3,3]
Output: 3
Explanation:
There are three different kinds of candies (1, 2 and 3), and two
↪ candies for each kind.
Optimal distribution: The sister has candies [1,2,3] and the
↪ brother has candies [1,2,3], too.
The sister has three different kinds of candies.
```

Example 2

```
Input: candies = [1,1,2,3]
Output: 2
Explanation:
For example, the sister has candies [2,3] and the brother has
↪ candies [1,1].
The sister has two different kinds of candies, the brother has
↪ only one kind of candies.
```

Note

- The length of the given array is in range [2, 10,000], and will be even.
- The number in given array is in range [-100,000, 100,000].

Analysis

Obviously, we only need to count the number of all categories, assuming it is `N`. If `N` is larger than the max number of candies the sister can get, the max categories are the cases where one candy corresponds to one category, otherwise the max category number is the total category number, i.e. all categories are given to the sister.

Assuming the number of candies is `n`, then

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

Solution

C

```c
int distributeCandies(int* candies, int candiesSize) {
    char cats[200001] = { 0 };
    int cat_number = 0;
    int max = candiesSize >> 1;
    for (int i = 0; i < candiesSize; ++i) {
        int cat = candies[i] + 100000;
        if (cats[cat] == 0) {
            cats[cat] = '1';
            ++cat_number;
            if (cat_number >= max) return max;
        }
    }
    return cat_number;
}
```

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 subarray 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);
}
```

669. Trim a Binary Search Tree

Difficulty

Easy

Tags

Binary Tree, Recursive Algorithm

Description

Given a binary search tree and the lowest and highest boundaries as `L` and `R`, trim the tree so that all its elements lies in `[L, R]` (R >= L). You might need to change the root of the tree, so the result should return the new root of the trimmed binary search tree.

Example 1

```
Input:
    1
   / \
  0   2

  L = 1
  R = 2
Output:
    1
     \
      2
```

Example 2

```
Input:
    3
   / \
  0   4
   \
    2
```

```
    /
   1

   L = 1
   R = 3
Output:
       3
      /
     2
    /
   1
```

Analysis

This is a basic Recursive Problem. In this problem, we should first consider whether the value of the node itself needs to be trimmed, then the left and right child. If the value of the node is less than `L`, we need to use the right child to replace it because the right child might be greater than `L`. And the same procedure is applied to the cases where the value is greater than `R`. After comparing this node itself, we next need to traverse to it left and right child.

Assuming that the number of nodes is n, we have

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

Solution

C

```c
typedef struct TreeNode TreeNode;

TreeNode *trim(TreeNode *node, int L, int R) {
    if (node == NULL) return NULL;
    // replace node with left child
    if (node->val > R) return trim(node->left, L, R);
```

```
    // replace node with right child
    if (node->val < L) return trim(node->right, L, R);
    // move to child
    node->left = trim(node->left, L, R);
    node->right = trim(node->right, L, R);
    return node;
}

struct TreeNode* trimBST(struct TreeNode* root, int L, int R) {
    return trim(root, L, R);
}
```

682. Baseball Game

Difficulty

Easy

Tags

Stack & Heap

Description

You're now a baseball game point recorder.

Given a list of strings, each string can be one of the 4 following types:

1. `Integer` (one round's score): Directly represents the number of points you get in this round.
2. `"+"` (one round's score): Represents that the points you get in this round are the sum of the last two `valid` round's points.
3. `"D"` (one round's score): Represents that the points you get in this round are the doubled data of the last `valid` round's points.
4. `"C"` (an operation, which isn't a round's score): Represents the last `valid` round's points you get were invalid and should be removed.

Each round's operation is permanent and could have an impact on the round before and the round after.

You need to return the sum of the points you could get in all the rounds.

Example 1

```
Input: ["5","2","C","D","+"]
Output: 30
Explanation:
Round 1: You could get 5 points. The sum is: 5.
Round 2: You could get 2 points. The sum is: 7.
Operation 1: The round 2's data was invalid. The sum is: 5.
```

```
Round 3: You could get 10 points (the round 2's data has been
↪ removed). The sum is: 15.
Round 4: You could get 5 + 10 = 15 points. The sum is: 30.
```

Example 2

```
Input: ["5","-2","4","C","D","9","+","+"]
Output: 27
Explanation:
Round 1: You could get 5 points. The sum is: 5.
Round 2: You could get -2 points. The sum is: 3.
Round 3: You could get 4 points. The sum is: 7.
Operation 1: The round 3's data is invalid. The sum is: 3.
Round 4: You could get -4 points (the round 3's data has been
↪ removed). The sum is: -1.
Round 5: You could get 9 points. The sum is: 8.
Round 6: You could get -4 + 9 = 5 points. The sum is 13.
Round 7: You could get 9 + 5 = 14 points. The sum is 27.
```

Note:

- The size of the input list will be between 1 and 1000.
- Every integer represented in the list will be between -30000 and 30000.

Analysis

Not too much to say, you just need to follow the instruction and use a stack to maintain all rounds.

We assume the total lengths of these string instructions are n, then

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

Solution

C

```c
int calPoints(char** ops, int opsSize) {
    int round[1000];
    int top = -1;
    int sum = 0;
    for (int i = 0; i < opsSize; ++i) {
        char *operation = ops[i];
        char op = operation[0];
        if ((op >= '0' && op <= '9') || (op == '-')) {
            // string integer to int
            int score = atoi(operation);
            sum += score;
            round[++top] = score;
        } else if (op == '+') {
            // top 2 elements in the stack
            int score = round[top] + round[top - 1];
            sum += score;
            round[++top] = score;
        } else if (op == 'D') {
            // double
            int score = round[top] << 1;
            sum += score;
            round[++top] = score;
        } else {
            // pop
            int score = -round[top--];
            sum += score;
        }
    }
    return sum;
}
```

728. Self Dividing Numbers

Difficulty

Easy

Tags

Math

Description

A self-dividing number is a number that is divisible by every digit it contains.

For example, 128 is a self-dividing number because `128 % 1 == 0`, `128 % 2 == 0`, and `128 % 8 == 0`.

Also, a self-dividing number is not allowed to contain the digit zero.

Given a lower and upper number bound, output a list of every possible self dividing number, including the bounds if possible.

Example 1

```
Input: left = 1, right = 22
Output: [1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 15, 22]
```

Note

- The boundaries of each input argument are `1 <= left <= right <= 10000`.

Analysis

This key problem is to fetch each digit of an integer and check whether this integer can divide each digit. In addition, as long as this integer contains `0`, it is not a self-dividing number.

- Time complexity: $\mathcal{O}(right - left)$
- Space complexity: $\mathcal{O}(1)$

Solution

C

```c
int* selfDividingNumbers(int left, int right, int* returnSize) {
    int *results = (int *)malloc((right - left + 1) * sizeof(int));
    int count = 0;
    for (int i = left; i <= right; ++i) {
        int number = i;
        int flag = 1;
        while (number > 0) {
            // fetch each digits
            int digit = number % 10;
            // contains 0 or cannot be divided
            if (digit == 0 || i % digit != 0) {
                flag = 0;
                break;
            }
            // shift right in hex mode
            number /= 10;
        }
        if (flag == 1) {
            results[count++] = i;
        }
    }
    *returnSize = count;
    return results;
}
```

763. Partition Labels

Difficulty

Medium

Tags

String & Array

Description

A string S of lowercase letters is given. We want to partition this string into as many parts as possible so that each letter appears in at most one part, and return a list of integers representing the size of these parts.

Example 1

```
Input: S = "ababcbacadefegdehijhklij"
Output: [9,7,8]
Explanation:
The partition is "ababcbaca", "defegde", "hijhklij".
This is a partition so that each letter appears in at most one
↪ part.
A partition like "ababcbacadefegde", "hijhklij" is incorrect,
↪ because it splits S into less parts.
```

Note

- `S` will have length in range `[1, 500]`.
- `S` will consist of lowercase letters (`'a'` to `'z'`) only.

Analysis

This is an interesting array problem, and maybe you will read several more times to comprehend it. In this problem we need to separate `S` into some parts, and each letter have to appear in at most one part, which means the last letter will never appear in successive parts. Therefore, we can naturally have an idea that we need to record the last appearance position of each letter.

In the example above, The last appearance position of `a` is `9`, and letters between `1` and `9` do not appear after `a`, so we can find the first part of `1` to `9`. Then, we know the second part starts at `d` of position `10`, and its last appearance position is `15`. But `e` appears after `15` at `16`, then we have to move the end of second part to `16`. And we find that `16` is exactly the end of the second part.

In conclusion, the basic idea is find the end of each part, which is the maximum position of last appearance in this part.

Assuming the length of `S` is n, we have

- Time complexity: $\mathcal{O}(n)$
- Space complexity: $\mathcal{O}(1)$ (we only need to store the last position of each letter, and the return size is obviously at worst 26, because each letter appears in only one part.)

Solution

C

```c
int* partitionLabels(char* S, int* returnSize) {
    int letter_last_pos[26];
    char c;
    // record each letter's last position
    for (int i = 0; (c = S[i]) != '\0'; ++i) {
        letter_last_pos[c - 'a'] = i;
    }
    int end;
    int *result = (int *)malloc(26 * sizeof(int));
    int size = 0;
    for (int i = 0; (c = S[i]) != '\0';) {
        // i is the start part, we need to find the end of this part
        end = letter_last_pos[c - 'a'];
        // compare the end of each letter with current end
        for (int j = i + 1; j < end; ++j) {
```

```
            int new_end = letter_last_pos[S[j] - 'a'];
            if (new_end > end) end = new_end;
        }
        result[size++] = end - i + 1;
        // move to next part
        i = end + 1;
    }
    *returnSize = size;
    return result;
}
```

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;
}
```

791. Custom Sort String

Difficulty

Medium

Tags

Hash Table, Sort, String & Array

Description

`S` and `T` are strings composed of lowercase letters. In `S`, no letter occurs more than once.

`S` was sorted in some custom order previously. We want to permute the characters of `T` so that they match the order that `S` was sorted. More specifically, if `x` occurs before `y` in `S`, then `x` should occur before `y` in the returned string.

Return any permutation of `T` (as a string) that satisfies this property.

Example 1

```
Input:
S = "cba"
T = "abcd"
Output:
"cbad"
Explanation:
"a", "b", "c" appear in S, so the order of "a", "b", "c" should
↪ be "c", "b", and "a".
Since "d" does not appear in S, it can be at any position in T. "
↪ dcba", "cdba", "cbda" are also valid outputs.
```

Note

- `S` has length at most `26`, and no character is repeated in `S`.
- `T` has length at most `200`.

- `S` and `T` consist of lowercase letters only.

Analysis

You could easily think up that you can customize the compare function of quick sort according to `S`. However, the time complexity will be $\mathcal{O}(n \log n)$ assuming the length of `T` is n.

We can use a linear method to calculate the count of each letter in `T` and reconstruct `T` according the appearance order of letters in `S`. By the way, we can use another array to store letters which are not appear in `S`.

- Time complexity: $\mathcal{O}(n)$ (because the length of `S` will not exceed `26`)
- Space complexity: $\mathcal{O}(n)$

Solution

C

```c
char* customSortString(char* S, char* T) {
    int appearance[26] = { 0 };
    int counts[26] = { 0 };
    char extra_letters[200];
    int extra_letters_count = 0;
    char c;
    // find which letters appear in S
    for (int i = 0; (c = S[i]) != '\0'; ++i) {
        appearance[c - 'a'] = 1;
    }
    // count the frequency of T
    for (int i = 0; (c = T[i]) != '\0'; ++i) {
        if (appearance[c - 'a'] == 0) extra_letters[extra_letters_count++] = c;
        else ++counts[c - 'a'];
    }
    int len = 0;
    // reconstruct T according to the order in S
    for (int i = 0; (c = S[i]) != '\0'; ++i) {
        int count = counts[c - 'a'];
        for (int k = 0; k < count; ++k) {
```

```
            T[len++] = c;
        }
    }
    // append the letters not appeared in S
    for (int i = 0; i < extra_letters_count; ++i) {
        T[len++] = extra_letters[i];
    }
    return T;
}
```

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
```

```
            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 $\mathcal{O}(1)$)
- Space Complexity: $\mathcal{O}(m)$ (We assume that each hash is stores as Integer)

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();
}
```

806. Number of Lines To Write String

Difficulty

Easy

Tags

String & Array

Description

We are to write the letters of a given string `S`, from left to right into lines. Each line has maximum width 100 units, and if writing a letter would cause the width of the line to exceed 100 units, it is written on the next line. We are given an array `widths`, an array where `widths[0]` is the width of `'a'`, `widths[1]` is the width of `'b'`, ..., and `widths[25]` is the width of `'z'`.

Now answer two questions: how many lines have at least one character from S, and what is the width used by the last such line? Return your answer as an integer list of length 2.

Example 1

```
Input:
widths = [10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
↪  10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10]
S = "abcdefghijklmnopqrstuvwxyz"
Output: [3, 60]
Explanation:
All letters have the same length of 10. To write all 26 letters,
we need two full lines and one line with 60 units.
```

Example 2

```
Input:
widths = [4, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
↪ 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10]
S = "bbbcccdddaaa"
```

```
Output: [2, 4]
Explanation:
All letters except 'a' have the same length of 10, and
"bbbcccdddaa" will cover 9 * 10 + 2 * 4 = 98 units.
For the last 'a', it is written on the second line because
there is only 2 units left in the first line.
So the answer is 2 lines, plus 4 units in the second line.
```

Note

- The length of `S` will be in the range `[1, 1000]`.
- `S` will only contain lowercase letters.
- `widths` is an array of length 26.
- `widths[i]` will be in the range of `[2, 10]`.

Analysis

This is a simple and classic typesetting problem, while the description and explanation may be unclear. Each line has a capacity of max character number, and if writing a new word causes the overflow, we need to break line, which means put the this word to a new line. This problem requires us to count the line number and the line width of the last line. Therefore, we just need to iterate the characters, calculating the current line width, and breaking line if needed.

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

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

Solution

C

```c
int* numberOfLines(int* widths, int widthsSize, char* S, int* returnSize) {
    int current_line_width = 0;
    int line_number = 1;
    char c;
    for (int i = 0; (c = S[i]) != '\0'; ++i) {
        int width = widths[c - 'a'];
        // writing new words
        current_line_width = current_line_width + width;
        if (current_line_width > 100) {
            // break line and put this word to the new line
            current_line_width = width;
            ++line_number;
        }
    }
    *returnSize = 2;
    int *result = (int *)malloc(2 * sizeof(int));
    result[0] = line_number;
    result[1] = current_line_width;
    return result;
}
```

811. Subdomain Visit Count

Difficulty

Easy

Tags

Hash Table

Description

A website domain like "discuss.leetcode.com" consists of various subdomains. At the top level, we have "com", at the next level, we have "leetcode.com", and at the lowest level, "discuss.leetcode.com". When we visit a domain like "discuss.leetcode.com", we will also visit the parent domains "leetcode.com" and "com" implicitly.

Now, call a "count-paired domain" to be a count (representing the number of visits this domain received), followed by a space, followed by the address. An example of a count-paired domain might be "9001 discuss.leetcode.com".

We are given a list `cpdomains` of count-paired domains. We would like a list of count-paired domains, (in the same format as the input, and in any order), that explicitly counts the number of visits to each subdomain.

Example 1

```
Input:
["9001 discuss.leetcode.com"]
Output:
["9001 discuss.leetcode.com", "9001 leetcode.com", "9001 com"]

Explanation:
We only have one website domain: "discuss.leetcode.com". As
↪ discussed above, the subdomain "leetcode.com" and "com" will
↪ also be visited. So they will all be visited 9001 times.
```

Example 2

```
Input:
["900 google.mail.com", "50 yahoo.com", "1 intel.mail.com", "5
↪ wiki.org"]
Output:
["901 mail.com","50 yahoo.com","900 google.mail.com","5 wiki.org
↪ ","5 org","1 intel.mail.com","951 com"]

Explanation:
We will visit "google.mail.com" 900 times, "yahoo.com" 50 times,
↪ "intel.mail.com" once and "wiki.org" 5 times. For the
↪ subdomains, we will visit "mail.com" 900 + 1 = 901 times, "
↪ com" 900 + 50 + 1 = 951 times, and "org" 5 times.
```

Note

- The length of `cpdomains` will not exceed `100`.
- The length of each domain name will not exceed `100`.
- Each address will have either 1 or 2 "." characters.
- The input count in any count-paired domain will not exceed `10000`.
- The answer output can be returned in any order.

Analysis

This is an another Hash Table problem. With in a domain, we need to extract each level of it. First of all, we need to iterate the cpdomain to find space, and chars before space is the count. And then we will find each dot to separate each level. And finally, we use a hash table to store and count.

Assuming the length of `cpdomains` is m, and the max length of each domain is n, then we have

- Time Complexity: $\mathcal{O}(mn)$
- Space Complexity: $\mathcal{O}(m)$

Solution

C++

```
vector<string> subdomainVisits(vector<string>& cpdomains) {
    typedef unordered_map<string, int> DomainCount;
    DomainCount domain_count(500);
    for (auto &cpdomain: cpdomains) {
        int count = 0;
        int length = cpdomain.length();
        for (int i = 0; i < length; ++i) {
            switch(cpdomain[i]) {
            case ' ':  // find space
                // then we can know the count
                count = stoi(cpdomain.substr(0, i));
                // lowest domain level
                domain_count[cpdomain.substr(i + 1, length)] += count;
                break;
            case '.':
                // find each domain level by dot
                domain_count[cpdomain.substr(i + 1, length)] += count;
                break;
            }
        }
    }
    vector<string> result;
    result.reserve(domain_count.size());
    for (auto &dc: domain_count) {
        result.emplace_back(to_string(dc.second) + " " + dc.first);
    }
    return result;
}
```

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.

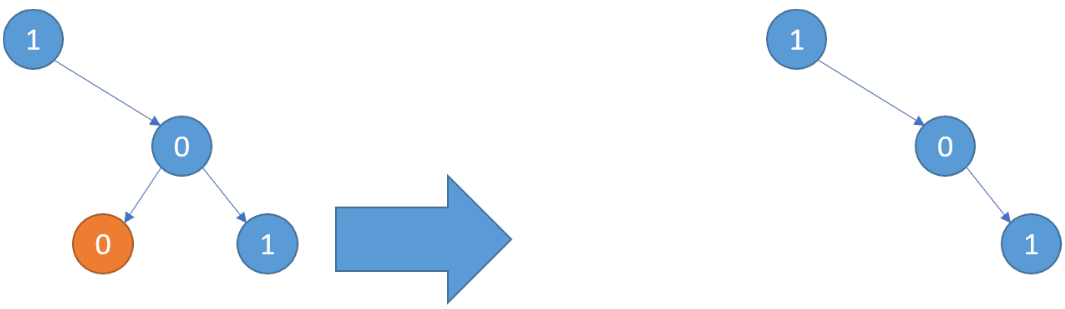
Return the same tree where every subtree (of the given tree) not containing a 1 has been removed.

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

Example 1

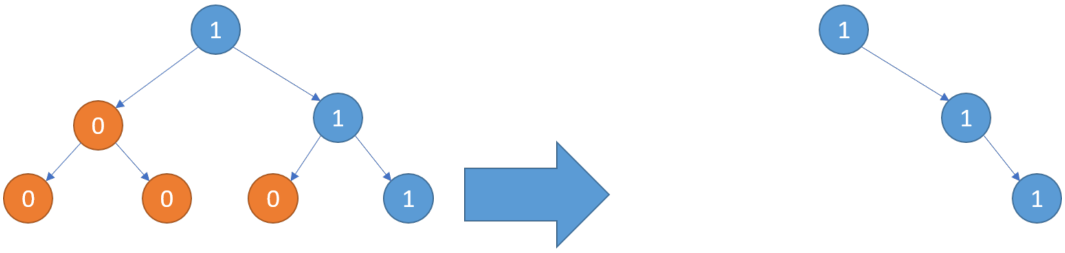
```
Input: [1, null, 0, 0, 1]
Output: [1, null, 0, null, 1]

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



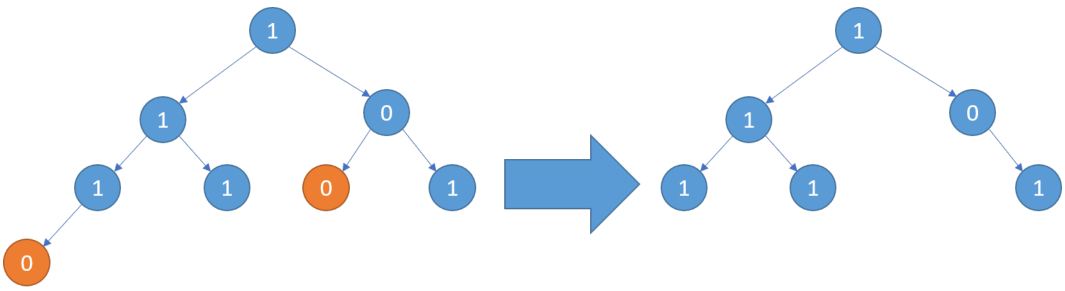
Example 2

```
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;
}
```

821. Shortest Distance to a Character

Difficulty

Easy

Tags

String & Array

Description

Given a string `S` and a character `C`, return an array of integers representing the shortest distance from the character `C` in the string.

Example 1

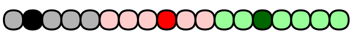
```
Input: S = "loveleetcode", C = 'e'
Output: [3, 2, 1, 0, 1, 0, 0, 1, 2, 2, 1, 0]
```

Note

- `S` string length is in `[1, 10000]`.
- `C` is a single character, and guaranteed to be in string `S`.
- All letters in `S` and `C` are lowercase.

Analysis

We can use the figure below to explain this algorithm:



The dark color is the specified characters, and we use different dark color to represent different positions. We can easily see that the light-color dots are closest to the same dark color. Therefore, the shortest distance should be calculated according to the dark-color dots. And obviously, the boundary of two dark-color dots is their middle element. So we can first find all positions of the specified character, and directly calculated the shortest distance of each character from the last middle boundary to the next middle boundary.

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

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

Solution

C

```c
int* shortestToChar(char* S, char C, int* returnSize) {
    char c;
    int str_length = 0;
    int positions[10000];
    int count = 0;
    // find the positions of specified character
    for (int i = 0; (c = S[i]) != '\0'; ++i) {
        if (c == C) positions[count++] = i;
        ++str_length;
    }
    int *result = (int *)malloc(str_length * sizeof(int));
    int start = 0, end;
    for (int i = 0; i < count - 1; ++i) {
        int position = positions[i];
        end = (position + positions[i + 1]) / 2;
        // range from the previous middle boundary to the next one
        for (int k = start; k <= end; ++k) {
            int diff = k - position;
            result[k] = diff < 0 ? -diff : diff;
        }
        start = end + 1;
    }
    end = str_length - 1;
    int position = positions[count - 1];
    // remaining element
    for (int k = start; k <= end; ++k) {
        int diff = k - position;
        result[k] = diff < 0 ? -diff : diff;
```

```
    }
    *returnSize = str_length;
    return result;
}
```

868. Binary Gap

Difficulty

Easy

Tags

String & Array

Description

Given a positive integer N, find and return the longest distance between two consecutive 1's in the binary representation of N.

If there aren't two consecutive 1's, return 0.

Example 1

```
Input: 22
Output: 2
Explanation:
22 in binary is 0b10110.
In the binary representation of 22, there are three ones, and two
↪  consecutive pairs of 1's.
The first consecutive pair of 1's have distance 2.
The second consecutive pair of 1's have distance 1.
The answer is the largest of these two distances, which is 2.
```

Example 2

```
Input: 5
Output: 2
Explanation:
5 in binary is 0b101.
```

Example 3

```
Input: 6
Output: 1
```

```
Explanation:
6 in binary is 0b110.
```

Example 4

```
Input: 8
Output: 0
Explanation:
8 in binary is 0b1000.
There aren't any consecutive pairs of 1's in the binary
↪ representation of 8, so we return 0.
```

Note

- $1 \leq N \leq 10^9$

Analysis

Using bitwise operation, we first find the first 1 from the right to left. Then we record the length of right shifting from the last 1.

- Time Complexity: $\mathcal{O}(\log(N))$ (The length of N's bits)
- Space Complexity: $\mathcal{O}(1)$

Solution

C

```c
int binaryGap(int N) {
    while ((N & 1) == 0) N >>= 1;
    N >>= 1;
    int max_gap = 0;
    int current_gap = 0;
    while (N > 0) {
        current_gap += 1;
        if (N & 1) {
            if (current_gap > max_gap) {
                max_gap = current_gap;
            }
```

```
            current_gap = 0;
        }
        N >>= 1;
    }
    return max_gap;
}
```

1480. Running Sum of 1d Array

Difficulty

Easy

Tags

String & Array

Description

Given an array `nums`. We define a running sum of an array as `runningSum[i] = sum(nums[0]...nums[i])`.

Return the running sum of `nums`.

Example 1

```
Input: nums = [1, 2, 3, 4]
Output: [1, 3, 6, 10]
Explanation: Running sum is obtained as follows: [1, 1 + 2, 1 + 2
↪  + 3, 1 + 2 + 3 + 4].
```

Example 2

```
Input: nums = [1, 1, 1, 1, 1]
Output: [1, 2, 3, 4, 5]
Explanation: Running sum is obtained as follows: [1, 1 + 1, 1 + 1
↪  + 1, 1 + 1 + 1 + 1, 1 + 1 + 1 + 1 + 1].
```

Example 3

```
Input: nums = [3, 1, 2, 10, 1]
Output: [3, 4, 6, 16, 17]
```

Constraints:

- `1 <= nums.length <= 1000`
- `-106ˆ<= nums[i] <= 106ˆ`

Analysis

This is a simple math problem. We just iterate the array and add the element. Here are some tricks:

- We can operate in place.
- For c, we can operate using pointer to accelerate.

We assume the length of `nums` is n:

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

Solution

C

```c
int* runningSum(int* nums, int numsSize, int* returnSize) {
    *returnSize = numsSize;
    for (int* s = nums + 1; s < nums + numsSize; s += 1) {
        *s += *(s - 1);
    }
    return nums;
}
```

Indexes of Solutions for Algorithms

Tags of Solutions for Algorithms

Binary Tree

617. Merge Two Binary Trees
654. Maximum Binary Tree
669. Trim a Binary Search Tree
814. Binary Tree Pruning

Bitwise Operation

338. Counting Bits
461. Hamming Distance
476. Number Complement

Cryptology

535. Encode and Decode TinyURL

Graph

797. All Paths From Source to Target

Hash Table

575. Distribute Candies
771. Jewels and Stones
791. Custom Sort String
804. Unique Morse Code Words
811. Subdomain Visit Count

Math

537. Complex Number Multiplication
657. Judge Route Circle
728. Self Dividing Numbers

Recursive Algorithm

617. Merge Two Binary Trees
669. Trim a Binary Search Tree
814. Binary Tree Pruning

Sort

791. Custom Sort String

Stack & Heap

682. Baseball Game

String & Array

344. Reverse String
419. Battleships in a Board
463. Island Perimeter
500. Keyboard Row
537. Complex Number Multiplication
557. Reverse Words in a String III
561. Array Partition I
575. Distribute Candies

763. Partition Labels

791. Custom Sort String

806. Number of Lines To Write String

821. Shortest Distance to a Character

868. Binary Gap

1480. Running Sum of 1d Array

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

595. Big Countries