春节7天练|Day6：图

你好，我是王争。初六好！

为了帮你巩固所学，真正掌握数据结构和算法，我整理了数据结构和算法中，必知必会的30个代码实现，分7天发布出来，供你复习巩固所用。今天是第六篇。

和之前一样，你可以花一点时间，来手写这些必知必会的代码。写完之后，你可以根据结果，回到相应章节，有针对性地进行复习。做到这些，相信你会有不一样的收获。

关于图的几个必知必会的代码实现

图

- 实现有向图、无向图、有权图、无权图的邻接矩阵和邻接表表示方法
- 实现图的深度优先搜索、广度优先搜索
- 实现Dijkstra算法、A*算法
- 实现拓扑排序的Kahn算法、DFS算法

对应的LeetCode练习题（@Smallfly 整理）

- Number of Islands（岛屿的个数）

英文版：https://leetcode.com/problems/number-of-islands/description/

中文版：https://leetcode-cn.com/problems/number-of-islands/description/

- Valid Sudoku（有效的数独）

英文版：https://leetcode.com/problems/valid-sudoku/

中文版：https://leetcode-cn.com/problems/valid-sudoku/

做完题目之后，你可以点击“请朋友读”，把测试题分享给你的朋友，说不定就帮他解决了一个难题。

祝你取得好成绩！明天见！

精选留言：

- 李皮皮皮皮皮 2019-02-10 08:07:02
  图很复杂 [1赞]

- 你看起来很好吃 2019-02-10 16:07:43

  岛屿个数python实现(广度优先搜索算法)：

```
def numIslands(self, grid):
if not grid:
return 0

count = 0
for i in range(len(grid)):
for j in range(len(grid[0])):
if grid[i][j] == '1':
self.dfs(grid, i, j)
count += 1
return count

def dfs(self, grid, i, j):
if i<0 or j<0 or i>=len(grid) or j>=len(grid[0]) or grid[i][j] != '1':
return
grid[i][j] = '#'
self.dfs(grid, i+1, j)
self.dfs(grid, i-1, j)
self.dfs(grid, i, j+1)
self.dfs(grid, i, j-1)
```

- _CountingStars 2019-02-10 13:55:42

  有效的数独 go 语言实现

```
package main

import (
"fmt"
)

func hasRepeatedNumbers(numbers []byte) bool {
```

```
var numbersExistFlag [9]bool
for _, num := range numbers {
if num == '.' {
continue
}
index := num - '0' - 1
if numbersExistFlag[index] {
return true
}
numbersExistFlag[index] = true
}
return false
}

func isValidSudoku(board [][]byte) bool {
sudokuSize := 9
sudokuUnitSize := 3
for _, line := range board {
if hasRepeatedNumbers(line) {
return false
}
}

for columnIndex := 0; columnIndex < sudokuSize; columnIndex++ {
columnNumbers := make([]byte, 0)
for lineIndex := 0; lineIndex < sudokuSize; lineIndex++ {
columnNumbers = append(columnNumbers, board[lineIndex][columnIndex])
}
if hasRepeatedNumbers(columnNumbers) {
return false
}
}
```

```
sudokuUnitCountEachLine := sudokuSize / sudokuUnitSize
for i := 0; i < sudokuUnitCountEachLine; i++ {
for j := 0; j < sudokuUnitCountEachLine; j++ {
sudokuUnitNumbers := make([]byte, 0)
for _, line := range board[i*3 : (i+1)*3] {
sudokuUnitNumbers = append(sudokuUnitNumbers, line[j*3:(j+1)*3]...)
}

if hasRepeatedNumbers(sudokuUnitNumbers) {
return false
}
}
}

return true
}

func main() {
testData1 := [][]byte{
{'5', '3', '.', '.', '7', '.', '.', '.', '.'},
{'6', '.', '.', '1', '9', '5', '.', '.', '.'},
{'.', '9', '8', '.', '.', '.', '.', '6', '.'},
{'8', '.', '.', '.', '6', '.', '.', '.', '3'},
{'4', '.', '.', '8', '.', '3', '.', '.', '1'},
{'7', '.', '.', '.', '2', '.', '.', '.', '6'},
{'.', '6', '.', '.', '.', '.', '2', '8', '.'},
{'.', '.', '.', '4', '1', '9', '.', '.', '5'},
{'.', '.', '.', '.', '8', '.', '.', '7', '9'}}
fmt.Println(isValidSudoku(testData1))
}
```

kai 2019-02-10 13:02:30

今天根据老师的课程，总结了一下图的相关知识点，然后用代码实现了一下图的相关的算法，感觉图还是要难于其他数据结构，需要接着多练习~

- 纯洁的憎恶 2019-02-10 10:19:57

  1.在邻接矩阵中找出连通图个数即可。在每个顶点执行DFS或BFS，执行次数即为岛屿数，也可以使用并查集。

  2. 依次考察9 9数独各行各列是否有重复数字（可以用9位数组统计），然后再考察每个3 3子矩阵是否有重复数字。都没有则成功。

- 峰 2019-02-10 10:03:02

  island个数，从一个点从发，判断一个island的逻辑是如果本身点是water，那么必然不是island，如果是陆地，说明它能扩展成一个island，那么向上下左右进行扩展，然后再以扩展的陆地点又一直递归扩展，直到所有边界为0。而判断island的个数，就在此基础上去遍历所有点，并加上一个boolean[][]记录每个点是否已经被遍历或者扩展过。

- C_love 2019-02-10 09:56:34

  Valid Sudoku

  ```
  class Solution {
  public boolean isValidSudoku(char[][] board) {
  for (int row = 0; row < 9; row++) {
  for (int col = 0; col < 9; col++) {
  if (board[row][col] == '.') continue;
  if (!isValid(board, row, col)) return false;
  }
  }
  return true;
  }

  private boolean isValid(char[][] board, final int row, final int col){
  char target=board[row][col];
  //check rows
  for (int i = 0; i < 9; i++) {
  if (i == row) continue;
  if (board[i][col] == target) return false;
  }
  ```

```
//check cols
for (int i = 0; i < 9; i++) {
if (i == col) continue;
if (board[row][i] == target) return false;
}

//check 3*3
int rowStart = row / 3 * 3, colStart = col / 3 * 3;
for (int i = rowStart; i < rowStart + 3; i++) {
for (int j = colStart; j < colStart + 3; j++) {
if (i == row && j == col) continue;
if (board[i][j] == target) return false;
}
}

return true;
}
}
```

- ext4 2019-02-10 09:35:07

  有效的数独

```
class Solution {
public:
bool isValidSudoku(vector< vector<char> >& board) {
set<char> numset;
for (int i = 0; i < 9; i++) {
numset.clear();
for (int j = 0; j < 9; j++) {
char val = board[i][j];
if (val != '.') {
if (numset.count(val) != 0) return false;
```

```
numset.insert(val);
}
}
}
for (int j = 0; j < 9; j++) {
numset.clear();
for (int i = 0; i < 9; i++) {
char val = board[i][j];
if (val != '.') {
if (numset.count(val) != 0) return false;
numset.insert(val);
}
}
}
for (int i = 0; i < 3; i++) {
for (int j = 0; j < 3; j++) {
numset.clear();
for (int p = 0; p < 3; p++) {
for (int q = 0; q < 3; q++) {
char val = board[i * 3 + p][j * 3 + q];
if (val != '.') {
if (numset.count(val) != 0) return false;
numset.insert(val);
}
}
}
}
}
return true;
}
};
```