[C++版本] 第一周 周日部分

Introduction

第一周，周日部分。

Content

Table

- ✓ 面试题 01.08. 零矩阵 （简单）
- ✓ 剑指Offer61.扑克牌中的顺子（中等）
- ✓ 面试题 16.11. 跳水板（简单）
- ✓ 面试题 01.05. 一次编辑（中等）
- ✓ 面试题 16.15. 珠玑妙算 （简单）
- ✓ 面试题 16.04. 井字游戏（中等）
- ✓ 55. 跳跃游戏 （中等）

- [x] 48. 旋转图像 （中等）经典
- [x] 54. 螺旋矩阵（中等）经典
- [x] 240. 搜索二维矩阵 II （中等）经典

面试题 01.08. 零矩阵

方法一：位图

time $O(n)$

space $O(n)$

```
class Solution {
 public:
  void setZeroes(vector<vector<int>> &matrix) {
      int n = matrix.size();
      if (n == 0) return;
      int m = matrix[0].size();
      vector<bool> zeroRows(n, false);
      vector<bool> zeroColumns(m, false);

      for (int i = 0; i < n; ++i) {
          for (int j = 0; j < m; ++j) {
              if (matrix[i][j] == 0) {
                  zeroRows[i] = true;
                  zeroColumns[j] = true;
              }
          }
      }

      for (int i = 0; i < n; ++i) {
          for (int j = 0; j < m; ++j) {
              if (zeroRows[i] || zeroColumns[j]) {
                  matrix[i][j] = 0;
              }
          }
      }
  }
};
```

方法二：哈希表

time $O(n)$

space $O(n)$

```
class Solution {
  public:
    void setZeroes(vector<vector<int>> &matrix) {
        set<int> row_set;
        set<int> col_set;
        for (int i = 0; i < matrix.size(); ++i) {
            for (int j = 0; j < matrix[i].size(); ++j) {
                if (matrix[i][j] == 0) {
                    row_set.insert(i);
                    col_set.insert(j);
                }
            }
        }

        for (int i = 0; i < matrix.size(); ++i) {
            for (int j = 0; j < matrix[i].size(); ++j) {
                if (row_set.find(i) != row_set.end()
                    || col_set.find(j) != col_set.end()) {
                    matrix[i][j] = 0;
                }
            }
        }
    }
};
```

剑指Offer61.扑克牌中的顺子（中等）

time $O(n)$

space $O(1)$

```
class Solution {
 public:
  bool isStraight(vector<int> &nums) {
      vector<bool> dup(14, false);
      int min = 100;
      int max = -1;
      for (int i = 0; i < 5; ++i) {
          if (nums[i] != 0) {
              if (dup[nums[i]]) {
                  return false;
              } else {
                  dup[nums[i]] = true;
              }
              if (nums[i] > max) max = nums[i];
              if (nums[i] < min) min = nums[i];
          }
```

```
        }
        return max - min < 5;
    }
};
```

面试题 16.11. 跳水板（简单）

time $O(n)$

space $O(1)$

```
class Solution {
  public:
    vector<int> divingBoard(int shorter, int longer, int k) {
        if (k == 0) return {};
        if (shorter == longer) return {k * shorter};

        vector<int> result;
        for (int i = 0; i <= k; ++i) {
            result.push_back(i * longer + (k - i) * shorter);
        }

        return result;
    }
};
```

面试题 01.05. 一次编辑

time $O(n)$

space $O(1)$

```
class Solution {
 public:
  bool oneEditAway(string first, string second) {
      int n = first.size();
      int m = second.size();
      if (abs(n - m) > 1) return false;

      int i = 0;
      int j = 0;
      while (i < n && j < m && first[i] == second[j]) {
          i++;
          j++;
      }
```

```
        if (n == m) {
            i++;
            j++;
        } else if (n > m) {
            i++;
        } else {
            j++;
        }

        while (i < n && j < m) {
            if (first[i] != second[j]) {
                return false;
            }
            i++;
            j++;
        }
        return true;
    }
};
```

面试题 16.15. 珠玑妙算

time $O(n^2)$

space $O(n)$

```
class Solution {
 public:
  vector<int> masterMind(string solution, string guess) {
      int n = solution.size();
      vector<bool> hited(n, false);
      vector<bool> used(n, false);

      // 先计算猜中的
      int hitCount = 0;
      for (int i = 0; i < n; ++i) {
          if (guess[i] == solution[i]) {
              hited[i] = true;
              used[i] = true;
              hitCount++;
          }
      }

      // 再计算伪猜中的
      int fakeHitCount = 0;
      for (int i = 0; i < n; ++i) {
          if (hited[i]) continue;
```

```
        for (int j = 0; j < n; ++j) {
            if (guess[i] == solution[j] && !used[j]) {
                used[j] = true;
                fakeHitCount++;
                break;
            }
        }
    }
    return {hitCount, fakeHitCount};
  }
};
```

面试题 16.04. 井字游戏

time $O(n^2)$

space $O(1)$

```
class Solution {
 public:
  string tictactoe(vector<string> &board) {
    int n = board.size();
    bool determined = false;

    // 检查行
    for (int i = 0; i < n; ++i) {
        if (board[i][0] == ' ') continue;
        determined = true;
        for (int j = 1; j < n; ++j) {
            if (board[i][j] != board[i][0]) {
                determined = false;
                break;
            }
        }

        string res(1, board[i][0]);
        if (determined) return res;
    }

    // 检查列
    for (int j = 0; j < n; ++j) {
        if (board[0][j] == ' ') continue;
        determined = true;
        for (int i = 1; i < n; ++i) {
            if (board[i][j] != board[0][j]) {
                determined = false;
                break;
                }
            }
            string res(1, board[0][j]);
            if (determined) return res;
        }

        // 检查对角线：左上 -> 右下
        if (board[0][0] != ' ') {
            int i = 1;
            int j = 1;

            determined = true;
            while (i < n && j < n) {
                if (board[i][j] != board[0][0]) {
                    determined = false;
                    break;
                }
                i++;
                j++;
            }
            string res(1, board[0][0]);
            if (determined) return res;
        }

        // 检查对角线：左下 -> 右上
        if (board[n - 1][0] != ' ') {
            int i = n - 2, j = 1;
            determined = true;
            while (i >= 0 && j < n) {
                if (board[i][j] != board[n - 1][0]) {
                    determined = false;
                    break;
                }
                i--, j++;
            }
            string str(1, board[n - 1][0]);
            if (determined) return str;
        }

        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < board[i].size(); ++j) {
                if (board[i][j] == ' ') return "Pending";
            }
        }
        return "Draw";
    }
};
```

55. 跳跃游戏

方法一 临时变量

time $O(n)$

space $O(1)$

```
class Solution {
public:
    bool canJump(vector<int>& nums) {
        int reachedMax = 0;
        for (int i = 0; i < nums.size(); ++i) {
            if (i > reachedMax) return false;
            if (i + nums[i] > reachedMax) {
                reachedMax = i + nums[i];
            }
            if (reachedMax >= nums.size() - 1) return true;
        }
        return false;
    }
};
```

方法二 标记数组

time $O(n)$

space $O(n)$

批注：争哥提了一嘴，但是没有实现，这里把它实现

```
class Solution {
 public:
  bool canJump(vector<int> &nums) {
      vector<bool> vec(nums.size(), false);
      int reachedMax = 0;
      for (int i = 0; i < nums.size(); ++i) {
          if (i > reachedMax) return false;
          vec[i] = true;
          if (i + nums[i] > reachedMax) {
              reachedMax = i + nums[i];
          }
      }
      return vec[nums.size() - 1];
  }
};
```

48. 旋转图像 （中等）经典

规律

- 1 上下翻转—— 行变，列不变

  a(i, j) -> a(n-i-1, j)

- 2 左右翻转—— 行不变，列变

  a(i, j) -> a(i, n - j -1)

- 3 \对角线翻转—— 行列交换

  a(i, j) -> a(j, i)

- 4 /对角线翻转—— 行列交换，并且都变

  a(i, j) -> a(n - 1 - j, n - 1 - i)

- 5 顺时针旋转90°——1 + 3

  先上下翻转，再\对角线翻转，即为

  a(i, j) -> a(n-i-1, j) -> a(j, n-i-1)

- 6 顺时针旋转180°， 1 + 2

  先上下翻转，再左右翻转，即为

  a(i, j) -> a(n-i-1, j) -> a(n-i-1, n -1-j)

- 7 顺时针旋转270°， 2 + 3

  先左右翻转，再\对角线翻转，即为

  a(i, j) -> a(i, n - j -1) -> a(n - j -1, i)

方法一：临时辅助数组

time：$O(n^2)$

space：$O(n^2)$

```
class Solution {
public:
    void rotate(vector<vector<int>>& matrix) {
        int n = matrix.size();
        vector<vector<int>> vec(n, vector<int>(n, 0));

        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                vec[j][n - 1 - i] = matrix[i][j];
            }
        }

        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                matrix[i][j] = vec[i][j];
            }
        }
    }
};
```

方法二：原地两次翻转代替旋转

time：$O(n^2)$

space：$O(1)$

```
class Solution {
public:
    void rotate(vector<vector<int>>& matrix) {
        int n = matrix.size();
        // 上下翻转
        for (int i = 0; i < n / 2; ++i) {
            for (int j = 0; j < n; ++j) {
                swap(matrix[n - 1 - i][j], matrix[i][j]);
            }
        }

        // \对角线
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < i; ++j) {
                swap(matrix[j][i], matrix[i][j]);
            }
        }
    }
};
```

批注：C++中直接就有swap()方法，所以不需要自己实现。

方法三：直接原地翻转

time：$O(n^2)$

space：$O(1)$

```
class Solution {
 public:
  void rotate(vector<vector<int>>& matrix) {
      int n = matrix.size();

      // 左上角
      int s1_i = 0;
      int s1_j = 0;

      while (n > 1) {
        // 右上角
        int s2_i = s1_i;
        int s2_j = s1_j + n - 1;

        // 右下角
        int s3_i = s1_i + n - 1;
        int s3_j = s1_j + n - 1;

        // 左下角
        int s4_i = s1_i + n - 1;
        int s4_j = s1_j;

        for (int move = 0; move <= n - 2; ++move) {
            int p1_i = s1_i;
            int p1_j = s1_j + move;

            int p2_i = s2_i + move;
            int p2_j = s2_j;

            int p3_i = s3_i;
            int p3_j = s3_j - move;

            int p4_i = s4_i - move;
            int p4_j = s4_j;

            swap(matrix,
                 p1_i, p1_j,
                 p2_i, p2_j,
                 p3_i, p3_j,
                 p4_i, p4_j);
        }
        s1_i++;
        s1_j++;
        n -= 2;
    }
}

void swap(vector<vector<int>>& matrix,
          int p1_i, int p1_j,
          int p2_i, int p2_j,
          int p3_i, int p3_j,
          int p4_i, int p4_j) {

    int tmp = matrix[p1_i][p1_j];
    matrix[p1_i][p1_j] = matrix[p4_i][p4_j];
    matrix[p4_i][p4_j] = matrix[p3_i][p3_j];
    matrix[p3_i][p3_j] = matrix[p2_i][p2_j];
    matrix[p2_i][p2_j] = tmp;
}
```

```
};
```

54. 螺旋矩阵（中等）经典

time $O(n)$

space $O(1)$

```
class Solution {
 public:
  vector<int> spiralOrder(vector<vector<int>> &matrix) {
      vector<int> res;
      if (matrix.empty()) return res;

      int m = matrix.size();
      int n = matrix[0].size();

      int left = 0;
      int right = n - 1;
      int top = 0;
      int bottom = m - 1;

      while (left <= right && top <= bottom) {
          // 上边: 左到右
          for (int j = left; j <= right; ++j) {
              res.push_back(matrix[top][j]);
          }

          // 右边: 上到下
          for (int i = top + 1; i <= bottom; ++i) {
              res.push_back(matrix[i][right]);
          }

          // 下边: 右到左
          if (top != bottom) { // 注意避免对于一行数据来回扫
              for (int j = right - 1; j >= left; --j) {
                  res.push_back(matrix[bottom][j]);
              }
          }

          // 左边: 下到上
          if (left != right) { // 注意避免对于一列数据来回扫
              for (int i = bottom - 1; i >= top + 1; --i) {
                  res.push_back(matrix[i][left]);
              }
          }
          left++;
```

```
            right--;
            top++;
            bottom--;
        }
        return res;
    }
};
```

240. 搜索二维矩阵 II

time $O(m + n)$

space $O(1)$

```
// 写法一：右上 -> 左下
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        int h = matrix.size();
        int w = matrix[0].size();
        int i = 0;
        int j = w - 1;

        while (i <= h - 1 && j >= 0) {
            if (matrix[i][j] == target) {
                return true;
            }
            if (matrix[i][j] > target) {
                j--;
                continue;
            }

            if (matrix[i][j] < target) {
                i++;
                continue;
            }
        }
        return false;
    }
};

// 写法二：左下 -> 右上
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        int h = matrix.size();
        int w = matrix[0].size();
        int i = h - 1;
        int j = 0;

        while (i >= 0 && j <= w - 1) {
            if (matrix[i][j] == target) {
                return true;
            }
            if (matrix[i][j] > target) {
                i--;
                continue;
            }

            if (matrix[i][j] < target) {
                j++;
                continue;
            }
        }
        return false;
    }
};
```