

61. 旋转链表

61. 旋转链表

难度 中等  234  收藏  分享  切换为英文  关注  反馈

给定一个链表，旋转链表，将链表每个节点向右移动 k 个位置，其中 k 是非负数。

示例 1:

输入: 1->2->3->4->5->NULL, $k = 2$
输出: 4->5->1->2->3->NULL
解释:
向右旋转 1 步: 5->1->2->3->4->NULL
向右旋转 2 步: 4->5->1->2->3->NULL

示例 2:

输入: 0->1->2->NULL, $k = 4$
输出: 2->0->1->NULL
解释:
向右旋转 1 步: 2->0->1->NULL
向右旋转 2 步: 1->2->0->NULL
向右旋转 3 步: 0->1->2->NULL
向右旋转 4 步: 2->0->1->NULL

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* rotateRight(ListNode* head, int k) {
        if(!head) return NULL;

        int n = 0;
        for(auto p = head;p = p->next) n ++;

        k %= n;
        auto first = head,second = head;
        while(k --) first = first->next;
        while(first->next)
        {
            first = first->next;
            second = second->next;
        }
        first->next = head;
        head = second->next;
        second->next = NULL;
    }
};
```

```
        return head;
    }
};
```

62. 不同路径

62. 不同路径

难度 中等 490 收藏 分享 切换为英文 关注 反馈

一个机器人位于一个 $m \times n$ 网格的左上角（起始点在下图中标记为“Start”）。

机器人每次只能向下或者向右移动一步。机器人试图达到网格的右下角（在下图中标记为“Finish”）。

问总共有多少条不同的路径？



例如，上图是一个 7×3 的网格。有多少可能的路径？

示例 1:

输入: $m = 3, n = 2$
输出: 3
解释:
从左上角开始，总共有 3 条路径可以到达右下角。
1. 向右 -> 向右 -> 向下
2. 向右 -> 向下 -> 向右
3. 向下 -> 向右 -> 向右

示例 2:

输入: $m = 7, n = 3$
输出: 28

```
class Solution {
public:
    int uniquePaths(int m, int n) {
        int dp[101][101];
        memset(dp, 0, sizeof(dp));
        for(int i = 1; i <= m; i++){
            for(int j = 1; j <= n; j++){
                if(i == 1 || j == 1)
                    dp[i][j] = 1;
                else
                    dp[i][j] = dp[i-1][j] + dp[i][j-1];
            }
        }
        return dp[m][n];
    }
};
```

```

    }
    }
    return dp[m][n];
}
};
// class Solution {
// public:
//     int uniquePaths(int m, int n) {
//         if (m == 1 || n == 1) return 1;
//         return uniquePaths(m, n - 1) + uniquePaths(m - 1, n);
//     }
// };

```

63. 不同路径 II

63. 不同路径 II

难度 中等  264  收藏  分享  切换为英文  关注  反馈

一个机器人位于一个 $m \times n$ 网格的左上角（起始点在下图中标记为“Start”）。

机器人每次只能向下或者向右移动一步。机器人试图达到网格的右下角（在下图中标记为“Finish”）。

现在考虑网格中有障碍物。那么从左上角到右下角将会有多少条不同的路径？



网格中的障碍物和空位置分别用 1 和 0 来表示。

说明： m 和 n 的值均不超过 100。

示例 1:

输入：

```

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

```

输出：2

解释：

3x3 网格的正中间有一个障碍物。

从左上角到右下角一共有 2 条不同的路径：

1. 向右 -> 向右 -> 向下 -> 向下
2. 向下 -> 向下 -> 向右 -> 向右

```

class Solution {
public:

```



```

int uniquePathsWithObstacles(vector<vector<int>>& g) {
    int n = g.size(), m = g[0].size();
    vector<vector<long long>> f(n, vector<long long>(m));
    for(int i = 0; i < n; i++)
        for(int j = 0; j < m; j++)
        {
            if(g[i][j]) continue;
            if(!i && !j) f[i][j] = 1;
            if(i > 0) f[i][j] += f[i - 1][j];
            if(j > 0) f[i][j] += f[i][j - 1];
        }
    return f[n - 1][m - 1];
}
};

```

64. 最小路径和

64. 最小路径和

难度 **中等**  437  收藏  分享  切换为英文  关注  反馈

给定一个包含非负整数的 $m \times n$ 网格，请找出一条从左上角到右下角的路径，使得路径上的数字总和为最小。

说明：每次只能向下或者向右移动一步。

示例：

输入：

```

[
  [1,3,1],
  [1,5,1],
  [4,2,1]
]

```

输出：7

解释：因为路径 1→3→1→1→1 的总和最小。

```







class Solution {
public:
    int minPathSum(vector<vector<int>>& grid) {
        int n = grid.size(), m = grid[0].size();
        if(n == 0 || m == 0) return 0;
        vector<vector<int>> dp(n, vector<int>(m, 0));
        for(int i = 0; i < n; i++)
        {
            for(int j = 0; j < m; j++)
            {
                if(i > 0) dp[i][j] = dp[i - 1][j];
                if(j > 0)
                {
                    if(i == 0) dp[i][j] = dp[i][j - 1];
                    else dp[i][j] = min(dp[i][j], dp[i][j - 1]);
                }
                dp[i][j] += grid[i][j];
            }
        }
    }
}

```

```
        return dp[n - 1][m - 1];
    }
};
```

65. 有效数字

65. 有效数字

难度 **困难**  106  收藏  分享  切换为英文  关注  反馈

验证给定的字符串是否可以解释为十进制数字。

例如:

```
"0" => true
" 0.1 " => true
"abc" => false
"1 a" => false
"2e10" => true
"-90e3 " => true
" 1e" => false
"e3" => false
" 6e-1" => true
"99e2.5 " => false
"53.5e93" => true
"--6 " => false
"-+3" => false
"95a54e53" => false
```

说明: 我们有意将问题陈述地比较模糊。在实现代码之前，你应当事先思考所有可能的情况。这里给出一份可能存在于有效十进制数字中的字符列表:

- 数字 0-9
- 指数 - "e"
- 正/负号 - "+" / "-"
- 小数点 - "."

当然，在输入中，这些字符的上下文也很重要。

```
class Solution {
public:
    bool isNumber(string s) {
        int i = 0;
        while (i < s.size() && s[i] == ' ') i ++ ;
        int j = s.size() - 1;
        while (j >= 0 && s[j] == ' ') j -- ;
        if (i > j) return false;
        s = s.substr(i, j - i + 1);

        if (s[0] == '-' || s[0] == '+') s = s.substr(1);
        if (s.empty() || s[0] == '.' && s.size() == 1) return false;

        int dot = 0, e = 0;
        for (int i = 0; i < s.size(); i ++ )
        {
            if (s[i] >= '0' && s[i] <= '9');
            else if (s[i] == '.')
            {
```

```

        dot ++ ;
        if (e || dot > 1) return false;
    }
    else if (s[i] == 'e' || s[i] == 'E')
    {
        e ++ ;
        if (i + 1 == s.size() || !i || e > 1 || i == 1 && s[0] == '.')
return false;
        if (s[i + 1] == '+' || s[i + 1] == '-')
        {
            if (i + 2 == s.size()) return false;
            i ++ ;
        }
    }
    else return false;
}
return true;
}
};

```

66. 加一

66. 加一

难度 简单

👍 457

❤ 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

给定一个由整数组成的非空数组所表示的非负整数，在该数的基础上加一。

最高位数字存放在数组的首位，数组中每个元素只存储单个数字。

你可以假设除了整数 0 之外，这个整数不会以零开头。

示例 1:

输入: [1,2,3]

输出: [1,2,4]

解释: 输入数组表示数字 123。

示例 2:

输入: [4,3,2,1]

输出: [4,3,2,2]

解释: 输入数组表示数字 4321。

```

class Solution {
public:
    vector<int> plusOne(vector<int>& digits) {
        int t = 1;
        for (int i = digits.size() - 1; i >= 0; i -- )
        {
            digits[i] += t;
            t = digits[i] / 10;
            digits[i] %= 10;
        }
        if (t)

```







```

    {
        digits.push_back(0);
        for (int i = digits.size() - 2; i >= 0; i -- )
            digits[i + 1] = digits[i];
        digits[0] = 1;
    }
    return digits;
}
};

```

67. 二进制求和

67. 二进制求和

难度 **简单**  345  收藏  分享  切换为英文  关注  反馈

给你两个二进制字符串，返回它们的和（用二进制表示）。

输入为 **非空** 字符串且只包含数字 **1** 和 **0**。

示例 1:

输入: a = "11", b = "1"
输出: "100"

示例 2:

输入: a = "1010", b = "1011"
输出: "10101"

提示:

- 每个字符串仅由字符 '0' 或 '1' 组成。
- $1 \leq a.length, b.length \leq 10^4$
- 字符串如果不是 "0"，就都不含前导零。

```

class Solution {
public:
    string addBinary(string a, string b) {
        if(a.size() < b.size()) swap(a,b);
        reverse(a.begin(),a.end());
        reverse(b.begin(),b.end());
        int t = 0;
        string res;
        int k = 0;
        while(k < b.size())
        {
            t += a[k] - '0' + b[k] - '0';
            res += to_string(t & 1);
            t /= 2;
            k ++;
        }
        while(k < a.size())

```

```

    {
        t += a[k] - '0';
        res += to_string(t & 1);
        t /= 2;
        k++;
    }
    if(t) res += '1';
    reverse(res.begin(), res.end());
    return res;
}
};

```

68. 文本左右对齐

68. 文本左右对齐

难度 **困难** 59 收藏 分享 切换为英文 关注 反馈

给定一个单词数组和一个长度 *maxWidth*，重新排版单词，使其成为每行恰好有 *maxWidth* 个字符，且左右两端对齐的文本。

你应该使用“贪心算法”来放置给定的单词；也就是说，尽可能多地往每行中放置单词。必要时可用空格 ‘ ’ 填充，使得每行恰好有 *maxWidth* 个字符。

要求尽可能均匀分配单词间的空格数量。如果某一行单词间的空格不能均匀分配，则左侧放置的空格数要多于右侧的空格数。

文本的最后一行应为左对齐，且单词之间不插入额外的空格。

说明:

- 单词是指由非空格字符组成的字符序列。
- 每个单词的长度大于 0，小于等于 *maxWidth*。
- 输入单词数组 *words* 至少包含一个单词。

示例:

输入:

```

words = ["This", "is", "an", "example", "of", "text", "justification."]
maxWidth = 16

```

输出:

```

[
    "This    is    an",
    "example  of text",
    "justification.  "
]

```

```

class Solution {
public:
    string space(int x)
    {
        string res;
        while (x -- ) res += ' ';
        return res;
    }

    vector<string> fullJustify(vector<string>& words, int maxWidth) {
        vector<string> res;
    }
};

```



```

for (int i = 0; i < words.size();)
{
    int j = i + 1, s = words[i].size(), rs = words[i].size();
    while (j < words.size() && s + 1 + words[j].size() <= maxWidth)
    {
        s += 1 + words[j].size();
        rs += words[j].size();
        j ++ ;
    }
    rs = maxWidth - rs;
    string line = words[i];
    if (j == words.size())
    {
        for (i ++; i < j; i ++ )
            line += ' ' + words[i];
        while (line.size() < maxWidth) line += ' ';
    }
    else if (j - i == 1) line += space(rs);
    else
    {
        int base = rs / (j - i - 1);
        int rem = rs % (j - i - 1);
        i ++ ;
        for (int k = 0; i < j; i ++, k ++ )
            line += space(base + (k < rem)) + words[i];
    }
    i = j;
    res.push_back(line);
}
return res;
};

```

69. x 的平方根

69. x 的平方根

难度 **简单** 345 收藏 分享 切换为英文 关注 反馈

实现 `int sqrt(int x)` 函数。

计算并返回 x 的平方根，其中 x 是非负整数。

由于返回类型是整数，结果只保留整数的部分，小数部分将被舍去。

示例 1:

输入: 4
输出: 2

示例 2:

输入: 8
输出: 2
说明: 8 的平方根是 2.82842...,
由于返回类型是整数，小数部分将被舍去。

```
class Solution {
public:
    int mySqrt(int x) {
        int l = 0, r = x;
        while(l < r)
        {
            int mid = l + r + 1ll >> 1;
            if(mid <= x / mid) l = mid;
            else r = mid - 1;
        }
        return r;
    }
};
```

70. 爬楼梯

70. 爬楼梯

难度 **简单** 928 收藏 分享 切换为英文 关注 反馈

假设你正在爬楼梯。需要 n 阶你才能到达楼顶。

每次你可以爬 1 或 2 个台阶。你有多少种不同的方法可以爬到楼顶呢？

注意：给定 n 是一个正整数。

示例 1:

输入： 2
输出： 2
解释： 有两种方法可以爬到楼顶。
1. 1 阶 + 1 阶
2. 2 阶

示例 2:

输入： 3
输出： 3
解释： 有三种方法可以爬到楼顶。
1. 1 阶 + 1 阶 + 1 阶
2. 1 阶 + 2 阶
3. 2 阶 + 1 阶



```

class Solution {
public:
    int climbStairs(int n) {
        int ans[100] = {0} ;
        ans[0] = 1;
        ans[1] = 1;
        for(int i = 2; i <= n; i++)
        {
            ans[i] = ans[i-1] + ans[i-2];
        }
        return ans[n];
    }
};

```

71. 简化路径

71. 简化路径

难度 中等  126  收藏  分享  切换为英文  关注  反馈

以 Unix 风格给出一个文件的**绝对路径**，你需要简化它。或者换句话说，将其转换为规范路径。

在 Unix 风格的文件系统中，一个点（.）表示当前目录本身；此外，两个点（..）表示将目录切换到上一级（指向父目录）；两者都可以是复杂相对路径的组成部分。更多信息请参阅：[Linux / Unix中的绝对路径 vs 相对路径](#)

请注意，返回的规范路径必须始终以斜杠 / 开头，并且两个目录名之间必须只有一个斜杠 /。最后一个目录名（如果存在）**不能**以 / 结尾。此外，规范路径必须是表示绝对路径的**最短字符串**。

示例 1:

输入："/home/"
 输出："/home"
 解释：注意，最后一个目录名后面没有斜杠。

示例 2:

输入："/../"
 输出："/"
 解释：从根目录向上一级是不可行的，因为根是你到达的最高级。

示例 3:

输入："/home//foo/"
 输出："/home/foo"
 解释：在规范路径中，多个连续斜杠需要用单个斜杠替换。

示例 4:

输入："/a/./b/../../c/"
 输出："/c"

```

class Solution {
public:

```

```

string simplifyPath(string path) {
    if (path.back() != '/') path += '/';
    string res, s;
    for (auto &c : path)
    {
        if (res.empty()) res += c;
        else if (c == '/')
        {
            if (s == "..")
            {
                if (res.size() > 1)
                {
                    res.pop_back();
                    while (res.back() != '/') res.pop_back();
                }
            }
            else if (s != "" && s != ".")
            {
                res += s + '/';
            }
            s = "";
        }
        else
        {
            s += c;
        }
    }
    if (res.size() > 1) res.pop_back();
    return res;
}
};

```

72. 编辑距离

72. 编辑距离

难度 困难

👍 781

❤ 收藏

🗒 分享

🌐 切换为英文

🔔 关注

📝 反馈

给你两个单词 *word1* 和 *word2*，请你计算出将 *word1* 转换成 *word2* 所使用的最少操作数。

你可以对一个单词进行如下三种操作：

1. 插入一个字符
2. 删除一个字符
3. 替换一个字符

示例 1:

```
输入：word1 = "horse", word2 = "ros"
输出：3
解释：
horse -> rorse (将 'h' 替换为 'r')
rorse -> rose (删除 'r')
rose -> ros (删除 'e')
```

示例 2:

```
输入：word1 = "intention", word2 = "execution"
输出：5
解释：
intention -> inention (删除 't')
inention -> enention (将 'i' 替换为 'e')
enention -> exention (将 'n' 替换为 'x')
exention -> exection (将 'n' 替换为 'c')
exection -> execution (插入 'u')
```

```
class Solution {
public:
    int minDistance(string word1, string word2) {
        int n = word1.size(), m = word2.size();
        vector<vector<int>> f(n + 1, vector<int>(m + 1));
        for(int i = 0; i <= n; i++) f[i][0] = i; //边界情况，把第一个字符串变成0，要进行
        i次删除操作
        for(int i = 0; i <= m; i++) f[0][i] = i; //边界情况，把第二个字符串变成0，要进行
        i次删除操作

        for(int i = 1; i <= n; i++)
        {
            for(int j = 1; j <= m; j++)
            {
                f[i][j] = min(f[i - 1][j], f[i][j - 1]) + 1;
                f[i][j] = min(f[i][j], f[i - 1][j - 1] + (word1[i - 1] != word2[j
- 1]));
            }
        }
        return f[n][m];
    }
};
```

73. 矩阵置零

73. 矩阵置零

难度 中等

👁 211

❤ 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

给定一个 $m \times n$ 的矩阵，如果一个元素为 0，则将其所在行和列的所有元素都设为 0。请使用原地算法。

示例 1:

```
输入：
[
  [1,1,1],
  [1,0,1],
  [1,1,1]
]
输出：
[
  [1,0,1],
  [0,0,0],
  [1,0,1]
]
```

示例 2:

```
输入：
[
  [0,1,2,0],
  [3,4,5,2],
  [1,3,1,5]
]
输出：
[
  [0,0,0,0],
  [0,4,5,0],
  [0,3,1,0]
]
```

```
class Solution {
public:
    void setZeroes(vector<vector<int>>& matrix) {
        if (matrix.empty()) return;
        int n = matrix.size(), m = matrix[0].size();
        int col0 = 1, row0 = 1;
        for (int i = 0; i < n; i++)
            if (!matrix[i][0]) col0 = 0;
        for (int i = 0; i < m; i++)
            if (!matrix[0][i]) row0 = 0;
        for (int i = 1; i < n; i++)
            for (int j = 1; j < m; j++)
                if (!matrix[i][j])
                {
                    matrix[i][0] = 0;
                    matrix[0][j] = 0;
                }
    }
}
```

```

        for (int i = 1; i < n; i ++ )
            if (!matrix[i][0])
                for (int j = 1; j < m; j ++ )
                    matrix[i][j] = 0;

        for (int i = 1; i < m; i ++ )
            if (!matrix[0][i])
                for (int j = 1; j < n; j ++ )
                    matrix[j][i] = 0;

        if (!col0)
            for (int i = 0; i < n; i ++ )
                matrix[i][0] = 0;

        if (!row0)
            for (int i = 0; i < m; i ++ )
                matrix[0][i] = 0;
    }
};

```

74. 搜索二维矩阵

74. 搜索二维矩阵

难度 中等  164  收藏  分享  切换为英文  关注  反馈

编写一个高效的算法来判断 $m \times n$ 矩阵中，是否存在一个目标值。该矩阵具有如下特性：

- 每行中的整数从左到右按升序排列。
- 每行的第一个整数大于前一行的最后一个整数。

示例 1:

```

输入：
matrix = [
  [1,   3,  5,  7],
  [10, 11, 16, 20],
  [23, 30, 34, 50]
]
target = 3
输出：true

```

示例 2:

```

输入：
matrix = [
  [1,   3,  5,  7],
  [10, 11, 16, 20],
  [23, 30, 34, 50]
]
target = 13
输出：false

```

```

class Solution {
public:

```

```
bool searchMatrix(vector<vector<int>>& matrix, int target) {
    if(matrix.empty() || matrix[0].empty()) return false;

    int n = matrix.size(), m = matrix[0].size();
    int l = 0, r = n * m - 1;
    while(l < r)
    {
        int mid = l + r >> 1;
        if(matrix[mid / m][mid % m] >= target) r = mid;
        else l = mid + 1;
    }
    if(matrix[l / m][l % m] != target) return false;
    return true;
}

};
```

75. 颜色分类

75. 颜色分类

难度 **中等**  389  收藏  分享  切换为英文  关注  反馈

给定一个包含红色、白色和蓝色，一共 n 个元素的数组，**原地**对它们进行排序，使得相同颜色的元素相邻，并按照红色、白色、蓝色顺序排列。

此题中，我们使用整数 0、1 和 2 分别表示红色、白色和蓝色。

注意:

不能使用代码库中的排序函数来解决这道题。

示例:

输入: [2,0,2,1,1,0]
输出: [0,0,1,1,2,2]

进阶:

- 一个直观的解决方案是使用计数排序的两趟扫描算法。
首先，迭代计算出0、1 和 2 元素的个数，然后按照0、1、2的排序，重写当前数组。
- 你能想出一个仅使用常数空间的一趟扫描算法吗？

```
class Solution {
public:
    void sortColors(vector<int>& nums) {
        int red = 0, blue = nums.size() - 1;
        for(int i = 0; i <= blue; i++)
        {
            if(nums[i] == 0) swap(nums[i], nums[red++]);
            else if(nums[i] == 2) swap(nums[i], nums[blue--]);
            else i++;
        }
    }
};
```

76. 最小覆盖子串

76. 最小覆盖子串

难度 困难

👍 421

📖 收藏

🔗 分享

🌐 切换为英文

🔔 关注

🗉 反馈

给你一个字符串 S 、一个字符串 T ，请在字符串 S 里面找出：包含 T 所有字母的最小子串。

示例：

```
输入：S = "ADOBECODEBANC", T = "ABC"
输出："BANC"
```

说明：

- 如果 S 中不存这样的子串，则返回空字符串 `""`。
- 如果 S 中存在这样的子串，我们保证它是唯一的答案。

```
class Solution {
public:
    string minWindow(string s, string t) {
        unordered_map<char,int> hash;

        for(auto c : t) hash[c] ++;
        int cnt = hash.size();

        string res;
        for(int i = 0,j = 0,c = 0;i < s.size();i ++)
        {
            if(hash[s[i]] == 1) c ++;
            hash[s[i]] --;
            while(hash[s[j]] < 0) hash[s[j ++ ]] ++;
            if(c == cnt)
            {
                if(res.empty() || res.size() > i - j + 1) res = s.substr(j,i - j
+ 1);
            }
        }
        return res;
    }
};
```

77. 组合

77. 组合

难度 中等

👍 242

📖 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

给定两个整数 n 和 k ，返回 $1 \dots n$ 中所有可能的 k 个数的组合。

示例:

输入: $n = 4, k = 2$

输出:

```
[
  [2,4],
  [3,4],
  [2,3],
  [1,2],
  [1,3],
  [1,4],
]
```

```
class Solution {
public:
    vector<vector<int>> ans;
    vector<int> path;

    vector<vector<int>> combine(int n, int k) {
        dfs(0, 1, n, k);
        return ans;
    }

    void dfs(int u, int start, int n, int k)
    {
        if (u == k)
        {
            ans.push_back(path);
            return ;
        }

        for (int i = start; i <= n; i ++ )
        {
            path.push_back(i);
            dfs(u + 1, i + 1, n, k);
            path.pop_back();
        }
    }
};
```

78. 子集

78. 子集

难度 中等

👍 530

📖 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

给定一组不含重复元素的整数数组 *nums*，返回该数组所有可能的子集（幂集）。

说明：解集不能包含重复的子集。

示例：

```
输入：nums = [1,2,3]
输出：
[
  [3],
  [1],
  [2],
  [1,2,3],
  [1,3],
  [2,3],
  [1,2],
  []
]
```

```
class Solution {
public:
    vector<vector<int>> subsets(vector<int>& nums) {
        vector<vector<int>> res;
        int n = nums.size();
        for(int i = 0; i < (1 << n); i++)
        {
            vector<int> temp;
            for(int j = 0; j < n; j++)
                if(i >> j & 1)
                    temp.push_back(nums[j]);
            res.push_back(temp);
        }
        return res;
    }
};
```

79. 单词搜索

79. 单词搜索

难度 中等

👍 377

❤ 收藏

🔗 分享

🌐 切换为英文

👤 关注

📝 反馈

给定一个二维网格和一个单词，找出该单词是否存在于网格中。

单词必须按照字母顺序，通过相邻的单元格内的字母构成，其中“相邻”单元格是那些水平相邻或垂直相邻的单元格。同一个单元格内的字母不允许被重复使用。

示例:

```
board =  
[  
  ['A','B','C','E'],  
  ['S','F','C','S'],  
  ['A','D','E','E']  
]
```

给定 word = "ABCCED", 返回 true

给定 word = "SEE", 返回 true

给定 word = "ABCB", 返回 false

提示:

- board 和 word 中只包含大写和小写英文字母。
- $1 \leq \text{board.length} \leq 200$
- $1 \leq \text{board}[i].\text{length} \leq 200$
- $1 \leq \text{word.length} \leq 10^3$

```
class Solution {  
public:  
    int n,m;  
    int dx[4] = {-1,0,1,0},dy[4] = {0,1,0,-1};  
    bool exist(vector<vector<char>>& board, string word) {  
        if(board.empty() || board[0].empty()) return false;  
        n = board.size(),m = board[0].size();  
  
        for(int i = 0;i < n;i ++)  
            for(int j = 0;j < m;j ++)  
                if(dfs(board,i,j,word,0))  
                    return true;  
        return false;  
    }  
  
    bool dfs(vector<vector<char>> &board,int x,int y,string &word,int u)  
    {  
        if(board[x][y] != word[u]) return false;  
        if(u == word.size() - 1) return true;  
  
        board[x][y] = '.';  
        for(int i = 0;i < 4;i ++)  
        {  
            int a = x + dx[i],b = y + dy[i];  
            if(a >= 0 && a < n && b >= 0 && b < m)
```







```

        if(dfs(board,a,b,word,u + 1))
            return true;
    }
    board[x][y] = word[u];
    return false;
}
};

```

80. 删除排序数组中的重复项 II

80. 删除排序数组中的重复项 II

难度 **中等**  207  收藏  分享  切换为英文  关注  反馈

给定一个排序数组，你需要在**原地**删除重复出现的元素，使得每个元素最多出现两次，返回移除后数组的新长度。

不要使用额外的数组空间，你必须在**原地**修改输入数组并在使用 $O(1)$ 额外空间的条件下完成。

示例 1:

给定 `nums = [1,1,1,2,2,3]`,

函数应返回新长度 `length = 5`，并且原数组的前五个元素被修改为 `1, 1, 2, 2, 3`。

你不需要考虑数组中超出新长度后面的元素。

示例 2:

给定 `nums = [0,0,1,1,1,1,2,3,3]`,

函数应返回新长度 `length = 7`，并且原数组的前五个元素被修改为 `0, 0, 1, 1, 2, 3, 3`。

你不需要考虑数组中超出新长度后面的元素。

```

class Solution {
public:
    int removeDuplicates(vector<int>& nums) {
        if(nums.size() < 3) return nums.size();
        int k = 1;
        for(int i = 2; i < nums.size(); i++)
            if(nums[i] != nums[k - 1])
                nums[++k] = nums[i];
        k++;
        return k;
    }
};

```