可信认证之九阴真经（第一卷）

作者：吴迈 w00448446

目 录

九阴真经第一式 ：单调栈
代表题目：84. 柱状图中最大的矩形

单调栈介绍

（https://zhuanlan.zhihu.com/p/26465701）单调栈是一种理解起来很容易，但是运用起来并不那么简单的数据结构。一句话解释单调栈，就是一个栈，里面的元素的大小按照他们所在栈内的位置，满足一定的单调性。那么到底什么时候用这个单调栈，怎么用单调栈呢。下面我们来看几个例子。

先来分享一道非常简单的，我本人在 google interview 中遇到的题目。（大雾，当时并没有做出来。）

题目是这样的，给一个数组，返回一个大小相同的数组。返回的数组的第 i 个位置的值应当是，对于原数组中的第 i 个元素，至少往右走多少步，才能遇到一个比自己大的元素（如果之后没有比自己大的元素，或者已经是最后一个元素，则在返回数组的对应位置放上-1）。

简单的例子：

input: 5, 3, 1, 2, 4
return: -1 3 1 1 -1

explaination: 对于第 0 个数字 5，之后没有比它更大的数字，因此是-1，对于第 1 个数字 3，需要走 3 步才能达到 4（第一个比 3 大的元素），对于第 2 和第 3 个数字，都只需要走 1 步，就可以遇到比自己大的元素。对于最后一个数字 4，因为之后没有更多的元素，所以是-1。

暴力做的结果就是 O(n^2)的时间复杂度，例如对于一个单调递减的数组，每次都要走到数组的末尾。

那么用单调栈怎么做呢？先来看代码：

```
vector<int> nextExceed(vector<int> &input) {
    vector<int> result (input.size(), -1);
    stack<int> monoStack;
    for(int i = 0; i < input.size(); ++i) {
        while(!monoStack.empty() && input[monoStack.top()] < input[i]) {
            result[monoStack.top()] = i - monoStack.top();
            monoStack.pop();
        }
        monoStack.push(i);
    }
    return result;
}
```

我们维护这样一个单调递减的 stack，stack 内部存的是原数组的每个 index。每当我们遇到一个比当前栈顶所对应的数（就是 input[monoStack.top()]）大的数的时候，我们就遇到了一个“大数“。这个”大数“比它之前多少个数大我们不知道，但是至少比当前栈顶所对应的数大。我们弹出栈内所有对应数比这个数小的栈内元素，并更新它们在返回数组中对应位置的值。因为这个栈本身的单调性，当我们栈顶元素所对应的数比这个元素大的时候，我们可以保证，栈内所有元素都比这个元素大。对于每一个元素，当它出栈的时候，说明它遇到了自己的 next greater element，我们也就要更新 return 数组中的对应位置的值。如果一个元素一直不曾出栈，那么说明不存在 next greater element，我们也就不用更新 return 数组了。

这里作者在数组末尾加入了一个 height 0，来强迫程序在结束前，将所有元素按照顺序弹出栈。是一个很巧妙的想法。在这个例子中，对于每一个元素都只有一次入栈和出栈的操作，因此时间复杂度只有

O(n)。

解决了这个开胃菜，我们来看一道稍微复杂一点的题目。Leetcode 84. Largest Rectangle in Histogram

Given n non-negative integers representing the histogram's bar height where the width of each bar is 1, find the area of largest rectangle in the histogram.
Above is a histogram where width of each bar is 1, given height = [2,1,5,6,2,3].
The largest rectangle is shown in the shaded area, which has area = 10 unit.
For example,
Given heights = [2,1,5,6,2,3],
return 10.

来看看 discuss 中 sipiprotoss5 的解法。

```
int largestRectangleArea(vector<int> &height) {
        int ret = 0;
        height.push_back(0);
        vector<int> index;
        for(int i = 0; i < height.size(); i++) {
            while(index.size() > 0 && height[index.back()] >= height[i]) {
                int h = height[index.back()];
                index.pop_back();
                int sidx = index.size() > 0 ? index.back() : -1;
                ret = max(ret, h * (i-sidx-1));
            }
            index.push_back(i);
        }
        return ret;
    }
```

看上去这个解法长得和刚才那道题的差不多。实际算法也很相近，作者维护了一个单调递增的栈，然后进行了一系列 xjb 操作。那么到底为什么可以这么做？我们需要分析一下，每一个元素都要入栈一次，出栈一次。入栈的时候是 for loop 的 iteration 走到它的时候，那出栈的时候意味着什么呢。想清楚了这一点，我们也就理解了上面的答案。在上一题，每个元素出栈，是说明它找到了它在原数组中的 next greater element.那这道题呢？元素出栈，意味着，我们已经计算了以它的顶为上边框的最大矩形。首先我们可以通过反证法轻松证明，最后的结果中的最大矩形的上边框，一定和某一个 bar 的顶重合，否则我们一定可以通过提高上边框来增加这个矩形的面积。这一步之后，我们还需要理解，这时候我们计

算的矩形的左右边框都已经到达了极限。结合栈内元素的单调性，我们知道左边的边框是栈顶的元素+1，栈顶元素所对应的 bar 一定比出栈元素对应的 bar 小，所以以出栈元素对应的 bar 为高的矩形无法往左边延展。结合代码，我们知道右边的边框是正在处理的 i，因为我们已经判断过这个第 i 个元素所对应的 bar 也一定比出栈元素对应的 bar 小，所以矩形无法往右边延展。这个元素和左右边框之间如果还有空隙，那么这些空隙里所存在的 bar，一定是因为维护栈的单调性而被弹出了。换言之，这些 bar 如果存在，那么一定比这个出栈元素所对应的 bar 高。既然这些 bar 的高度更高，那么就可以被纳入这个最大矩形的计算中（例如一个“凹”字）。因此我们证明了，当我们将第 i 个元素弹出栈的时候，我们计算了以 hight[i]为高的最大矩形的面积。

下面我们来看另一个可以借助单调栈的题目，Leetcode 85. Maximal Rectangle.

Given a 2D binary matrix filled with 0's and 1's, find the largest rectangle containing only 1's and return its area.
For example, given the following matrix:
1 0 1 0 0
1 0 1 1 1
1 1 1 1 1
1 0 0 1 0
Return 6.

这道题可以轻松地转化成上一题。对于每一行，我们都可以构建一个 histogram，然后计算。在构建新的 histogram 的时候，我们不需要全部遍历，只需要对已有的 histogram 进行略微的修改（运用 DP 的思想）。为了在视觉上更加清晰，我直接调用了上一题中的函数。思路还是异常简单的。

```
int maximalRectangle(vector<vector<char>>& matrix) {
    if (matrix.empty())  return 0;
    vector<int> height(matrix[0].size(), 0);
    int maxRect= 0;
    for(int i = 0; i < matrix.size(); ++i) {
        for(int j = 0; j < height.size(); ++j) {
            if(matrix[i][j] == '0')    height[j] = 0;
            else    ++height[j];
        }
```

```
            maxRect = max(maxRect, largestRectangleArea(height));
            height.pop_back();
        }
        return maxRect;
    }
    int largestRectangleArea(vector<int> &height) {
            int ret = 0;
            height.push_back(0);
            vector<int> index;
            for(int i = 0; i < height.size(); i++) {
                while(index.size() > 0 && height[index.back()] >= height[i]) {
                    int h = height[index.back()];
                    index.pop_back();
                    int sidx = index.size() > 0 ? index.back() : -1;
                    ret = max(ret, h * (i-sidx-1));
                }
                index.push_back(i);
            }
            return ret;
    }
```

最后再总结一下单调栈。单调栈这种数据结构，通常应用在一维数组上。如果遇到的问题，和前后元素之间的大小关系有关系的话，（例如第一题中我们要找比某个元素大的元素，第二个题目中，前后的 bar 的高低影响了最终矩形的计算），我们可以试图用单调栈来解决。在思考如何使用单调栈的时候，可以回忆一下这两题的解题套路，然后想清楚，如果使用单调栈，每个元素出栈时候的意义。最后的时间复杂度，因为每个元素都出栈入栈各一次，所以是线性时间的复杂度。

单调栈的性质：

1.单调栈里的元素具有单调性；

2.元素加入栈前，会在栈顶端把破坏栈单调性的元素都删除；

3.使用单调栈可以找到元素向左遍历第一个比他小的元素，也可以找到元素向左遍历第一个比他大的元素。

对于第三条性质的解释（最常用的性质）：

对于单调栈的第三条性质，你可能会产生疑问，为什么使用单调栈可以找到元素向左遍历第一个比他大的元素，而不是最后一个比他大的元素呢？我们可以从单调栈中元素的单调性来解释这个问题，由于单调栈中的元素只能是单调递增或者是单调递减的，所以我们可以分别讨论这两种情况（假设不存在两个相同的元素）：

1.当单调栈中的元素是单调递增的时候，根据上面我们从数组的角度阐述单调栈的性质的叙述，可以得出：

（1）.当 a > b 时，则将元素 a 插入栈顶，新的栈顶则为 a

（2）.当 a < b 时，则将从当前栈顶位置向前查找（边查找，栈顶元素边出栈），直到找到第一个比 a 小的数，停止查找，将元素 a 插入栈顶（在当前找到的数之后，即此时元素 a 找到了自己的“位置”）

2.当单调栈中的元素是单调递减的时候，则有：

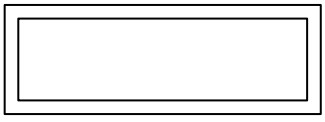
（1）.当 a < b 时，则将元素 a 插入栈顶，新的栈顶则为 a

（2）.当 a > b 时，则将从当前栈顶位置向前查找（边查找，栈顶元素边出栈），直到找到第一个比 a 大的数，停止查找，将元素 a

插入栈顶（在当前找到的数之后，即此时元素 a 找到了自己的“位置”）

（原文链接：https://blog.csdn.net/liujian20150808/article/details/50752861）

触类旁通：
柱状图中的最大矩形（ 84）的一种解法
单调栈：时间复杂度 O(N)



```
/*
 * Copyright (c) Huawei Technologies Co., Ltd. 2019-2020. All rights reserved.
 * Description: 84. 柱状图中的最大矩形
 * Author: w00448446
 * Create: 2020-03-20
 */
```

```
#include <stdio.h>

#define MAX_STACK_LEN 50000
int g_stack[MAX_STACK_LEN];
int g_stackLen;

int GetMax(int a, int b)
{
    if (a >= b) {
        return a;
    }
    return b;
}
void InitStack()
{
    g_stack[0] = -1;
    g_stackLen = 1;
}

// 输入保证数组都是非负元素，考试时都假定输入已做过可信校验
int largestRectangleArea(const int* heights, int heightsSize)
{
    int maxArea = 0;
    int area;
    int stackIndex;
    InitStack();
    if (heightsSize >= MAX_STACK_LEN) {
        return -1;
    }
    for (int i = 0; i < heightsSize; i++) {
        // 不满足单调递增,则计算面积并出栈
        while ((g_stack[g_stackLen - 1] >= 0) && (heights[i] < heights[g_stack[g_stackLen - 1]])) {
            area = ((i - g_stack[g_stackLen - 1 - 1]) - 1) * heights[g_stack[g_stackLen - 1]];
            maxArea = GetMax(maxArea, area);
            // Pop
            g_stackLen--;
        }
        // 满足单调递增，入栈
        g_stack[g_stackLen++] = i;
    }
    // 最后对栈内元素进行特殊处理，计算面积
    stackIndex = g_stack[g_stackLen - 1];
    while (g_stackLen > 1) {
```

```
        area = (stackIndex - g_stack[g_stackLen - 1 - 1]) * heights[g_stack[g_stackLen - 1]];
        maxArea = GetMax(maxArea, area);
        // Pop
        g_stackLen--;
    }
    return maxArea;
}
void Test1()
{
    int nums[] = { 2, 1, 5, 6, 2, 3 };
    int numsSize = sizeof(nums) / sizeof(nums[0]);
    int rslt;
    rslt = largestRectangleArea(nums, numsSize);
    if (rslt == 10) {
        printf("Test1 pass \n");
    } else {
        printf("Test1 fail \n");
    }
}
void main()
{
    Test1();
    return;
}
```

每日温度(739)
题目描述

参考代码：

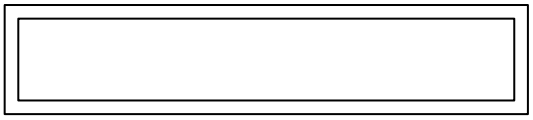
```
// 739. 每日温度
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <ctype.h>
#include <math.h>

#define MAX_DAILY_TEMPERATURES_NUM 30010
int *g_destStack;
int g_destLen;
int g_tempStack[MAX_DAILY_TEMPERATURES_NUM];
int g_tempLen;
int GetMax(int a, int b)
{
    if (a >= b) {
        return a;
    }
    return b;
}

bool MallocDailyTemperatures(int* t, int tSize)
```

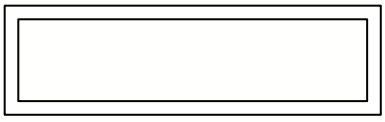
```
{
    int i;
    g_destLen = 0;
    if (tSize == 0) {
        return false;
    }
    g_destStack = (int *)malloc(tSize * sizeof(int));
    if (g_destStack == NULL) {
        return false;
    }

    for (i = 0; i < tSize; i++) {
        g_destStack[i] = 0;
    }

    return true;
}

/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
int *dailyTemperatures(int* t, int tSize, int* returnSize)
{
    int i;
    int stackIndex;
    bool rslt;
    g_tempLen = 0;
    rslt = MallocDailyTemperatures(t, tSize);
    if (!rslt) {
        (*returnSize) = 0;
        return NULL;
    }

    g_tempStack[0] = 0;
        g_tempLen++;
    for (i = 1; i < tSize; i++) {
        while (g_tempLen > 0) {
            stackIndex = g_tempStack[g_tempLen - 1];

            // 不满足单调递减

            if (t[i] > t[stackIndex]) {
                g_destStack[stackIndex] = i - stackIndex;

                // 出栈
```

```
            g_tempLen--;
            continue;
        }
        break;
    }

    // 满足单调递减，入栈
    g_tempStack[g_tempLen++] = i;
}

(*returnSize) = tSize;
return g_destStack;
}
```

下一个更大元素 II（503）

最大矩形（85）





```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>

// #include <securec.h>
int **g_matrix;
int g_matrixSize;
int g_matrixColSize;

#define MAX_STACK_LEN 10000
int g_stack[MAX_STACK_LEN];
int g_stackLen;

int GetMax(int x, int y)
{
    if (x >= y) {
        return x;
    }

    return y;
}

void InitStack()
{
    g_stack[0] = -1;
    g_stackLen = 1;
}

bool MallocMatrix(const char **matrix, const int matrixSize, const int *matrixColSize)
{
    int i;

    g_matrix = NULL;
    g_matrixSize = 0;

    if (matrixColSize == NULL) {
        return false;
    }

    if ((matrixSize == 0) || matrixColSize[0] == 0) {
        return false;
    }
```

```
    g_matrixColSize = matrixColSize[0];

    // 动态分配二维数组
    g_matrix = (int **)malloc(matrixSize * sizeof(int *));
    if (g_matrix == NULL) {
        return false;
    }
    memset(g_matrix, 0, matrixSize * sizeof(int *));

    for (i = 0; i < matrixSize; i++) {
        g_matrix[i] = (int *)malloc(matrixColSize[0] * sizeof(int));
        if (g_matrix[i] == 0) {
            return false;
        }
        memset(g_matrix[i], 0, matrixColSize[0] * sizeof(int));
        g_matrixSize++;
    }

    return true;
}

void FreeMatrix()
{
    int i;

    if (g_matrix == NULL) {
        return;
    }

    for (i = 0; i < g_matrixSize; i++) {
        if (g_matrix[i] == 0) {
            break;
        }
        free(g_matrix[i]);
    }

    return;
}

// 计算每个连续 1 的个数
void InitMatrixSum(const char **matrix, const int matrixSize, const int *matrixColSize)
```

```
{
    int i;
    int j;
    for (i = 0; i < matrixColSize[0]; i++) {
        for (j = matrixSize - 1; j >= 0; j--) {
            if (matrix[j][i] != '1') {
                g_matrix[j][i] = 0;
                continue;
            }

            if ((j + 1) >= matrixSize) {
                g_matrix[j][i] = 1;
                continue;
            }
            g_matrix[j][i] = g_matrix[j + 1][i] + 1;
        }
    }
}


// 输入保证数组都是非负元素，考试时都假定输入已做过可信校验
int largestRectangleArea(const int* heights, int heightsSize)
{
    int maxArea = 0;
    int area;
    int stackIndex;
    InitStack();
    if (heightsSize >= MAX_STACK_LEN) {
        return -1;
    }
    for (int i = 0; i < heightsSize; i++) {

        // 不满足单调递增,则计算面积并出栈

        while ((g_stack[g_stackLen - 1] >= 0) && (heights[i] < heights[g_stack[g_stackLen - 1]])) {
            area = ((i - g_stack[g_stackLen - 1 - 1]) - 1) * heights[g_stack[g_stackLen - 1]];
            maxArea = GetMax(maxArea, area);
            // Pop
            g_stackLen--;
        }

        // 满足单调递增，入栈

        g_stack[g_stackLen++] = i;
    }
```

```
    // 最后对栈内元素进行特殊处理，计算面积
    stackIndex = g_stack[g_stackLen - 1];
    while (g_stackLen > 1) {
        area = (stackIndex - g_stack[g_stackLen - 1 - 1]) * heights[g_stack[g_stackLen - 1]];
        maxArea = GetMax(maxArea, area);
        // Pop
        g_stackLen--;
    }
    return maxArea;
}

// 计算每个连续 1 的个数
unsigned int GetMaxRectangle()
{
    int max = 0;
    int tmpMax;
    int i;

    for (i = 0; i < g_matrixSize; i++) {
        // 复用 Leet84 题结果，先取出一列的数据
        tmpMax = largestRectangleArea(g_matrix[i], g_matrixColSize);
        max = GetMax(max, tmpMax);
    }

    return max;
}
int maximalRectangle(const char **matrix, int matrixSize, const int *matrixColSize)
{
    bool rslt = MallocMatrix(matrix, matrixSize, matrixColSize);

    if (!rslt) {
        FreeMatrix();
        return 0;
    }

    InitMatrixSum(matrix, matrixSize, matrixColSize);
    int max = GetMaxRectangle();
    FreeMatrix();
    return max;
}
```

接雨水（42_困难_性能敏感）

方法 1，暴力解：

找出每根柱子左右的最高点，取两边最高柱子最小的高度，与中间柱子高度相减就是凹槽的高度。

复杂性分析

时间复杂度： $O(n2)$。数组中的每个元素都需要向左向右扫描。

空间复杂度 O(1)$O(1)$ 的额外空间。

方法 2：栈的应用

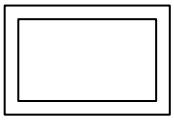
我们在遍历数组时维护一个栈。如果当前的条形块小于或等于栈顶的条形块，我们将条形块的索引入栈，意思是当前的条形块被栈中的前一个条形块界定。如果我们发现一个条形块长于栈顶，我们可以确定栈顶的条形块被当前条形块和栈的前一个条形块界定，因此我们可以弹出栈顶元素并且累加答案到 ans 。算法使用栈来存储条形块的索引下标。

遍历数组：

C++

```
int trap(vector<int>& height)
{
    int ans = 0, current = 0;
    stack<int> st;
    while (current < height.size()) {
        while (!st.empty() && height[current] > height[st.top()]) {
            int top = st.top();
            st.pop();
            if (st.empty())
                break;
            int distance = current - st.top() - 1;
            int bounded_height = min(height[current], height[st.top()]) - height[top];
            ans += distance * bounded_height;
        }
        st.push(current++);
    }
    return ans;
}
```

附 C 语言解法：大家体会下单调栈解法中的三个基本步骤！



```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
```

```
#define MAX_STACK_LEN 20003

int g_stack[MAX_STACK_LEN + 1];

int g_stackLen;


int GetMin(int x, int y)

{

    if (x >= y) {

        return y;

    }

    return x;

}

void InitStack()

{

    g_stack[0] = 0;

    g_stackLen = 0;

}


// 单调递减栈

int trap(const int *height, int heightSize)

{

    int i;

    int area = 0;

    int baseHigh;

    InitStack();

    for (i = 0; i < heightSize; i++) {
```

```
        // 步骤 1: 不满足单调递减，开始计算接水面积, 整体回溯
        while ((g_stackLen > 0) && (height[i] >= height[g_stack[g_stackLen - 1]])) {
            if (g_stackLen <= 1) {
                g_stackLen--;
                break;
            }

            baseHigh = height[g_stack[g_stackLen - 1]];
            // pop
            g_stackLen--;

            if (baseHigh == height[i]) {
                continue;
            }

            // 计算面积
            area += (GetMin(height[g_stack[g_stackLen - 1]], height[i]) - baseHigh) * (i - g_stack[g_stackLen
- 1] - 1);
        }

        // 步骤 2: 满足单调递减，入栈;
        g_stack[g_stackLen++] = i;
    }

    // 步骤 3: 对栈里面的数据进行处理: 都是单调递减性，因而不需要处理
    return area;
```

```
}


void Test1()
{
    int nums[] = {0,1,0,2,1,0,1,3,2,1,2,1};
    int numsSize = sizeof(nums) / sizeof(nums[0]);
    int rslt;
    rslt = trap(nums, numsSize);
    if (rslt == 6) {
        printf("Test1 pass \n");
    } else {
        printf("Test1 fail \n");
    }
}
void Test2()
{
    int nums[] = {0,1};
    int numsSize = sizeof(nums) / sizeof(nums[0]);
    int rslt;
    rslt = trap(nums, numsSize);
    if (rslt == 0) {
        printf("Test2 pass \n");
    } else {
        printf("Test2 fail \n");
    }
```

```
}

void Test3()
{
    int nums[] = {5,3,1,2,4,1,5};
    int numsSize = sizeof(nums) / sizeof(nums[0]);
    int rslt;
    rslt = trap(nums, numsSize);
    if (rslt == 14) {
        printf("Test3 pass \n");
    } else {
        printf("Test3 fail \n");
    }
}

void Test4()
{
    int nums[] = {5,2,1,3,6};
    int numsSize = sizeof(nums) / sizeof(nums[0]);
    int rslt;
    rslt = trap(nums, numsSize);
    if (rslt == 9) {
        printf("Test4 pass \n");
    } else {
        printf("Test4 fail \n");
```

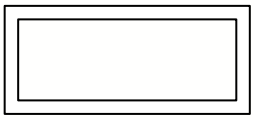
```
    }
}

void main(void)
{
    Test1();
    Test2();
    Test3();
    Test4();
    return;
}
```

股票价格跨度（901）

滑动窗口最大值（239_困难_O（n）时间复杂度解决）

最大宽度坡(962)

采用单调栈的思路，先收集峰谷，再从右往左遍历，获取目标解。O(N)时间复杂。



```
/*
* Copyright (c) Huawei Technologies Co., Ltd. 2019-2020. All rights reserved.
* Description: 962. 最大宽度坡
* Author: w00448446
* Create: 2020-03-21
*/
```

```
#include <stdio.h>
#define MAX_STACK_LEN 50002
int g_stack[MAX_STACK_LEN];
int g_stackLen;

int GetMax(int a, int b)
{
    if (a >= b) {
        return a;
    }
    return b;
}


// 先尝试收集所有的峰谷，作为初始成员, 注: 2 <= arraySize <= 50000

void InitStack(const int* array, int arraySize)
{

        // 显示将数组的第一个元素下标索引入栈

        g_stackLen = 0;
    g_stack[g_stackLen++] = 0;
    for (int i = 1; i < arraySize; i++) {

        // 如果元素比前面的元素小，则入栈

        if (array[i] < array[g_stack[g_stackLen - 1]]) {
            g_stack[g_stackLen++] = i;
        }
    }
}


// 从后向前遍历，获取最大坡长, 注: 2 <= arraySize <= 50000

int CalMaxWidthRamp(const int* array, int arraySize)
{
    int max = 0;
    for (int i = arraySize - 1; i >= 0; i--) {
        while ((g_stackLen > 0) && (array[i] >= array[g_stack[g_stackLen - 1]])) {
            max = GetMax(max, (i - g_stack[g_stackLen - 1]));
            g_stackLen--;
        }
    }
    return max;
}
```

```
// 找一个元素到峰谷的最大距离, 2 <= arraySize <= 50000
int maxWidthRamp(const int* array, int arraySize)
{
    InitStack(array, arraySize);
    int max = CalMaxWidthRamp(array, arraySize);
    return max;
}
```

九阴真经第二式 ：并查集
代表题目：朋友圈

并查集介绍

（https://blog.csdn.net/qq_41593380/article/details/81146850）江湖上散落着各式各样的大侠，有上千个之多。他们没有什么正当职业，整天背着剑在外面走来走去，碰到和自己不是一路人的，就免不了要打一架。但大侠们有一个优点就是讲义气，绝对不打自己的朋友。而且他们信奉“朋友的朋友就是我的朋友”，只要是能通过朋友关系串联起来的，不管拐了多少个弯，都认为是自己人。这样一来，江湖上就形成了一个一个的帮派，通过两两之间的朋友关系串联起来。而不在同一个帮派的人，无论如何都无法通过朋友关系连起来，于是就可以放心往死了打。但是两个原本互不相识的人，如何判断是否属于一个朋友圈呢？

我们可以在每个朋友圈内推举出一个比较有名望的人，作为该圈子的代表人物。这样，每个圈子就可以这样命名“中国同胞队”美国同胞队”……两人只要互相对一下自己的队长是不是同一个人，就可以确定敌友关系了。

但是还有问题啊，大侠们只知道自己直接的朋友是谁，很多人压根就不认识队长　　要判断自己的队长是谁，只能漫无目的的通过朋友的朋友关系问下去：“你是不是队长？你是不是队长？”这样，想打一架得先问个几十年，饿都饿死了，受不了。这样一来，队长面子上也挂不住了，不仅效率太低，还有可能陷入无限循环中。于是队长下令，重新组队。队内所有人实行分等级制度，形成树状结构，我队长就是根节点，下面分别是二级队员、三级队员。每个人只要记住自己的上级是谁就行了。遇到判断敌友的时候，只要一层层向上问，直到最高层，就可以在短时间内确定队长是谁了。由于我们关心的只是两个人之间是否是一个帮派的，至于他们是如何通过朋友关系相关联的，以及每个圈子内部的结构是怎样的，甚至队长是谁，都不重要了。所以我们可以放任队长随意重新组队，只要不搞错敌友关系就好了。于是，门派产生了。

下面我们来看并查集的实现。 int pre[1000]; 这个数组，记录了每个大侠的上级是谁。大侠们从 1 或者 0 开始编号（依据题意而定），pre[15]=3 就表示 15 号大侠的上级是 3 号大侠。如果一个人的上级就是他自己，那说明他就是掌门人了，查找到此为止。也有孤家寡人自成一派的，比如欧阳锋，那么他的上级就是他自己。每个人都只认自己的上级。比如胡青牛同学只知道自己的上级是杨左使。张无忌是谁？不认识！要想知道自己的掌门是谁，只能一级级查上去。

find 这个函数就是找掌门用的，意义再清楚不过了（路径压缩算法先不论，后面再说）。

```
int unionsearch(int root) //查找根结点
{
    int son, tmp;
    son = root;
```

```
while(root != pre[root]) //我的上级不是掌门
    root = pre[root];
while(son != root) //我就找他的上级，直到掌门出现
{
    tmp = pre[son];
    pre[son] = root;
    son = tmp;
}
return root; //掌门驾到~~
```

再来看看 join 函数，就是在两个点之间连一条线，这样一来，原先它们所在的两个板块的所有点就都可以互通了。这在图上很好办，画条线就行了。但我们现在是用并查集来描述武林中的状况的，一共只有一个 pre[]数组，该如何实现呢？ 还是举江湖的例子，假设现在武林中的形势如图所示。虚竹帅锅与周芷若 MM 是我非常喜欢的两个人物，他们的终极 boss 分别是玄慈方丈和灭绝师太，那明显就是两个阵营了。我不希望他们互相打架，就对他俩说：“你们两位拉拉勾，做好朋友吧。”他们看在我的面子上，同意了。这一同意可非同小可，整个少林和峨眉派的人就不能打架了。这么重大的变化，可如何实现呀，要改动多少地方？其实非常简单，我对玄慈方丈说：“大师，麻烦你把你的上级改为灭绝师太吧。这样一来，两派原先的所有人员的终极 boss 都是师太，那还打个球啊！ 反正我们关心的只是连通性，门派内部的结构不要紧的。”玄慈一听肯定火大了：“我靠，凭什么是我变成她手下呀，怎么不反过来？我抗议！”于是，两人相约一战，杀的是天昏地暗，风云为之变色啊，但是啊，这场战争终究会有胜负，胜者为王。弱者就被吞并了。反正谁加入谁效果是一样的，门派就由两个变成一个了。这段函数的意思明白了吧？

```
void join(int root1, int root2) //虚竹和周芷若做朋友
{
    int x, y;
    x = unionsearch(root1);//我老大是玄慈
```

```
    y = unionsearch(root2);//我老大是灭绝
    if(x != y)
        pre[x] = y; //打一仗，谁赢就当对方老大
}
```

再来看看路径压缩算法。建立门派的过程是用 join 函数两个人两个人地连接起来的，谁当谁的手下完全随机。最后的树状结构会变成什么样，我也无法预知，一字长蛇阵也有可能。这样查找的效率就会比较低下。最理想的情况就是所有人的直接上级都是掌门，一共就两级结构，只要找一次就找到掌门了。哪怕不能完全做到，也最好尽量接近。这样就产生了路径压缩算法。

设想这样一个场景：两个互不相识的大侠碰面了，想知道能不能干一场。 于是赶紧打电话问自己的上级："你是不是掌门？" 上级说："我不是呀，我的上级是谁谁谁，你问问他看看。" 一路问下去，原来两人的最终 boss 都是东厂曹公公。 "哎呀呀，原来是自己人，有礼有礼，在下三营六组白面葫芦娃!" "幸会幸会，在下九营十八组仙子狗尾巴花！" 两人高高兴兴地手拉手喝酒去了。 "等等等等，两位大侠请留步，还有事情没完成呢！"我叫住他俩。 "哦，对了，还要做路径压缩。"两人醒悟。 白面葫芦娃打电话给他的上级六组长："组长啊，我查过了，其实偶们的掌门是曹公公。不如偶们一起结拜在曹公公手下吧，省得级别太低，以后查找掌门麻烦。" "唔，有道理。" 白面葫芦娃接着打电话给刚才拜访过的三营长……仙子狗尾巴花也做了同样的事情。 这样，查询中所有涉及到的人物都聚集在曹公公的直接领导下。每次查询都做了优化处理，所以整个门派树的层数都会维持在比较低的水平上。路径压缩的代码，看得懂很好，看不懂可以自己模拟一下，很简单的一个递归而已。总之它所实现的功能就是这么个意思。

触类旁通
朋友圈的参考解法（547）

```
/*
 * Copyright (c) Huawei Technologies Co., Ltd. 2019-2019. All rights reserved.
 * Description: 547. 朋友圈
 * Author: w00448446
 * Create: 2019-11-27
 */
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <ctype.h>
#include <math.h>

int *g_dest;

bool Init(int mSize)
```

```
{
    int i;
    if (mSize < 1) {
        return false;
    }
    g_dest = (int *)malloc(mSize * sizeof(int));
    if (g_dest == NULL) {
        return false;
    }

    for (i = 0; i < mSize; i++) {
        g_dest[i] = i;
    }

    return true;
}

int Find(int index)
{
        if (g_dest[index] == index) {
                return index;
        }
        return g_dest[index] = Find(g_dest[index]);
}

int FindRoot(int i)
{
    while (g_dest[i] != i) {
        i = g_dest[i];
    }

    return i;
}
void ProcCircle(int **m, int mSize)
{
    int i, j;
    int rootI, rootJ;
    for (i = 0; i < mSize; i++) {
        for (j = (i + 1); j < mSize; j++) {
            if (m[i][j] != 1) {
                continue;
            }
            rootI = FindRoot(i);
```

```
            rootJ = FindRoot(j);
            if (rootI == rootJ) {
                continue;
            }
            g_dest[rootI] = rootJ;
        }
    }

    return;
}
int GetCircleNum(int mSize)
{
    int i;
    int sum = 0;
    for (i = 0; i < mSize; i++) {
        if (g_dest[i] == i) {
            sum++;
        }
    }
    return sum;
}
void FreeCircle()
{
    if (g_dest != NULL) {
        free(g_dest);
        g_dest = 0;
    }
}
int findCircleNum(int **m, int mSize, int* mColSize)
{
    bool rslt;
    int sum;

    rslt = Init(mSize);
    if (!rslt) {
        return 0;
    }

    ProcCircle(m, mSize);
    sum = GetCircleNum(mSize);

    FreeCircle();
    return sum;
```

}

冗余连接（684）

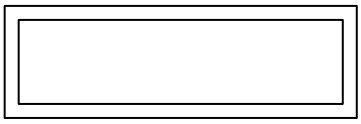
岛屿数量（200_必做）

```
/*
* Copyright (c) Huawei Technologies Co., Ltd. 2012-2019. All rights reserved.
* Description: 200. 岛屿数量
* Author: w00448446
* Create: 2019-10-27
 */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include <ctype.h>
#include "securec.h"
#define POINT_MAX_VALUE 10000
#define POINT_MIN_VALUE (-10000)
#define POINT_MAX_NUM 2
int g_direction[][POINT_MAX_NUM] = { { -1, 0 }, { 0, 1 }, { 1, 0 }, { 0, -1 } };
int g_len = (int)sizeof(g_direction) / sizeof(g_direction[0]);
int GetMax(int a, int b)
{
    if (a >= b) {
        return a;
    }
    return b;
}
void Dfs(char **grid, int gridSize, int *gridColSize, int x, int y)
{
    int i;
    int nextX, nextY;
    grid[x][y] = '2';
```

```
    for (i = 0; i < g_len; i++) {
        nextX = (int) (x + g_direction[i][0]);
        nextY = (int) (y + g_direction[i][1]);
        if ((nextX >= 0) && (nextX < gridSize) && (nextY >= 0) && (nextY < gridColSize[0])) {
            if (grid[nextX][nextY] == '2') {
                continue;
            }

            if (grid[nextX][nextY] == '0') {
                continue;
            }

            Dfs(grid, gridSize, gridColSize, nextX, nextY);
        }
    }
}
int numIslands(char **grid, int gridSize, int *gridColSize)
{
    int i, j;
    int sum = 0;

    for (i = 0; i < gridSize; i++) {
        for (j = 0; j < gridColSize[0]; j++) {
            if (grid[i][j] == '0') {
                continue;
            }

            if (grid[i][j] == '2') {
                continue;
            }

            if (grid[i][j] == '1') {
                sum++;
                Dfs(grid, gridSize, gridColSize, i, j);
            }
        }
    }
    return sum;
}

void TestCase1()
{
```

```
    int rslt;
    char array1[] = "11110";
    char array2[] = "11010";
    char array3[] = "11000";
    char array4[] = "00000";
    char *grid[] = { array1, array2, array3, array4 };
    int gridSize = sizeof(grid) / sizeof(char *);
    int gridColSize[] = { 5, 5, 5, 5 };
    rslt = numIslands(grid, gridSize, gridColSize);
    if (rslt == 1) {
        printf("T1:succ");
    }
}

void TestCase2()
{
    int rslt;
    char array1[] = "11000";
    char array2[] = "11000";
    char array3[] = "00100";
    char array4[] = "00011";
    char *grid[] = { array1, array2, array3, array4 };
    int gridSize = sizeof(grid) / sizeof(char *);
    int gridColSize[] = { 5, 5, 5, 5 };
    rslt = numIslands(grid, gridSize, gridColSize);
    if (rslt == 3) {
        printf("T2:succ");
    }
}

int main()
{
    TestCase2();
    int i;
    if (scanf_s("%d", &i) != 0) {
    }
    return 0;
}
```

句子相似性 II (737_会员)



```
/*
 * Copyright (c) Huawei Technologies Co., Ltd. 2012-2018. All rights reserved.
 * Description: 737. 句子相似性 II
 * Author: w00448446
 * Create: 2019-11-27
 */
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <ctype.h>
#include <math.h>

#define MAX_WORDS_NUM 4010

typedef struct {
    int pre;
    char *words;
} Word;

Word g_pair[MAX_WORDS_NUM];
int g_pairLen;

int PairCmp(const void *w1, const void *w2)
{
    return strcmp(((Word*)w1)->words, ((Word*)w2)->words);
}
```

```
int GetRoot(int i)
{
    while (g_pair[i].pre != i) {
        i = g_pair[i].pre;
    }
    return i;
}
void Init(char ***pairs, int pairsSize)
{
    int i, j;
    int pre1, pre2;
    Word pair1, pair2;
    Word *word1 = NULL;
    Word *word2 = NULL;

    g_pairLen = 0;
    if (pairsSize < 1) {
        return;
    }

    for (i = 0; i < pairsSize; i++) {
        g_pair[g_pairLen++].words = pairs[i][0];
        g_pair[g_pairLen++].words = pairs[i][1];
    }

    // 排序
    qsort(g_pair, g_pairLen, sizeof(Word), PairCmp);

    // 去除重复
    for (i = 1, j = 0; i < g_pairLen; i++) {
        if (strcmp(g_pair[i].words, g_pair[j].words) == 0) {
            continue;
        }
        j++;
        g_pair[j].words = g_pair[i].words;
    }
    g_pairLen = j + 1;

    // 初始化 pre 值
    for (i = 0; i < g_pairLen; i++) {
        g_pair[i].pre = i;
    }
```

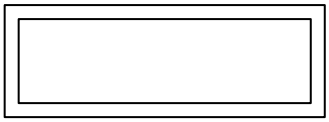
```
    // 根据 pair,生成映射
    for (i = 0; i < pairsSize; i++) {
        pair1.words = pairs[i][0];
        pair2.words = pairs[i][1];
        word1 = bsearch(&pair1, g_pair, g_pairLen, sizeof(Word), PairCmp);
        word2 = bsearch(&pair2, g_pair, g_pairLen, sizeof(Word), PairCmp);
        if ((word1 == NULL) || (word2 == NULL)) {
            printf("search fail, not expect!\n");
            continue;
        }

        pre1 = GetRoot(word1->pre);
        pre2 = GetRoot(word2->pre);
        if (pre1 == pre2) {
            continue;
        }
        g_pair[pre1].pre = pre2;
    }
    return ;
}

bool IsSimilar(char **words1, int words1Size, char **words2, int words2Size)
{
    int i;
    int pre1, pre2;
    Word pair1, pair2;
    Word *word1 = NULL;
    Word *word2 = NULL;
    if (words1Size != words2Size) {
        return false;
    }

    for (i = 0; i < words1Size; i++) {
        // 查找
        pair1.words = words1[i];
        pair2.words = words2[i];
        word1 = bsearch(&pair1, g_pair, g_pairLen, sizeof(Word), PairCmp);
        word2 = bsearch(&pair2, g_pair, g_pairLen, sizeof(Word), PairCmp);
        if ((word1 == NULL) || (word2 == NULL)) {
            if (strcmp(words1[i], words2[i]) == 0) {
                continue;
            }
            return false;
```

```
        }

        pre1 = GetRoot(word1->pre);
        pre2 = GetRoot(word2->pre);
        if (pre1 == pre2) {
            continue;
        }

        return false;
    }

    return true;
}
bool areSentencesSimilarTwo(char **words1, int words1Size, char **words2, int words2Size,
                            char ***pairs, int pairsSize, int *pairsColSize)
{
    Init(pairs, pairsSize);
    bool rslt = IsSimilar(words1, words1Size, words2, words2Size);
    return rslt;
}

void Test1()
{
    char *word1[] = { "great", "acting",  "skills" };
    char *word2[] = { "fine", "drama",  "talent" };
    char *pair1[] = { "great", "fine" };
    char *pair2[] = { "acting", "drama" };
    char *pair3[] = { "skills", "talent" };
    char **g[] = { pair1,         pair2,          pair3 };

    int word1Size = sizeof(word1) / sizeof(word1[0]);
    int word2Size = sizeof(word2) / sizeof(word2[0]);
    int pairsSize = sizeof(g) / sizeof(g[0]);
    int pairsColSize[] = { 2 };
    int rslt;
    rslt = areSentencesSimilarTwo(word1, word1Size, word2, word2Size, g, pairsSize, pairsColSize);
    if (rslt == 1) {
        printf("t1: succ\n");
    }
}
void main()
{
    Test1();
```

```
    return;
}
```

得分最高的路径（1102_会员）

使用并查集的一种解法：
将所有路径数据按照从大到小排序；
然后建立圈子：依次加入数据，如果出现了从起始点到结束点的圈子，则说明找到目标解。



```
/*
 * Copyright (c) Huawei Technologies Co., Ltd. 2019-2019. All rights reserved.
 * Description: 1102. 得分最高的路径
 * Author: w00448446
 * Create: 2019-12-13
 */
#include <stdio.h>
#include <stdlib.h>
```

```
#include <stdbool.h>

#define MAX_NUM          10006
#define DIRECTION_NODE_NUM 2
#define DIRECTION_LEN     4

typedef struct {
    int currIndex;
    int preIndex;
    int marked;
    int val;
} Node;

int g_direction[][DIRECTION_NODE_NUM] = { { -1, 0 }, { 1, 0 }, { 0, -1 }, { 0, 1 } };

int g_col;
int g_row;
Node g_node[MAX_NUM];
Node g_sortNode[MAX_NUM];
int g_num;

int NodeCmp(const void *node1, const void *node2)
{
    return (((Node *)node2)->val) - (((Node *)node1)->val);
}
void InitNode(const int **a, int aSize, const int *aColSize)
{
    int i, j;
    g_col = aColSize[0];
    g_row = aSize;
    g_num = 0;
    for (i = 0; i < aSize; i++) {
        for (j = 0; j < g_col; j++) {
            g_node[g_num].val = a[i][j];
            g_node[g_num].currIndex = i * g_col + j;
            g_node[g_num].preIndex = g_node[g_num].currIndex;
            g_node[g_num].marked = 0;
            g_sortNode[g_num] = g_node[g_num];
            g_num++;
        }
    }
    // 从大到小排序
    qsort(g_sortNode, g_num, sizeof(Node), NodeCmp);
```

```
}

bool IsNodeValid(int i, int j)
{
    if (i < 0) {
        return false;
    }

    if (i >= g_row) {
        return false;
    }

    if (j < 0) {
        return false;
    }

    if (j >= g_col) {
        return false;
    }

    return true;
}

int FindRoot(int index)
{
    if (index == g_node[index].preIndex) {
        return index;
    }
    return g_node[index].preIndex = FindRoot(g_node[index].preIndex);
}

void UnionNode(int index)
{
    int i;
    int currI, currJ, currRoot;
    int nextI, nextJ, nextMap, nextRoot;

    currI = g_node[index].currIndex / g_col;
    currJ = g_node[index].currIndex % g_col;
    for (i = 0; i < DIRECTION_LEN; i++) {
        nextI = currI + g_direction[i][0];
        nextJ = currJ + g_direction[i][1];
```

```
        if (!IsNodeValid(nextI, nextJ)) {
            continue;
        }
        // 如果有效点, 找到父亲节点进行并操作
        nextMap = nextI * g_col + nextJ;
        if (g_node[nextMap].marked == 0) {
            continue;
        }

        nextRoot = FindRoot(nextMap);
        currRoot = FindRoot(index);
        if (nextRoot == currRoot) {
            continue;
        }

        if (nextRoot <= currRoot) {
            g_node[currRoot].preIndex = nextRoot;
            continue;
        }

        // 进行并操作
        g_node[nextRoot].preIndex = currRoot;
    }
}

int Traverse()
{
    int i, index;
    int startValid = 0;
    const int START_INDEX = 0;
    int startRoot;
    int endValid = 0;
    int endIndex = g_num - 1;
    int endRoot;

    for (i = 0; i < g_num; i++) {
        // 查找上下左右，进行归并
        index = g_sortNode[i].currIndex;
        g_node[index].marked = 1;
        UnionNode(index);

        if (index == START_INDEX) {
            startValid = 1;
```

```
        }

        if (index == endIndex) {
            endValid = 1;
        }

        if ((startValid == 0) || (endValid == 0)) {
            continue;
        }

        // 检测起始和结束点是否在一个圈子里面，若是则找到目标解
        startRoot = FindRoot(START_INDEX);
        endRoot = FindRoot(endIndex);
        if (endRoot == startRoot) {
            return g_sortNode[i].val;
        }
    }
    return g_sortNode[0].val;
}
int maximumMinimumPath(const int **a, int aSize, const int *aColSize)
{
    InitNode(a, aSize, aColSize);
    int max = Traverse();
    return max;
}

void Test1()
{
    int m1[] = { 5,  4,  5 };
    int m2[] = { 1,  2,  6 };
    int m3[] = { 7,  4,  6 };
    int *g[] = { m1, m2, m3 };
    int len = sizeof(g) / sizeof(g[0]);
    int col[] = { 3, 3, 3 };
    int rslt = maximumMinimumPath(g, len, col);
    if (rslt == 4) {
        printf("T1: pass\n");
    } else {
        printf("T1: fail\n");
    }
}

void Test2()
```

```
{
    int m1[] = { 2,  2, 1, 2, 2, 2 };
    int m2[] = { 1,  2, 2, 2, 1, 2 };
    int *g[] = { m1, m2 };
    int len = sizeof(g) / sizeof(g[0]);
    int col[] = { 6, 6, 6 };
    int rslt = maximumMinimumPath(g, len, col);
    if (rslt == 2) {
        printf("T2: pass\n");
    } else {
        printf("T2: fail\n");
    }
}
void main(void)
{
    Test2();
    Test1();
    return;
}
```

最低成本联通所有城市（1135_会员）

最小生成树
（https://blog.csdn.net/luoshixian099/article/details/51908175）
关于图的几个概念定义：

连通图：在无向图中，若任意两个顶点 vivi 与 vjvj 都有路径相通，则称该无向图为连通图。
强连通图：在有向图中，若任意两个顶点 vivi 与 vjvj 都有路径相通，则称该有向图为强连通图。
连通网：在连通图中，若图的边具有一定的意义，每一条边都对应着一个数，称为权；权代表着连接连个顶点的代价，称这种连通图叫做连通网。
生成树：一个连通图的生成树是指一个连通子图，它含有图中全部 n 个顶点，但只有足以构成一棵树的 n-1 条边。一颗有 n 个顶点的生成树有且仅有 n-1 条边，如果生成树中再添加一条边，则必定成环。
最小生成树：在连通网的所有生成树中，所有边的代价和最小的生成树，称为最小生成树。

以图辨树（261_会员）

```
/*
* Copyright (c) Huawei Technologies Co., Ltd. 2012-2019. All rights reserved.
* Description: 261. 以图判树 securec.h
* Author: w00448446
* Create: 2019-11-14
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#define MAX_VERTEX_NUM 10000
int g_vertex[MAX_VERTEX_NUM];
int GetMin(int a, int b)
{
    if (a <= b) {
        return a;
    }
    return b;
}
int FindRoot(int a)
{
    while (g_vertex[a] != a) {
        a = g_vertex[a];
    }

    return a;
}
bool validTree(int n, int** edges, int edgesSize, int* edgesColSize)
{
    int i;
    int a, b;
    if (edgesSize != (n - 1)) {
        return false;
```

```
    }

    if (edgesSize == 0 && n == 1) {
        return true;
    }

    for (i = 0; i < n; i++) {
        g_vertex[i] = i;
    }

    for (i = 0; i < edgesSize; i++) {
        a = FindRoot(edges[i][0]);
        b = FindRoot(edges[i][1]);

        if (a == b) {
            return false;
        }

        if (edges[i][0] < edges[i][1]) {
            g_vertex[edges[i][1]] = a;
        } else {
            g_vertex[edges[i][0]] = b;
        }
    }
    return true;
}
void TestCase1()
{
    int m1[] = {0, 1};
    int m2[] = {0, 2};
    int m3[] = {0, 3};
    int m4[] = {1, 4};
    int *g[] = {m1, m2, m3, m4};
    int length = sizeof(g) / sizeof(g[0]);
    int col[] = {2,2,2,2};
    int rslt;
    int n = 5;
    rslt = validTree(n, g, length, col);
    if (rslt == 1) {
        printf("t1: pass\n");
    }
}
void main()
```

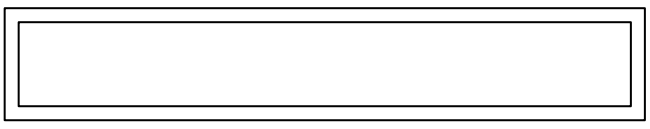
```
{
    TestCase1();
}
```

按字典序排列最小的等效字符串（1061_会员）

无向图中连通分量的数目（323_会员）

尽量减少恶意软件的传播（924_困难）

注：将 initial 视为临时圈长，那么所拥有最大个数的圈，就是目标解。



```
#include <stdlib.h>
#include <string.h>
#include "securec.h"
#define MAX_LEN 301

int g_preNode[MAX_LEN];
int g_hashInfo[MAX_LEN];

int FindRoot(int pos)
{
    int son = pos;
    int temp;

    if (g_preNode[pos] == -1) {
        return pos;
    }

    while (g_preNode[pos] != -1) {
        pos = g_preNode[pos];
    }

    while (g_preNode[son] != -1) {
        temp = g_preNode[son];
        g_preNode[son] = pos;
        son = temp;
    }

    return pos;
}

void UnionRoot(int x, int y)
{
    int a = FindRoot(x);
    int b = FindRoot(y);

    if (a != b) {
        g_preNode[a] = b;
    }
}

int minMalwareSpread(int **graph, int graphSize, int *graphColSize, int *initial, int initialSize)
{
    int temp;
```

```
    int max = 0;
    int result;

    if (graphSize <= 0 || initialSize <= 0) {
        return 0;
    }

    result = initial[0];
    (void)memset_s(g_preNode, MAX_LEN * sizeof(int), -1, MAX_LEN * sizeof(int));
    (void)memset_s(g_hashInfo, MAX_LEN * sizeof(int), 0, MAX_LEN * sizeof(int));


    // step1：建圈

    for (int i = 0; i < graphSize; i++) {
        for (int j = 0; j < graphColSize[i]; j++) {
            if (graph[i][j] == 1) {
                UnionRoot(i, j);
            }
        }
    }


    // step2：统计各圈长下挂节点数

    for (int i = 0; i < graphSize; i++) {
        temp = FindRoot(i);
        if (temp != -1) {
            g_hashInfo[temp]++;
        }
    }


    // step3：获取 initial 列表中节点所属圈中圈长下挂节点数最大的节点

    for (int i = 0; i < initialSize; i++) {
        temp = FindRoot(initial[i]);
        if ((temp != -1) && ((max < g_hashInfo[temp]) || ((max == g_hashInfo[temp]) && initial[i] < result))) {
            max = g_hashInfo[temp];
            result = initial[i];
        }
    }

    return result;
}
void main(void)
```

```
{
    return;
}
```

九阴真经第三式 ：滑动窗口
代表题目：尽可能使字符串相等（1208）

滑动窗口介绍

（https://zhuanlan.zhihu.com/p/61564531）滑动窗口法，也叫尺取法（可能也不一定相等，大概就是这样 =。=），可以用来解决一些查找满足一定条件的连续区间的性质（长度等）的问题。由于区间连续，因此当区间发生变化时，可以通过旧有的计算结

果对搜索空间进行剪枝，这样便减少了重复计算，降低了时间复杂度。往往类似于“请找到满足 xx 的最 x 的区间（子串、子数组）的 xx”这类问题都可以使用该方法进行解决。

Leetcode 209. 长度最小的子数组

给定一个含有 n 个正整数的数组和一个正整数 s ，找出该数组中满足其和 ≥ s 的长度最小的连续子数组。如果不存在符合条件的连续子数组，返回 0。示例:

输入: s = 7, nums = [2,3,1,2,4,3]

输出: 2
解释: 子数组 [4,3] 是该条件下的长度最小的连续子数组。

Leetcode 3. 无重复字符的最长子串

给定一个字符串，请你找出其中不含有重复字符的 最长子串 的长度。

示例 1:

输入: "abcabcbb"

输出: 3

解释: 因为无重复字符的最长子串是 "abc"，所以其长度为 3。

示例 2:

输入: "bbbbb"

输出: 1

解释: 因为无重复字符的最长子串是 "b"，所以其长度为 1。

示例 3:

输入: "pwwkew"

输出: 3

解释: 因为无重复字符的最长子串是 "wke"，所以其长度为 3。
请注意，你的答案必须是 子串 的长度，"pwke" 是一个子序列，不是子串。

Leetcode 1004. 最大连续 1 的个数 III

给定一个由若干 0 和 1 组成的数组 A，我们最多可以将 K 个值从 0 变成 1 。

返回仅包含 1 的最长（连续）子数组的长度。

示例 1：

输入：A = [1,1,1,0,0,0,1,1,1,1,0], K = 2

输出：6

解释：

[1,1,1,0,0,1,1,1,1,1,1]

粗体数字从 0 翻转到 1，最长的子数组长度为 6。

示例 2：

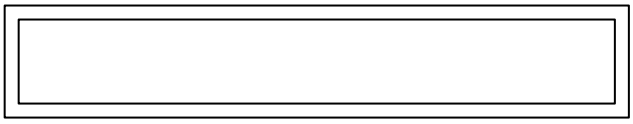
输入：A = [0,0,1,1,0,0,1,1,1,0,1,1,0,0,0,1,1,1,1], K = 3

输出：10

解释：

[0,0,1,1,1,1,1,1,1,1,1,1,0,0,0,1,1,1,1]
粗体数字从 0 翻转到 1，最长的子数组长度为 10。

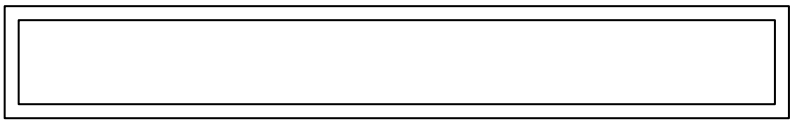
触类旁通
尽可能使字符串相等（1208）的一种解法



```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
#define MAX_STRING_LEN 100001
int g_num;
char g_stringValue[MAX_STRING_LEN];
void InitStringValue(const char *s, const char *t)
{
    char *tmpS = s;
    char *tmpT = t;

    g_num = 0;
    while (*tmpS != 0) {
        g_stringValue[g_num] = abs(*tmpS - *tmpT);
        g_num++;
        tmpS++;
        tmpT++;
    }
    return;
}
int equalSubstring(char *s, char *t, int maxCost)
{
    int i, j;
    int sum = 0;
    int maxCount = 0;
    int begin = 0;
```

```
    InitStringValue(s, t);
    for (i = 0; i < g_num; i++) {
        sum += g_stringValue[i];
        if (sum > maxCost) {
            maxCount = (maxCount >= (i - begin)) ? maxCount : (i - begin);
            sum -= g_stringValue[begin];
            begin++;
        }

    }

    maxCount = (maxCount >= (i - begin)) ? maxCount : (i - begin);
    return maxCount;
}
```

340. 至多包含 K 个不同字符的最长子串（会员_困难）



```
/*
 * Copyright (c) Huawei Technologies Co., Ltd. 2012-2018. All rights reserved.
 * Description: 项目 LEET_340_LongestSubstringforKDiffChar 的源文件
 * Author: f00485759
 * Create: 2019-12-16
 */
#include <stdio.h>
#include <stdbool.h>
#include "securec.h"
#define MAX_HASH_NUM 256
#define MAX(a, b) ((a) > (b) ? (a) : (b))
int g_hash[MAX_HASH_NUM];
void Init(void)
```

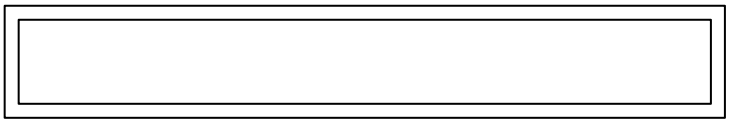
```
{
    (void)memset(g_hash, 0, sizeof(g_hash));
}
void HashChar(char c)
{
    g_hash[c]++;
}
void DeHashChar(char c)
{
    if (g_hash[c] > 0) {
        g_hash[c]--;
    }
}
bool IsExtendKChar(int k)
{
    int count = 0;
    for (int i = 0; i < MAX_HASH_NUM; i++) {
        if (g_hash[i] != 0) {
            count++; // 可以通过变量来记忆，减少计算量
        }
        if (count >= k) {
            return true;
        }
    }

    return false;
}

bool IsRepeat(char c)
{
    if (g_hash[c] != 0) {
        return true;
    }

    return false;
}
int lengthOfLongestSubstringKDistinct(const char *s, int k)
{
    int start, end, maxlen;
    int slen = (int)strlen(s);
    Init();
    start = 0;
    maxlen = 0;
```

```
    for (end = 0; end < slen; end++) {
        if (!IsExtendKChar(k) || IsRepeat(s[end])) {
            HashChar(s[end]);
            continue;
        }

        if (start == end) {
            break;
        }

        maxlen = MAX(maxlen, end - start);
        DeHashChar(s[start]);
        start++;

        while (IsExtendKChar(k)) {
            DeHashChar(s[start]);
            start++;
        }
        HashChar(s[end]);
    }
    maxlen = MAX(maxlen, end - start);
    return maxlen;
}
int main()
{
    const char *s = "a";
    int k = 0;

    int maxlen = lengthOfLongestSubstringKDistinct(s, k);

    printf("%d\n", maxlen);

    return 0;
}
```

1151. 最少交换次数来组合所有的 1（会员）

159. 至多包含两个不同字符的最长子串（会员）

1100. 长度为 K 的无重复字符子串（会员）



```
#define MAX_LEN 512
int g_array[MAX_LEN];

int numKLenSubstrNoRepeats(char * s, int k){
    int right = 0;
    int left = 0;
    int result = 0;
    int count = 0;
    int len;

    len = strlen(s);
    memset(g_array, 0, sizeof(g_array));

    while (right < len) {
        if (g_array[s[right]] == 0) {
            count++;
        }
        g_array[s[right]]++;
        while (right - left + 1 > k) {
            g_array[s[left]]--;
            if (g_array[s[left]] == 0) {
                count--;
            }
            left++;
        }
```

```
            printf("%d, %d, %d\n", right, left, count);
            if ((right - left + 1 == k) && (count == k)) {
                result++;
            }

            right++;
        }

        return result;
    }
```

九阴真经第四式 ：前缀和 & HASH
代表题目：560. 和为 K 的子数组

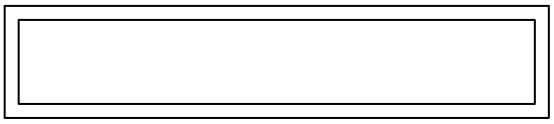
注：前缀和+hash，可以在 O(n)复杂度来解决此问题。

前缀和介绍

前缀和（Prefix Sum）定义（https://www.jianshu.com/p/3021429f38d4）：
给定一个数组 A[1..n]，前缀和数组 PrefixSum[1..n]定义为：PrefixSum[i] = A[0]+A[1]+...+A[i-1]；
例如：A[5,6,7,8] --> PrefixSum[5,11,18,26]
PrefixSum[0] =A[0] ;
PrefixSum[1] =A[0] + A[1] ;
PrefixSum[2] =A[0] + A[1] + A[2] ;
PrefixSum[3] =A[0] + A[1] + A[2] + A[3] = PrefixSum[2] + A[3] ;

触类旁通
523. 连续的子数组和

注：《= O(n^2)复杂度解决



// 使用 Hash O（n）复杂度，因没有进行动态内存申请，所以空间占用较大，留给大家进一步优化

```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>

#define MAX_HASH_LEN 20001
#define MAX_HASH_NODE_LEN (MAX_HASH_LEN + MAX_HASH_LEN + 1)

typedef struct {
    int valid;
    int value;
    int count;
} HashNode;

HashNode g_hashMap[MAX_HASH_NODE_LEN];
int g_max;

int GetHashCode(int val)
{
```

```
    int hashCode = (abs(val) * 131) % MAX_HASH_LEN;
    return hashCode;
}

int GetHashMapIndex(int val)
{
    int hashCode;
    hashCode = GetHashCode(val);
    for (int i = hashCode; i < MAX_HASH_NODE_LEN; i++) {
        if (g_hashMap[i].valid == 0) {
            return i;
        }

        if (g_hashMap[i].value == val) {
            return i;
        }
    }
    // 异常保护
    printf("GetHashMapIndex:error\n");
    return 0;
}

void UpdateHashMap(int index, int val)
{
    if (g_hashMap[index].valid == 0) {
        g_hashMap[index].valid = 1;
        g_hashMap[index].value = val;
        g_hashMap[index].count = 1;
        return;
    }

    if (g_hashMap[index].value != val) {
        printf("UpdateHashMap: error\n");
        return;
    }

    g_hashMap[index].count++;
    return ;
}

void Init()
{
    g_max = 0;
```

```
    for (int i = 0; i < MAX_HASH_NODE_LEN; i++) {
        g_hashMap[i].valid = 0;
        g_hashMap[i].count = 0;
        g_hashMap[i].value = 0;
    }

    // 将 0 初始化为有效值，但初始计数为 0
    g_hashMap[0].valid = 1;
    g_hashMap[0].count = 1;
}
int subarraySum(int* nums, int numsSize, int k)
{
    int index;
    int sum = 0;
    int preSum = 0;

    Init();
    for (int i = 0; i < numsSize; i++) {
        sum += nums[i];

        // 获取 preSum 的个数
        preSum = sum - k;
        index = GetHashMapIndex(preSum);
        if (g_hashMap[index].valid == 1) {
            g_max += g_hashMap[index].count;
        }

        // 获取 HashIndex
        index = GetHashMapIndex(sum);

        // 放入 Hash 表中
        UpdateHashMap(index, sum);
    }

    return g_max;
}

void Test1()
{
    int nums[] = {1,1,1};
    int numsSize = sizeof(nums) / sizeof(nums[0]);
    int k = 2;
    int rslt;
```

```
    rslt = subarraySum(nums, numsSize, k);
    if (rslt == 2) {
        printf("Test1 pass \n");
    } else {
        printf("Test1 fail \n");
    }
}
void main()
{
    Test1();
    return;
}
```

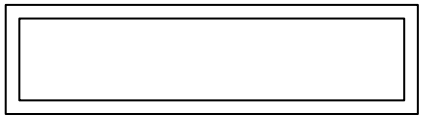
974. 和可被 K 整除的子数组

九阴真经第五式 ：差分
代表题目：**1094. 拼车**

差分介绍
对于数组 array[N]中的某一段进行增减操作，通过差分可在 O（n）时间内完成。如

```
trips = [[2,1,5],[3,3,7]]
```

第一步：更新 array[1] = 2, array[5] = -2;
第二步：更新 array[3] = 3, array[7] = -3;
第三步：进行求和,得到结果 array[] = {0,2,2,5,5,3,3,0};

一种解法： 

```
#include <stdio.h>

#define MAX_NUM 1001


int g_locationInfo[MAX_NUM]; /* 用于存放每个位置的人数 */


bool carPooling(int** trips, int tripsSize, int* tripsColSize, int capacity)
{
    int temp;
    int i;
    int j;

    if (trips == NULL || tripsSize <= 0 || tripsColSize == NULL || capacity <= 0) {
        return false;
    }

    memset(g_locationInfo, 0, MAX_NUM * sizeof(int));

    for (i = 0; i < tripsSize; i++) {
        temp = tripsColSize[i] - 1;

        for (j = trips[i][1]; j < trips[i][temp]; j++) { /* 最后一站为下车站 */

            g_locationInfo[j] += trips[i][0];
        }
    }

    for (i = 0; i < MAX_NUM; i++) {
        if (g_locationInfo[i] > capacity) {
            return false;
        }
    }

    return true;
}
```

触类旁通

1109. 航班预订统计

121. 买卖股票的最佳时机(简单)

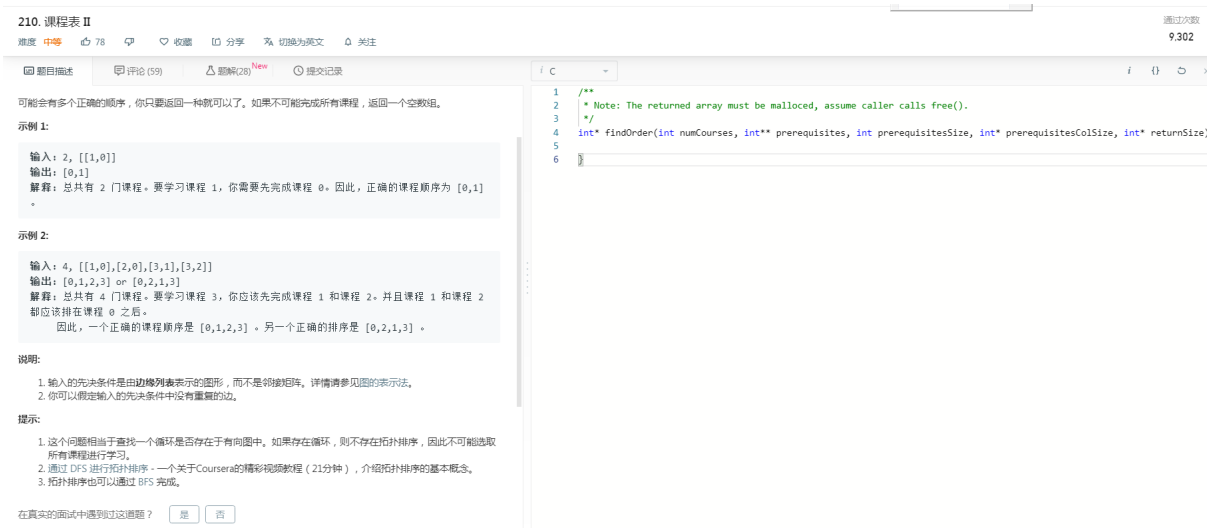
122. 买卖股票的最佳时机 II

```
int maxProfit(int *prices, int pricesSize)
{
    int purchasePrice;
    int profit = 0;
    if (prices == NULL || pricesSize <= 0) {
        return 0;
    }
    purchasePrice = prices[0];
    for (int i = 1; i < pricesSize; i++) {
        if (purchasePrice < prices[i]) {
            profit += (prices[i] - purchasePrice);
        }
        purchasePrice = prices[i];
    }
    return profit;
}
```

253. 会议室 II （会员）

```
#include <stdio.h>
#include <securec.h>
#define MAX_LEN 1000000
int g_hashInfo[MAX_LEN];
int minMeetingRooms(int **intervals, int intervalsSize, int *intervalsColSize)
{
    int max = 0;
    int sum = 0;
    if (intervals == NULL || intervalsSize <= 0 || intervalsColSize == NULL) {
        return 0;
    }
    (void)memset_s(g_hashInfo, MAX_LEN * sizeof(int), 0, MAX_LEN * sizeof(int));
    for (int i = 0; i < intervalsSize; i++) {
        g_hashInfo[intervals[i][0]]++;
        g_hashInfo[intervals[i][1]]--;
    }
    for (int i = 0; i < MAX_LEN; i++) {
        sum += g_hashInfo[i];
        if (max < sum) {
            max = sum;
        }
    }
    return max;
}
```

九阴真经第六式 ：拓扑排序（专业级）
代表题目：210. 课程表 II



拓扑排序介绍
（https://blog.csdn.net/lisonglisonglisong/article/details/45543451）
一、什么是拓扑排序
在图论中，拓扑排序（Topological Sorting）是一个有向无环图（DAG, Directed Acyclic Graph）的所有顶点的线性序列。且该序列必须满足下面两个条件：
每个顶点出现且只出现一次。
若存在一条从顶点 A 到顶点 B 的路径，那么在序列中顶点 A 出现在顶点 B 的前面。
有向无环图（DAG）才有拓扑排序，非 DAG 图没有拓扑排序一说。

例如，下面这个图：

它是一个 DAG 图，那么如何写出它的拓扑排序呢？这里说一种比较常用的方法：
从 DAG 图中选择一个 没有前驱（即入度为 0）的顶点并输出。
从图中删除该顶点和所有以它为起点的有向边。
重复 1 和 2 直到当前的 DAG 图为空或当前图中不存在无前驱的顶点为止。后一种情况说明有向图中必然存在环。

于是，得到拓扑排序后的结果是 { 1, 2, 4, 3, 5 }。通常，一个有向无环图可以有**一个或多个**拓扑排序序列。

二、拓扑排序的应用

拓扑排序通常用来“排序”具有依赖关系的任务。比如，如果用一个 DAG 图来表示一个工程，其中每个顶点表示工程中的一个任务，用有向边<A,B><A,B>表示在做任务 B 之前必须先完成任务 A。故在这个工程中，任意两个任务要么具有确定的先后关系，要么是没有关系，绝对不存在互相矛盾的关系（即环路）。

触类旁通
课程表 II（210）的一种解法

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <ctype.h>
typedef struct {
    int num;
    int *array;
} Node;
int *g_dest;
int g_destLen;
int g_front;

Node *g_preNode;
Node *g_outNode;
int GetMax(int a, int b)
{
    if (a >= b) {
        return a;
    }
    return b;
}
bool Init(int numCourses, int **prerequisites, int prerequisitesSize, int *prerequisitesColSize)
{
    int i;
    int input;
    int output;

    g_preNode = (Node *)malloc(sizeof(Node) * numCourses);
    if (g_preNode == NULL) {
```

```
    return false;
}
g_outNode = (Node *)malloc(sizeof(Node) * numCourses);
if (g_outNode == NULL) {
    return false;
}
for (i = 0; i < numCourses; i++) {
    g_preNode[i].num = 0;
    g_preNode[i].array = NULL;
    g_outNode[i].num = 0;
    g_outNode[i].array = NULL;
}

for (i = 0; i < prerequisitesSize; i++) {
    input = prerequisites[i][1];
    output = prerequisites[i][0];
    g_preNode[output].num++;
    g_outNode[input].num++;
}

// 动态分配数组
for (i = 0; i < numCourses; i++) {
    if (g_preNode[i].num > 0) {
        g_preNode[i].array = (int *)malloc(sizeof(int) * g_preNode[i].num);
        if (g_preNode[i].array == NULL) {
            return false;
        }
        g_preNode[i].num = 0;
    }

    if (g_outNode[i].num > 0) {
        g_outNode[i].array = (int *)malloc(sizeof(int) * g_outNode[i].num);
        if (g_outNode[i].array == NULL) {
            return false;
        }
        g_outNode[i].num = 0;
    }
}

for (i = 0; i < prerequisitesSize; i++) {
    input = prerequisites[i][1];
    output = prerequisites[i][0];
    g_preNode[output].array[g_preNode[output].num] = input;
```

```
        g_preNode[output].num++;
        g_outNode[input].array[g_outNode[input].num] = output;
        g_outNode[input].num++;
    }

    g_dest = (int *)malloc(numCourses * sizeof(int));
    if (g_dest == NULL) {
        return false;
    }
    g_destLen = 0;
    g_front = 0;
    return true;
}

void FreeNode(int numCourses)
{
    int i;
    if (g_preNode != NULL) {
        for (i = 0; i < numCourses; i++) {
            if (g_preNode[i].array != NULL) {
                free(g_preNode[i].array);
            }
        }
        free(g_preNode);
        g_preNode = NULL;
    }

    if (g_outNode != NULL) {
        for (i = 0; i < numCourses; i++) {
            if (g_outNode[i].array != NULL) {
                free(g_outNode[i].array);
                g_outNode[i].array = NULL;
            }
        }
        free(g_outNode);
        g_outNode = NULL;
    }
}

void Bfs(int numCourses)
{
    int i;
    int curr;
```

```
    int out;

    for (i = 0; i < numCourses; i++) {
        if (g_preNode[i].num == 0) {
            g_dest[g_destLen++] = i;
        }
    }

    while (g_front < g_destLen) {
        curr = g_dest[g_front];
        g_front++;
        for (i = 0; i < g_outNode[curr].num; i++) {
            out = g_outNode[curr].array[i];

            // 删除, 为 0，则进入 BFS 队列
            g_preNode[out].num--;
            if (g_preNode[out].num == 0) {
                g_dest[g_destLen] = out;
                g_destLen++;
            }
        }
    }
}

/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
int *findOrder(int numCourses, int **prerequisites, int prerequisitesSize, int *prerequisitesColSize,
               int *returnSize)
{
    bool rslt;
    rslt = Init(numCourses, prerequisites, prerequisitesSize, prerequisitesColSize);
    if (!rslt) {
        FreeNode(numCourses);
        (*returnSize) = 0;
        return NULL;
    }
    Bfs(numCourses);
    FreeNode(numCourses);
    (*returnSize) = g_destLen;
    if (g_destLen < numCourses) {
        (*returnSize) = 0;
    }
```

```
    return g_dest;
}

void TestCase1()
{
    int m1[] = { 1, 0 };
    int *g[] = { m1 };
    int prerequisitesColSize[] = { 2 };
    int numCourses = 2;
    int prerequisitesSize = 1;
    int returnSize;
    int *rslt;
    rslt = findOrder(numCourses, g, 1, prerequisitesColSize, &returnSize);
    if (returnSize == 2) {
        printf("1: pass");
    }
}
void TestCase2()
{
    int m1[] = { 1, 0 };
    int m2[] = { 2, 0 };
    int m3[] = { 3, 1 };
    int m4[] = { 3, 2 };
    int *g[] = { m1, m2, m3, m4 };
    int prerequisitesColSize[] = { 2 };
    int numCourses = 4;
    int prerequisitesSize = 4;
    int returnSize;
    int *rslt;
    rslt = findOrder(numCourses, g, 1, prerequisitesColSize, &returnSize);
    if (returnSize == 4) {
        printf("1: pass");
    }
}
int main()
{
    TestCase2();
    TestCase1();
    return 0;
}
```

444. 序列重建 （会员）

注：进行拓扑排序：
(5,2),(2,6),(6,3),(4,1),(1,5),(5,2)

269. 火星词典

示例 3：

```
输入：
[
  "z",
  "x",
  "z"
]

输出："

解释：此顺序是非法的，因此返回 ""。
```

注意：

1. 你可以默认输入的全部都是小写字母
2. 假如，a 的字母排列顺序优先于 b，那么在给定的词典当中 a 定先出现在 b 前面
3. 若给定的顺序是不合法的，则返回空字符串即可
4. 若存在多种可能的合法字母顺序，请返回其中任意一种顺序即可

九阴真经第七式 ：字符串
代表题目：5. 最长回文子串

字符串介绍
字符串可以涉及非常多的考点：如递归、栈、hash、dfs、bfs、动态规划等，需要重点关注。

触类旁通
93. 复原 IP 地址（dfs）

43. 字符串相乘

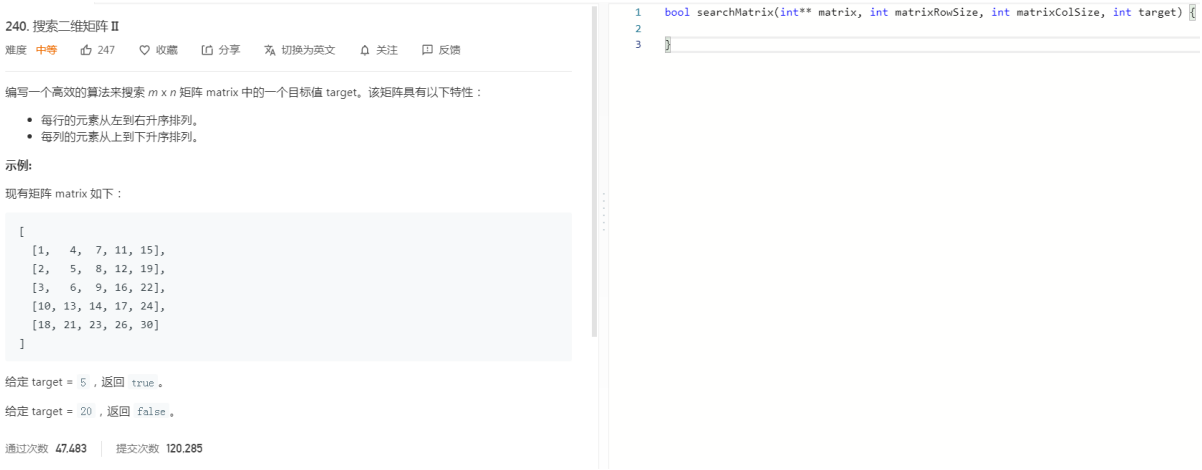
227. 基本计算器 II

给出一个表达式，计算结果，就是一个计算器类型的问题，也是栈的经典应用。

栈：添加和删除元素都在队尾进行，先进后出，后进先出，类比于子***，入栈和出栈。
思路：
将表达式（中缀）转化为后缀
将后缀计算出结果
具体规则为：
1.中缀转后缀：
- 数字直接输出到后缀表达式
- 栈为空时，遇到运算符，直接入栈
- 遇到运算符，弹出所有优先级大于或等于该运算符的栈顶元素，并将该运算符入栈
- 将栈中元素依次出栈
例如：表达式：3+2 * 2
遇到 3，直接输出到后缀表达式中，栈中元素为空，结果为： 栈： 空； 后缀表达式：3
遇到符号“+”，入栈，结果为： 栈：+ ； 后缀表达式：3
遇到 2，直接输出，结果为： 栈：+； 后缀表达式： 3 2
遇到乘号*，入栈，结果为： 栈： * + ；后缀表达式：3 2
遇到 2，直接输出，结果为： 栈： * + ；后缀表达式：3 2 2
最后，将元素出栈：结果为：后缀表达式：3 2 2 * +

2.计算后缀：
- 遇到数字，入栈
- 遇到运算符，弹出栈顶两个元素，做运算，并将结果入栈
- 重复上述步骤，直到表达式最右端

例如上述得到的后缀表达式为 3 2 2 * +
3 2 2 都是数字，入栈，结果为：栈：2 2 3
遇到* 号， 2 2 出栈，并计算，2*2 = 4, 4 入栈，结果为：栈：4 3 ，表达式还剩一个加号
遇到+ 号，栈顶两个元素出栈并运算，4 3 做加法，4+3 =7
后缀表达式空了，计算完毕，输出结果：7
九阴真经第八式 ：二分查找
代表题目：240. 搜索二维矩阵 II



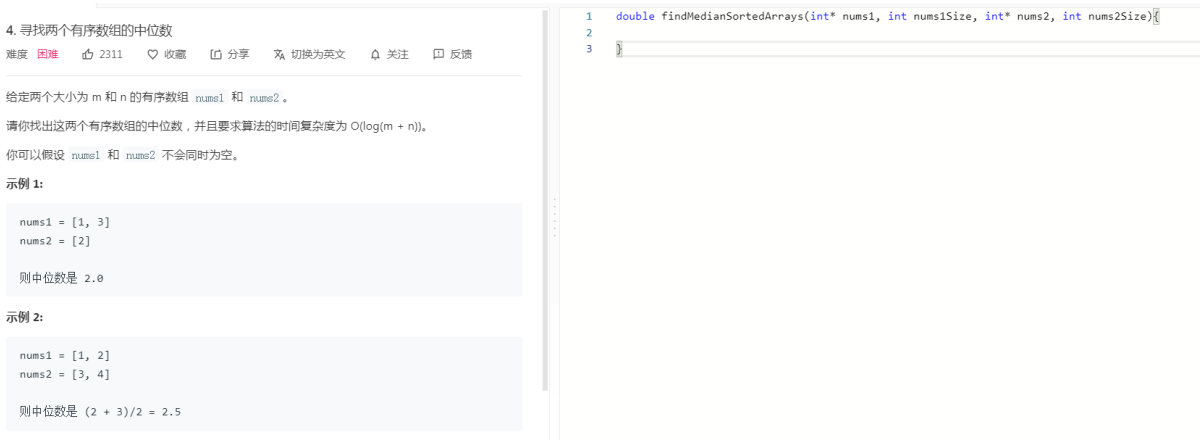
二分查找介绍

力扣：二分查找也称折半查找（Binary Search），它是一种效率较高的查找方法，前提是数据结构必须先排好序，可以在数据规模的对数时间复杂度内完成查找。但是，二分查找要求线性表具有有随机访问的特点（例如数组），也要求线性表能够根据中间元素的特点推测它两侧元素的性质，以达到缩减问题规模的效果。

二分查找问题也是面试中经常考到的问题，虽然它的思想很简单，但写好二分查找算法并不是一件容易的事情。

触类旁通

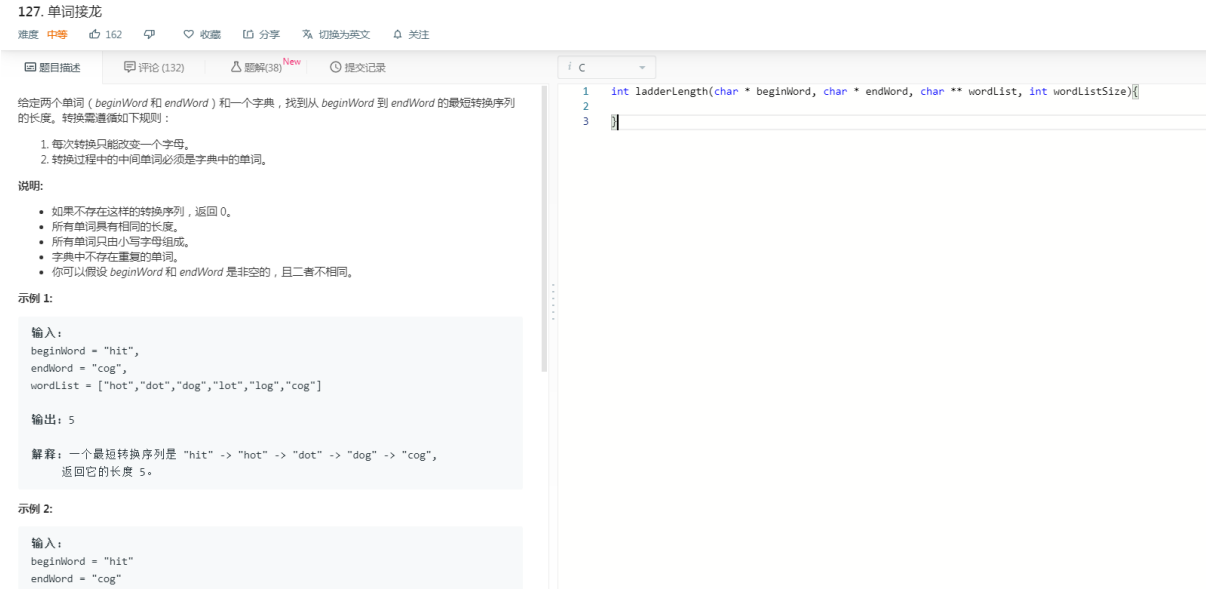
4. 寻找两个有序数组的中位数



33. 搜索旋转排序数组

九阴真经第九式 ：BFS

代表题目：127. 单词接龙



127. 单词接龙

难度 中等 162 收藏 分享 切换为英文 关注

题目描述 评论 (132) 题解(38) New 提交记录

给定两个单词（*beginWord* 和 *endWord*）和一个字典，找到从 *beginWord* 到 *endWord* 的最短转换序列的长度。转换需遵循如下规则：

1. 每次转换只能改变一个字母。
2. 转换过程中的中间单词必须是字典中的单词。

说明:

- 如果不存在这样的转换序列，返回 0。
- 所有单词具有相同的长度。
- 所有单词只由小写字母组成。
- 字典中不存在重复的单词。
- 你可以假设 *beginWord* 和 *endWord* 是非空的，且二者不相同。

示例 1:

```
输入：
beginWord = "hit",
endWord = "cog",
wordList = ["hot","dot","dog","lot","log","cog"]

输出：5

解释：一个最短转换序列是 "hit" -> "hot" -> "dot" -> "dog" -> "cog",
     返回它的长度 5。
```

示例 2:

```
输入：
beginWord = "hit"
endWord = "cog"
```

C

```
int ladderLength(char * beginWord, char * endWord, char ** wordList, int wordListSize){

}
```

Bfs 介绍
（https://zhuanlan.zhihu.com/p/62884729）

BFS（广度优先搜索） 常用来解决最短路径问题。第一次遍历到目的节点时，所经过的路径是最短路径。几个要点：

只能用来求解无权图的最短路径问题

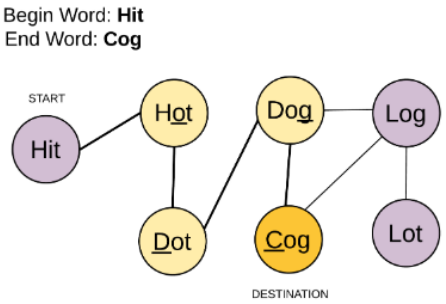
队列：用来存储每一层遍历得到的节点

标记：对于遍历过的结点，应将其标记，以防重复访问。

单词接龙

力扣 (LeetCode) 17.1k 阅读 官方题解 广度优先搜索 Java Python

拥有一个 `beginWord` 和一个 `endWord` ，分别表示图上的 `start node` 和 `end node` 。我们希望利用一些中间节点（单词）从 `start node` 到 `end node` ，中间节点是 `wordList` 给定的单词。我们对这个单词接龙每个步骤的唯一条件是相邻单词只可以改变 `一个字母` 。



我们将问题抽象在一个无向无权图中，每个单词作为节点，差距只有一个字母的两个单词之间连一条边。问题变成找到从起点到终点的最短路径，如果存在的话。因此可以使用 `广度优先搜索` 方法。

注：

广度搜索时候，如果曾经加入过，后续就不用再加入了；
加入队列时候，需要标记当前层级，方便后续直接返回目标解；
上代码：

```
#include <stdio.h>
#include <securec.h>

#define WORD_NUM 10000
#define WORD_LEN 100

struct wordStru {
    char word[WORD_LEN];
    int distance;
};

struct queueStru {
    int head;
    int tail;
    int distance;
    struct wordStru wordInfo[WORD_NUM];
};

struct queueStru g_queueInfo;
char *g_endWord;
int g_flag[WORD_NUM];
```

```
bool SimilarWord(char *a, char *b)
{
    int count = 0;

    for (int i = 0; i < strlen(a); i++) {
        if (a[i] != b[i]) {
            count++;
        }
    }

    if (count == 1) {
        return true;
    }

    return false;
}

int BFS(char ** wordList, int wordListSize)
{
    int distance;
    char *word = NULL;

    while (g_queueInfo.head < g_queueInfo.tail) {
        distance = g_queueInfo.wordInfo[g_queueInfo.head].distance;
        word = g_queueInfo.wordInfo[g_queueInfo.head++].word;
        if (strcmp(word, g_endWord) == 0) {
            return distance + 1;
        }

        for (int i = 0; i < wordListSize; i++) {
            if (g_flag[i] == 0 && SimilarWord(word, wordList[i])) {
                g_flag[i] = 1;
                g_queueInfo.wordInfo[g_queueInfo.tail].distance = distance + 1;
                memcpy_s(g_queueInfo.wordInfo[g_queueInfo.tail++].word,    WORD_LEN,    wordList[i],
strlen(wordList[i]));
            }
        }
    }

    return 0;
}
```

```
int ladderLength(char * beginWord, char * endWord, char ** wordList, int wordListSize)
{
    if (beginWord == NULL || endWord == NULL || wordList == NULL || wordListSize <= 0) {
        return 0;
    }

    memset_s(&g_queueInfo, sizeof(g_queueInfo), 0, sizeof(g_queueInfo));
    memset_s(&g_flag, sizeof(g_flag), 0, sizeof(g_flag));
    memcpy_s(g_queueInfo.wordInfo[g_queueInfo.tail++].word, WORD_LEN, beginWord, strlen(beginWord));
    g_endWord = endWord;

    return BFS(wordList, wordListSize);
}
```

触类旁通

139. 单词拆分

130. 被围绕的区域

130. 被围绕的区域

难度 中等 118 收藏 分享 切换为英文 关注

题目描述 评论 (127) 题解(43) New 提交记录

给定一个二维的矩阵，包含 'X' 和 'O'（字母 O）。

找到所有被 'X' 围绕的区域，并将这些区域里所有的 'O' 用 'X' 填充。

示例:

```
X X X X
X O O X
X X O X
X O X X
```

运行你的函数后，矩阵变为：

```
X X X X
X X X X
X X X X
X O X X
```

解释:

被围绕的区间不会存在于边界上，换句话说，任何边界上的 'O' 都不会被填充为 'X'。 任何不在边界上，或不与边界上的 'O' 相连的 'O' 最终都会被填充为 'X'。如果两个元素在水平或垂直方向相邻，则称它们是“相连”的。

在真实的面试中遇到过这道题？ 是 否

C

```c
void solve(char** board, int boardSize, int* boardColSize){

}
```

317. 离建筑物最近的距离（困难）

317. 离建筑物最近的距离

难度 困难 5 收藏 分享 切换为英文 关注

题目描述 评论 (4) 题解(8) New 提交记录

你是个房地产开发商，想要选择一片空地建一栋大楼。你想把这栋大楼够造在一个距离周边设施都比较方便的地方，通过调研，你希望从它出发能在 **最短的距离和** 内抵达周边全部的建筑物。请你计算出这个最佳的选址到周边全部建筑物的 **最短距离和**。

注意：

你只能通过向上、下、左、右四个方向上移动。

给你一个由 0、1 和 2 组成的二维网格，其中：

- **0** 代表你可以自由通过和选择建造的空地
- **1** 代表你无非通行的建筑物
- **2** 代表你无非通行的障碍物

示例：

输入： [[1,0,2,0,1],[0,0,0,0,0],[0,0,1,0,0]]

```
1 - 0 - 2 - 0 - 1
|   |   |   |   |
0 - 0 - 0 - 0 - 0
|   |   |   |   |
0 - 0 - 1 - 0 - 0
```

输出： 7

解析：
给定三个建筑物 (0,0)、(0,4) 和 (2,2) 以及一个位于 (0,2) 的障碍物。
由于总距离之和 3+3+1=7 最优，所以位置 (1,2) 是符合要求的最优地点，故返回7。

C

```c
int shortestDistance(int** grid, int gridSize, int* gridColSize){

}
```

505. 迷宫 II （会员）

505. 迷宫 II

难度 中等 15 收藏 分享 切换为英文 关注

通过次数 636 提交

题目描述 评论 (5) 题解(10) New 提交记录

由空地和墙组成的迷宫中有一个球。球可以向**上下左右**四个方向滚动，但在遇到墙壁前不会停止滚动。当球停下时，可以选择下一个方向。

给定球的**起始位置**，**目的地**和**迷宫**，找出让球停在目的地的最短距离。距离的定义是球从起始位置（不包括）到目的地（包括）经过的**空地**个数。如果球无法停在目的地，返回 -1。

迷宫由一个0和1的二维数组表示。 1表示墙壁，0表示空地。你可以假定迷宫的边缘都是墙壁。起始位置和目的地的坐标通过行号和列号给出。

示例 1:

输入 1: 迷宫由以下二维数组表示

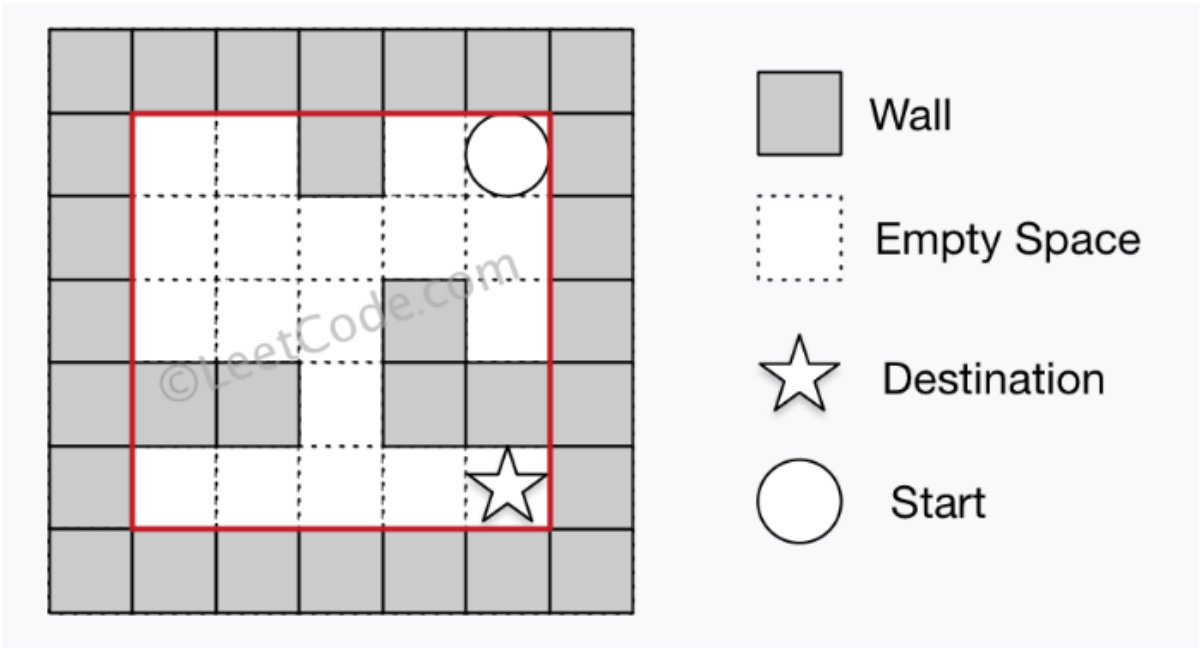
```
0 0 1 0 0
0 0 0 0 0
0 0 0 1 0
1 1 0 1 1
0 0 0 0 0
```

输入 2: 起始位置坐标 (rowStart, colStart) = (0, 4)
输入 3: 目的地坐标 (rowDest, colDest) = (4, 4)

输出: 12

解析: 一条最短路径 : left -> down -> left -> down -> right -> down -> right。
总距离为 1 + 1 + 3 + 1 + 2 + 2 + 2 = 12。

```
int shortestDistance(int** maze, int mazeSize, int* mazeColSize, int* start, int startSize, int* destination, int destinationSize){

}
```

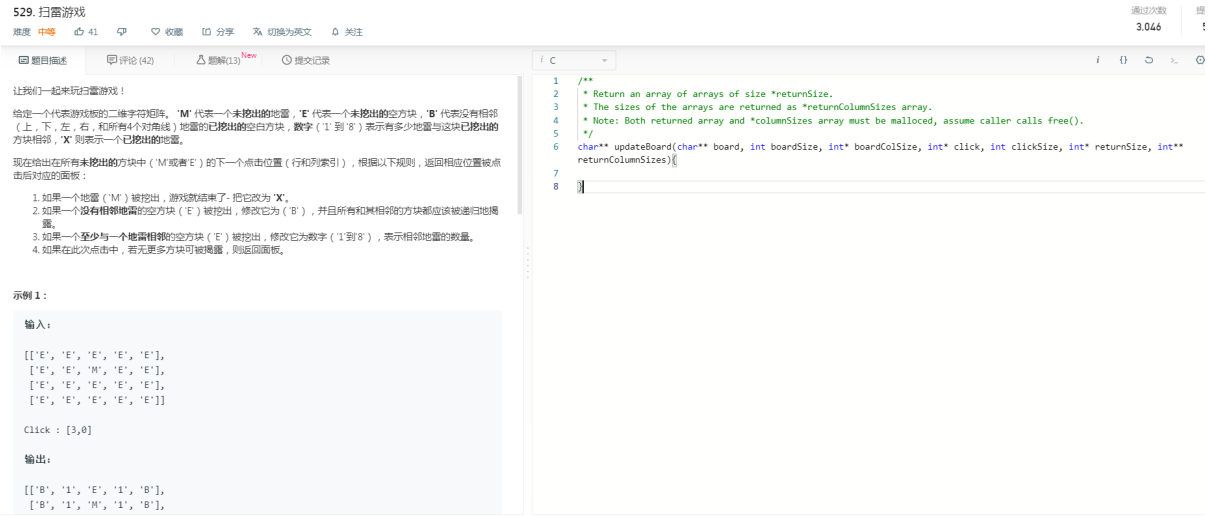


注意:

1. 迷宫中只有一个球和一个目的地。
2. 球和目的地都在空地上，且初始时它们不在同一位置。
3. 给定的迷宫不包括边界 (如图中的红色矩形), 但你可以假设迷宫的边缘都是墙壁。
4. 迷宫至少包括2块空地，行数和列数均不超过100。

在真实的面试中遇到过这道题？ 是 否

529. 扫雷游戏




```
/*
 * Copyright (c) Huawei Technologies Co., Ltd. 2012-2019. All rights reserved.
 * Description:529. 扫雷游戏
 * Author: w00448446
 * Create: 2019-10-21
 */
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
// #include <securec.h>
#define MAX_MATRIX_NUMBER 51
#define MAX_QUEUE_NUMBER  (MAX_MATRIX_NUMBER * MAX_MATRIX_NUMBER)

int g_direction[][2] = {{0, -1}, {1, 0},{0, 1}, {-1, 0}, {1, -1}, {1, 1},{-1, 1}, {-1, -1}};
int g_directionLen = sizeof(g_direction) / sizeof(g_direction[0]);

int g_queue[MAX_QUEUE_NUMBER][2];
int g_front;
int g_length;

char** MallocMineBoard(char** board, int boardSize, int* boardColSize,
int* returnSize, int** returnColumnSizes)
{
    char** mineBoard = NULL;
    int *columSizes = NULL;
    int i;
    int rslt;
```

```
    mineBoard = (char **)malloc(boardSize * sizeof(char *));
    if (mineBoard == NULL) {
        return NULL;
    }

    columSizes = (int *)malloc(boardSize * sizeof(int));
    if (columSizes == NULL) {
        return NULL;
    }
    for (i = 0; i < boardSize; i++) {
        columSizes[i] = boardColSize[i];
    }
    *returnColumnSizes = columSizes;
    *returnSize = boardSize;
    for (i = 0; i < boardSize; i++) {
        mineBoard[i] = (char *)malloc(boardColSize[i] * sizeof(char));
        if (mineBoard[i] == NULL) {
            return NULL;
        }
        //rslt = memcpy_s(mineBoard[i], boardColSize[i], board[i], boardColSize[i]);
        memcpy(mineBoard[i], board[i], boardColSize[i]);

    }

    return mineBoard;
}



int IsPointValid(char** board, int boardSize, int* boardColSize, int pointX, int pointY)
{
    if (pointX < 0) {
        return false;
    }

    if (pointX >= boardSize) {
        return false;
    }

    if (pointY < 0) {
        return false;
    }
```

```
    if (pointY >= boardColSize[0]) {
        return false;
    }

    return true;
}


int GetNeighborMineNum(char** board, int boardSize, int* boardColSize, int pointX, int pointY)
{
    int i;
    int nextX, nextY;
    int sum = 0;

    for (i = 0; i < g_directionLen; i++) {
        nextX = pointX + g_direction[i][0];
        nextY = pointY + g_direction[i][1];


        // 如果无效，则找下一个点

        if (!IsPointValid(board, boardSize, boardColSize, nextX, nextY)) {
            continue;
        }


        // 找到地雷的处理

        if (board[nextX][nextY] == 'M' ) {
            sum++;
        }
    }

    return sum;

}

void Bfs(char** board, int boardSize, int* boardColSize)
{
    int i;
    int pointX, pointY;
    int nextX, nextY;
    int sum = 0;
    while (g_front < g_length) {

        // 取出首节点
```

```
        pointX = g_queue[g_front][0];
        pointY = g_queue[g_front][1];
        g_front++;

        for (i = 0; i < g_directionLen; i++) {
            nextX = pointX + g_direction[i][0];
            nextY = pointY + g_direction[i][1];

            // 如果无效，则找下一个点
            if (!IsPointValid(board, boardSize, boardColSize, nextX, nextY)) {
                continue;
            }

            if (board[nextX][nextY] != 'E') {
                continue;
            }

            sum = GetNeighborMineNum(board, boardSize, boardColSize, nextX, nextY);
            if (sum > 0) {
                board[nextX][nextY] = sum + '0';
                continue;
            }

            // 如果都是空节点，则更新为 B
            board[nextX][nextY] = 'B';

            // 相邻节点入队
            g_queue[g_length][0] = nextX;
            g_queue[g_length][1] = nextY;
            g_length++;
        }
    }
    return ;
}

/**
 * Return an array of arrays of size *returnSize.
 * The sizes of the arrays are returned as *returnColumnSizes array.
 * Note: Both returned array and *columnSizes array must be malloced, assume caller calls free().
```

```
 */
char** updateBoard(char** board, int boardSize, int* boardColSize, int* click, int clickSize,
    int* returnSize, int** returnColumnSizes)
{

    char **mineBoard = NULL;
    int pointX = click[0];
    int pointY = click[1];
    int sum;
    char ch;

    mineBoard = MallocMineBoard(board, boardSize, boardColSize, returnSize, returnColumnSizes);
    if (mineBoard == NULL) {
        return NULL;
    }

    ch = mineBoard[pointX][pointY];
    if (ch == 'M') {
        mineBoard[pointX][pointY] = 'X';
        return mineBoard;
    }

    sum = GetNeighborMineNum(board, boardSize, boardColSize, pointX, pointY);
    if (sum > 0) {
        mineBoard[pointX][pointY] = sum + '0';
        return mineBoard;
    }


    // 广度优先搜索:没有相邻地雷的空方块

    g_length = 0;
    g_front = 0;
    g_queue[g_length][0] = pointX;
    g_queue[g_length][1] = pointY;
    g_length++;
    mineBoard[pointX][pointY] = 'B';

    Bfs(mineBoard, boardSize, boardColSize);

    return mineBoard;
}
```

1263. 推箱子（困难）

1263. 推箱子

难度 困难 16 收藏 分享 切换为英文 关注

题目描述 评论 (15) 题解(19) New 提交记录

「推箱子」是一款风靡全球的益智小游戏，玩家需要将箱子推到仓库中的目标位置。

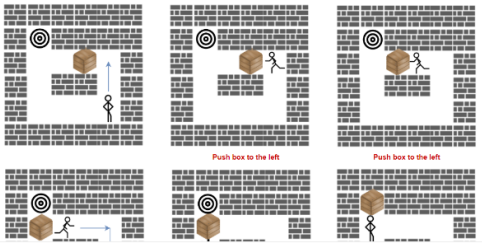
游戏地图用大小为 `n * m` 的网格 `grid` 表示，其中每个元素可以是墙、地板或者是箱子。

现在你将作为玩家参与游戏，按规则将箱子 `'B'` 移动到目标位置 `'T'` ：

- 玩家用字符 `'S'` 表示，只要他在地板上，就可以在网格中向上、下、左、右四个方向移动。
- 地板用字符 `'.'` 表示，意味着可以自由行走。
- 墙用字符 `'#'` 表示，意味着障碍物，不能通行。
- 箱子仅有一个，用字符 `'B'` 表示。相应地，网格上有一个目标位置 `'T'`。
- 玩家需要站在箱子旁边，然后沿着箱子的方向进行移动，此时箱子会被移动到相邻的地板单元格。记作一次「推动」。
- 玩家无法越过箱子。

返回将箱子推到目标位置的最小 **推动** 次数，如果无法做到，请返回 `-1`。

示例 1：



C

```c
int minPushBox(char** grid, int gridSize, int* gridColSize){

}
```

1197. 进击的骑士

1197. 进击的骑士

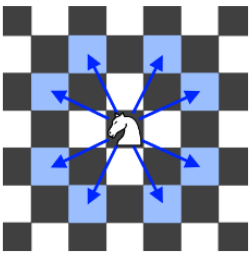
难度 中等 5 收藏 分享 切换为英文 关注

题目描述 评论 题解(6) New 提交记录

一个坐标可以从 `-infinity` 延伸到 `+infinity` 的 **无限大的** 棋盘上，你的 **骑士** 驻扎在坐标为 `[0, 0]` 的方格里。

骑士的走法和中国象棋中的马相似，走“日”字：即先向左（或右）走 1 格，再向上（或下）走 2 格；或先向左（或右）走 2 格，再向上（或下）走 1 格。

每次移动，他都可以按图示八个方向之一前进。



现在，骑士需要前去征服坐标为 `[x, y]` 的部落，请你为他规划路线。

最后返回所需的最小移动次数即可。本题确保答案是一定存在的。

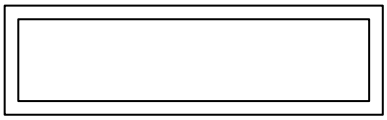
示例 1：

```
输入：x = 2, y = 1
输出：1
解释：[0, 0] → [2, 1]
```

C

```c
int minKnightMoves(int x, int y){

}
```

815. 公交路线（困难_必做）



934. 最短的桥

九阴真经第十式 ：DFS
代表题目：934. 最短的桥

Dfs 介绍

DFS 实质就是一种枚举，不过借助递归实现；

DFS 的基本模式：

```
void dfs(int step)
{
    判断边界；

    for（int i=1;i<=n;++i)//尝试每一种可能；
    {
    dfs(step+1);/'/继续下一步
    }
    返回；
}
```

触类旁通

1102. 得分最高的路径

685. 冗余连接 II（困难）

531. 孤独像素 I

533. 孤独像素 II

332. 重新安排行程

337. 打家劫舍 III

113. 路径总和 II

九阴真经第十一式 ：动态规划
代表题目：213. 打家劫舍 II

动态规划介绍
https://blog.csdn.net/qq_37763204/article/details/79394397
https://blog.csdn.net/u013309870/article/details/75193592

触类旁通
1043. 分隔数组以得到最大和

416. 分割等和子集

123. 买卖股票的最佳时机 III

一、穷举框架
首先，还是一样的思路：如何穷举？这里的穷举思路和上篇文章递归的思想不太一样。

递归其实是符合我们思考的逻辑的，一步步推进，遇到无法解决的就丢给递归，一不小心就做出来了，可读性还很好。缺点就是一旦出错，你也不容易找到错误出现的原因。比如上篇文章的递归解法，肯定

还有计算冗余，但确实不容易找到。

而这里，我们不用递归思想进行穷举，而是利用「状态」进行穷举。我们具体到每一天，看看总共有几种可能的「状态」，再找出每个「状态」对应的「选择」。我们要穷举所有「状态」，穷举的目的是根据对应的「选择」更新状态。听起来抽象，你只要记住「状态」和「选择」两个词就行，下面实操一下就很容易明白了。

Python
```
for 状态 1 in 状态 1 的所有取值：
    for 状态 2 in 状态 2 的所有取值：
        for ...
            dp[状态 1][状态 2][...] = 择优(选择 1，选择 2...)
```
比如说这个问题，每天都有三种「选择」：买入、卖出、无操作，我们用 buybuy，sellsell，restrest 表示这三种选择。但问题是，并不是每天都可以任意选择这三种选择的，因为 sellsell 必须在 buybuy 之后，buybuy 必须在 sellsell 之后。那么 restrest 操作还应该分两种状态，一种是 buybuy 之后的 restrest（持有了股票），一种是 sellsell 之后的 restrest（没有持有股票）。而且别忘了，我们还有交易次数 kk 的限制，就是说你 buybuy 还只能在 k > 0 的前提下操作。

很复杂对吧，不要怕，我们现在的目的只是穷举，你有再多的状态，老夫要做的就是一把梭全部列举出来。这个问题的「状态」有三个，第一个是天数，第二个是允许交易的最大次数，第三个是当前的持有状态（即之前说的 restrest 的状态，我们不妨用 11 表示持有，00 表示没有持有）。然后我们用一个三维数组就可以装下这几种状态的全部组合：

Python
```
dp[i][k][0 or 1]
0 <= i <= n-1, 1 <= k <= K
n 为天数，大 K 为最多交易数
此问题共 n × K × 2 种状态，全部穷举就能搞定。

for 0 <= i < n:
    for 1 <= k <= K:
        for s in {0, 1}:
            dp[i][k][s] = max(buy, sell, rest)
```
而且我们可以用自然语言描述出每一个状态的含义，比如说 dp[3][2][1] 的含义就是：今天是第三天，我现在手上持有着股票，至今最多进行 22 次交易。再比如 dp[2][3][0] 的含义：今天是第二天，我现在手上没有持有股票，至今最多进行 33 次交易。很容易理解，对吧？

我们想求的最终答案是 dp[n - 1][K][0]，即最后一天，最多允许 K 次交易，最多获得多少利润。读者可能问为什么不是 dp[n - 1][K][1]？因为 [1] 代表手上还持有股票，[0] 表示手上的股票已经卖出去了，很显然后者得到的利润一定大于前者。

记住如何解释「状态」，一旦你觉得哪里不好理解，把它翻译成自然语言就容易理解了。

二、状态转移框架
现在，我们完成了「状态」的穷举，我们开始思考每种「状态」有哪些「选择」，应该如何更新「状态」。只看「持有状态」，可以画个状态转移图。

通过这个图可以很清楚地看到，每种状态（0 和 1）是如何转移而来的。根据这个图，我们来写一下状态转移方程：

Python
```
dp[i][k][0] = max(dp[i-1][k][0], dp[i-1][k][1] + prices[i])
              max(   选择 rest ，          选择 sell      )
```

解释：今天我没有持有股票，有两种可能：
要么是我昨天就没有持有，然后今天选择 rest，所以我今天还是没有持有；
要么是我昨天持有股票，但是今天我 sell 了，所以我今天没有持有股票了。

```
dp[i][k][1] = max(dp[i-1][k][1], dp[i-1][k-1][0] - prices[i])
              max(   选择 rest ，          选择 buy         )
```

解释：今天我持有着股票，有两种可能：
要么我昨天就持有着股票，然后今天选择 rest，所以我今天还持有着股票；
要么我昨天本没有持有，但今天我选择 buy，所以今天我就持有股票了。
这个解释应该很清楚了，如果 buybuy，就要从利润中减去 prices[i]prices[i]，如果 sellsell，就要给利润增加 prices[i]prices[i]。今天的最大利润就是这两种可能选择中较大的那个。而且注意 kk 的限制，我们在选择 buybuy 的时候，把 kk 减小了 11，很好理解吧，当然你也可以在 sellsell 的时候减 11，一样的。

现在，我们已经完成了动态规划中最困难的一步：状态转移方程。如果之前的内容你都可以理解，那么你已经可以秒杀所有问题了，只要套这个框架就行了。不过还差最后一点点，就是定义 base case，即最简单的情况。

Python
```
dp[-1][k][0] = 0
```
解释：因为 i 是从 0 开始的，所以 i = -1 意味着还没有开始，这时候的利润当然是 0 。
```
dp[-1][k][1] = -infinity
```
解释：还没开始的时候，是不可能持有股票的，用负无穷表示这种不可能。
```
dp[i][0][0] = 0
```
解释：因为 k 是从 1 开始的，所以 k = 0 意味着根本不允许交易，这时候利润当然是 0 。
```
dp[i][0][1] = -infinity
```
解释：不允许交易的情况下，是不可能持有股票的，用负无穷表示这种不可能。
把上面的状态转移方程总结一下：

Python
base case：
dp[-1][k][0] = dp[i][0][0] = 0
dp[-1][k][1] = dp[i][0][1] = -infinity

状态转移方程：
dp[i][k][0] = max(dp[i-1][k][0], dp[i-1][k][1] + prices[i])
dp[i][k][1] = max(dp[i-1][k][1], dp[i-1][k-1][0] - prices[i])
读者可能会问，这个数组索引是 -1 怎么编程表示出来呢，负无穷怎么表示呢？这都是细节问题，有很多方法实现。现在完整的框架已经完成，下面开始具体化。
链接：https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock-iii/solution/yi-ge-tong-yong-fang-fa-tuan-mie-6-dao-gu-piao-wen/

62. 不同路径

63. 不同路径 II

651. 4 键键盘（会员）

361. 轰炸敌人

1066. 校园自行车分配 II（会员）

750. 角矩形的数量（会员）

1230. 抛掷硬币

1055. 形成字符串的最短路径（会员）

九阴真经第十二式 ：贪心算法
代表题目：**452. 用最少数量的箭引爆气球**

注：排序后，第一箭能射穿所有 end《= start 的球。

贪心算法介绍

触类旁通
1231. 分享巧克力（会员）

1247. 交换字符使得字符串相同

45. 跳跃游戏 II

621. 任务调度器

376. 摆动序列

九阴真经第十三式 ：字典树
代表题目：820. 单词的压缩编码

注：本题可信考了两次。

解法二：
直接按前序进行排序，如果第一个在第二个里面，则忽略第一个。 如：em emit，则忽略 em。

字典树介绍

触类旁通
1231. 分享巧克力（会员）

648. 单词替换

208. 实现 Trie (前缀树)

真题实战
http://3ms.huawei.com/km/blogs/details/7657743