春节7天练讲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

kai

实现图的深度优先搜索、广度优先搜索：

```
import java.util.ArrayList;
import java.util.HashSet;
import java.util.LinkedList;
import java.util.Queue;

public class BFSAndDFS {

class Node {
public int value; //Node 值
public int in; //入度：指向该节点的边有几条
public int out; //出度：指向其他节点的边有几条
public ArrayList<Node> nexts;
public ArrayList<Edge> edges;

public Node(int value) {
```

```
this.value = value;
this.in = 0;
this.out = 0;
this.nexts = new ArrayList<>();
this.edges = new ArrayList<>();
}
}

public static void bfs(Node node) {
if (node == null) {
return;
}

Queue<Node> queue = new LinkedList<>();
HashSet<Node> set = new HashSet<>();
queue.add(node);
set.add(node);
while (!queue.isEmpty()) {
Node cur = queue.poll();
System.out.print(cur.value + " ");
for (Node next : cur.nexts) {
if (!set.contains(next)) {
queue.add(next);
set.add(next);
}
}
}
}

public static void dfs(Node node) {
if (node == null) {
return;
}

Stack<Node> stack = new Stack<>();
HashSet<Node> set = new HashSet<>();
stack.push(node);
set.add(node);
System.out.print(node.value + " ");
while (!stack.isEmpty()) {
Node cur = stack.pop();
for (Node next : cur.nexts) {
if (!set.contains(next)) {
stack.push(cur);
stack.push(next);
set.add(next);
System.out.print(next.value + " ");
break;
}
}
}
```

```
}
}
```
2019-02-11 10:54

kai

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~

2019-02-10 13:02

李皮皮皮皮皮

图很复杂

2019-02-10 08:07

Nereus

并查集—go实现

```
func numIslands(grid [][]byte) int {
if len(grid) == 0 {
return 0
}

N := len(grid)*len(grid[0]) + 1

u := NewUnionSet(N)

for i := 0; i < len(grid); i ++ {
for j := 0; j < len(grid[i]); j ++ {
if grid[i][j] == '1' {
// 联通下边
if i+1 < len(grid) {
if grid[i+1][j] == '1' {
u.join(i*len(grid[i])+j, (i+1)*len(grid[i])+j)
}
}

// 联通右边
if j+1 < len(grid[i]) {
if grid[i][j+1] == '1' {
u.join(i*len(grid[i])+j, i*len(grid[i])+j+1)
}
}
} else {
u.join(i*len(grid[i])+j, N-1)
}
}
}

return u.counts() -1
}

type UnionSet []int

func NewUnionSet(n int) UnionSet {
var u UnionSet
```

```
u = make([]int, n)
for i := 0; i < len(u); i ++ {
u[i] = i
}
return u

}

func (u UnionSet) find(i int) int {
tmp := i
for u[tmp] != tmp {
tmp = u[tmp]
}

j := i
for j != tmp {
tt := u[j]
u[j] = tmp
j = tt
}

return tmp
}

func (u UnionSet) connected(i, j int) bool {
return u.find(i) == u.find(j)
}

func (u UnionSet) counts() int {
var count int
for idx, rec := range u {
if idx == rec {
count++
}
}
return count
}

func (u UnionSet) join(i, j int) {
x, y := u.find(i), u.find(j)
if x != y {
if y > x {
u[x] = y
} else {
u[y] = x
}
}
}
```

2019-02-14 17:34

拉欧

Valid Sudoku（有效的数独）go语言实现

```
func isValidSudoku(board [][]byte) bool {

isValid:=true
for i:=0;i<9;i++{
for j:=0;j<9;j++{
if board[i][j]=='.' {
continue
}else{
if !judgeLine(board,i,j){
return false
}
}
}
}

return isValid
}

func judgeLine(board [][]byte,i,j int) bool{
hash:=make(map[byte]int,9)
for k:=0;k<9;k++{
if board[i][k]!='.'{
if hash[board[i][k]]==0{
hash[board[i][k]]=1
}else{
return false
}
}
}
hash=make(map[byte]int,9)
for k:=0;k<9;k++{
if board[k][j]!='.' {
if hash[board[k][j]]==0{
hash[board[k][j]]=1
}else{
return false
}
}
}
hash=make(map[byte]int,9)
for m:=i/3*3;m<i/3*3+3;m++{
for n:=j/3*3;n<j/3*3+3;n++{
if board[m][n]!='.'{
if hash[board[m][n]]==0{
hash[board[m][n]]=1
}else{
return false
}
}
```

```
}
}
return true

}
```

2019-02-14 14:18

拉欧

Number of Islands（岛屿的个数）go语言实现，亲测通过：

```
func numIslands(grid [][]byte) int {

isSearch:=make([][]int,len(grid))
island:=0
for i:=0;i<len(isSearch);i++{
isSearch[i]=make([]int,len(grid[0]))
}
for i,line:=range grid{
for j,_:=range line{
if isSearch[i][j]==0 && grid[i][j]=='1'{
Search(grid,isSearch,i,j)
island++
}

}
}
return island
}

func Search(grid [][]byte,isSearch [][]int, i int,j int){
if isSearch[i][j]==1{
return
}
isSearch[i][j]=1
if grid[i][j]=='1'{
if i>=1{
Search(grid,isSearch,i-1,j)
}
if i<len(grid)-1{
Search(grid,isSearch,i+1,j)
}
if j>=1{
Search(grid,isSearch,i,j-1)
}
if j<len(grid[0])-1{
Search(grid,isSearch,i,j+1)
}
}else{
return
}
```

```
}
```

2019-02-14 10:45

molybdenum

island 我用的深搜，把所有的1探索，用visited保存访问过访问的，搜索次数便是岛屿个数

2019-02-11 13:10

小美

岛屿数Java实现

```
public int numIslands(char[][] grid) {
int m = grid.length;
if (m == 0) return 0;
int n = grid[0].length;

int ans = 0;
for (int y = 0; y < m; ++y)
for (int x = 0; x < n; ++x)
if (grid[y][x] == '1') {
++ans;
dfs(grid, x, y, n, m);
}

return ans;
}

private void dfs(char[][] grid, int x, int y, int n, int m) {
if (x < 0 || y < 0 || x >= n || y >= m || grid[y][x] == '0')
return;
grid[y][x] = '0';
dfs(grid, x + 1, y, n, m);
dfs(grid, x - 1, y, n, m);
dfs(grid, x, y + 1, n, m);
dfs(grid, x, y - 1, n, m);
}
```

2019-02-11 10:30

黄丹

已经初六啦，就快要到去学校的时间了，难受。

图的邻接矩阵表示法是使用一个二维数组int[0..n-1][0...n-1]来保存顶点和边的，对于无权图，1表示有边，0表示两个顶点没有变，有权图，值代表权重。

图的邻接表则是采用数组+链表的结构来表示的，数组里存的是顶点，链表存储的是边的信息，当然链表也可以换做二叉搜索树，散列表等高效查找的数据结构。

今天的两道leetcode题的解题思路和代码如下：

1. Number of Islands （岛屿的个数）

解题思路：遍历数组，遇到1时，使用深度/广度遍历，将连通的1都置为0，然后将岛屿个数加1.

代码：https://github.com/yyxd/leetcode/blob/master/src/leetcode/graph/Problem200_NumberofIslands.java

2. Valid Sudoku （有效的数独）

解题思路：emm，不知道为什么这道题要放在图论的专题下，我的解法就是横着一行行判断，竖着一列列的判断，然后每个3*3的子块进行判断。没有用到图的知识。

代码：https://github.com/yyxd/leetcode/blob/master/src/leetcode/graph/Problem36_ValidSudoku.java

2019-02-10 23:05

虎虎

基于临接表实现的联通分量求法， go 语言实现：

```
package graph_basics

type Components struct {
graph Graph
visited []bool
id []int
ccount int
}

func InitComponents(g Graph) *Components {
return &Components{
graph: g,
visited: make([]bool, g.V()),
id: make([]int, g.V()),
ccount: 0,
}
}

func (c *Components) dfs(index int) {
c.visited[index] = true
c.id[index] = c.ccount
adj := c.graph.Iterator(index)
for i := range adj {
if !c.visited[adj[i]] {
c.dfs(adj[i])
}
}
}

func (c *Components) CalculateComponents() {
for i := 0; i < c.graph.V(); i++ {
if c.visited[i] {
continue
}
c.dfs(i)
c.ccount++
}
}

func (c *Components) Count() int {
return c.ccount
}

func (c *Components) IsConnected(p int, q int) bool {
return c.id[p] == c.id[q]
}
```

临接表的实现：

```
package graph_basics

import "fmt"
```

```
type SparseGraph struct {
v int
e int
direct bool
g [][]int
}

func InitSparseGraph(n int, direct bool) *SparseGraph {
graph := make([][]int, n)
return &SparseGraph{
v: n,
e: 0,
direct: direct,
g: graph,
}
}

func (sg *SparseGraph) V() int {
return sg.v
}

func (sg *SparseGraph) E() int {
return sg.e
}

func (sg *SparseGraph) AddEdge(p int, q int) {
sg.g[p] = append(sg.g[p], q)
if !sg.direct {
sg.g[q] = append(sg.g[q], p)
}

sg.e++
}

func (sg *SparseGraph) HasEdge(p int, q int) bool {
for i := 0; i < len(sg.g[p]); i++ {
if sg.g[p][i] == q {
return true
}
}
return false
}

func (sg *SparseGraph) Show() {
for i := range sg.g {
fmt.Printf("vertex %d :\t", i)
for j := range sg.g[i] {
fmt.Printf("%d\t", sg.g[i][j])
}
fmt.Println()
```

```
}
}

func (sg *SparseGraph) Iterator(v int) []int {
return sg.g[v]
}
```
2019-02-10 21:32

你看起来很好吃

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)
```
2019-02-10 16:07

_CountingStars

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))
}
```

2019-02-10 13:55

纯洁的憎恶

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。

2019-02-10 10:19

峰

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。

2019-02-10 10:03

C_love

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;
}
```

}
2019-02-10 09:56

ext4

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;
}
};
```

2019-02-10 09:35